CENTRO DE INVESTIGACION Y ESTUDIOS AVANZADOS

DEL INSTITUTO POLITECNICO NACIONAL

DEPARTAMENTO DE INGENIERIA ELECTRICA

SECCION COMPUTACION

SIMULACION DE SISTEMAS DISCRETOS

TESIS QUE PRESENTA EL ING. FELIPE LOPEZ SUAREZ, PARA OBTENER EL GRADO DE MAESTRO EN CIENCIAS EN LA ESPECIALIDAD DE INGENIERIA ELECTRICA.

MEXICO, D.F. FEBRERO DE 1991

AGRADECIMIENTOS:

A LA UNESCO POR LA AYUDA ECONOMICA QUE ME PROPORCIONO DURANTE MIS ESTUDIOS DE MAESTRIA.

AGRADEZCO DE FORMA ESPECIAL AL DR. ENRIQUE ARCE MEDINA, POR SU GRAN AMISTAD Y AYUDA QUE ME BRINDO PARA EL DESARROLLO DE ESTE TRABAJO.

TAMBIEN AL DR. MANUEL GUZMAN RENTERIA POR SUS CONSEJOS EN LOS INICIOS DE LA CONSTRUCCION DE ESTE PROYECTO. Y POR LA GRAN AYUDA QUE ME BRINDO AL INICIO DE MIS ESTUDIOS DE MAESTRIA.

AGRADEZCO A LA DRA. ANA MARIA MARTINEZ POR SU AMISTAD, CONSEJOS Y ENSAÑANZAS DURANTE MIS ESTUDIOS, TAMBIEN POR SU COLABORACION EN LA REVISION DE ESTA TESIS.

A LOS PROFESORES M.C. OSCAR OLMEDO Y M.C. FELIU SAGOLS POR SU LECTURA, CONSEJOS Y AYUDA PARA MEJORAR EL ESCRITO.

DEDICO CON TODO CARIÑO A MIS PADRES: DR. SERGIO LOPEZ CERVANTES Y SRA. SONIA SUAREZ MARTINEZ, AGRADECIENDOLES TODO SU CARIÑO, PACIENCIA, APOYO MORAL Y ECONOMICO QUE ME HAN BRINDADO TODO MI VIDA.

A MIS HERMANOS: SONIA, JUDITH, JOSE ANTONIO Y AGUSTIN, POR SU APOYO MORAL QUE ME HAN OTORGADO.

AGRADEZCO A BELINDA VIZUET R. POR SU CARIÑO, COMPAÑIA Y AYUDA QUE ME HA BRINDADO EN TODO MOMENTO.

FELIPE LOPEZ SUAREZ

INDICE

CAPITULO I.

.1. INTRODUCCION

La simulación es una de las mas poderosas herramientas de analisis disponibles para aquellas personas que son responsables del diseño y operacion de sistemas complejos. Utilizando software especializado, la simulación predice el comportamiento de los sistemas por calculo de movimientos e interacciones entre los componentes del sistema. Evaluando el flujo de las partes dentro de las maquinas y estaciones de trabajo, el analista puede evaluar los planes fisicos, selección de equipos y procedimientos de operación; efectuando experimentos en el modelo, en vez del sistema. La simulación lo hace posible al examinar cambios, o nuevos diseños antes de efectuarse la instalacion en el sistema real.

Debido al poder de analisis de sistemas que presenta la simulacion se desarrolló el simulador de eventos discretos que se presenta en esta tesis **SIMDISC**, con el principal objetivo de ser interactivo con el usuario. Es decir el usuario unicamente se limita a establecer la red de actividades a simular y las propiedades de cada actividad. Por lo consiguiente esta herramienta no requiere de la introducción de lineas de programación en un código o lenguage especializado, haciendolo muy accesible a personas con pocos conocimientos de programación.

El simulador está limitado a cierto tipo de sistemas, que son los sistemas de colas, o de lineas de espera, habiendo una amplia gama de procesos industriales que cuentan con este tipo de sistemas, como el tránsito de clientes en bancos, centros comerciales, etc.

La ventaja que ofrece este simulador sobre los lenguajes de simulación existentes tales como GPSS, GASP, SIMULA, etc. es que el usuario no requiere del conocimiento de ningun lenguaje, para la utlización del simulador, haciendolo por consiguiente ideal para fines didácticos y prácticos. Otra de las ventajas es que se puede observar en la pantalla de la computadora como ocurren los eventos en el sistema a traves del tiempo, haciendo posible analizar mas especificamente el desarrollo del sistema.

Una vez terminada la simulación SIMDISC da una serie de parametros importantes que miden el desarrollo del mismo, y estos resultados se pueden obtener tanto en la pantalla como en la impresora. Los resultados pueden ser desplegados en cualquier momento de la simulación con el fin de rastrear el comportamiento de las actividades a traves del tiempo del sistema bajo estudio.

I.2. ANTECEDENTES.

Simulación es una técnica numerica que realiza experimentos en una computadora digital. Estos experimentos involucran ciertos tipos de modelos matematicos y lógicos que describen el comportamiento de sistemas de negocios, económicos, sociales, biológicos, físicos o químicos a traves de largos periodos de tiempo [2].

Una simulacion es una imitación de la operación de un proceso del mundo real o de un sistema hipotético atraves del tiempo. Involucra la generación de una historia artificial del sistema, y la observación de dicha historia nos proporciona las inferencias de las caracteristicas de operación como ocurriria en el mundo real.

La simulación trata primordialmente con dos objetivos; primero el interes de la investigación y la explicación del comportamiento de los sistemas para diferentes estímulos a través de las respuestas y segundo, optimización y control. Así, si el modelo de simulación es adecuadamente implementado y validado para un cierto rango de operación, se puede inferir de los resultados lo mismo que para el sistema real dentro del mismo rango de operacion.

Cambios potenciales en el sistema pueden ser primeramente simulados para predecir el impacto en la ejecucion del sistema. La simulacion puede ser tambien utilizada para el estudio de sistemas en un estado de diseño previo a la construccion del sistema.

También puede ser utilizada como una herramienta de análisis o para la predicción de efectos en los cambios del sistema.

Muchos de los sistemas del mundo real son bastante complejos, cuyos modelos son virtualmente imposibles de resolver matemáticamente. En estos casos, la simulación mediante una computadora puede ser utilizada para imitar el funcionamiento del sistema a traves del tiempo. Se obtienen resultados como si el sistema real hubiese sido observado. Los resultados de la simulacion son usados para estimar las medidas de desarrollo en el sistema.

Aunque la simulación es una herramienta de analisis en muchos casos, el analista del sistema debe considerar las ventajas y desventajas antes de proponer la metodologia de un caso en particular. Las ventajas de la simulación son las siguientes:

1. Una vez construido el modelo, este puede ser repetidamente analizado para proponer diseños.

2. Los resultados de la simulacion son menos costosos que experimentar con el sistema real.

3. Los metodos de simulación son usualmente mas fáciles de aplicar que los métodos analiticos. Debido a esto, existen mas usuarios de métodos de simulación que de metodos analiticos.

Las desventajas a considerar antes de utilizar la simulacion son:

1. Los modelos de simulación para computadoras digitales requieren de una gran cantidad de tiempo para su construccion y validacion.

2. La simulacion es utilizada inadecuadamente en sistemas, en los cuales las técnicas analíticas son suficientes para modelar el sistema. Esta situación ocurre a los usuarios que llegan a familiarizarse con la metodologia de simulación y se olvidan de los tratamientos matematicos.

Para modelar un sistema, es necesario entender el concepto de sistema y de los limites del sistema. Un sistema esta definido como un grupo de objetos que se encuentran interrelacionados para la realizacion de un objetivo [2].

Un sistema es a menudo afectado por cambios que ocurren fuera del sistema. Estos cambios son llamados ocurrencias en el entorno del sistema. En el proceso de simulacion, es necesario definir los limites entre el sistema y su medio ambiente [2]

Terminologia de simulacion de sistemas.

Dentro de la simulacion de sistemas existen conceptos de gran importancia, que se utilizaran a lo largo de este trabajo; los cuales se definen a continuacion:

Los componentes de un sistema son: la *entidad*, que es el objeto de interés en el sistema. Estas entidades tienen *atributos* que son sus propiedades. Y finalmente una *actividad* que representa un periodo de tiempo de longitud especifica. Por ejemplo para un banco los clientes podrian ser las entidades y el estado de cuenta de su chequera es el atributo de cada cliente y la acción de depósitar dinero en una caja es la actividad.

El estado de un sistema está definido como el conjunto de valores de las variables necesarias para describir el sistema en cualquier momento, relativo a los objetivos de estudio. Y un *evento* esta definido como una ocurrencia instantanea que puede cambiar el estado de un sistema.

Existen diferentes clases de sistemas, los cuales se pueden dividir en tres grupos que son: los *sistemas discretos*, son aquellos en los cuales las variables estado cambian en intervalos discretos de tiempo. Por ejemplo los bancos, donde la variable de estado son los clientes que se encuentran dentro del banco, los cuales varian dependiendo del tiempo de arribo y el tiempo que tarda un cajero en atender al cliente. Los *sistemas continuos* son aquellos en los cuales las variables de estado cambian continuamente a través del tiempo. Por ejemplo es el proceso de calentamiento de un reactor quimico, en donde la variable de estado es la temperatura, la cual varia continuamente y finalmente los *sistemas combinados* que son aquellos en los cuales se presentan variaciones discretas y continuas de las variables de estado.

Un modelo está definido como una representación de un sistema con el propósito del estudio de este. Para muchos estudios, no es necesario considerar todos los detalles de un sistema, por lo que un modelo no es unicamente el substituto del sistema, sino tambien una simplificacion del sistema. En otras palabras, el modelo puede ser suficientemente detallado para permitir conclusiones validas, acerca del sistema real. Diferentes modelos de un mismo sistema pueden ser construidos con el propósito de analizar los cambios [2].

Los modelos pueden ser clasificados en estáticos o dinamicos, deterministicos o estocásticos, y discretos o continuos. Un *modelo estático de simulación*, también llamado simulación de Monte Carlo, representa al sistema en un punto particular en el tiempo. Un *modelo dinámico de simulacion* representa los cambios del sistema a través del tiempo.

Los modelos de simulación que no contienen variables aleatorias son considerados como *deterministicos*. Un *modelo estocastico de simulación* tiene una o mas variables aleatorias como entradas. Entradas aleatorias producen salidas o resultados aleatorios. Por lo consiguiente los resultados pueden ser considerados como una estimación de las caracteristicas verdaderas de un modelo.

Los modelos de simulacion continua y discreta de definen de forma analoga a los sistemas continuos y discretos previamente mencionados.

Los pasos en el estudio de la simulación son los siguientes.

1. Planteamiento del problema. Consiste en el establecimiento preciso del problema, la definición de los objetivos y la recolección de datos de información.

2. Fase de Preparación. En esta fase se incluye:

a) Formulacion y diseño del modelo.
b) Prueba de la consistencia y veracidad del modelo.
c) Diseño de experimentos y plan de simulacion.

3. Fase de Producción. Incluye:

a) Desarrollo de experimentos.
b) Análisis y evaluación de resultados.
c) Remodelación y depuracion del programa de simulación.

I.3. LENGUAJES DE SIMULACION.

Las primeras etapas de un estudio de simulación se refieren a la definicion del sistema a ser modelado y a su descripcion en terminos de relaciones logicas de sus variables y diagramas de flujo. Sin embargo llega el momento de describir el modelo en un lenguaje que sea aceptado por la computadora que se va a usar. En esta etapa se tiene que decidir por alguna de las siguientes dos opciones:

1. Desarrollar el software requerido para el estudio de la simulación.

2. Adquirir algun paquete de simulación tal como GPSS, GASP, SIMULA, etc.

Para la segunda alternativa existen un gran numero de paquetes, resulta dificil decidir cual paquete se ajusta mejor a la aplicacion determinada. La selección del paquete depende del conocimiento que tenga el analista sobre el, y si se sabe aplicar.

Entre las ventajas principales del los lenguajes de simulacion, se mencionan las siguientes:

1. Reduccion en la tarea de programación.
2. Mejor definición del sistema.
3. Mayor flexibilidad para cambios.
4. Mejor diferenciación de las entidades que forman el sistema.
5. Se relacionan mejor las entidades.

Los lenguajes de simulacion que existen actualmente tienen una serie de caracteristicas propias que los diferencian.

Las caracteristicas son las siguientes:

a) El algoritmo para generar numeros aleatorios.
b) Los procedimientos utilizados para generar las variables aleatorias no uniformes.
c) La forma de avance del "reloj de simulación", la cual puede ser de dos formas incrementos de tiempo fijo o incrementos al proximo evento.
d) El análisis estadistico de los resultados de la simulacion.
e) El formato de salida de los resultados.
f) El lenguaje en el cual el paquete está construido.
g) Los diferentes tipos de computadoras sobre las que opera el paquete.

I.4. MODELOS DE COLAS.

Los modelos de colas pueden ser resueltos matematicamente o analizados por medio de la simulación, proveen al analista con una poderosa herramienta para el diseño y evaluación del desarrollo de los sistemas de colas. Típicas medidas del desarrollo del sistema incluyen el tiempo en el que una actividad se desarrolla, la longitud de la linea de espera o cola de la actividad, y el tiempo de estancia de las entidades en el sistema.

La teoria de colas y/o la simulacion es usada para predecir aquellos parametros del desarrollo del sistema como una funcion de las condiciones de entrada. Los parametros de entrada incluyen el tiempo de llegada entre las entidades, el porcentaje de tiempo que una actividad se encuentra trabajando, y el número y configuracion de las actividades. Algunos de los parametros de entrada se encuentran bajo el control directo del analista. Por lo tanto, los parametros de desarrollo se encuentran bajo control indirecto, previendo que la relacion entre los parametros de desarrollo y las condiciones de entrada, estén relacionados para el sistema propuesto.

Caracteristicas de los modelos de colas.

Los elementos importantes en los sistemas de colas son las entidades y las actividades.

La Población. La población potencial de entidades, puede asumirse como finita o infinita. En sistemas con una gran población de entidades potenciales generalmente se supone infinita.

Para dichos sistemas, esta suposicion generalmente es inocua y ademas puede simplificar el modelo. Generalmente es seguro el uso de modelos con poblacion infinita, siempre que el numero de entidades que son atendidas o en espera de un servicio en cualquier tiempo es una pequeña parte de la poblacion potencial de entidades. La principal diferencia entre poblaciones finitas o infinitas es como esta establecido el tiempo de arribo entre las entidades. En un modelo de población infinita, el tiempo de arribo no está afectado por el numero de entidades que han dejado la población y se han unido a un sistema de colas. Cuando el proceso de arribo es homogéneo a traves del tiempo, el arribo generalmente es constante. Por otra parte para modelos de poblaciones finitas, el tiempo de arribo en los sistemas de colas dependera del numero de entidades que seran servicas y que se encuentren en espera.

Capacidad del sistema. En muchos de los sistemas de colas hay un limite para el numero de entidades que pueden estar en las lineas de espera o en servicio. Una entidad que llega al sistema, y lo encuentra saturado o lleno, no entra al sistema pero regresa inmediatamente a la poblacion rechazo al ambiente de muestreo.

El proceso de arribo al sistema. El proceso de arribo para modelos de poblaciones infinitas están usualmente caracterizados en términos de los tiempos entre los arribos de entidades sucesivas. Los arribos pueden ocurrir en tiempos programados o en tiempos aleatorios. Cuando son tiempos aleatorios, los tiempos entre arribos están generalmente representados por una distribución probablistica. Las entidades pueden arribar una a la vez o en lotes. El lote puede ser de una cantidad constante o aleatoria.

Comportamiento y disciplina de las colas. El comportamiento de las colas se refiere a las acciones de las entidades mientras comienza la espera en la cola de un servicio. En algunas situaciones, no hay la posibilidad de que una entidad al llegar pueda desistir de entrar al sistema al encontrarlo lleno o saltar de una linea de espera a otra línea al haber escogido una actividad.

La disciplina de las colas se refiere al ordenamiento lógico de las entidades en la cola y determina cuál entidad será la que recibirá el servicio.

Las disciplinas más comunes son:

FIFO. Primero que entra, primero que sale (First Input First Output).
SIRO. Servicio en orden aleatorio (Service In Random Order).
SPT. El tiempo de proceso más corto (Shortest Proess Time).
SP. Servicio de acuerdo a prioridades (Service Priorities).

La disciplina FIFO implica que el servicio comienza en el mismo orden de los arribos, pero las entidades pueden dejar el sistema en diferente orden debido a las diferentes tiempos de servicio de las actividades.

Tiempos de servicio. El tiempo de servicio puede ser constante o de duración variable. En el último caso puede ser descrito por una variable aleatoria. Para algunos sistemas, el tiempo de servicio depende de la hora del día, o de la longitud de la línea de espera.

5. TRABAJOS AFINES

La idea de aplicar técnicas de simulación para sistemas de manufacturación no es nuevo, desde el desarrollo de la metodología de simulación, cientos de modelos han sido construidos considerando varios aspectos en sistemas de producción [14].

Una variedad de trabajos han sido aplicados a la simulación de sistemas de producción. Algunos de los primeros modelos fueron desarrollados usando la técnica de procesamiento en lotes, pero los modelos más recientemente desarrollados se han dirigido al uso de métodos interactivos.

Algunos de los sistemas de simulación han estado cerca de la idea de modelar completamente el ambiente de sistemas de producción. Uno de los primeros trabajos de simulación relacionados con sistemas de producción fue presentado por Krajewsky con el sistema (MASS) Manufacturing System Simulator, pero el sistema es limitado desde el punto de vista de producción-inventario, además MASS no proporciona completa interacción con el usuario [14].

Engelke construyó un paquete de software genérico para modelos de manofacturación llamado "The Integrated Manufacturing Modeling System" (IMMS) (Engelke 1985). La entrada a IMMS es desarrollada por medio la localización de simbolos en una pantalla de graficos y la especificación de los parametros se realizan mediante la introduccion de tablas. IMMS utiliza la técnica de teoria de colas para redes para el modelo de sistemas de producción [14].

Crookall reporta el desarrollo de un sistema computacional integrado para una fábrica combinando los conceptos de simulación y base de datos. Este modelo representa una tecnica inovadora para el modelaje y reconoce la importancia de la información concurrente disponible para el modelaje de sistemas de producción, pero esto aparenta que el alcance de este modelo esta de alguna forma limitado [14].

En 1986, IBM anunció un modelo para la simulacion de escenarios de manufacturación usando el lenguaje GPSS (Boletin técnico de IBM 1986). La linea de desarrollo de este software esta inclinado hacia los detalles de sistemas de producción [14].

Conclusiones.

Este trabajo se encuentra organizado de tres capitulos. En el primero se presenta toda la terminologia y caracteristicas que representan a los sistemas discretos. En el segundo capitulo se presentan los elementos técnicos que constituyen al simulador SIMDISC, tales como estructura de datos que maneja, estrategias que utiliza para la solución de problemas tipo y cálculo de parametros estadísticos. Finalmente el tercer capitulo presenta ejemplos de aplicacion del simulador en areas de producción, estudios de transporte e investigación de operaciones.

CAPITULO II.

II.1. ESTRUCTURA DEL PROGRAMA.

El simulador se encuentra construido en lenguaje turbo pascal 4.0, con procedimientos que manipulan a las corrutinas en ensamblador, sus requerimientos de hardware son:

- Dos unidades de disco o un disco duro.
- Tarjeta de graficos CGA.
- Memoria de 256 Kbytes como minimo.

SIMDISC (Simulador de Sistemas Discretos) se desarrollo con la finalidad de ser completamente interactivo con el usuario, con el objetivo principal de ayudar en el diseño y estudio de sistemas con problemas de lineas de espera o colas.

El sistema se encuentra orientado a la simulación de procesos de manufacturación, procesos quimicos, problemas de transporte, sistemas economicos, etc. o sea todos aquellos que presenten variables aleatorias dentro del sistema. No se incluye la simulación continua debido a la filosofia de tener una completa interaccion con el usuario; con la cual fué construido el simulador. Ya que la simulación continua requiere de la especificación de funciones para describir el comportamiento de las variables del sistema. Para la definición de las funciones que son caracteristicas para cada sistema en particular existen dos alternativas; la primera es mediante la programacion de la funcion; alternativa inadecuada para el simulador SIMDISC ya que sale de su filosofia, y la segunda opcion seria mediante la definición de la función a través de bloques operacionales como son sumadores, integradores, divisores, etc., alternativa viable, pero ocasionaria el crecimiento y complejidad en gran proporcion del programa.

La estructura del programa se observa en la figura 1., SIMDISC cuenta con cuatro modulos importantes los cuales son: el manejador de archivos, los procedimientos de simulación, los procedimientos de animación y por último el generador de reportes, el programa puede desplegar los resultados de la simulación tanto en la pantalla como en la impresora.

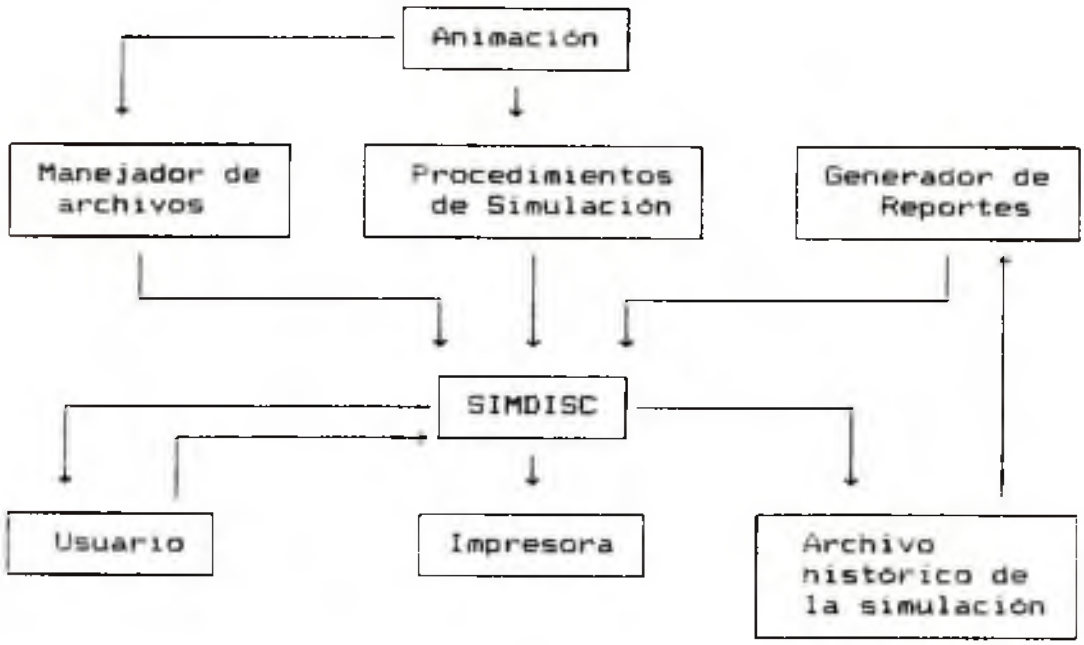


Figura 1.

El simulador se encuentra construido a base de menús jerárquicos que van llevado de la mano al usuario para la construcción de la red como para la especificación de los datos necesarios para efectuar la simulación. El programa envía mensajes de error en caso de introducir datos, parámetros o conexiones entre actividades no válidos para la red de actividades que se encuentre editando o simulando.

La figura 2 muestra el diagrama jerárquico de los menús del programa. Con este diagrama se observa toda la gama de opciones que presenta el simulador para la construcción, simulación y producción de reportes.

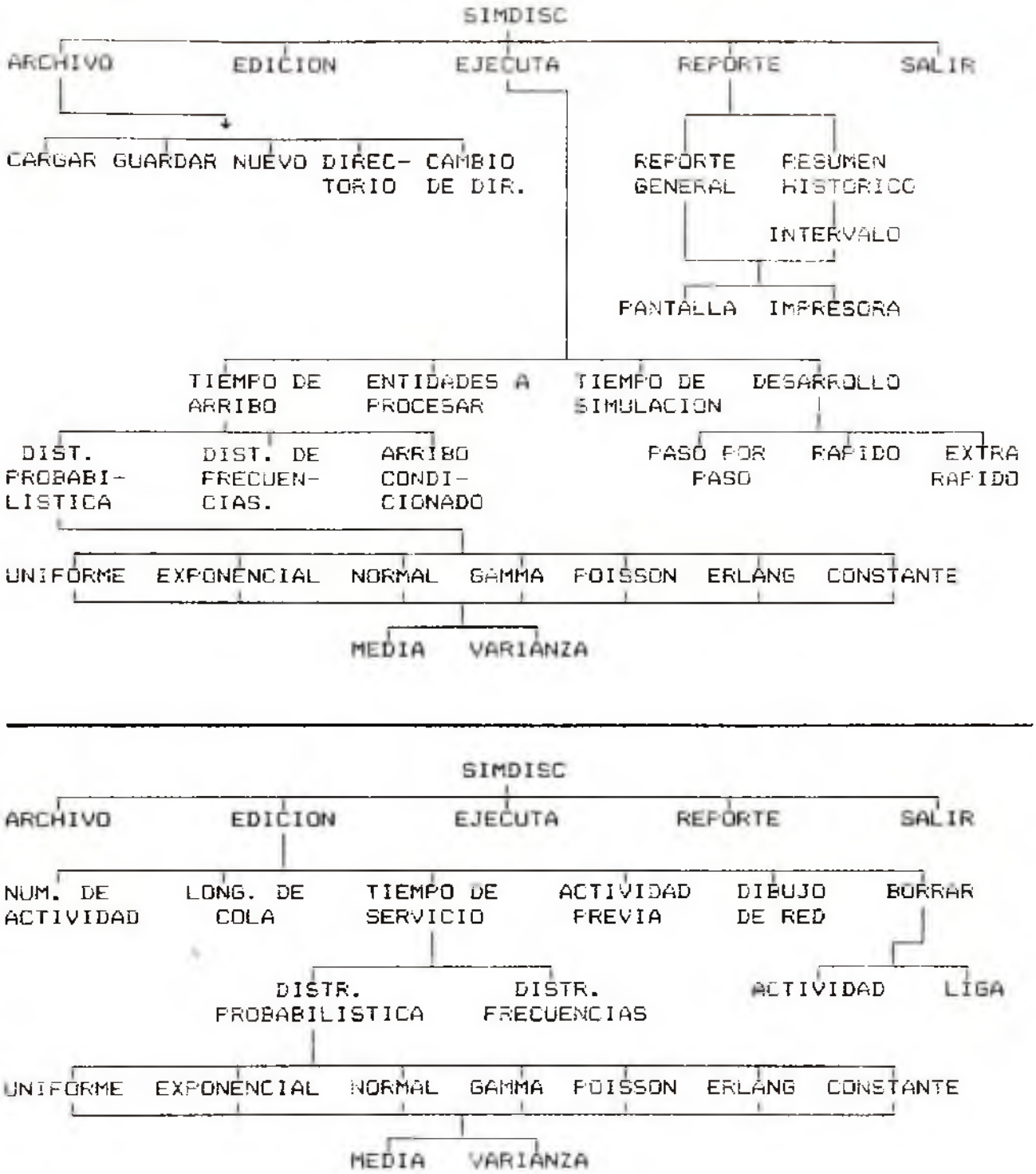


Figura 2. Diagrama jerárquico de SIMDISC.

II.2. MODULOS DEL PROGRAMA.

El modulo de manejador de archivos se encarga de manipular el manejo de ventanas, manejo de menus, construccion de la red y lectura de los parametros para cada una de las actividades, también se encarga de manipular los archivos de trabajo para el simulador y desplegar las ventanas de ayuda para cada uno de los menus.

Este módulo también se encarga de manejar las pantallas de edición de la red, ya que la red se puede dibujar sobre un conjunto de pantallas, pero el módulo requiere de la ayuda de procedimientos que se encuentran dentro del modulo de animación para poder manipular los bloques representativos de las actividades en la edición de estas, así como el dibujo de lineas, flechas y principalmente el corrimiento entre las pantallas de dibujo.

Los principales procedimientos que constituyen este modulo son:

Función runmenu. Maneja los menus, regresando como valor el digito correspondiente a la opción seleccionada. Vease el apendice [D] dentro de la unidad de procedimientos menus.pas.

Procedimiento helpmanag. Despliega la ventana de ayuda correspondiente al menú que se encuentre activo.

Procedimiento editmanag. Controla al menú de edicion, construyendo la red y validando los datos de las actividades que constituyen la red, así como la localización de las actividades en la pantalla de dibujo.

Procedimiento Filemanag. Maneja al menú de archivos controlando las unidades de disco y sus directorios, también manipula los archivos que contienen la información de las redes de actividades con las que puede trabajar el simulador.

El módulo de simulación contiene los procedimientos que se encargan de efectuar la simulación, ésta se puede llevar a cabo en tres formas diferentes que son paso por paso, rápido y extra rápido, si se seleccionan las dos primeras opciones de desarrollo, el modulo requerirá de los procedimientos que estan dentro del módulo de animación para observar los cambios de estado de las actividades. Los procedimientos que constituyen este modulo serán explicados a fondo en la siguiente seccion. Cada cambio de estado de las actividades de la red se almacena en disco para contruir la historia de la simulación, el archivo que guarda esta historia es EVENTS.SIM.

El modulo de generacion de reportes se encarga de desplegar los resultados estadisticos e interpretar los números que describen la historia de la simulación contenida en el archivo EVENTS.SIM. Los principales procedimientos que constituyen a este modulo son:

Procedimiento make_history. Se encarga de interpretar la información contenida en el archivo events.sim para producir un reporte de tiempos de duración de los tres diferentes estados para cada actividad, los tres estados que puede tener una actividad son ocioso, ocupado y bloqueado; cada estado esta representado por un simbolo . la construcción del histórico se va dejando en memoria, para desplegarse posteriormente en la pantalla. El tamaño de histórico dependerá de dos factores, el primero es el incremento de tiempo que se especifique para su construcción, y segundo la frecuencia con la cual ocurrieron los cambios de estado entre las actividades durante la simulación.

Procedimiento display_history. Este procedimiento se encarga de desplegar el histórico en la pantalla, donde este se puede analizar por renglones desde el tiempo cero de la simulación hasta el termino de esta.

Procedimiento print_history. Este procedimiento construye y envia el histórico a la impresora al mismo tiempo, por lo que únicamente requiere del incremento de tiempo que se especifica para la construcción del historico.

Procedimiento display_report. Este procedimiento despliega en la pantalla el reporte estadistico de cada actividad, mostrando los maximos de cada parametro estadistico. El procedimiento se encarga de formatear la salida del reporte ya que no realiza ningún cálculo de parámetros, se limita a establecer los máximos de los parámetros para después mostrarlos en el reporte con el simbolo [+].

Procedimiento print_report. Tiene la misma función que el procedimiento anterior con la diferencia de mandar el reporte a la impresora.

Algunos parametros estadisticos se calculan en el transcurso de la simulacion, y los demas se calculan al final de la simulación. El *procedimiento initialize* se encarga de calcular estos parametros estadisticos al final de la simulación. Este procedimiento se explicara más detalladamente en la siguiente seccion.

Los procedimientos anteriores se encuentran en el apéndice [C] dentro de la unidad de procedimientos auxproc.pas.

II.3. IMPLEMETACION DE CORRUTINAS.

Aunque existen muchos lenguajes de simulación, hay barreras para el uso de estos como: su indisponibilidad para pequeños centros de computo y la renuencia del usuario para aprender otro lenguaje. Debido a que los sistemas de simulacion no son accesibles o que puedan ser modificados por el usuario, ellos parecen ser demasiado especificos o bastante generales y complejos para una tarea en particular. El programador entonces utiliza un lenguaje con el cual se encuentre familiarizado como por ejemplo Pascal, Fortran, ganando flexibilidad, pero teniendo que manipular las dificultades que presentan la programacion de eventos, el manejo de colas, y manipulación de datos.

Las corrutinas es una técnica de programación que representa adecuadamente a las las entidades de un sistema, debido a que una entidad se puede encontrar en diferentes estados que son suspendida, en espera del cumplimiento de una condición dentro del sistema y bloqueada.

En los sistemas de multiprogramación, el CPU conmuta de un proceso a otro, ejecutando cada uno de estos proceso en un intervalo muy pequeño de tiempo. El CPU, puede realizar un programa a la vez, pero puede trabajar con varios programas en el curso de un segundo, dando asi una ilusión al usuario de paralelismo. A la conmutación rápida del CPU entre los programas se le llama pseudoparalelismo, a diferencia del paralelismo que se efectua por hardware, cuando el procesador trabaja mientras uno o más dispositivos de entrada y salida se encuentran operando [11].

Las corrutinas son rutinas que se comunican con cualquier otra de una manera simetrica. El mecanismo normal de las subrutinas es asimetrico; permite el llamado de una rutina, parando la ejecución de las intrucciones y la activación de otra subrutina la cual es entonces ejecutada desde su primera instrucción hasta la ultima, regresando a la rutina previa para continuar con sus intrucciones. Una corrutina, opera como sigue: Comienza con la primera instruccion cuando ésta es activada la primera vez, continúa la ejecución hasta que se suspenda por si misma y transfiera el control a otra corrutina. [6]

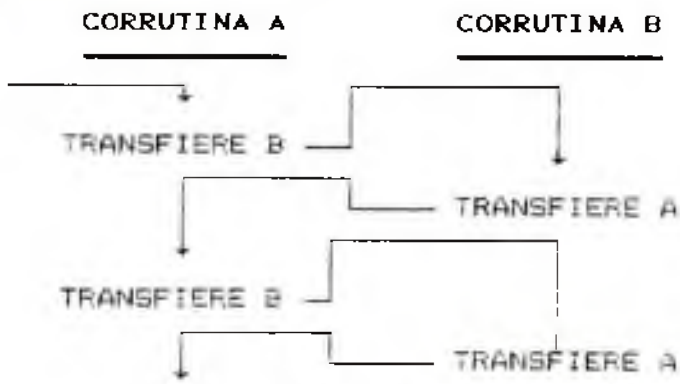


Figura 3.

La corrutina suspendida puede reactivarse cuando se lleva a cabo una transferencia de control explícita por medio de otra corrutina, regresando a ejecutar las instrucciones siguientes al punto de suspension figura 3.

Un concepto clave en los sistemas concurrentes es el proceso. Un proceso es un programa en ejecución. Este esta constituido por codigo ejecutable, una area de datos y un stack, su propio contador de programa, su apuntador de stack y otros registros, y toda la información necesaria para que se ejecute el programa. [1]

Cuando un proceso es suspendido temporalmente, se espera a que sea reiniciado exactamente en el mismo estado en que se detuvo.

Esto significa que toda la información acerca del proceso debe de ser explicitamente guardada en algun lugar durante la suspensión.

Asi un proceso tiene asociado una direccion al área donde se guardaron sus parametros.

Apuntador al stack.

Cada corrutina debe de tener asignado un apuntador al stack, ya que este apuntara al inicio del stack, para la activacion del proceso. Este apuntador esta declarado como sigue:

```
type      ptr_word: ^word;
```

Permitiendonos apuntar a un word [1] en la área de heap, o memoria disponible [12]. Y haciendo posible la permutación de procesos en la transferencia del control entre corrutinas.

Existen varios etapas en la manipulación de las corrutinas que son las siguientes: inicialización del núcleo, creacion de la corrutina, activación, suspensión y reactivacion. A continuacion se describira cada una de las etapas:

Inicializacion del nucleo.

Debido a la transferencia de control entre las corrutinas, estas deben de contar con su propio stack, cabe mencionar que el stack se crea como un bloque de memoria de un tamaño especifico, localizado en el heap [12]. El cual contendra la informacion del estado en que se encontraba la ejecucion de la corrutina antes de suspenderse. Es necesario tomar en consideración que el programa principal debe de ser visto como una corrutina y al efectuar la transferencia a una nueva corrutina, se almacenara el estado del programa principal en el stack del mismo, al que se le asigna implicitamente [12].

Para evitar la especificación explicita de la corrutina que se suspende y la corrutina que se activará, al llevarse a cabo un cambio de contexto [11], el núcleo debe de contener la dirección de memoria [1] del apuntador al stack del programa que se encuentra ejecutandose, especificándose unicamente la corrutina que se desea activar. Por lo que es necesario inicializar el nucleo con la dirección en memoria del apuntador al stack del programa principal.

Este procedimiento se puede analizar en el apéndice [C] con el nombre de *initnucleus*. El cual requiere de un solo parámetro que es el apuntador que se le asigna al programa principal. Cabe mencionar que este apuntador puede no estar inicializado, debido a que unicamente se requiere su dirección en memoria.

Creación de la corrutina.

En este paso se prepara o se inicializa el stack de la corrutina para su subsecuente utilización.

En esta función no se efectúa la petición de memoria, debido a que el simulador efectúa previamente la petición de la memoria requerida, con la finalidad de evitar las operaciones de mantenimiento del heap. En la sección posterior del manejo de memoria se explicará más ampliamente este aspecto.

Como se mencionó anteriormente el stack se obtiene de la memoria disponible, el tamaño del stack, debe de ser de una cantidad en bytes [1] apropiada para efectuar todas las instrucciones que contiene la corrutina. Se recomienda utilizar como mínimo un tamaño de 1000 bytes para el stack. Ya que la mayoria de las operaciones de entrada y salida, así como el paso de parámetros entre los procedimientos se llevan a cabo mediante el stack.

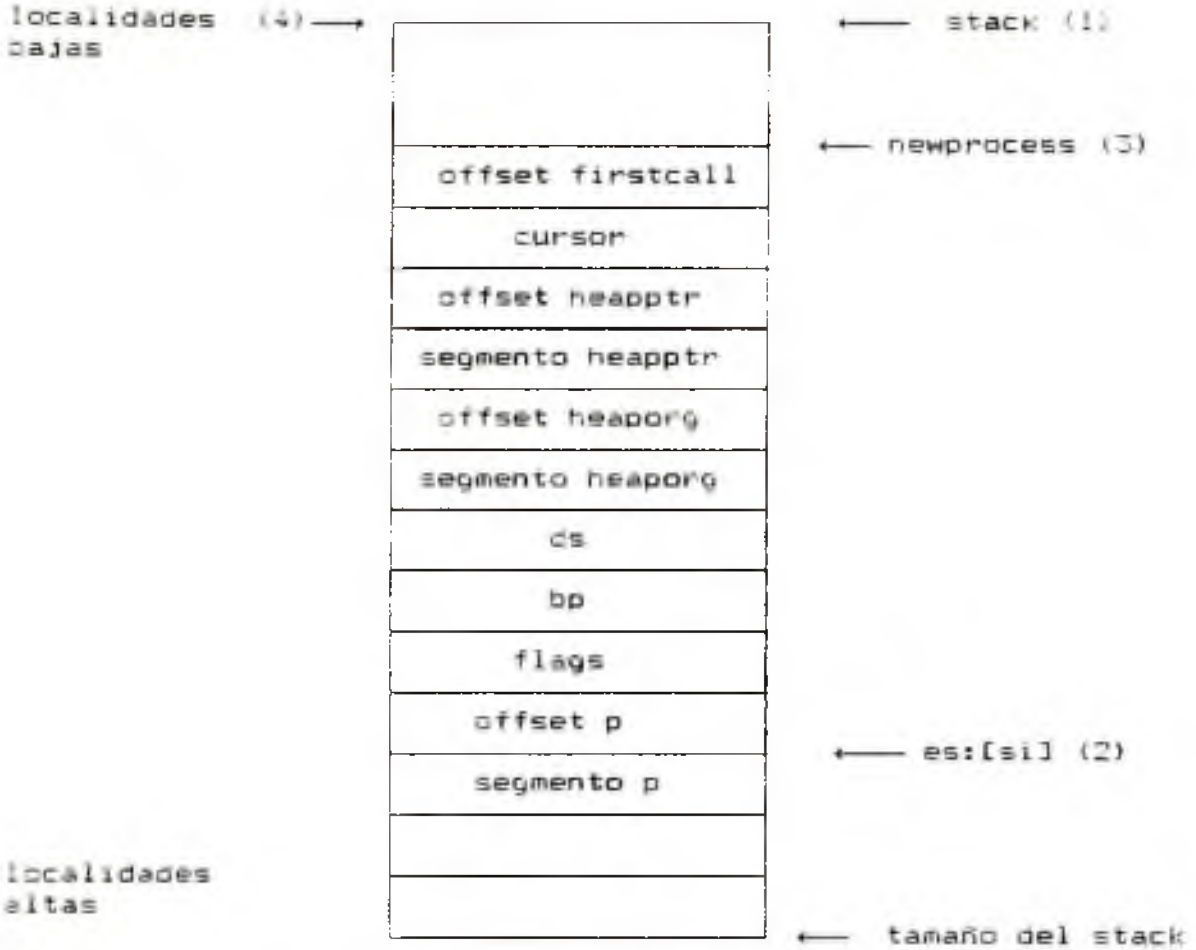
La función que lleva a cabo la creación de la corrutina, es la *función newprocess* la cual tiene tres argumentos: el primer argumento es la dirección en memoria del procedimiento que trabajara como corrutina, el segundo argumento es el apuntador al bloque de memoria que se utilizará como stack para el procedimiento, y el tercer parámetro es el tamaño del bloque de memoria.

El stack de una corrutina debe de tener asignados los apuntadores heapptr y heaporg [12], apuntando al tope del stack con el fin de poder crear el error de sobreflujo del stack [12] en casos en que el tamaño de la corrutina sea pequeño para el procedimiento. Pero se tiene la limitante de que una corrutina no puede hacer requerimientos de bloques de memoria, ya que el valor de los apuntadores previamente mencionados se han modificado de valor, provocando fallas en el funcionamiento del manejador de memoria de turbo pascal figura 4.

Activación y suspensión de corrutinas.

La transferencia de control entre corrutinas se lleva a cabo por el *procedimiento transfer* el cual unicamente requiere como argumento el apuntador al stack de la corrutina que se desea activar.

En la figura 5 se puede observar como se lleva a cabo el cambio de contexto entre las corrutinas, el cambio se lleva en tres pasos. El primero es el guardado del stack de la corrutina que se está dejando. El segundo es el cambio al nuevo stack y por último la activación de la nueva corrutina. La variable *addr_current*, contiene la dirección en memoria del apuntador al stack de la corrutina que se encuentre activa.



(1) La variable stack apunta al inicio del bloque de memoria al entrar a la *función newprocess*.

(2) Localidad de memoria donde se inicia a preparar el stack, vease *procedure initprocess*; apéndice C.

(3) La *función newprocess* al terminar su ejecución devuelve un apuntador a memoria, cuyo valor es el tope del stack para la corrutina.

(4) Lugar donde se inicializan los apuntadores heapptr y heaporg, para la corrutina.

Figura 4.Funcion newprocess.

Paso 1. tranfer (p1). Antes de efectuarse el transfer, la variable addr_current guarda la dirección en memoria del tope del stack correspondiente al proceso p0.

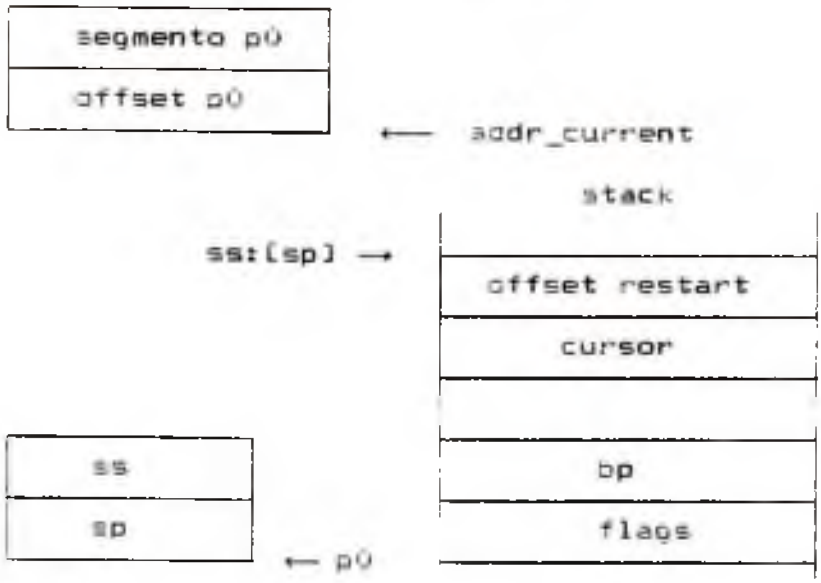


Figura 5.

Paso 2. transfer (p1).

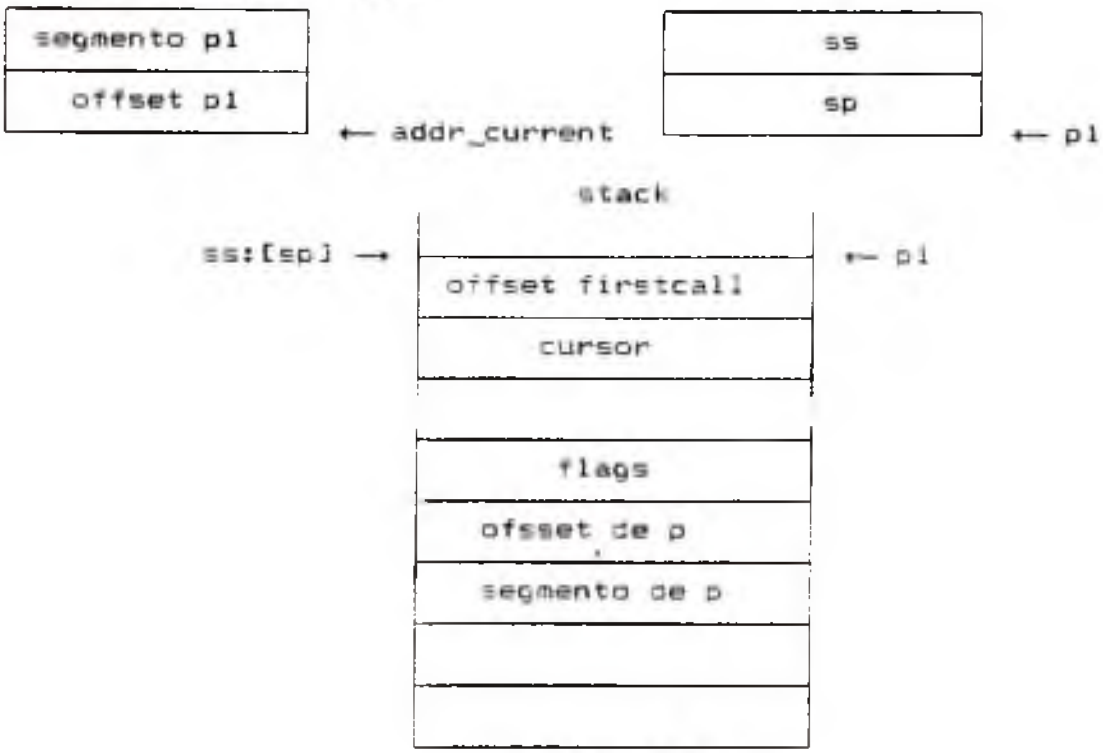


Figura 6.

Con el siguiente programa ilustra como se llevan a cabo los pasos previamente especificados.

```
program   ejemplo1;

uses globals, nucleo;        { utiliza procedimientos externos .
                             { ver apendice C }
var
  p0,  p1,  p2,   s1,   s2: ptr_word;

procedure prueba;            { corrutina }
begin
  writeln ('prueba: statment 1');
  transfer (p2);             { suspende ejecución y }
                             { transfiere el control al proceso p2 }
  writeln  ('prueba:  statment  2');
  transfer (p2);
end;

procedure test;              { corrutina }
begin
  writeln ('test: statment  1');
  transfer (p1);
  writeln ('test: statment 2');
  transfer  (p0);            { suspende ejecución y transfiere el }
end;                         { el control al proceso p0 }

begin
  initnucleus (p0);          { inicializa el nucleo }
  getmem  (s1,  1000);       { petición de memoria para stacks}
  getmem (s2,  1000);
  p1 := newprocess (@prueba, s1, 1000); {inicializa los procesos}
  p2 := newprocess (@test, s2, 1000);
  writeln  ("programa  principal");
  transfer (p1);             { transfiere el control al proceso p1 }
  writeln ('termina'); { reasume el control }
  freemem (s1, 1000);  { libera la memoria de los stacks }
  freemem (s2, 1000);
end.
```

La salida que produce el programa anterior es la siguiente:

```
programa principal     { despliega el mensaje del programa principal }
prueba: statment 1     { ejecuta la primera instr. del  proceso  p1  y
                         permuta al proceso p2 }
test: statment 1       { ejecuta la primera instr. del  proceso  p2  y
                         permuta a la tercera instr. del proceso p1 }
prueba: statment 2     { ejecuta la tercera instr. del  proceso  p1  y
                         permuta a la tercera instr. del proceso p2 }
test: statment 2       { ejcuta la tercera instr.  del  proceso  p2  y
                         permuta al programa principal }
termina                { regresa al programa principal }
```

II.4. ESTRUCTURA DE DATOS DEL SIMULADOR.

Se encuentran dos tipos de estructuras que son importantes en el simulador, una es la que representa a las actividades en el simulador y la segunda es la que representa a las entidades u ordenes de trabajo.

Estructura para la red de actividades.

En esta estructura se especifican como están interconectadas las actividades dentro de la red. La estructura grafos dirigidos [13] se utiliza para representar la red de actividades.

La estructura que se utiliza para representar las actividades se llama *node* (vease apendice B) contiene las variables necesarias tanto para su especificación asi como su posición en la pantalla del proceso, junto con las variables de simulación, a continuación se muestra la estructura y sus variables especificando la información que contendrá la variable.

Descripción de variables del registro tipo node:

identif. Es el numero de identificación correspondiente a la actividad.

sons. Contiene las actividades que son sucesivas a la actividad. Es un vector en donde se encuentran los números de identificación de las actividades sucesoras.

queue. Es la lista de ordenes de trabajo o entidades esperando a ser atendidos por la actividad.

lqcap. Es la cantidad máxima de entidades que puede contener la actividad en su línea de espera (longitud máxima de la cola).

lq. Es la variable que contendrá la longitud de la cola durante la simulación.

lqmax. Contedra la cantidad máxima de entidades que se encontraron en la línea de espera de la actividad, durante la simulación.

distr_type. Contiene el tipo de distribución probabilistica que describe el tiempo de servicio de la actividad. Cada distribución está representado por un digito entre el 0 y el 6. (vease *funcion distribution_type* en apendice B).

serv_time. Es la media del tiempo de servicio, correspondiente a la distribución probabilistica.

variance. Es la varianza del tiempo de servicio, correspondiente a la distribución probabilistica.

state. Es el estado correspondiente a la actividad en el tiempo T. Los estados de una actividad se especifican mediante dígitos y son los siguientes ocioso (0), trabajando (1), bloqueado (2).

tblock. Contendra el tiempo total que estuvo la actividad en estado de bloqueo.

tlastblock. Guardara el tiempo en que inicio el estado de bloqueo en la actividad.

n_block. Contador del número de veces que la actividad se encontro en estado de bloqueo.

twait. Contendra la sumatoria de los tiempos de residencia de las entidades en la actividad, desde su llegada a la linea de espera hasta que termina de ser servida por la actividad.

tidle. Almacenara el tiempo neto que duró la actividad en estado ocioso durante la simulación.

tlastidle. Guardara el tiempo en el cual comienza el estado ocioso de la actividad, durante la simulación.

jobout. Es la cantidad de entidades que fueron completamente atendidas por la actividad.

jobin. Es la cantidad de entidades que entran a requerir servicio a la actividad, formandose en la linea de espera o entrando directamente a ser atendida por la actividad.

tprevlq. Es el tiempo en cual se efectua un cambio en la longitud de cola, se utiliza con la finalidad de calcular la duración de la longitud de la linea de espera durante la simulación.

timelq. Vector que almacena los tiempos de duración de las diferentes longitudes que tuvo la linea de espera durante la simulación.

nwq. Es la cantidad de entidades que dejan la linea de espera.

twq. Es el tiempo promedio gastado en la linea de espera por las entidades.

timef. Vector que contendra los tiempos correspondientes a las frecuencias acumuladas de la tabla de frecuencias, que se utiliza para la descripcion del tiempo de servicio de la actividad.

frec. Vector que contendra las frecuencias acumuladas para la tabla mencionada anteriormente.

flag_frec. variable lógica que especificara la existencia de la tabla de frecuencias acumulativas para la descripción del tiempo de servicio de la actividad.

x, y. Almacenarán las coordenadas reales en la pantalla de dibujo de la red de actividades.

prev_hold. Variable lógica que se utiliza para la selección de las posibles actividades sucesoras a la actividad actual.

nextn. Es un apuntador a la siguiente actividad en la lista de actividades de la red. Ya que para representación de grafos dirigidos se utiliza una lista ligada [9].

El registro [8] *lineprod* es el nodo inicial del grafo, este nodo contendrá unicamente la información de las actividades de entrada al proceso, ya que mediante estas, entran al sistema las entidades u órdenes de trabajo. Y es la cabeza de la lista de actividades de la red.

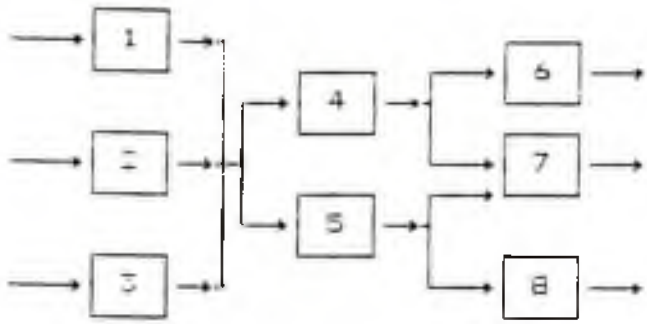
A continuación se presentarán algunas redes de actividades y como se encuentran representadas por esta estructura.

Red 1.

ACTIVIDAD	*ACTIVIDADES SUCESORAS*
1	2
2	3
3	

ACTIVIDADES DE ENTRADA: 1

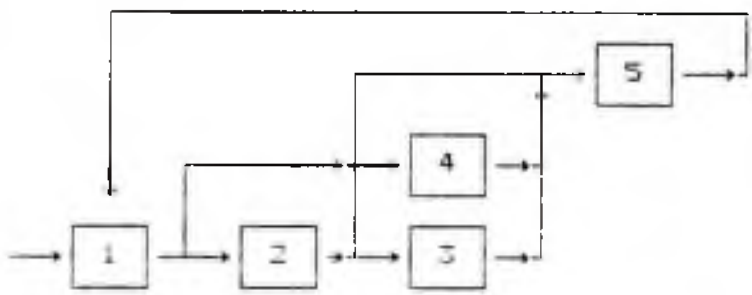
Red 2.



ACTIVIDAD	ACTIVIDADES SUSCESORAS
1	4, 5
2	4, 5
3	4, 5
4	6, 7
5	7, 8
6	
7	
8	

ACTIVIDADES DE ENTRADA: 1, 2, 3

Red 3.



ACTIVIDAD	ACTIVIDADES SUSCESORAS
1	2, 3, 4, 5
2	3, 4, 5
3	5
4	5
5	1

ACTIVIDADES DE ENTRADA: 1

Estructura para el manejo de entidades.

Debido a que las entidades fluyen a través de una red de actividades, es necesario introducir un algoritmo que decida por donde se debe dirigir la entidad una vez que entra a la red. Por lo que cada entidad es una corrutina, y por lo consiguiente cada corrutina puede seguir la trayectoria seleccionada por su propio algoritmo.

Como las entidades llegan a la red, la recorren, y por ultimo terminan de ser atendidas, éstas se deben de crear y destruir, por lo que es necesario, crear o destruir corrutinas con el fin de evitar el agotamiento de la memoria disponible [8], ya que cada corrutina requiere una cantidad de memoria para su stack.

La estructura que se utiliza para representar a las entidades o corrutinas tiene como nombre *descr* y sus variables son las siguientes:

Descripción de variables del registro tipo descr:

corout. Es una variable tipo apuntador, que apunta al tope del stack, ya que con este apuntador se efectuará el cambio de contexto entre las corrutinas.

init_ptr. Variable tipo apuntador, el cual apunta al principio del bloque de memoria. Se utliza para crear la corrutina y para la devolución del bloque de memoria al heap, una vez terminada la simulación.

next. Variable tipo apuntador, el cual se utiliza para mantener la lista ligada entre las corrutinas.

key. Esta variable se utiliza para guardar el tiempo en el cual será reactivada la corrutina.

sym. Se utiliza para identificación de la corrutina.

flag_busy. Variable lógica que indicará si el bloque de memoria esta ocupado por una corrutina o si esta disponible para una nueva. El funcionamiento de esta variable quedará mejor comprendida en la sección de manejo de memoria por el simulador.

II.5. ALGORITMO DE SIMULACION.

Los algoritmos de simulación más frecuentemente usados son *en función del tiempo* y *por evento*. En el primer algoritmo se cuenta con una variable que representará el reloj de simulación. El programa de simulacion ejecuta un ciclo, en el cual cada iteracción del ciclo el reloj de simulación es avanzado por un incremento pequeño de tiempo [dT]. Cada entidad se checa para verificar si ocaciona un cambio en el estado del sistema durante el intervalo [dT] previo. Este algoritmo tiene el inconveniente que para incrementos de tiempo muy pequeños, en muchos de los incrementos no existen cambios de estado en el sistema, lo que ocaciona una simulación bastante lenta.

Para el segundo tipo de algoritmo la estrategia consiste en avanzar el reloj de simulación directamente a la ocurrencia del próximo evento. No se pierde tiempo de simulación en aquellos intervalos en los que no existen eventos. El programa de simulación determina el próximo evento manteniendo una lista ordenada de eventos que pueden ocurrir en el futuro. Este algoritmo es el seleccionado para llevar a cabo la simulación.

Para realizar la simulación de cualquier red se requiere de un algoritmo que decida la trayectoria a seguir dentro de la red de actividades. El algoritmo tiene que resolver el manejo de las entidades dentro de la red, en función del estado en que se encuentren las actividades. El algoritmo controla la llegada de las entidades al sistema como el flujo de estas a través de la red. A continuación se muestra el algoritmo e ilustraciones sencillas que demuestran su funcionamiento. Para facilitar la explicación de las ilustraciones unicamente se estableceran problemas con linea de espera en los ejemplos del 2 al 4. La simbologia que se utiliza en los ejemplos es la siguiente:

simbolo	Descripción
o	entidad
—→	flujo de entidades
1	actividad no. 1
W	actividad ocupada
B	actividad bloqueada
I	actividad ociosa

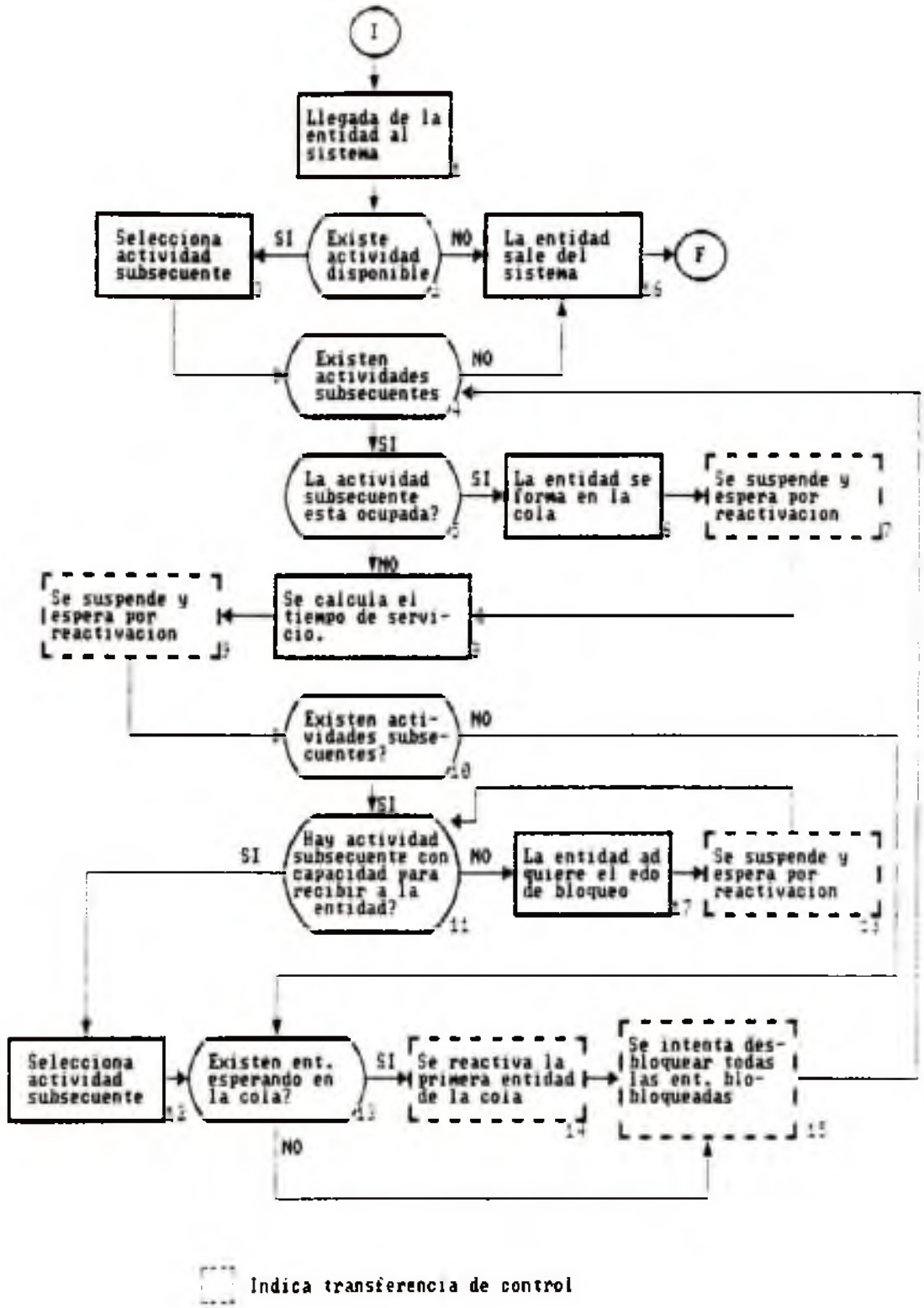


Figura 7. Algoritmo utilizado por las entidades para el recorrido dentro de la red de actividades.

Administración de entidades en el algoritmo.

Enseguida se analizará como se lleva a cabo la administración de las entidades por el algoritmo y los procedimientos que lo integran así como las variables que se utilizan para la simulacion.

Se cuenta con una variable global de tipo real *time* la cual llevará el reloj de simulacion. La variable *simduration* contendrá el tiempo a simular. Se tiene la lista *timelist* la cual contiene la lista de eventos que ocurrirán en el futuro y la lista *blocklist* que contendrá la lista de entidades que se encuentran bloqueadas. El apuntador *running* se utiliza para especificar la entidad o descriptor de corrutina que se encuentra en el momento activa. La variable *main* es el descriptor asociado al stack del programa principal, se utiliza para transferir el control al programa principal una vez terminada la simulación.

Los procedimientos que se utilizan para el manejo de entidades se describirán a continuación y el código de programación se encuentra en el apendice B.

Problema 1.

Uno de los primeros proolemas que tiene que resolver el algoritmo es la decisión de como se distribuirán las entidades a su llegada al sistema, en las actividades de entrada.

Existen procesos en los cuales existe unicamente una sola actividad de entrada u otros en los cuales existen dos o más actividades de entrada en paralelo, en donde cada actividad puede tener una longitud de cola de espera o en caso contrario carecer de esta.

Se recomienda tener adjunto la figura 7 que es el algoritmo que seguiran las entidades, ya que se hara referencia a esta figura, durante la explicación de los problemas.

Analizando primeramente el caso mas sencillo, la de una actividad de entrada sin cola de espera.

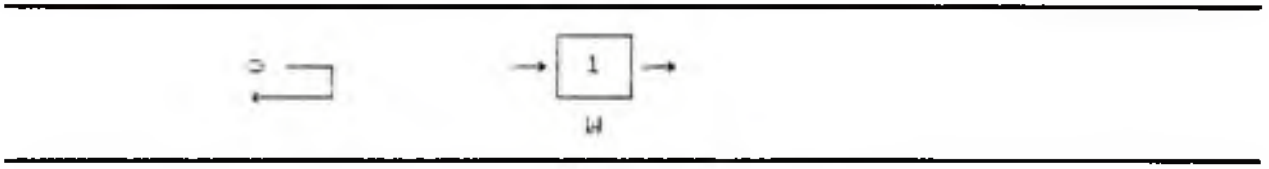


Figura 8.

Cabe mencionar que cada corrutina tiene asociado un valor de identificación (una letra minuscula) y el tiempo de reactivacion de la corrutina. Ejemplo:

```
ja - 12.87      corrutina [a] asociada al procedimiento job. con un
                tiempo de reactivación de 12.87
m  - 145.8      corrutina asociada al procedimiento initialize.
g  - 3.5        corrutina asociada al procedimiento generate.
```

Para las actividades, cada una tiene asociado un valor de nivel o profundidad, ya que la red de actividades se encuentra representada por la estructura de datos *grafo dirigido* [13]. Para el problema la actividad 1 tiene un nivel asociado de 0.

Al inicio de la simulación la variable *simduration* tiene el valor del tiempo que durará la simulación.

El *procedimiento initialize* es el que inicia la simulación, inicializando las variables de simulación y efectúa la petición de memoria para el stack de las corrutinas. Determina la cantidad de stacks que fueron creadas, esta cantidad se guarda en la variable *corout_avail*. Una vez terminada la simulación se transfiere el control a este *procedimiento*, ya que funge como la corrutina asociada al stack principal y finalmente devuelve al heap [12], la memoria previamente requerida para los stacks. Ya que éste procedimiento fungirá como una corrutina, hay que asignarle un tiempo de reactivación que será igual a *simduration* + 0.1. (en el estado de las variables, al *procedimiento initialize* lo representa la corrutina (m)).

La corrutina a la cual se efectúa la primera transferencia de control es *generate*, que genera la primera entidad al tiempo 0 de simulación. Este procedimiento se utiliza para generar las entidades de acuerdo al tipo de arribo que se haya especificado. Para distinguir la forma de arribo se cuenta con la variable de tipo byte *type_arrival* la cual puede contener los siguientes valores:

```
0       tiempo de arribo descrito por una distribución probabilistica.
1       tiempo esta representado por una tabla de frecuencias.
2       indica que los arribos se efectuaran en forma condicional a
        la terminación de servicio de una de las actividades
        sucesoras.
```

Si se desea generar una cantidad especifica de entidades a simular, este valor se almacena en la variable *processjobs*. El procedimiento que controla este tipo de arribo es la corrutina *cond_gen* que es similar a la corrutina *generate* pero esta se utiliza para generar entidades en donde el tiempo de arribo esta descrito por las primeras dos formas de arribo previamente mencionadas. (en el estado de las variables, al *procedimiento generate* lo representa la corrutina (g)).

Las corrutina asociada para las entidades es el *procedimiento job*, que contiene al algoritmo de la figura 7. Ya que se encarga de decidir el flujo de la entidad a traves de la red de actividades.

El estado de las variables despues de la transferencia de control del *procedimiento generate* a la primera corrutina creada es el siguiente:

simduration:100.0 *time*: 0.00 *running*: j_a - 0.0

Actividad: estado:
1 0 (ocioso)

timelist: *blocklist*:
g - 0.0
m - 100.1

La variable *running* apunta a la corrutina que posee el control, por lo que en este momento, lo tiene la corrutina (j_a). Realiza los bloques 2, 3, 4, 5 y 8 (fig. 7). Ya calculado el tiempo de servicio, enseguida se suspende la corrutina (j_a) mediante el *procedimiento hold*, el cual inserta la corrutina (j_a) en la lista *timelist* de acuerdo al tiempo de reactivación bloque 9 (fig. 7). Si el tiempo de servicio para la entidad o corrutina (j_a) es de 3.3 entonces el estado de las variabales es el siguiente:

simduration:100.0 *time*: 0.00 *running*: g - 0.0

Actividad: estado: entidad:
1 1 (trabajando) a

timelist: *blocklist*:
j_a - 3.3
m - 100.1

El *procedimiento hold* una vez que inserta a la entidad en *timelist* selecciona la corrutina a la cual se hará la transferencia de control con el *procedimiento schedule* y finalmente hace el cambio de control mediante el *procedimiento transfer*.

El *procedimiento schedule* toma a la corrutina que se encuentre al inicio de la lista *timelist*, y lo asigna a la variable *running* y actualizando la variable time con el valor asociado a la corrutina. En el estado de variables anterior la corrutina [g] se encontraba al inicio de la lista *timelist*, antes del llamado al *procedimiento schedule*.

En este momento la corrutina que posee el control es el *procedimiento generate* el cual le corresponde calcular el tiempo en el cual se generará la próxima corrutina asociada al *procedimiento job*, una vez calculado este tiempo se suspende con el *procedimiento hold* (previamente descrito).

Suponiendo que el tiempo calculado es de 2.0, entonces el estado es el siguiente:

simduration:100.0 *time*: 0.00 *running*:

Actividad:	estado:		entidad:
1	1	(trabajando)	a

timelist:
g - 2.0
ja- 3.3
m - 100.1

blocklist:

Una vez suspendida la corrutina por el *procedimiento hold*, se llama al *procedimiento schedule*, que produce el siguiente estado:

simduration:100.0 *time*: 2.00 *running*: g - 2.0

Actividad:	estado:		entidad:
1	1	(trabajando)	a

timelist:
ja - 3.3
m - 100.1

blocklist:

Se observa que por el tiempo de reactivación de la corrutina (g), ésta vuelve a tomar el control, por lo que genera una nueva corrutina que tendrá la letra de identificación (b). El paso de control a la nueva corrutina, lo realiza el *procedimiento activate*, el cual; lo primero que realiza es la verificación de la existencia de un stack disponible para la nueva corrutina, si la memoria esta saturada entonces, despliega el mensaje:

<< No existe suficiente memoria para la simulación >>

Y transfiere el control a la corrutina asociada a la variable *main*. Si existe un stack disponible entonces pasa a inicializar el stack para la corrutina con el *procedimiento newprocess*, y enseguida se inserta esta nueva corrutina al inicio de la lista *timelist*. y finalmente transfiere el control a la nueva corutina.

El estado de las variables después del *procedimiento activate* es el siguiente:

simduration:100.0 *time*: 2.00 *running*: jb- 0.0

Actividad:	estado:		entidad:
1	1	(trabajando)	a

timelist:
g - 2.0
ja- 3.3
m - 100.1

blocklist:

En este momento la corrutina (jb) ejecuta los bloques 2 y 16 (fig. 7), encontrando la actividad 1 ocupada por lo tiene que abandonar el sistema. La terminación de una corrutina se lleva acabo mediante el *procedimiento terminate*, el cual únicamente determina la siguiente corrutina que tomará el control.

El desarrollo de la simulación se llevará a cabo de forma similar durante el tiempo de simulación. Enseguida se analizará como se lleva a cabo la terminación de la simulación. Suponiendo que el estado de las variables es el siguiente:

simduration:100.0 *time*: 98.4 *running*: jv- 98.4

Actividad:	estado:		entidad:
1	1	(trabajando)	a

timelist:
m - 100.1
g - 101.4

blocklist:

El control lo tiene la corrutina (jv), el cual calcula su tiempo de servicio que es de 4.3 y llama al *procedimiento hold* suspendiendose con un tiempo de reactivación de 98.4 + 4.3 = 102.7, enseguida se ejecuta el *procedimiento schedule* el cual transfiere el control a la corrutina (m), terminando la simulación ya que regresa el control al *procedimiento initialize*.

Problema 2.

Ahora supongamos que la actividad anterior cuenta con una longitud de cola de 3. Al arribo de la primera entidad se realizan los bloques 2, 3, 4, 5, 8 y 9 (fig. 7), para las demás entidades subsecuentes que arriben a la actividad pasarán a formarse al final de la cola de espera, hasta que la capacidad de la cola se sature; bloques 2, 3, 4, 5, 6 y 7 (fig. 7).

Suponiendo que el estado de la actividad en cierto momento de la simulación es como se muestra en la figura 9. Al arribo de una entidad, ésta encuentra al sistema saturado por lo que abandona al sistema.

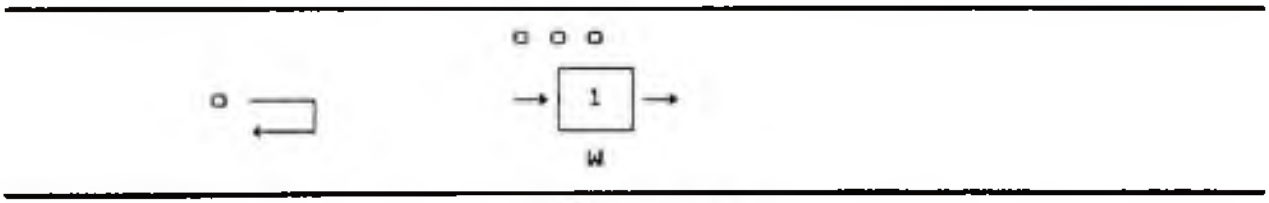


Figura 9.

Cuando la actividad termine de dar servicio a la entidad, la actividad tomará a la primera entidad que se encuentra en la cola para darle servicio figura 10. Bloques 10, 11, 12, 13 y 14 (fig 7).

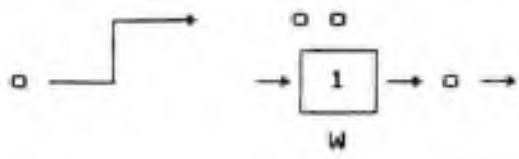


Figura 10.

A partir de este momento la actividad podrá recibir la próxima entidad que arribe al sistema.

Para este problema se analizará como se insertan las corrutinas en la línea de espera de una actividad. Suponiendo que el estado de las variables es el siguiente:

simduration:100.0 *time*: 10.3 *running*: jb - 0.0

Actividad:	estado:	entidad:	línea de espera:
1	1	a	

timelist:
g - 10.3
ja - 13.4
m - 100.1

blocklist:

En este momento se puede observar que se acaba de crear una nueva corrutina (b), la cual realiza los bloques 2, 3, 4, 5, 6 y 7 (fig. 7), el evento que implica que una entidad se forme en la cola de una actividad se lleva a cabo mediante el *procedimiento wait*, el cual deja la corrutina en estado de espera, insertandola al final de la lista queue asociada a la actividad [apendice D], enseguida hace el llamado al *procedimiento schedule* y transfiere el control a la corrutina seleccionada. El estado de las variables después del llamado al *procedimiento wait* es el siguiente:

simduration:100.0 *time*: 10.3 *running*: g -10.3

Actividad:	estado:	entidad:	línea de espera:
1	1	a	jb

timelist:
ja - 13.4
m - 100.1

blocklist:

Spongamos que el tiempo en el cual se generará la próxima corrutina es de 4.3, enseguida la corrutina generate se suspende con un tiempo de reactivación e 10.3 + 4.3 = 14.4, después la corrutina (ja) adquiere el control determinando que termina su servicio en la actividad, entonces realiza los bloques 10, 13 y 14 (fig. 7), ocasionando la reactivación de la corrutina (jb), la cual ejecutan los bloques 8 y 9 (fig. 7), obteniendo un tiempo de servicio de 5.0 por lo que se reactivará hasta el tiempo 13.4 + 5.0 = 18.4.

La reactivación de una corrutina que se encuentre en la línea de espera de una actividad, la lleva a cabo el *procedimiento signal*. El llamado de éste procedimiento lo realiza la corrutina que ha terminado de ser atendida por la actividad. El procedimiento realiza las siguientes acciones: inserta al inicio de la lista *timelist* la corrutina que acaba de terminar el servicio en la actividad, enseguida toma la primera corrutina que se encuentre en la línea de espera de la actividad y le transfiere el control. Entonces la corrutina reactivada se suspenderá con el *procedimiento hold*, el cual transferirá el control nuevamente a la corrutina suspedida que se encuentra al inicio de la lista *timelist*.

La siguiente corrutina que se activa es (ja), ejecutando los bloques 15, 2 y 16 (fig. 7). El estado de las variables hasta éste momento es el siguiente:

simduration:100.0 *time*: 13.4 *running*: ja- 13.4

Actividad:	estado:	entidad:	línea de espera:
1	1	b	

timelist:
g - 14.4
jb - 18.4
m - 100.1

blocklist:

Problema 3.

El siguiente caso a analizar es cuando se encuentran varias actividades de entrada en paraleio, pero niguna de ellas cuenta con línea de espera, para el ejemplo especificaremos tres actividades de entrada como se muestra en la siguiente figura 11.

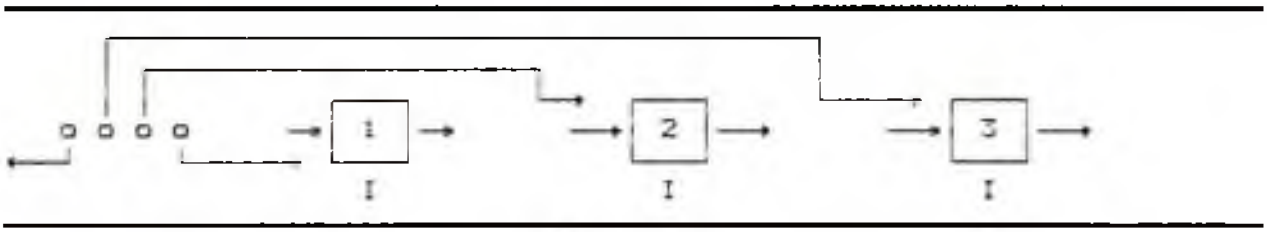


Figura 11.

Inicialmente las tres actividades se encuentran en estado ocioso. Al arribo de la tres primeras entidades, éstas pasan a ser atendidas por las actividades de acuerdo a los bloques 2, 3, 4, 5 8 y 9 (fig. 7).

Cuando las tres actividades se encuentren ocupadas y llega una entidad, esta se pierde del sistema. La actividad que termine primero de dar servicio será la que tome a la próxima entidad que arribe al sistema.

Las variables se encuentran en el siguiente estado despues del arribo de las primeras tres entidades:

simduration:100.0 *time*: 13.4 *running*: g - 13.4

Actividad:	estado:	entidad:
1	1	a
2	1	b
3	1	c

timelist:
ja- 15.6
jb - 18.4
jc - 19.3
m - 100.1

blocklist:

Problema 4.

El problema establece un caso general, donde tres actividades en paralelo contando cada una con su línea de espera. Si la actividad 1 tiene una longitud de cola de 3, la actividad 2 tiene una cola de 2 y la actividad 3 una cola de 4.

Al arribo de las tres primeras entidades, éstas se distribuyen en las actividades de igual forma que en el problema anterior. La decisión de que línea de espera seleccionarán las próximas entidades que arriben al sistema esta descrito por la *función gatefree*, el cual decide si la entidad puede avanzar a una actividad sucesora a la actividad en la cual se encuentra, si no es posible que avance o si la actividad no tiene actividades sucesoras regresa el valor de *nil* indicando que no se encuentra una actividad disponible. En caso contrario regresa como valor el apuntador a la actividad sucesora.

Este procedimiento efectúa la selección de la posible actividad sucesora mediante el siguiente algoritmo: ordena en forma ascendente las posibles actividades sucesoras de acuerdo a la siguiente expresión.

longitud de la cola + estado de la actividad

Enseguida prueba para cada una de las actividades sucesoras la siguiente condición:

la actividad se encuentra en estado ocioso o la linea de espera no se encuentra saturada ?.

La primera actividad que confirme la condición anterior será la seleccionada como la actividad subsecuente.

Para el arribo de las siguientes entidades estas se van distribuyendo en el sistema de acuerdo a los bloques 2, 3, 4, 5, 6 y 7 (fig. 7). Enseguida se analiza el siguiente caso si durante la simulación el sistema tiene el siguiente estado (figura 12):

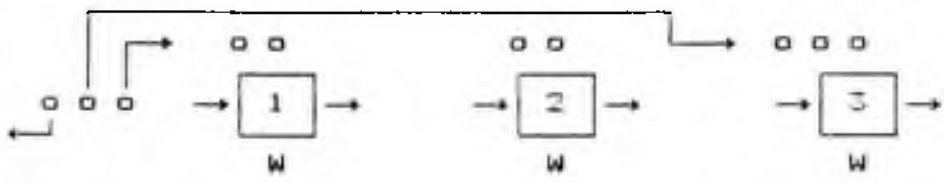


Figura 12.

Al arribo de la siguiente entidad, ésta pasa a formarse en la línea de espera de la actividad 1, ya que las líneas de espera de las actividades 1 y 2 son iguales en longitud, pero la cola de la actividad 2 se encuentra saturada y la cola de la actividad 1 todavía puede contener una entidad más. En este momento las colas de las actividades 1 y 2 se encuentran saturadas, y no podrán contener mas entidades que arriben en el futuro.

La siguiente entidad que lleque al sistema se formará en la línea de espera de la actividad 3, ya que es la única cola que no se encuentra saturada.

Para las siguientes entidades que arriben al sistema, éstas se perderán, debido a que las líneas de espera de las tres actividades se encuentran saturadas.

El estado de las variables correspondiente a la figura 12, es la siguiente:

simduration:100.0 *time*: 23.1 *running*: g - 23.1

Actividad:	estado:	entidad:	línea de espera:
1	1	a	d g
2	1	b	e h
3	1	c	f i j

timelist:
j_a- 25.7
j_b - 19.4
j_c - 22.3
m - 100.1

blocklist:

El evento siguiente es el arribo de la corrutina (j_k), la cual se perderá del sistema ya que se encuentra saturado. Suponiendo que el tiempo en cual se generará la siguiente corrutina es dentro de 1.5 unidades, entonces la corrutina generate se suspenderá con un tiempo de reactivación igual a 26.2 unidades. La corrutina que se reactivará será (j_a) dejando el estado de las variables como sigue:

simduration:100.0 *time*: 25.7 *running*: ja- 25.7

Actividad:	estado:	entidad:	línea de espera:
1	0		d g
2	1	b	e h
3	1	c	f i j

timelist:
g - 26.2
jb - 27.4
jc - 29.3
m - 100.1

blocklist:

En este momento el algoritmo ejecuta los bloques 10, 13 y 14 (fig. 7). Pasando a recibir servicio la corrutina (jd) que se encontraba suspendida en el bloque 7 (fig. 7), para continuar con los bloques 8 y 9. La corrutina (ja) vuelve a tomar el control en el bloque 15 (*procedimiento signal*) y ejecuta los bloques 2 y 16.

La siguiente corrutina que arribe al sistema pasará a formarse al final de la línea de espera de la actividad 1, dejando al sistema como sigue:

simduration:100.0 *time*: 25.7 *running*: ja- 25.7

Actividad:	estado:	entidad:	línea de espera:
1	1	d	f l
2	1	b	e h
3	1	c	f i j

timelist:
g - 26.2
jb - 27.4
jc - 29.3
jd - 31.6
m - 100.1

blocklist:

Problema 5.

El problema consiste de la red de actividades más sencilla en problemas de actividades en serie. Considerando que ninguna de las actividades posee cola de espera. La figura 13 muestra la interconexión entre las actividades.

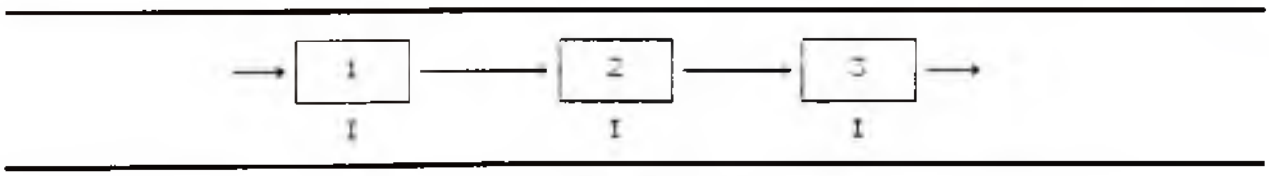


Figura 13.

Los arribos de las entidades para el problema se comportan de igual forma que el ejemplo 1, para la actividad de entrada 1.

La simulación del sistema se desarrollará normalmente, si el tiempo de servicio de la actividad 3 es menor que el tiempo de servicio de la actividad 2 y ésta a la vez menor que el tiempo de la actividad 1. Las entidades entrarán al sistema sin sufrir ningún retardo dentro de éste.

Pero el problema se suscita cuando los tiempos de servicio de las últimas actividades es mayor que los tiempos de las actividades predecesoras, ocasionando bloqueos en el sistema. Para ilustrar el anterior problema supongamos que el sistema se encuentra en el estado de la figura 14.

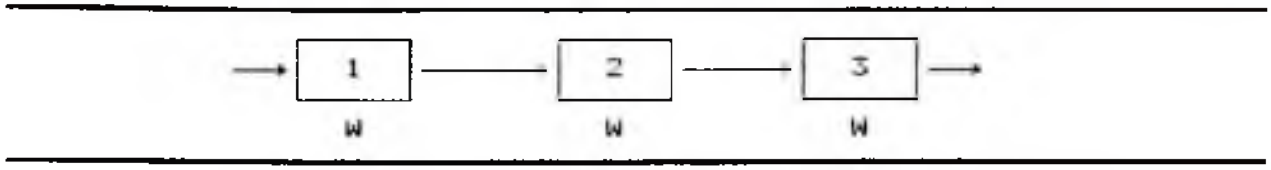


Figura 14.

El estado de las variables es el siguiente:

simduration:100.0 *time*: 25.7 *running*: jb - 25.7

Actividad:	estado:	entidad:
1	1	c
2	1	b
3	1	a

timelist:
jc - 29.3
ja - 31.6
g - 33.5
m - 100.1

blocklist:

En este momento la corrutina que tiene el control es (jb), que ha terminado de ser servida por la actividad 2, debiendo pasar a la actividad 3, pero ésta todavía se encuentra ocupada y además no tiene linea de espera, por lo que la actividad 2 y la corrutina (jb) adquieren el estado de bloqueo. Dejando el estado de las variables como sigue:

simduration:100.0 *time*: 29.3 *running*: jc - 29.3

Actividad:	estado:	entidad:
1	1	c
2	2	b
3	1	a

timelist:	*blocklist*:
ja - 31.6	jb
g - 33.5	
m - 100.1	

Se puede observar que la corrutina (jb) se encuentra formada en la lista *blocklist*. Las corrutinas bloqueadas se insertarán al final de la lista de acuerdo al orden de como se vayan bloqueando.

El evento de bloqueo de una corrutina lo realiza el *procedimiento wait* el cual inserta la corrutina bloqueada en la lista *blocklist*, al final de esta. Enseguida hace el llamado del *procedimiento schedule* para determinar la próxima corrutina que adquirirá el control y finalmente realiza el cambio de control con el *procedimiento transfer*.

La actividad 2 no podrá recibir ninguna otra entidad proveniente de la actividad 1, hasta que la actividad 3 se desocupe. Si la actividad 1 termina antes que la actividad 3 entonces la actividad 1 también se bloqueará, y no podrá recibir entidades que traten de ingresar al sistema, por lo que el sistema se encuentra bloqueado por la actividad 3. La siguiente figura 15 muestra el estado del sistema hasta este momento.

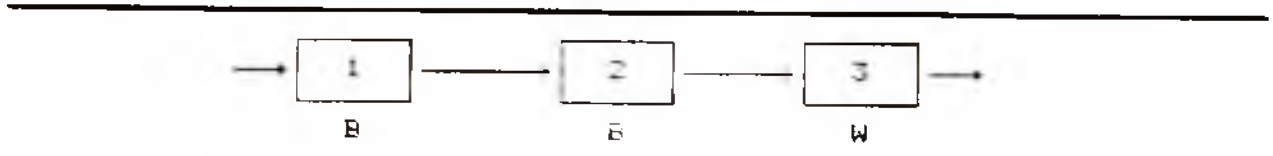


Figura 15.

El estado de las variables es el siguiente:

simduration:100.0 *time*: 31.6 *running*: ja - 31.6

Actividad:	estado:	entidad:
1	2	c
2	2	b
3	1	a

timelist:	*blocklist*:
g - 33.5	jb
m - 100.1	jc

En este momento, el control lo tiene la corrutina (ja), que es la que está ocasionando el bloqueo en el sistema. De acuerdo al algoritmo de la fig. 7, ésta corrutina se encuentra en el bloque 10 y continúa su desarrollo por los bloques 13 y 15. En el último bloque, se realiza el llamado al *procedimiento signal*, el cual reactiva a la corrutina que se encuentre al inicio de la lista *blocklist*, tomando el control la corrutina (jb); la cual, inicia su desarrollo a partir del bloque 18 y continuando con los bloques 11, 12, 13 y 15, también en ésta corrutina se realiza un llamado al *procedimiento signal*, reactivando a la corrutina (ja).

El estado de las variables en este momento es el siguiente:

simduration:100.0 *time*: 31.6 *running*: jc - 31.6

Actividad:	estado:	entidad:
1	2	c
2	2	b
3	1	a

timelist:
jb - 31.6
ja - 31.6
g - 33.5
m - 100.1

blocklist:

La corrutina (ja) inicia su desarrollo a partir del bloque 18 y continúa su ejecución con los bloques 11, 12 y 15 (fig. 7), ya que no existen más corrutinas que desbloquear; continúa con lo bloques 2, 3, 4, 5, 8 y 9, suspendiendose con el *procedimiento hold* y pasando el control a la corrutina (jb), que continúa su desarrollo a partir del bloque 15 y continuando con los bloques 2, 3, 4, 5, 8 y 9, suspendiendose y pasando el control a la corrutina (ja), la cual también se reactiva en el bloque 15 y continúa con los bloques 4 y 16, saliendo del sistema. El estado de las variables en este momento es el siguiente:

simduration:100.0 *time*: 33.5 *running*: g - 33.5

Actividad:	estado:	entidad:
1	0	
2	1	c
3	1	b

timelist:
jc - 38.3
jb - 42.5
m - 100.1

blocklist:

Y el sistema se encuentra como se observa en la figura 16.

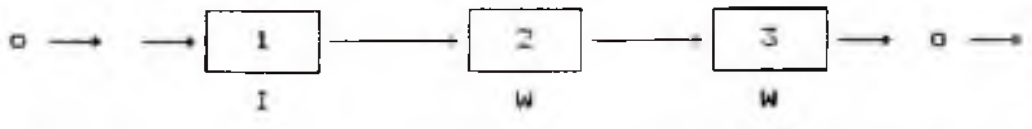


Figura 16.

La siguiente entidad que arribe al sistema entrará directamente a recibir servicio por la actividad 1.

Problema 6.

Ahora consideremos el siguiente problema (figura 17). Considerando que el sistema se encuentra en el siguiente estado:

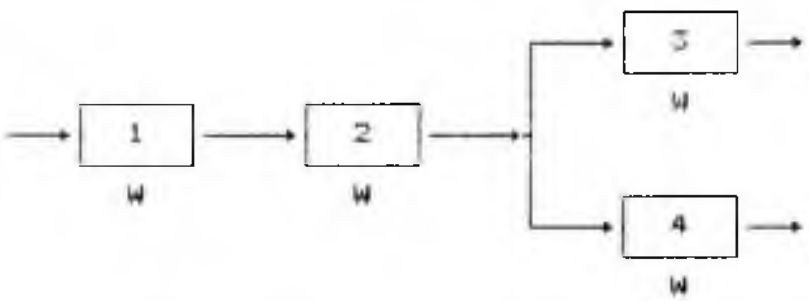


Figura 17.

Considerando que ninguna de las actividades tiene línea de espera y los tiempo de servicio de las actividades 3 y 4 iguales y el tiempo de servicio de la actividad 1 es menor que el tiempo de servicio de la actividad 2.

El estado de la variables se encuentra como sigue:

simduration:100.0 *time*: 26.5 *running*: jd - 26.5

Actividad:	estado:	entidad:
1	2	d
2	1	c
3	1	a
4	1	b

timelist:
jc - 28.5
ja - 30.1
jb - 35.4
g - 36.7
m - 100.1

blocklist:

En este momento el control lo tienen la corrutina (jd), la cual desea pasar a ser servida por la actividad 2; pera ésta no se encuentra disponible por lo que la actividad 1 y la corrutina (jd) adquieren el estado de bloqueo. Dejando al sistema en el siguiente estado:

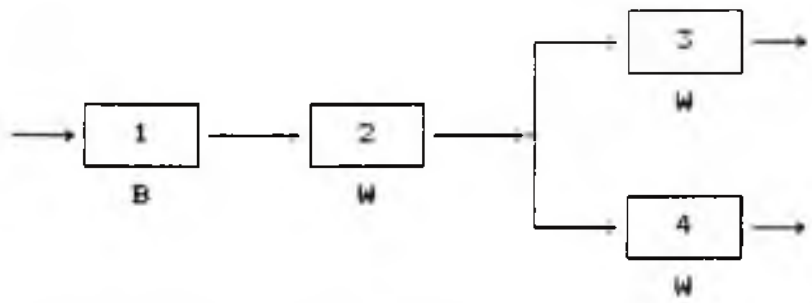


Figura 18.

El siguiente evento es la terminación de servicio de la actividad 2 y el estado de las variables es el siguiente:

simduration:100.0 *time*: 28.5 *running*: jc - 28.5

Actividad:	estado:	entidad:
1	2	d
2	1	c
3	1	a
4	1	b

timelist:	*blocklist*:
ja - 30.1	jd
jb - 35.4	
g - 36.7	
m - 100.1	

La corrutina (jc) al terminar de atenderse, encuentra que las actividades sucesoras se encuentran ocupadas, adquiriendo el estado de bloqueo e insertándose en la lista *blocklist*. Dejando el estado de las variables como sigue:

simduration:100.0 *time*: 30.1 *running*: ja - 30.1

Actividad:	estado:	entidad:
1	2	d
2	2	c
3	0	
4	1	b

timelist:	*blocklist*:
jb - 35.4	jd
g - 36.7	jc
m - 100.1	

Al tomar el control la corrutina (ja), indicando la terminación de su servicio; desbloquea la corrutina (jd), la cual al reactivarse encuentra que la actividad 2 continúa bloqueada, por lo cual nuevamente se suspende y pasa a formarse al final de la lista *blocklist*; pasando el control nuevamente a la corrutina (ja). En este momento el estado de las variables es el siguiente:

simduration: 100.0 *time*: 30.1 *running*: ja - 30.1

Actividad:	estado:	entidad:
1	2	d
2	2	c
3	0	
4	1	b

timelist:	*blocklist*:
jb - 35.4	jc
g - 36.7	jd
m - 100.1	

Al adquirir nuevamente el control la corrutina (ja), ésta se encuentra tratando de desbloquear a todas las corrutinas existentes en la lista *blocklist*, por lo que pasa el control ahora a la corrutina (jc); la cual, encuentra que si puede avanzar a la actividad 3, ejecutando los bloques 18, 11, 12 y 15 (fig. 7), en éste último bloque la corrutina (jc), también intenta desbloquear a la corrutinas bloqueadas, pasando el control a la corrutina (jd). En este momento el estado de las variables se encuentra como sigue:

simduration: 100.0 *time*: 30.1 *running*: jd - 30.1

Actividad:	estado:	entidad:
1	2	d
2	0	
3	0	
4	1	b

timelist:	*blocklist*:
jc - 30.1	
ja - 30.1	
jb - 35.4	
g - 36.7	
m - 100.1	

Teniendo el control la corrutina (jd), encuentra que ahora si puede avanzar a la siguiente actividad bloque 11 (fig. 7), realizando los siguientes bloques 12, 13, 15, 4, 5, 8 y 9, en el último bloque se suspende y transfiere el control a la corrutina (jc); la cual se reactiva en el bloque 15 y continúa su ejecución con los bloques 4, 5, 8 y 9; bloque donde se suspende y pasa el control a la corrutina (ja), la cual ejecuta los bloques 15, 4 y 16, terminando sus existencia dentro del sistema. El estado de variables se encuentra hasta éste momento como sigue:

simduration:100.0 *time*: 35.4 *running*: jb - 35.4

Actividad:	estado:	entidad:
1	0	
2	1	d
3	1	c
4	1	b

timelist:
g - 36.7
ja - 45.2
jc - 46.3
m - 100.1

blocklist:

En este momento, queda resuelto el problema de bloqueo en el sistema.

II.6. CALCULO DE PARAMETROS ESTADISTICOS.

Los parámetros estadisticos que se reportan corresponden a una sola actividad dentro de la red de actividades. Los parámetros que se reporta son los siguientes:

- Entidades procesadas.
- Entidades pendientes.
- Longitud media de cola.
- Longitud maxima de cola.
- Tiempo media de espera.
- Número de veces bloqueada.
- Porcentaje de tiempo en estado de bloqueo.
- Porcentaje de tiempo en estado ocioso.
- Tiempo medio de residencia.

Los dos primeros parámetros no requieren de un procedimiento muy complejo para su cálculo, para las *entidades procesadas* únicamente se lleva el conteo de las entidades que terminan de ser servidas por la actividad, y para las *entidades pendientes* se resta las entidades que entraron a la línea de espera de la actividad menos las entidades procesadas.

Longitud media de cola.

Para poder conceptualizar mejor la forma de cálculo de este parámetro se explicará en base a un ejemplo: Considerando un sistema de colas cualquiera el cual es simulado un tiempo T, y llamemos Lq(t) el número de entidades que se encuentran en la cola de una actividad cualquiera al tiempo t. La simulación de una actividad del sistema es presentado en la siguiente figura.

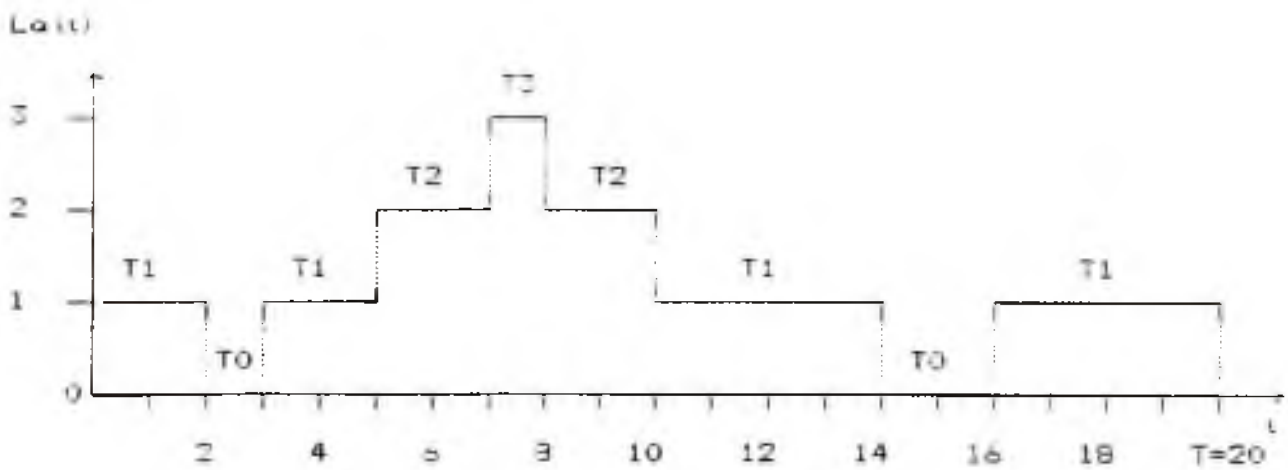


Figura 20.

T_i denota el intervalo de tiempo total durante [0, T] en el cual la cola de una actividad contiene i entidades. En la figura se observa que $T_0 = 3$, $T_1 = 12$, $T_2 = 4$, $T_3 = 1$. En general $\sum_{i=0}^{\infty} T_i = T$.

El promedio de entidades que se encuentran en la línea de espera esta denotado por Lq y se encuentra definido por:

$$Lq = \frac{1}{T} \sum_{i=0}^{\infty} i T_i$$

Para nuestro ejemplo:

$$Lq = \frac{1}{20} [0\ (3) + 1\ (12) + 2\ (4) + 3\ (1)] = 1.15$$

Longitud máxima de cola.

En este parámetro nos indica el máximo de entidades que se encontraron en la cola al mismo tiempo, para la grafica anterior 3 es la longitud máxima.

Tiempo medio de espera.

Es el tiempo promedio de estancia de las entidades en la cola y esta denotado por Wq y simulando un sistema cualquiera durante un periodo de tiempo T y guardando el tiempo que cada entidad gasta en la línea de espera o cola, digamos $W_1, W_2, \ldots, W_N$, donde N es el número de entidades que entraron a la línea de espera durante [0,T]. El tiempo promedio de espera se encuentra definido por:

$$Wq = \frac{1}{N} \sum_{i=1}^{N} W_i$$

TIEMPO DE ESPERA EN LA COLA DE UNA ACTIVIDAD

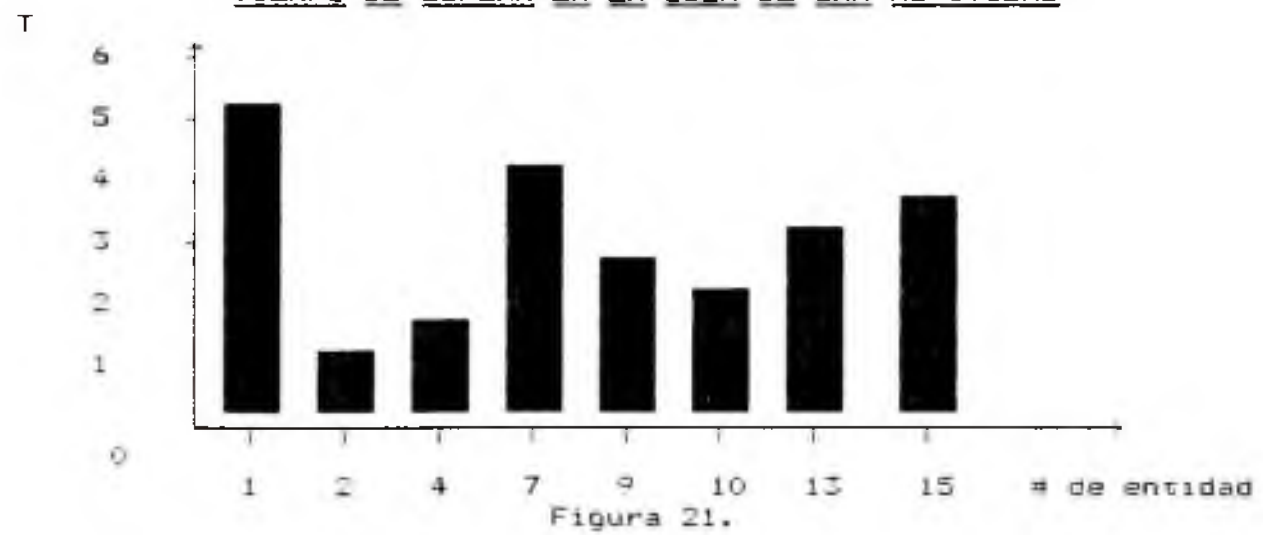


Figura 21.

Para el ejemplo de la figura 21.

$$W_Q = \frac{1}{8} [\ 5 + 1 + 1.5 + 4 + 3 + 2.5 + 3.5 + 4\ ] = 3.06$$

Número de veces bloqueada.

Es la cantidad de veces que la actividad se encontró en estado de bloqueo. En el ejemplo de la figura 22 el valor es:

$$N_B = 3$$

Porcentaje de tiempo en estado de bloqueo.

Es el porcentaje del tiempo de simulación T, que la actividad permaneció en estado de bloqueo, se denota por T_B, si durante el periodo [0,T] ocurrieron N bloqueos, entonces se registraron N periodos de tiempo de bloqueo, digamos B_1, B_2,...,B_N, entonces el porcentaje de tiempo de bloqueo se encuentra definido por:

$$T_B = \frac{1}{T} \sum_{i=1}^{N} B_i$$

HISTORICO DE LA SIMULACION DE UNA ACTIVIDAD

ESTADOS BLOQUEO: ▒ TRABAJANDO: ▒

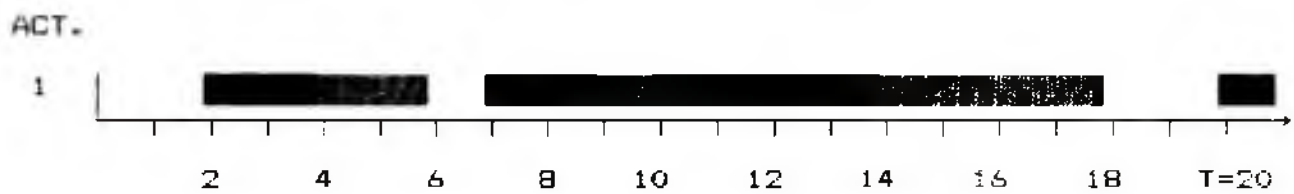


Figura 22.

El porcentaje de tiempo en estado de bloqueo para la figura 22 es:

$$T_B = \frac{1}{20} [\ 2 + 1 + 4\ ] = 0.35$$

Porcentaje de tiempo en estado ocioso.

Es el porcentaje del tiempo de simulación T, durante el cual la actividad se encontró sin realizar ninguna actividad, ya sea por no tener ninguna entidad en espera de servicio o por encontrarse en estado de bloqueo. Se encuentra denotado por T_I, y los periodos de tiempo en estado ocioso durante el periodo [0,T] por I_1, I_2, I_3,...,

Se encuentra definido por:

$$T_I = \frac{1}{\gamma} \left\{ \sum_{i=1}^{\infty} I_i + \sum_{i=1}^{N} B_i \right\}$$

Para el ejemplo de la figura 22.

$$T_I = \frac{1}{20} \left\{ [\ 2 + 1 + 2\ ] + [\ 2 + 1 + 4\ ] \right\} = 0.6$$

Tiempo medio de residencia.

Es el tiempo promedio de residencia de las entidades en la actividad, se denoto por T_R. Los tiempos se calculan desde que la entidad entra a formarse en la línea de espera o entra en servicio directamente si se encuentra desocupada la actividad, hasta que termina de ser servida, estos tiempos estan denotados por R_1, R_2, ..., R_N, en donde N es la cantidad de entidades que terminaron de ser atendidas. El parámetro se define como sigue:

$$T_R = \frac{1}{N} \sum_{i=1}^{N} R_i$$

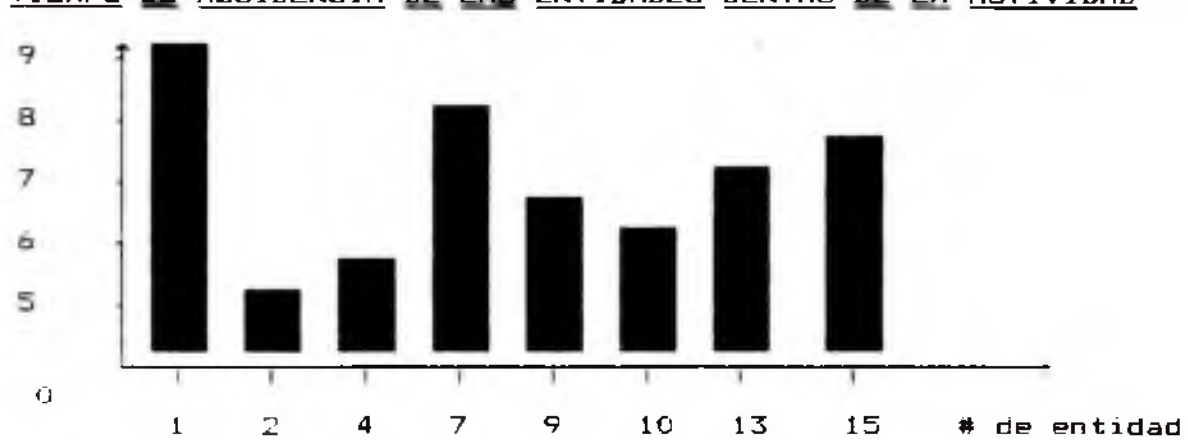


Figura 23.

Para el ejemplo de la figura 23.

$$T_R = \frac{1}{8} [\ 9 + 5 + 5.5 + 7 + 6.5 + 6 + 7 + 7.5\ ] = 6.6875$$

II.7. GENERACION DE NUMEROS ALEATORIOS

En todos los experimentos de simulación existe la necesidad de generar valores de variables aleatorias que representan a una cierta distribución de probabilidad. Durante un experimento de simulación, el proceso de generar un valor aleatorio de una distribución particular puede repetirse tantas veces como se desee y tantas veces como distribuciones de probabilidad existan en el experimento de simulación. Es conveniente señalar que el proceso de generación de variables no uniformes se hace a partir de la generación de números rectangulares (rango 0..1).

Independientemente de el procedimiento que se utilice para la generación de los números rectangulares, éstos deben de poseer ciertas caracteristicas que aseguren o aumenten la confiabilidad de los resultados obtenidos de la simulación. Tales caracteristicas son:

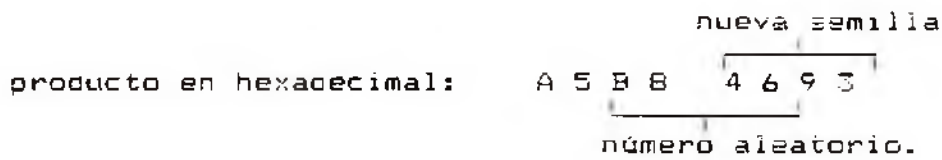
1. Uniformente distribuidos.
2. Estadisticamente independiente.
3. Reproducibles.
4. Periodo largo (sin repetición dentro de una longitud determinada de la sucesión).
5. Generados a través de un método rapido.
6. Generados a través de un método que no requiera mucha capacidad de almacenamiento de la computadora.

El procedimiento que se utiliza en este simulador para la generación de números aleatorios es el método del lenguaje GPSS el cual es una modificación de los métodos congruencial multiplicativo y el "midsquare". En donde se cuentan con dos clases de números llamados multiplicador y semilla. Estos números deben ser enteros positivos menores que 2^{31}, la semilla y el multiplicador se inicializan por el programador pero la semilla es la única que cambia de valor durante la operación de el generador. El algoritmo es el siguiente:

- La semilla es multiplicada por el multiplicador especificado anteriormente, produciendo un número de 32 bits.

- los 16 bits intermedios serán el nuevo numero pseudoaleatorio y los 16 bits menos significativos pasarán a ser la nueva semilla.

La siguiente figura muestra el algoritmo anteriormente descrito.



De acuerdo a pruebas realizadas en la referencia [3] en donde se efectúa la simulación de un sistema M/M/1, con diferentes generadores, se concluye que el método GPSS es el mejor en la producción de números pseudoaleatorios en el campo de la teoría de colas.

II.8. CALCULO DE DISTRIBUCIONES PROBABILISTICAS.

En todo modelo de simulación estocástico, existen una o varias variables aleatorias interactuando. Generalmente, estas variables siguen distribuciones de probabilidad teóricas o empíricas diferentes a la distribución uniforme. Por consiguiente, para simular este tipo de variables, es necesario contar con un generador de números uniformes y una función que a través de un método específico transforme estos números en valores de la distribución deseada. Existen varios procedimientos para lograr este objetivo, los más comunes y más difundidos se pueden mencionar: 1) El método de la transformada inversa, 2) El método de aceptación y rechazo, 3) El método de composición y 4) Procedimientos especiales.

Las distribuciones con las que cuenta el simulador son las siguientes:

- Constante.
- Uniforme.
- Exponencial.
- Erlang-K
- Gamma
- Normal.
- Poisson.

A continuación se describirán los algoritmos que se utilizan para generar cada una de las variables aleatorias, los procedimientos se encuentran mayor detallados en la referencia [4], en donde fueron consultados.

Cada actividad en la red cuenta con su propia función de distribución que describe su tiempo de servicio. Las funciones requieren como parámetros la media y la varianza, para generar números aleatorios. Cada actividad cuenta con las siguientes variables que describen a la función y a sus parámetros, *serv_time* almecena la media, *variance* almacena la varianza, *distr_type* guarda el tipo de función ya que estas se referencian mediante un dígito. *flag_frec* variable lógica que se utiliza para especificar que el tiempo de servicio se encuentra descrito por una tabla de distribución de frecuencias.

Constante.

Este tipo de distribución únicamente requiere del valor constante que se desea generar cada vez que sea llamada esta función.

Uniforme.

Las variables uniformes se generar por medio del método de la transformada inversa, la función que genera las variables aleatorias uniformes es la siguiente:

Considerando una variable aleatoria X que esta uniformemente distribuida con un intervalo [a, b].

$$X = a + (b - a)\ R$$

R = variable pseudoaleatoria.

Exponencial.

El método de la transformada inversa se utiliza para generar variables exponenciales. La función es la siguiente:

$$X = -\ media * Ln\ R$$

de donde:

media: es el valor que se introduce como parámetro en la definición de la distribución.
R: variable pseudoaleatoria.

Erlang-k.

La variable Erlang es generada a partir de la siguiente fórmula. El método de covolución [2] se utiliza para generar esta variable.

$$X = \frac{-1}{K\ \theta} Ln \left(\prod_{i=1}^{k} Ri \right)$$

de donde:

$$\theta = \frac{1}{media}$$

$$K = \frac{1}{varianza\ \theta^2}$$

Ri = Variable Pseudoaleatoria.

Ha sido demostrado por algunos matemáticos que esta distribución es justamente la suma de K variables aleatorias exponenciales cada una con un valor esperado igual a:

$$\frac{-1}{media} = \frac{-1}{K\ \theta}$$

Con la anterior ecuación se optimiza la producción de variables aleatorias de tipo erlang.

Gamma.

El siguiente algoritmo genera variables aleatorias de tipo gamma variando con un parámetro de escala θ y un parametro de forma β, es decir con una media de $1/\theta$ y una varianza de $1/\beta\theta^2$. Los pasos son los siguientes:

Paso 1. Calcule $a = (2\beta - 1)^{1/2}$
calcule $b = 2\beta - \ln 4 + 1 / a$.

Paso 2. Genere dos variables aleatoria R_1 y R_2.

Paso 3. Calcule $X = \beta [R_1/(1 - R_1)]^a$.

Paso 4a. Si $X > b - \ln (R_1^2 R_2)$, elimine X y regrese al paso 2.

Paso 4b. Si $X \leq b - \ln (R_1^2 R_2)$ entonces X es la variable deseada.

las variables generadas en el paso 4b. tendrán una media y una varianza igual a β. Pero si se desea que se tenga una media $1/\theta$ y varianza $1/\beta\theta^2$ entonces se realiza el siguiente calculo.

Paso 5. Substituya X por $X/\beta\theta$.

El método de aceptación y rechazo se utilizó para generar esta variable.

Normal.

Puesto que no es posible expresar la distribución acumulada de la distribución normal en forma explicita, entonces no es posible utilizar para la generación de números al azar. el método de la transformada inversa. En lugar de este método, se puede hacer uso del teorema del limite central. el cual establece que la suma de n variables aleatorias independientes se aproxima a una distribución normal a medidad que n se aproxima a infinito, el teorema es el siguiente:

Si $x_1, x_2, \ldots, x_n$ es una secuencia de n variables aleatorias independientes con media $(x_i) = \mu$ y var $(x_i) = \sigma_i^2$ (ambas finitas) y $Y = a_1x_1 + a_2x_2 + \ldots + a_nx_n$; entonces bajo ciertas condiciones generales:

$$Z = \frac{Y - \sum_{i=1}^{n} a_i\mu_i}{\sqrt{\sum_{i=1}^{n} a_i^2\sigma_i^2}}$$

tiene una distribución estandar a medida que n se aproxima a infinito. Si las variables que se están sumando son uniformes en el intervalo [0..1], entonces:

$$Z = \frac{\sum_{i=1}^{n} R_i - n/2}{\sqrt{n/12}}$$

tiene una distribución normal estándar. Puesto que la normal estándar de una variable aleatoria x distribuida normalmente se obtiene como:

$$Z = \frac{x - \mu}{\sigma}$$

también se ha comprobado que utilizando un valor de n = 12, laconfiabilidad de los valores simulados es bastante aceptable. Además, es muy poco probable obtener valores simulados de x en la región $\mu + 6\sigma < x < \mu - 6\sigma$. También, es obvio que utilizando un valor de n = 12, la fórmua se simplifica a:

$$x = \mu + \sigma \left(\sum_{i=1}^{12} R_i - 6 \right)$$

Poisson.

El procedimiento para generar variables aleatorias de tipo poisson, n, se encuentra definido por los siguientes pasos:

$$\alpha = \text{media}$$

Paso 1. Inicialice n = 0, p = 1.

Paso 2. Genere el número aleatorio R_{n+1} y sustituya p por $p * R_{n+1}$.

Paso 3. Si $p < e^{-\alpha}$, entonces se acepta N = n. En caso contrario icrementese la n por uno, y regrese al paso 2.

Cuando $\alpha \geq 15$, el algoritmo anterior toma demasiados pasos para generar el número aleatorio, entonces se utiliza la técnica de aproximación basada en la distribución normal que trabaja bastante mejor cuando la media [α] es grande.

$$Z = \frac{N - \alpha}{\sqrt{\alpha}}$$

es aproximadamente distribuido con una media de cero y varianza de 1, la cual indica una técnica de aproximación. Primero se genera una variable normal estandar Z. Enseguida se genera la variable poisson deseada por:

$$N = \alpha + \sqrt{\alpha}\ Z - 0.5$$

II.9. DISTRIBUCION DE FRECUENCIAS.

Cuando el tiempo de servicio de una actividad o el tiempo de arribo de las entidades al sistema, no puede ser descrito por una distribución probabilistica, entonces se establece una tabla de frecuencias, la cual unicamente cuenta con dos columnas que son tiempo vs. frecuencia. Esta tabla tiene que ser transformada a tiempo vs. frecuencia acumulada. Se introduce el segundo tipo de tabla debido a que no todos los problemas cuentan con las frecuencias de eventos que ocurren en el mismo tiempo, sino que presentan la tabla de frecuencias acumuladas como dato, por lo que es necesario introducir esta tabla para estadarizar la entrada de la tabla. La siguiente tabla muestra los dos tipos de frecuencias y apartir de esta tabla se explicara como se generan los tiempos aleatorios.

tiempo	*No. de medidas*	*% ocurrencia*	*Acumulado %*
10	2	2	2
11	8	8	10
12	14	15	25
13	32	33	58
14	22	23	81
15	14	15	96
16	4	4	100

Figura 24.

El método para la generación de los números aleatorios consiste en los siguientes pasos.

- Se genera un número pseudoaleatorio entre [0..1] llamemoslo, R.
- Enseguida se busca el tiempo que cumpla la siguiente condicion:

frecuencia acumulada / 100 ≤ R.

La siguiente tabla muestra los valores de tiempo obtenidos para diferentes números aleatorios.

Número Aleatorio	tiempo
0.11	12
0.28	13
0.08	11
0.56	13
0.09	11

Figura 25.

Si se desea consultar más a fondo sobre el método anteriormente descrito consulte la referencia [8].

II.10. MANEJO DE LA MEMORIA DISPONIBLE PARA EL SIMULADOR.

Cuando se inicia la construcción de la red, las actividades se van creando de acuerdo a la necesidad del usuario, lo que quiere decir que el simulador no tiene límite en la cantidad de actividades a definir, ya que la estructura que representa a las actividades esta definida como una variable dinámica [8], y se localizan en la región del heap [8].

Una vez construida la red de actividades, el siguiente paso es la simulación de la misma, aquí es donde entra el manejo del resto de la memoria disponible. Como previamente se mencionó la necesidad de crear y destruir corrutinas, lo que implica a estar llamando los procedimientos de petición y liberación de bloques de memoria. Pero debido a que cada corrutina cuenta con sus propios apuntadores heapptr y heaporg, estos apuntadores son los que utiliza el manejador de memoria de turbo pascal para efectuar las operaciones de mantenimiento de bloques de memoria, no es recomendable efectuar peticiones de memoria desde las corrutinas, ya que se podría perder la referencia de un bloque de memoria apartado una vez que se haya efectuado el cambio de contexto entre las corrutinas y por consiguiente se modifican los valores de los dos apuntadores anteriormente mencionados. Ocasionando al manejador de memoria conflictos en la administración de la memoria disponible.

Para evitar el problema de efectuar peticiones de memoria desde las corrutinas, se efectúa al principio de la simulación la petición de memoria que se va a requerir para llevar a cabo la simulación. Y esta memoria será administrada por los procedimientos activate y terminate (apéndice C).

Al iniciarse la simulación, se efectúa la petición de 50 bloques de memoria de un tamaño cada uno de 2500 bytes. Existe un vector llamado *job_descr*; que tiene un tamaño de 50 variables de tipo descr. Si la cantidad de memoria disponible no alcanza para la petición de los 50 bloques, se almacenará en la variable *corout_avail* la cantidad de bloques que se alcanzaron a pedir. Esta variable nos indicará la cantidad de corrutinas disponibles para el simulador. El tamaño se considera suficiente para la simulación.

Al iniciarse una corrutina se llama al procedimiento activate el cual se encarga de encontrar un bloque disponible dentro del vector *job_descr*. El bloque que se encuentre disponible tiene en su registro asociado la variable *flag_busy* con el valor de false, enseguida se cambia el valor de la variable a true; indicando que el bloque de memoria se encuentra ocupado.

Para terminar con la existencia de una corrutina se llama al procedimiento terminate, el cual unicamente habilita el bloque para que pueda ser utilizado por una nueva corrutina, cambiando el valor de la variable *flag_busy* a false, del registro asociado.

Con el algoritmo anterior se incrementa la velocidad en la activación de corrutinas evitando el llamado de procedimientos del manejador de memoria de turbo pascal. Se evita la posible saturación del vector *job_descr* a menos que se encuentren más de 50 corrutinas activas en el mismo momento.

La siguiente figura 15 muestra como se encuentran asignadas las variables de la estructura *descr* al bloque de memoria que apunta.

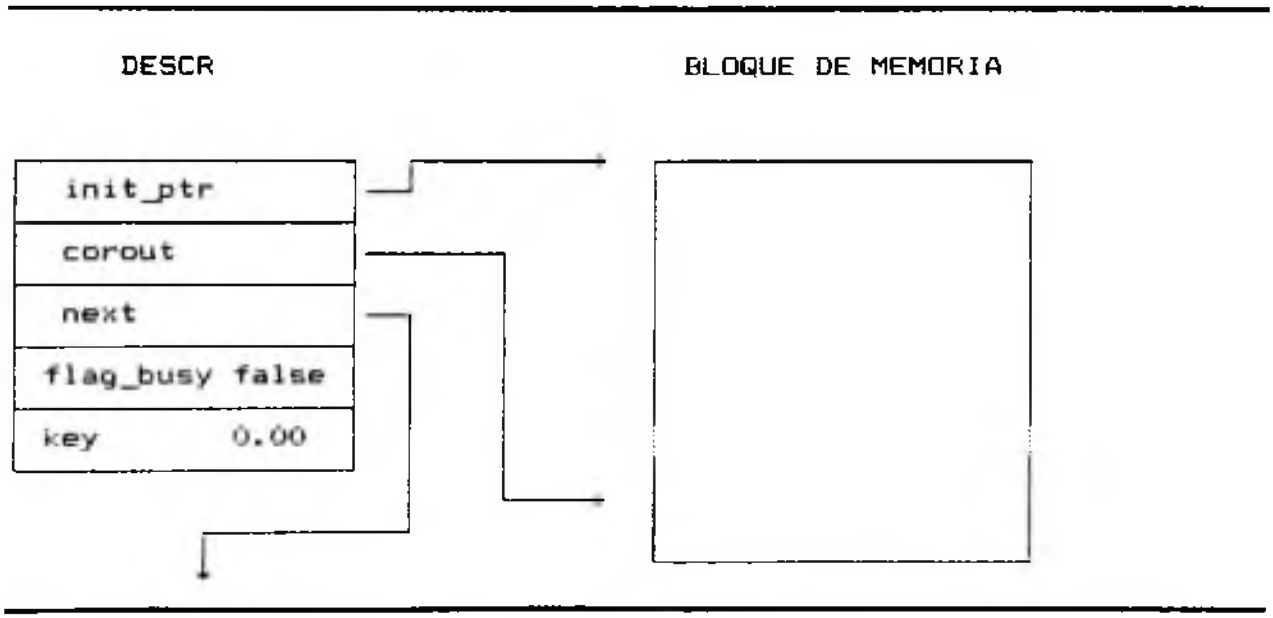


Figura 26.

II.11. CONCLUSIONES

- Este paquete ofrece algunas ventajas considerables sobre los lenguajes de simulación tales como GPSS, SLAM, SIMSCRIPT, etc. las ventajas son:

  - Ofrece una total interacción con el usuario, el cual no requiere de conocimientos de programación para la utilización del mismo, siendo una ventaja bastante significativa, ya que existe un gran rechazo por parte de los usuarios para el estudio de nuevos lenguajes de programación.

  - El simulador tiene la opción de poder observar en la pantalla el desarrollo de la simulación, cualidad recomendable para fines didácticos, diseño de sistemas y optimización de los mismos. Siendo también una ventaja sobre los lenguajes, ya que éstos únicamente producen una serie de resultados al final de la simulación o en el mejor de los casos producen un listado de los eventos que se efectuaron durante la simulación, al final de cuentas siguen siendo números, que son difíciles de inter pretar.

- El manejo del reloj de tiempo se lleva a cabo mediante el criterio de avance al próximo evento, el cual acelera la simulación, avanzando el reloj al tiempo en el cual se llevará a cabo el próximo evento.

- Se hace uso de corrutinas con la finalidad de poder tener una mejor representación de las entidades que fluyen dentro del sistema, disminuyendo la complejidad del algoritmo, en comparación si éste estuviese construido de forma secuencial.

- Para aquellos sistemas en los cuales existan problemas de congestionamiento o "Cuellos de Botella", el algoritmo se encuentra diseñado para tratar de desbloquear a las actividades lo más rápido posible evitando que el congestionamiento se siga extendiendo a las actividades predecesoras.

- Al inicio de la simulación se realiza la petición de toda la memoria para el desarrollo de la misma, optimizando la velocidad del algoritmo; evitando las operaciones de altas y bajas de bloques de memoria necesarios para la simulación.

CAPITULO III. EJEMPLOS DE APLICACION.

Ejemplo No. 1. **SIMULACION DE UNA RED SENCILLA DE ACTIVIDADES.**

El objetivo de este ejemplo es describir como se utiliza el simulador, desde la construcción de la red, almacenamiento de la red en el disco, simulación y obtención de reportes.

La red con la que se trabajará es la que se describe en la siguiente figura 27. Ninguna de las actividades cuenta con cola de espera.

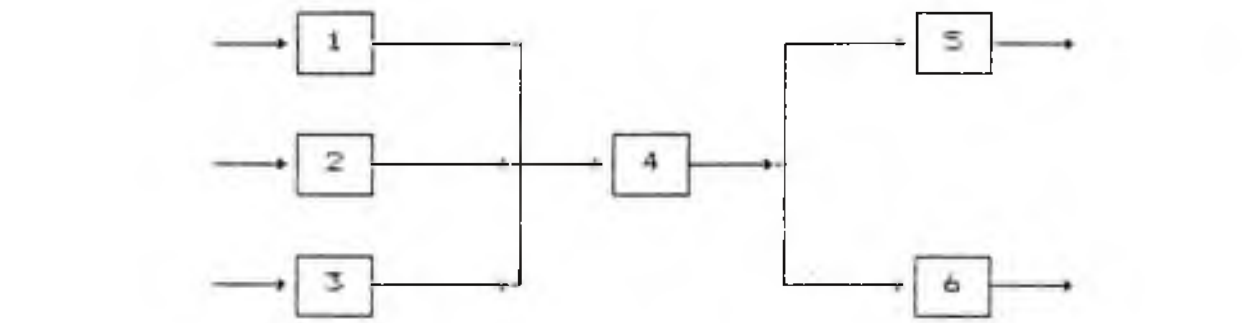


Figura 27.

Los tiempos de servicio de las actividades son los siguientes:

Actividad	Tiempo de Servicio	
1	Uniforme	10.54 min. ± 6.7 min.
2	Uniforme	13.50 min. ± 8.2 min.
3	Uniforme	11.23 min. ± 5.6 min.
4	Exponencial	26.87 min. ± 15.9 min.
5	Normal	18.24 min. ± 6.7 min.
6	Normal	17.67 min. ± 8.9 min.

El arribo de las entidades ésta descrito por la función Poisson 15.9 min ± 5.1 min. Se requiere simular el sistema por un periodo de 12 hrs.

Construcción de la red.

Se recomienda tener adjunto el apéndice [B], para poder observar las teclas y opciones que ofrece cada menú en la construccion de la red.

La red se contruye en la ventana del menú Edición. Primeramente se deben de introducir las actividades que serán de entrada a la red. Las actividades se especifican de la siguiente forma:

- Se especifica el número que identificará a la actividad, mediante la opción *Número de actividad*. El número no debe de seguir un orden forsozamente.

- Se establece la longitud de la cola que tendrá la actividad.

- Se establece el tiempo de servicio, por una distribución probabilística especificando el tipo de función, y sus parámetros que son la media y la varianza (figura 28).

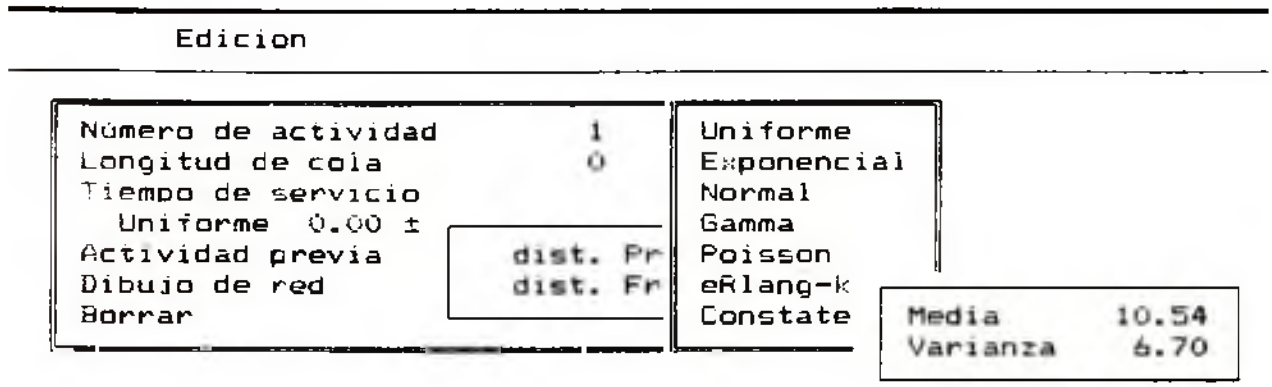


Figura 28.

Una vez introducidas las características de la actividad, estas serán desplegadas dentro de las sección de Resumen de actividades (figura 29). Aunque la actividad aparezca dentro de la sección, esta no pertenece todavía a la red, ya que todavía no se especifica la actividad que precede a la actividad que se encuentra editando.

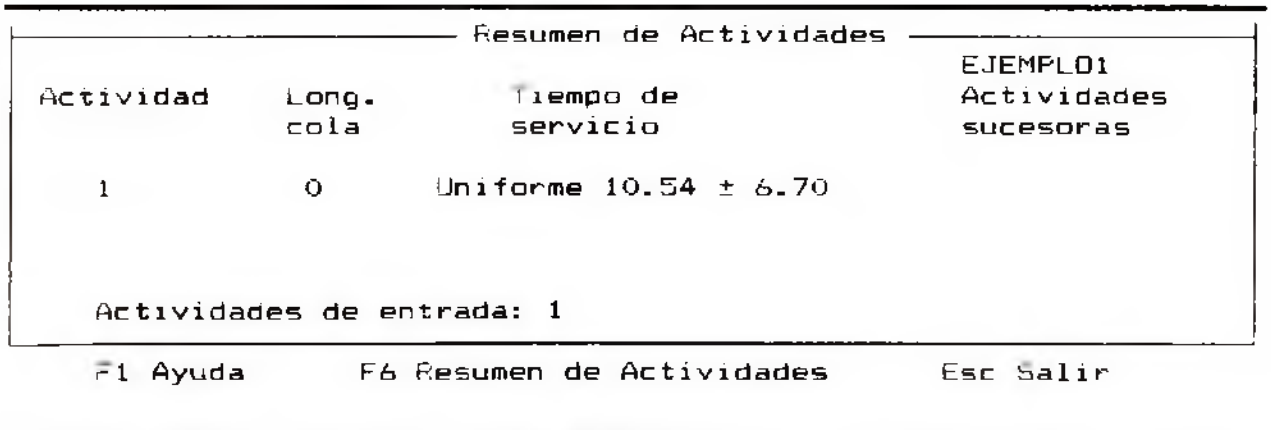


Figura 29.

- A continuación se une la actividad a la red, y se realiza con la opción *Actividad previa*. En caso de ser una actividad de entrada se introduce 0, y el número de la actividad aparecerá en el renglón de actividades de entrada. (figura. 29). En caso de no ser actividad de entrada se introduce el número de la actividad que será previa a la actividad que se ésta editando.

- Una vez unida la actividad a la red se procede a construir el dibujo de la red, seleccionando la opción *Dibujo de red*, apareciendo un cubo con el número de actividad, en la esquina superior izquierda de la pantalla, se localiza el cubo en el lugar deseado, utilizando las teclas de flechas para su desplazamiento en la pantalla. Se recomienda dibujar la líneas, una vez que se hayan establecido todos los cubos representativos de las actividades.

El paso anterior se puede saltar ya que una vez contruida toda la red se procede a hacer el dibujo de la misma. Lo único que hay que considerar es que se puede editar una actividad a la vez, por lo que unicamente se puede localizar en el dibujo la actividad que se encuentre editando.

- Una vez introducidas todas las actividades de entrada, se prosigue a construir la red con las demás actividades. El métodd que se utiliza es igual al anterior excepto en la especificación de las actividades previas que se describe a continuación.

Para nuestro ejemplo digamos que se encuentra editando la actividad número 4, una vez establecidos sus parámetros de tiempo de servicio y longitud de cola, se inicia su conexión con las actividades que preceden a la actividad, al seleccionar la opción *Actividad previa*, introducir 1, al realizarse esta operación se podrá observar en la sección *Resumen de Actividades*, que para la actividad 1 aparece la actividad 4 en la columna de actividades sucesoras (figura 30). De esta forma quedan interconectadas las actividades 1 y 4. En forma similar se establecen las ligas entre las actividades 2 y 3 con la actividad 4. Hay que señalar que la actividad previa que se especifique debe de existir, en caso contrario se producirá un error.

Resumen de Actividades

EJEMPLO1

Actividad	Long. cola	Tiempo de servicio	Actividades sucesoras
1	0	Uniforme 10.54 ± 6.70	4
2	0	Uniforme 13.50 ± 8.2	4
3	0	Uniforme 11.23 ± 5.6	4
4	0	Exponencial 26.87 ± 15.9	5 6

Actividades de entrada: 1 2 3

F1 Ayuda F6 Resumen de Actividades Esc Salir

Figura 30.

Al terminar de introducir todas las interconexiones que tiene la actividad 4 con las actividades que le preceden la sección de Resumen de Actividades se encuentra como lo muestra la figura 30. Con el método anterior se interconectan las actividades 5 y 6 con la actividad 4.

Guardar la red.

Se recomienda guardar periodicamente la red, con el fin de almacenar el avance de contrucción de la red, y evitar la pérdida de esta por una falla de energía eléctrica en la computadora.

Para guardar la red se realizan los siguientes pasos:

- Dejar el menú de Edición y pasar al menú de Archivo.

- Seleccionar la opción *Cambio dir.* con la finalidad de especificar la unidad de disco y directorio donde se guardarán los archivos de la red figura 31.

- Enseguida se selecciona la opción Guardar, si es la primera vez que se guarda la red, el sistema preguntará por el nuevo nombre, ya que el simulador asigna por omisión el nombre de WORK a una nueva red que se esté construyendo. Para las siguientes operaciones de guardado el simulador ya no preguntará por el nombre de la red, ya que esta tiene un nombre asignado. Para nuestro ejemplo introducir el nombre EJEMPLO1 como se muestra en la figura 33.

Cargar un archivo al simulador.

Para cargar una red del disco primeramente se debe de especificar la unidad de disco y directorio donde se encuentra la red, mediante la opción cAmbio dir del menú Archivo. Supongamos que el disco de ejemplos se encuentra en la unidad de disco B y los ejemplos se encuentran en el directorio Ejemplos del mismo disco. En la siguiente figura 31 se muestra la especificación del cambio del directorio.

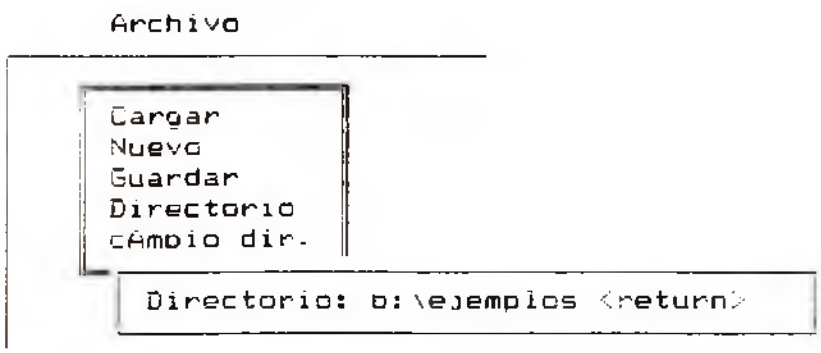


Figura 31. Cambio de Directorio.

Enseguida se selecciona la opción *Cargar* dentro del mismo menú, y se introduce el nombre de la red que se desee alimentar al programa, pero si se quiere seleccionar la red presione <space> y enseguida <return> después se desplegará una ventana con las posibles redes que se pueden cargar al simulador. La figura 32 muestra el procedimiento.

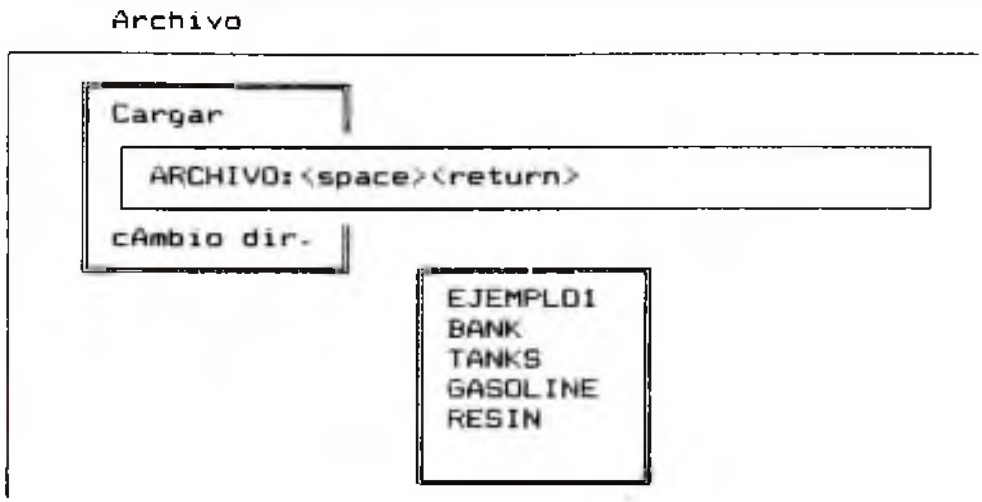


Figura 32. Selección de un archivo de trabajo.

Enseguida seleccione la red con las teclas de flechas y presione <return> y la red se cargará automáticamente al programa.

Simulación de la red.

Una vez guardada la red, se procede a simularla mediante la opción *Ejecuta* del menú principal.

Para llevar a cabo la simulación se establece primeramente el tiempo de arribo, mediante una distribución probabilística, debido a que el problema así lo requiere.

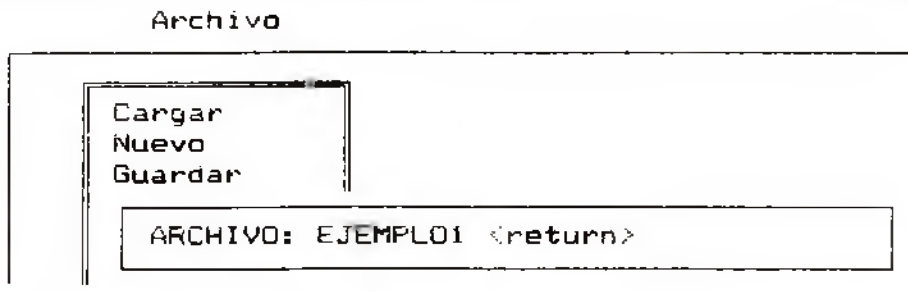


Figura 33.

Enseguida se establece el tiempo de simulación, ya que el problema plantea el periodo de tiempo a simular, se introduce la cantidad equivalente a 12 hrs en minutos que es igual a 720.

Una vez establecidos los parámetros anteriores se procede a seleccionar el tipo de desarrollo, en donde se selecciona a gusto del usuario, el tipo de desarrollo que se desee efectuar, mediante la opción desarrollo.

Hasta este momento el menú de eJecutar se encuentra como lo muestra la figura 34.

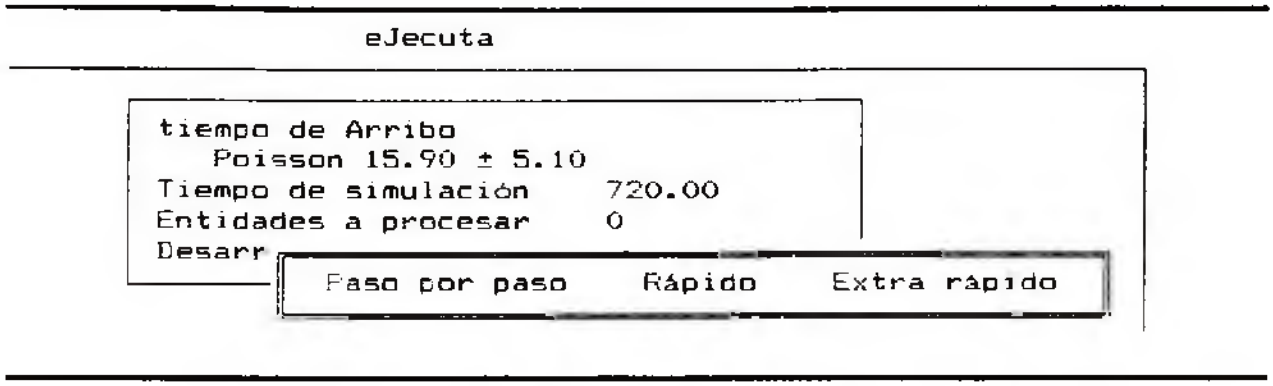


Figura 34.

Reporte de simulación.

Para obtener los resultados de la simulación, se seleciona la opción *Reporte* del menú principal.

Para obtener el Reporte general, unicamente seleccionese la opción correspondiente, y aparecerá en la pantalla, una tabla con parámetros estadísticos reportados para cada actividad. Para observar las demás actividades utilice las teclas de flechas para recorrer las actividades dentro del reporte.

Para el Resumen histórico se especifica el incremento de tiempo con el cual será construido. Se recomienda establecer un intervalo de 5.

Las figuras 36 y 37, muestran la salida de los reportes a la impresora.

Interpretación de resultados.

Este problema es típico para poder observar los problemas de bloqueo, para mostrar los criterios que se utilizan para optimizar el sistema en función de los resultados estadísticos. Y mediante el resumen histórico analizar los intervalos de tiempo en que las actividades se encuentran ociosas, bloqueadas o trabajando.

Primeramente analizaremos la cantidad de entidades que procesan las tres actividades de entrada. La actividad 1 es la que procesa el mayor número de entidades con 4, enseguida le sigue la actividad 2 con 2 entidades y por ultimo la actividad 3 con unicamente una entidad, lo que nos indica que no hay una distribución uniforme en la atención a las entidades entre las actividades, donde se puede proponer las siguientes modificaciones al sistema:

Primero eliminar la actividad 3 que es la que menos entidades procesa, segundo si se desea no perder las entidades que arriben al sistema, se debe de establecer una linea de espera para las actividades 1 y 2. Una vez establecidos estos cambios en las actividades de entrada el paso a seguir es simular el sistema para observar su comportamiento.

Para resolver el problema de bloqueo de las actividades 1, 2 y 3 se tiene que investigar qué actividad es la que ocasiona el bloqueo en las tres actividades anteriores. Del resumen histórico se observa que cuando algunas de las actividades de entrada termina de dar servicio a su entidad, la actividad 4 todavia se encuentra procesando una entidad, provocando el bloqueo. La solución que se puede sugerir a este sistema es instalar otra actividad en paralelo a la actividad 4, para agilizar el flujo de las entidades dentro del sistema.

Finalmente hay que estudiar la optimización de las últimas dos actividades 5 y 6. Del reporte general observamos que el porcentaje de tiempo ocioso de la actividad 6 fue del 100.00 % lo que indica que se encontró durante toda la simulación en estado ocioso; por lo que es obvio la eliminación de esta actividad. Pero hay que tener en cuenta que la actividad 4 tendrá una actividad en paralelo, ocasionando seguramente que la actividad 5 no de abasto a las dos actividades precedentes, por lo que la decisión de eliminar a la actividad 6 queda sujeta al estudio del nuevo sistema. El nuevo sistema que se propone es el siguiente:

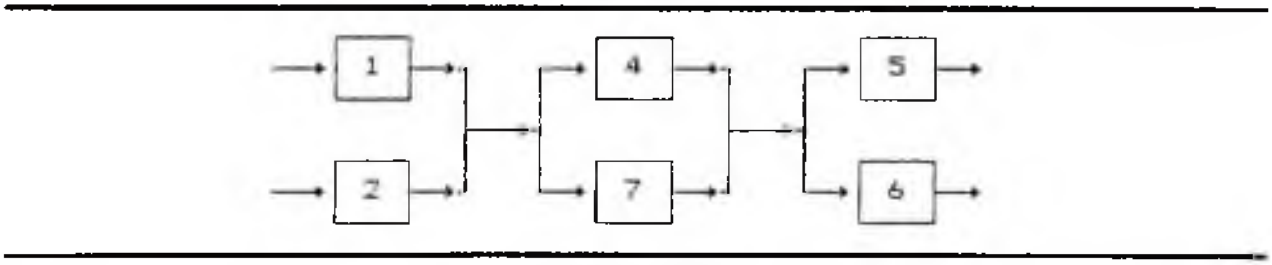


Figura 35.

*** REPORTE GENERAL ***

*** PARAMETROS DEL SISTEMA ***

Num. de entidades terminadas:3
Num. de entidades perdidas: 5
Tiempo de simulación: 120.10
Tiempo de arribo entre entidades: Poisson 15.90 +- 5.10

*** REPORTE DE PARAMETROS POR ACTIVIDAD ***

Número de actividad	1	2	3
Tiempo de servicio	10.54	13.50	11.23
Entidades procesadas	+ 4	2	1
Entidades pendientes	+ 0	0	0
Long. media de cola	0.00	0.00	0.00
Long. máxima de cola	0	0	0
Tiempo medio de espera	0.00	0.00	0.00
Num. de veces bloqueada	+ 2	2	1
% T. en estado de bloqueo	+ 15.90	15.42	0.00
% T. en estado ocioso	43.18	+ 77.54	61.02
Tiempo medio de residencia	11.85	16.41	0.00

*** REPORTE DE PARAMETROS POR ACTIVIDAD ***

Número de actividad	4	5	6
Tiempo de servicio	26.87	18.24	17.67
Entidades procesadas	3	3	0
Entidades pendientes	1	0	0
Long. media de cola	0.00	0.00	0.00
Long. máxima de cola	0	0	0
Tiempo medio de espera	0.00	0.00	0.00
Num. de veces bloqueada	0	0	0
% T. en estado de bloqueo	0.00	0.00	0.00
% T. en estado ocioso	26.10	34.77	100.00
Tiempo medio de residencia	16.61	+ 18.81	0.00

Figura 36.(continuación)

*** RESUMEN HISTORICO ***

#.- Bloqueo [a,z].- Ocupada

ACTIVIDADES

TIEMPO	1	2	3	4	5	6
0.00	a					
	a					
	a					
	a					
9.25				a		
10.02					a	
					a	
					a	
17.95	b				a	
	b				a	
	b				a	
	b				a	
	b				a	
30.26	b					
32.12				b		
				b		
35.52	c			b		
	c			b		
40.42	#			b		
	#			b		
	#			b		
48.37	#	d		b		
	#	d		b		
	#	d		b		
	#	d		b		
59.51		d		c	b	
61.08	e	d		c	b	
62.68	e	#		c	b	
	e	#		c	b	
	e	#		c	b	
	e	#		c	b	
73.29	e	#	f	c	b	
	e	#	f	c	b	
77.50	#	#	f	c	b	
78.70	#	#	f	c		
81.19	#		f	d	c	
82.28	#		#	d	c	
	#		#	d	c	
	#		#	d	c	
88.92	#	g	#	d	c	
	#	g	#	d	c	
	#	g	#	d	c	
	#	g	#	d	c	

TIEMPO	1	2	3	4	5	6
98.19	#	g	#	d		
	#	g	#	d		
	#	g	#	d		
	#	g	#	d		
108.35	#	#	#	d		

Figura 37.

Ejemplo No. 2. **SIMULACION DE UNA TIENDA DE ABARROTES.**

Este sistema, representa el modelo de un servidor con línea de espera. La simulación se desarrollará hasta que 1000 clientes hayan sido atendidos. El tiempo de arribo entre los clientes está descrito por una distribución exponencial con una media de 4.5 minutos y el tiempo de servicio es aproximado a una distribución normal de 3.2 minutos y desviación estadard de 0.6 minutos. Cuando el cajero está ocupado, los clientes se forman de acuerdo al orden de llegada.

El objetivo de este ejemplo, es el de establecer una comparación de resultados con diferentes lenguajes de simulación. El mismo modelo se simuló en los siguientes lenguajes: FORTRAN, GASP, SIMSCRIPT, GPSS y SLAM. [2]

Se presentarán únicamente los resultados finales de la simulación, omitiendo la programación del modelo en cada lenguaje. [2]

Primeramente se presentará la tabla de resultados de las simulaciones realizadas en cada uno de los lenguajes previamente mencionados.

	LENGUAJES			
	FORTRAN	*SIMSCRIPT*	*GPSS*	*SLAM*
% DE UTILIZACION DEL SERVIDOR	0.60	0.67	0.688	0.7694
MAXIMA LONGITUD DE COLA	5	8	11	8
TIEMPO MEDIO DE SERVICIO	4.59	6.51	NR	NR
TIEMPO DE SIMULACION	4460.68	4794.13	4626.28	4164.00

Figura 38.

La tabla de resultados que reporta SIMDISC son:

*** REPORTE GENERAL ***

*** PARAMETROS DEL SISTEMA ***

Num. de entidades terminadas: 1000
Num. de entidades perdidas: 0
Tiempo de simulación: 4702.14
Tiempo de arribo entre entidades: Exponencial 4.50 +- 0.00

*** REPORTE DE PARAMETROS POR ACTIVIDAD ***

Número de actividad		1
Tiempo de servicio		3.20
Entidades procesadas	+	1000
Entidades pendientes		0
Long. media de cola	+	0.53
Long. máxima de cola	+	7
Tiempo medio de espera	+	5.77
Num. de veces bloqueada		0
% T. en estado de bloqueo		0.00
% T. en estado ocioso	+	35.48
Tiempo medio de residencia	+	2.95

Figura 39.

Para poder conocer el % de utilización, unicamente se resta 100 - % T. en estado ocioso = 100 - 35.48 = 64.52.

Análisis de Resultados.

Primeramente se analizará el parámetro de % de utilización del servidor, el cual es similar, en los 4 lenguajes como en SIMDISC; éste parámetro varía dentro del rango 0.6 a 0.77, indicando que casi las 3/4 partes del tiempo, el servidor permaneció ocupado.

La máxima longitud de cola, también varía dentro de un rango de 5 a 11 clientes. Este parámetro varía bastante de un lenguaje a otro, ya que depende del algoritmo utilizado para la generación de números aleatorios, debido a que cada lenguaje inicializa su generador con una diferente semilla [5], la cual es de bastante importancia para la obtención de una buena aleatoriedad en los números.

El tiempo medio de servicio en el cual fueron atendidos los 1000 clientes varía de 2.95 a 6.51. El tiempo de servicio que reporta SIMDISC, es el que más se acerca al tiempo de servicio teórico, ya que el promedio de clientes que esperarón en la línea de espera fué de 0.53, indicando que la gran mayoría de los clientes no tuvieron que esperar.

Finalmente el tiempo de simulación se encuentra dentro del rango de 4160 a 4795 minutos, donde la amplitud del rango es de 635 minutos aproximadamente 10 horas de diferencia, (el rango se considera bastante amplio) éste parámetro al igual que la máxima longitud de cola depende del generador de números aleatorios.

SIMDISC utiliza un generador de números aleatorios similar al de los lenguajes SIMSCRIPT y GASP, observándose que entre estos tres lenguajes existe una gran similitud entre los resultados. (Vease Figuras 38 y 39).

Ejemplo No. 3. **SIMULACION DE TANQUES DE ABASTECIMIENTO.**

El sistema a simular consiste de un grupo de 15 tanques. Los tanques son cargados con petróleo crudo en Valdez, Alaska y son transladados a Seattle, Washington donde son descargados. Cada tanque tiene una capacidad de 150 tb (miles de barriles de crudo). Existen suficientes muelles en Alaska para que todos los tanques puedan ser cargados simultaneamente. El tiempo para cargar el tanque es exponencialmente distribuido con una media igual a 3 días. Todos los tanques son descargados en el mismo muelle en Seattle con un tiempo de descarga de 0.75 días. El tiempo de viaje de Alaska a Washington cuando el tanque es cargado es normalmente distribuido con una media de cinco días y una desviación standar de 1.5 días. Normalmente a un tanque nunca le toma menos de 3.5 días o mas de 8 días en el viaje de Alaska a Seattle. El tiempo de viaje del tanque descargado de Seattle a Alaska es normalmente distribuido con una media de cuatro días, y una desviación standar de un día con un mínimo de 3 días y un máximo de siete días. Se requiere simular el abastecimiento de tanques para 365 días para obtener una estimación de las siguientes cantidades. Utilización del muelle, Tiempo de viaje redondo del tanque, Tiempo de espera del tanque, y el número de tanques esparando para ser descargados.

Las condiciones iniciales para la simulación especifican que el número de tanques llegan a los muelles de abastecimiento en un intervalo de 0.5 días, iniciando con el primero en el tiempo cero, los muelles de descarga están ociosos, y no hay tanques esperando a ser descargados.

La construcción de la red se muestra en la figura 30. El reporte general de la simulación se muestra es la figura 31. Para el resumen histórico unicamente se despliega una porción de éste debido a su amplitud, en la figura 32.

Descripción del problema.

La analizar un problema se debe de identificar los siguientes datos: las entidades, a las actividades y sus tiempos de servicio y por último el tiempo de arribo de las entidades al sistema. Para el problema, las entidades son los 15 tanques, y las actividades son las siguientes:

actividad	*Distribucion*
- Cargado de tanques en Muelle de Alaska	Exponencial 3 días
- Viaje de Alaska a Washington	Normal 5 ± 1.5 días
- Descarga de tanques en Seattle	Constante 0.75 días
- Viaje de Washington a Alaska	Normal 4 ± 1 días.

El tiempo de arribo es constante de 0.5 días.

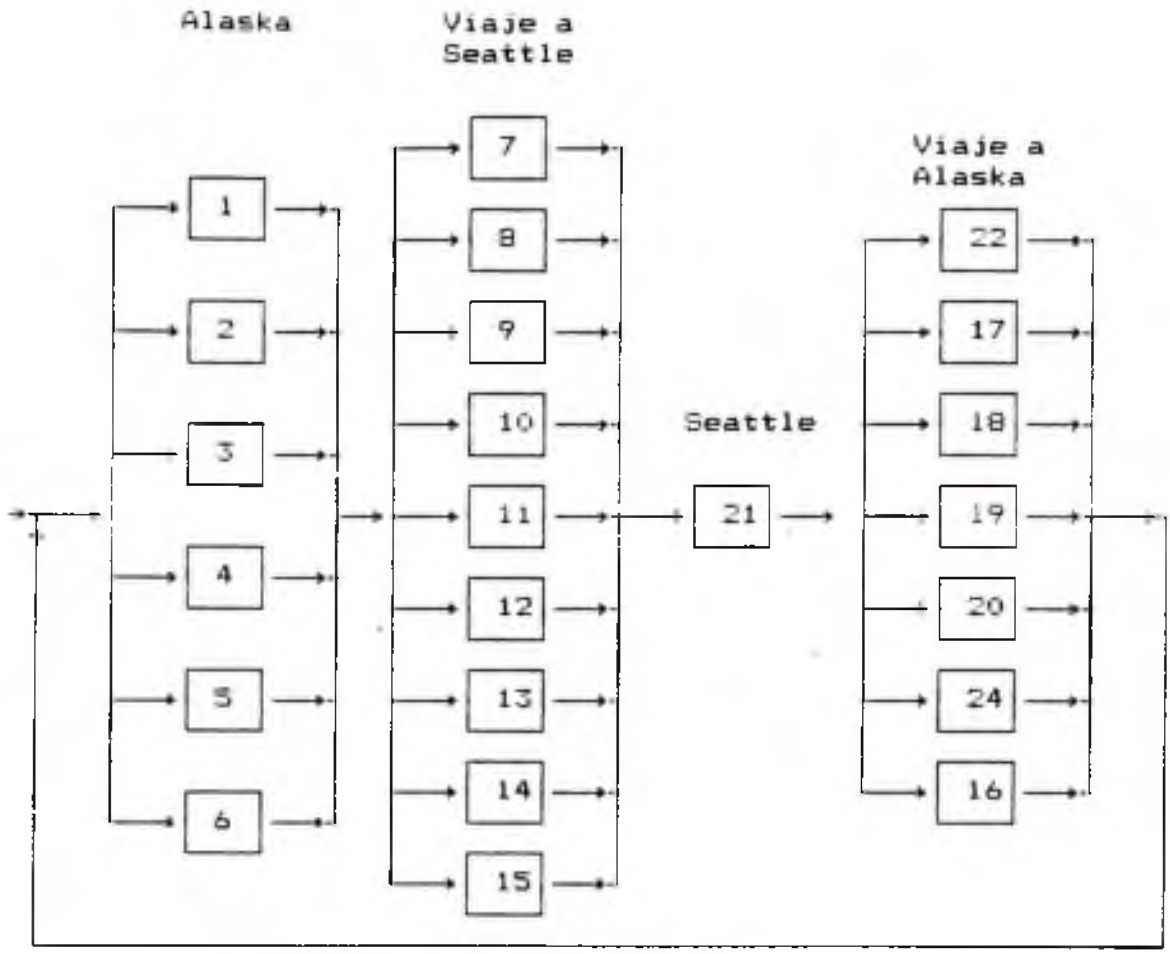

Alaska
Viaje a Seattle
Seattle
Viaje a Alaska
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
21
22
17
18
19
20
24
16

Figura 40 .

Descripción del problema

La simulación se efectuará por un periodo de 365 días, pero esta se define por la cantidad de entidades a procesar, ya que el sistema requiere unicamente de 12 entidades, en cambio si se especificase por el tiempo de simulación arribaran más de 12 entidades al sistema, por lo que la simulación no se efectuará de acuerdo a las necesidades del problema.

Cabe mencionar que para este ejemplo, la simulación se terminará por medio de la tecla <F10> (interrumpe simulación), cuando el tiempo de simulación llegue a 365, ya que el tipo de red es ciclica, por lo que las entidades circularán dentro de la red sin poder salir de ella.

Análisis de resultados.

De los resultados que obtenemos de este ejemplo, nos muestran la siguiente información, el tiempo que permanece ocupado el muelle de Seattle, al que le corresponde la actividad no. 21, la cual nos reporta los siguientes parámetros estadisticos: La longitud media de su línea de espera 0.35 tanques, la longitud máxima de la línea de espera durante la simulación 5 tanques, El tiempo de espera de los tanques en la cola 1.49 días. La cantidad de tiempo que permanecio la actividad 21 en estado ocioso:

$$T_{ocioso} = 366.62 * 0.2989 = 15.645 \text{ días}$$

El último parámetro nos dice el tiempo que tarda un tanque en ser atendido por el muelle, desde que llega a la línea de espera de éste, hasta que termina de ser atendido por la actividad, y comparándolo con el tiempo de servicio teórico de la actividad, se pueden concluir dos cosas; la primera es que las entidades o tanques permanecen 5 veces más del tiempo que deben de tardar en la actividad 21, y la segunda se recomienda establecer una actividad en paralelo a la actividad 21 con el fin de agilizar el flujo de los tanques a través de este muelle y puedan realizarse mas viajes de abastecimiento de los tanques.

El problema desea conocer el tiempo que le lleva a un tanque realizar un viaje redondo, este tipo de información no la proporciona el simulador explícitamente, pero se puede obtener analizando el resumen histórico de la simulación, de donde podemos observar que el tanque [a] llega al muelle en Alaska al tiempo 0.0 y regresa al mismo lugar hasta el tiempo 12.16, lo cual nos indica que para la primera vuelta del tanque [a] requirió de 12.16 días, para la siguiente vuelta el tanque regresa al tiempo 26.45 y el tiempo que tardó en dar la segunda vuelta fué de 26.45 - 12.16 = 14.29 días. La siguiente tabla muestra los tiempos que tardaron los tanques a, b, c, d, e, en sus primeras vueltas:

Tabla de duración de viajes para diferentes tanques. Tiempo [días].

Tanque	a	b	c	d	e
vuelta 1	12.16	13.83	14.38	10.71	11.94
vuelta 2	14.29	10.65	15.29	11.89	14.90

De la tabla anterior podemos establecer el tiempo aproximado que tarda un tanque en dar la vuelta en el sistema, efectuando un promedio de los tiempos reportados en la tabla obtenemos que un tanque tarda aproximadamente 12.45 días en realizar la vuelta.

*** REPORTE GENERAL ***

*** PARAMETROS DEL SISTEMA ***

Num. de entidades terminadas: 0
Num. de entidades perdidas: 0
Tiempo de simulación: 366.62
Tiempo de arribo entre entidades: Constante 0.50 +- 0.00

*** REPORTE DE PARAMETROS POR ACTIVIDAD ***

Número de actividad	1	2	3
Tiempo de servicio	3.00	3.00	3.00
Entidades procesadas	+ 97	87	73
Entidades pendientes	+ 1	1	0
Long. media de cola	0.00	0.00	0.00
Long. máxima de cola	0	0	0
Tiempo medio de espera	0.00	0.00	0.00
Num. de veces bloqueada	0	0	0
% T. en estado de bloqueo	0.00	0.00	0.00
% T. en estado ocioso	20.35	28.11	39.31
Tiempo medio de residencia	3.00	3.00	3.00

Figura 41.

*** REPORTE DE PARAMETROS POR ACTIVIDAD ***

Número de actividad	4	5	6
Tiempo de servicio	3.00	3.00	3.00
Entidades procesadas	57	25	9
Entidades pendientes	0	0	0
Long. media de cola	0.00	0.00	0.00
Long. máxima de cola	0	0	0
Tiempo medio de espera	0.00	0.00	0.00
Num. de veces bloqueada	+ 1	1	1
% T. en estado de bloqueo	0.08	+ 0.22	0.20
% T. en estado ocioso	53.10	76.23	62.65
Tiempo medio de residencia	3.01	3.03	3.08

*** REPORTE DE PARAMETROS POR ACTIVIDAD ***

Número de actividad	7	8	9
Tiempo de servicio	5.00	5.00	5.00
Entidades procesadas	59	55	60
Entidades pendientes	1	1	1
Long. media de cola	0.00	0.00	0.00
Long. máxima de cola	0	0	0
Tiempo medio de espera	0.00	0.00	0.00
Num. de veces bloqueada	0	0	0
% T. en estado de bloqueo	0.00	0.00	0.00
% T. en estado ocioso	18.73	28.44	21.72
Tiempo medio de residencia	4.97	4.71	4.72

Figura 41. continuación

*** REPORTE DE PARAMETROS POR ACTIVIDAD ***

Número de actividad	10	11	12
Tiempo de servicio	5.00	5.00	5.00
Entidades procesadas	54	51	41
Entidades pendientes	1	1	0
Long. media de cola	0.00	0.00	0.00
Long. máxima de cola	0	0	0
Tiempo medio de espera	0.00	0.00	0.00
Num. de veces bloqueada	0	0	0
% T. en estado de bloqueo	0.00	0.00	0.00
% T. en estado ocioso	32.20	31.05	45.02
Tiempo medio de residencia	4.58	4.93	4.88

*** REPORTE DE PARAMETROS POR ACTIVIDAD ***

Número de actividad	21	22	17
Tiempo de servicio	0.75	4.00	4.00
Entidades procesadas	86	81	79
Entidades pendientes	1	1	1
Long. media de cola	0.35	0.00	0.00
Long. máxima de cola	+ 5	0	0
Tiempo medio de espera	+ 1.49	0.00	0.00
Num. de veces bloqueada	0	0	0
% T. en estado de bloqueo	0.00	0.00	0.00
% T. en estado ocioso	29.89	17.90	20.61
Tiempo medio de residencia	4.48	3.70	3.62

Figura 41. continuación

*** REPORTE DE PARAMETROS POR ACTIVIDAD ***

Número de actividad	18	19	20
Tiempo de servicio	4.00	4.00	4.00
Entidades procesadas	67	62	39
Entidades pendientes	1	1	0
Long. media de cola	0.00	0.00	0.00
Long. máxima de cola	0	0	0
Tiempo medio de espera	0.00	0.00	0.00
Num. de veces bloqueada	0	0	0
% T. en estado de bloqueo	0.00	0.00	0.00
% T. en estado ocioso	30.35	36.92	58.77
Tiempo medio de residencia	3.80	3.67	3.76

*** REPORTE DE PARAMETROS POR ACTIVIDAD ***

Número de actividad	24	13	14
Tiempo de servicio	4.00	5.00	5.00
Entidades procesadas	10	15	6
Entidades pendientes	0	0	0
Long. media de cola	0.00	0.00	0.00
Long. máxima de cola	0	0	0
Tiempo medio de espera	0.00	0.00	0.00
Num. de veces bloqueada	0	0	0
% T. en estado de bloqueo	0.00	0.00	0.00
% T. en estado ocioso	+ 82.90	68.36	68.34
Tiempo medio de residencia	3.79	+ 5.46	4.67

Figura 41. continuación

*** REPORTE DE PARAMETROS POR ACTIVIDAD ***

Número de actividad	15	16
Tiempo de servicio	5.00	4.00
Entidades procesadas	2	0
Entidades pendientes	0	0
Long. media de cola	0.00	0.00
Long. máxima de cola	0	0
Tiempo medio de espera	0.00	0.00
Num. de veces bloqueada	0	0
% T. en estado de bloqueo	0.00	0.00
% T. en estado ocioso	4.86	0.00
Tiempo medio de residencia	4.37	0.00

Figura 41. continuación

*** RESUMEN HISTORICO ***

#.- Bloqueo [a,z].- Ocupada

ACTIVIDADES

TIEMPO	1	2	3	4	5	6	7	8	9	10	11	12	21	27	28	19	20	24	13	14	15	16
0.00	a																					
0.50	a	b																				
1.00	a	b	c																			
1.50	a	b	c	d																		
2.00	a	b	c	d	e																	
2.50	a	b	c	d	e	f																
3.00	g	b	c	d	e	f	a															
3.50	g	h	c	d	e	f	a		b													
4.00	g	h	i	d	e	f	a		b		c											
4.50	g	h	i	j	e	f	a	d	b		c											
5.00	g	h	i	j	k	f	a	d	b	e	c											
5.50	g	h	i	j	k	l	a	d	b	e	c	f										
6.00		h	i	j	k	l	a	d	b	e	c	f							g			
6.50			i	j	k	l	a	d	b	e	c	f							g	h		
7.00				j	k	l	a	d	b	e	c	f							g	h	i	
7.50				#	k	l	a	d	b	e	c	f							g	h	i	
7.81					k	l	a	j	b	e	c	f	d						g	h	i	
8.00					#	l	a	j	b	e	c	f	d						g	h	i	
8.50					#	#	a	j	b	e	c	f	d						g	h	i	
8.56					#	#	a	j	b	e	c	f		d					g	h	i	
8.79						#	k	j	b	e	c	f	a	d					g	h	i	
9.22							k	j	b	l	c	f	a	d					g	h	i	
9.53							k	j		l	c	f	a	d					g	h	i	
9.54							k	j		l	c	f	e	d	a				g	h	i	
9.76							k	j		l	c		e	d	a				g	h	i	
10.29							k	j		l	c		b	d	a	e			g	h	i	
10.41							k	j		l	c		b	d	a	e			g	h		
10.61							k	j		l			b	d	a	e			g	h		
11.03							k	j		l			b	d	a	e			g			
11.04							k	j		l			f	d	a	e	b		g			
11.79							k	j		l			i	d	a	e	b	f	g			
12.07							k			l			i	d	a	e	b	f	g			
12.16	a						k			l			i	d		e	b	f	g			
12.21	a	d					k			l			i			e	b	f	g			
12.24	a	d								l			i			e	b	f	g			
12.54	a	d								l			c	i		e	b	f	g			
12.84	a	d											c	i		e	b	f	g			
13.02	a	d											c	i		e	b	f				
13.29	a	d											h	i	c	e	b	f				
13.94	a	d	e										h	i	c		b	f				
14.04	a	d	e										j	i	c	h	b	f				
14.33	a	d	e	b									j	i	c	h		f				
14.79	a	d	e	b									k	i	c	h	j	f				
15.16		d	e	b			a						k	i	c	h	j	f				

Figura 42.

*** RESUMEN HISTORICO ***

#.- Bloqueo [a,z].- Ocupada

ACTIVIDADES

TIEMPO	1	2	3	4	5	6	7	8	9	10	11	12	21	22	27	28	19	20	24	13	14	15	16
15.21			e	b			a		d				k	i	c	h	j	f					
15.38	c		e	b			a		d				k	i		h	j	f					
15.54	c		e	b			a		d				l	i	k	h	j	f					
15.60	c	f	e	b			a		d				l	i	k	h	j						
16.29	c	f	e	b			a		d				g	i	k	h	j	1					
16.44	c	f	e	b	h		a		d				g	i	k		j	1					
16.88	c	f	e	b	h	j	a		d				g	i	k			1					
16.94	c	f		b	h	j	a		d		e		g	i	k			1					
17.04	c	f		b	h	j	a		d		e			i	k	g		1					
17.33	c	f			h	j	a	b	d		e			i	k	g		1					
17.61	c	f	i		h	j	a	b	d		e				k	g		1					
18.22	c	f	i	1	h	j	a	b	d		e				k	g							
18.38		f	i	1	h	j	a	b	d	c	e				k	g							
18.60			i	1	h	j	a	b	d	c	e	f			k	g							
18.99	k		i	1	h	j	a	b	d	c	e	f				g							
19.44	k		i	1		j	a	b	d	c	e	f				g				h			
19.70	k		i	1		j	a	b		c	e	f	d			g				h			
19.75	k	g	i	1		j	a	b		c	e	f	d							h			
19.88	k	g	i	1			a	b	j	c	e	f	d							h			
20.45	k	g	i	1			a	b	j	c	e	f		d						h			
20.61	k	g		1			a	b	j	c	e	f		d						h	i		
21.22	k	g					a	b	j	c	e	f		d						h	i	l	
21.53	k	g					a		j	c	e	f	b	d						h	i	l	
21.54	k	g							j	c	e	f	b	d						h	i	l	
21.54	k	g							j	c		f	b	d						h	i	l	
21.71	k	g							j	c			b	d						h	i	l	
21.73	k	g							j				b	d						h	i	l	
21.99		g					k		j				b	d						h	i	l	
22.28		g					k		j				a	d	b					h	i	l	
22.75							k		j	g			a	d	b					h	i	l	
23.03							k		j	g			e	d	b	a				h	i	l	
23.78							k		j	g			f	d	b	a	e			h	i	l	
24.10	d						k		j	g			f		b	a	e			h	i	l	
24.53	d						k		j	g			c	f	b	a	e			h	i	l	
24.98	d	b					k		j	g			c	f		a	e			h	i	l	
25.28	d	b					k		j	g				f	c	a	e			h	i	l	
25.28	d	b					k		j	g			i	f	c	a	e			h		l	
25.30	d	b					k		j	g			i	f	c	a	e					l	
26.03	d	b					k		j	g			h	f	c	a	e	i				l	
26.45	d	b	a				k		j	g			h	f	c		e	i				l	
26.54	d	b	a				k		j	g			h	f	c		e	i					
26.66	d	b	a				k			g			h	f	c		e	i					
26.78	d	b	a				k			g			l	f	c	h	e	i					
27.10		b	a				k	d		g			l	f	c	h	e	i					

Figura 42. continuación

*** RESUMEN HISTORICO ***

#.- Bloqueo [a,z].- Ocupada

ACTIVIDADES

TIEMPO	1	2	3	4	5	6	7	8	9	10	11	12	21	22	17	18	19	20	24	13	14	15	16
27.53		b	a				k	d		g		.	j	f	c	h	e	i	l				
27.62		b	a					d		g			j	f	c	h	e	i	l				
27.68	f	b	a					d		g			j		c	h	e	i	l				
27.98	f		a					d	b	g			j		c	h	e	i	l				
28.28	f		a					d	b	g			k	j	c	h	e	i	l				
28.74	f	j	a					d	b	g			k	j	c	h	e		l				
28.78	f	j	a					d	b				k	j	c	h	e		l				
28.84	f	j	a	e				d	b				k	j	c	h			l				
29.03	f	j	a	e				d	b				g	j	c	h	k		l				
29.45	f	j		e				d	b		a		g	j	c	h	k		l				
29.78	f	j		e				d	b		a			j	c	h	k	g	l				
29.96	f	j		e					b		a		d	j	c	h	k	g	l				
30.29	f	j	h	e					b		a		d	j	c		k	g	l				
30.67	f	j	h	e	c				b		a		d	j			k	g	j				
30.68		j	h	e	c		f		b		a		d	j			k	g	j				
30.71		j	h	e	c		f		b		a			j	d		k	g	l				
31.37	1	j	h	e	c		f		b		a			j	d		k	g					
31.74	1		h	e	c		f		b	j	a			j	d		k	g					
31.79	1	k	h	e	c		f		b	j	a			j	d			g					
31.84	1	k	h		c		f	e	b	j	a			j	d			g					
32.67	1	k	h	j	c		f	e	b	j	a				d			g					
33.29	1	k		j	c		f	e	b	j	a	h			d			g					
33.42	1	k	d	j	c		f	e	b	j	a	h						g					
33.64	1	k	d	j	c		f	e		j	a	h	b					g					
33.67	1	k	d	j			f	e	c	j	a	h	b					g					
34.37		k	d	j			f	e	c	j	a	h	b					g		1			
34.39		k	d	j			f	e	c	j	a	h		b				g		1			
34.79			d	j			f	e	c	j	a	h		b				g		j	k		
34.87			d	j			f	e	c	j		h	a	b				g		j	k		
34.98	g		d	j			f	e	c	j		h	a	b						j	k		
35.59	g		d	j			f	e	c			h	a	b						j	k		
35.62	g		d	j			f	e	c			h	j	b	a					j	k		
35.67	g		d				f	e	c	j		h	j	b	a					j	k		
35.69	g		d					e	c	j		h	j	b	a					j	k		
36.37	g		d					e	c	j		h	f	b	a	j				j	k		
36.42	g							e	c	j	d	h	f	b	a	j				j	k		
37.01	g							e		j	d	h	f	b	a	j				j	k		
37.09	g									j	d	h	f	b	a	j				j	k		
37.12	g									j	d	h	c	b	a	j	f			j	k		
37.27	g	a								j	d	h	c	b		j	f			j	k		
37.87	g	a	b							j	d	h	c			j	f			j	k		
37.87	g	a	b							j	d	h	e	c		j	f			j	k		
37.99		a	b				g			j	d	h	e	c		j	f			j	k		
38.62		a	b				g			j	d	h		c	e	j	f			j	k		

Figura 42. continuación

*** RESUMEN HISTORICO ***

#.- Bloqueo [a,z].- Ocupada

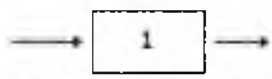
ACTIVIDADES

TIEMPO	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
39.32		a	b				g			j	d		h	c	e	i	f							k	
39.52		a	b				g			j	d		h	c	e	i	f							k	
40.07		a	b				g			j	d		l	c	e	i	f	h						k	
40.15		a	b				g				d		l	c	e	i	f	h						k	
40.27			b				g		a		d		l	c	e	i	f	h						k	
40.32	f		b				g		a		d		l	c	e	i		h						k	
40.35	f		b				g		a		d		l	c	e	i		h							
40.63	f	i	b				g		a		d		l	c	e			h							
40.75	f	i	b				g		a				l	c	e			h							
40.77	f	i	b	c			g		a				l		e			h							
40.82	f	i	b	c			g		a				j	l	e			h							
40.87	f	i		c			g		a		b		j	l	e			h							
41.57	f	i		c			g		a		b		k	l	e	j		h							
42.32	f	i		c			g		a		b		d	l	e	j	k	h							
42.62	f	i		c					a		b		d	l	e	j	k	h							
43.07	f	i		c					a		b		g	l	e	j	k	h	d						
43.08	f	i	e	c					a		b		g	l		j	k	h	d						
43.32		i	e	c			f		a		b		g	l		j	k	h	d						
43.63			e	c			f		a	i	b		g	l		j	k	h	d						
43.72	h		e	c			f		a	i	b		g	l		j	k		d						
43.77	h		e				f	c	a	i	b		g	l		j	k		d						
43.82	h		e				f	c	a	i	b			l	g	j	k		d						
44.53	h		e				f	c		i	b		a	l	g	j	k		d						
45.14	h	l	e				f	c		i	b		a		g	j	k		d						

Figura 42. continuación

Ejemplo No. 4. **SIMULACION DE UN BANCO.**

El banco tiene dos ventanillas, cada una atendida por un cobrador y con su propia línea de espera. Las ventanillas son adyacentes. De observaciones previas se ha determinado que el tiempo de arribo de los clientes durante las horas críticas es exponencialmente distribuida con una media entre arribos de 0.5 unidades de tiempo. Las congestiones ocurren unicamente durante las horas críticas, y unicamente este periodo será analizado. El tiempo de servicio es normalmente distribuido para cada cobrador con una media de una unidad de tiempo y una desviación standar de 0.3 unidades de tiempo. Se ha observado que el cliente tiene preferencia por la ventanilla uno, si los dos cobradores se encuentran ocupados o si las líneas de espera son iguales. Para todos los demás tiempos, a veces el cliente escoge la línea de espera mas corta. Después que el cliente ha entrado al sistema, no puede salir hasta ser servido. No puede cambiar de líneas de espera. Debido a limitaciones de espacio solamente tres clientes pueden esperar en la línea de espera. Estos clientes más el cliente que se encuentra en servico por el cobrador, permite un máximo de ocho personas para el sistema. Si el sistema esta lleno cuando el cliente llega, desiste y deja el sistema. Las condiciones iniciales es que el primer arribo es al tiempo cero, los cobradores se encuentran ociosos, y no existen clientes en la línea de espera.



ventanilla No. 1.

ventanilla No. 2.

Figura 43.

Descripción del problema.

Las actividades en el problema son las dos ventanillas que tienen un tiempo de servicio normalmente distribuido con una media de 1 y varianza de 0.3 unidades de tiempo. Las actividades tendrán una cola máxima de 3 clientes. El tiempo de arribo de los clientes es exponencial de 0.5 unidades de tiempo.

Los resultados de la simulación se observan en la figuras 44 y 45, para el reporte general y resumen histórico respectivamente.

Interpretación de resultados.

Del reporte general podemos observar la diferencia de trabajo entre los dos cobradores, aunque en el parámetro de entidades procesadas indican que atendieron a la misma cantidad de clientes, el porcentaje de tiempo ocioso es mayor para el cobrador en la vantanilla 2, esto es debido a las condiciones del problema que especifican que al arribo de un cliente se tiene preferencia por la ventanilla 1 en caso de que ambas ventanillas se encuentren con clientes atendiendose. La longitud máxima de la línea de espera es de un cliente, y el tiempo de espera del cliente en la línea de espera es menor a la unidad de tiempo para la ventanilla 1 es de 0.22 unidades de tiempo y para la ventanilla 2 es de 0.16 unidades de tiempo, por lo que el sistema no presenta problemas de espera bastante grandes para los clientes. Por último el tiempo promedio que tarda un cliente en el sistema es de 1.20 unidades cuando este selecciona la ventanilla 1, y 0.91 unidades de tiempo para cuando el cliente escoge la ventanilla 2.

Se puede observar de acuerdo a los parámetros anteriores que los clientes que seleccionan la ventanilla 2 permanecen menos tiempo en el sistema, debido a que cuando arriban a este la ventanilla 1 se encuentra atendiendo a un cliente y la ventanilla 2 se encuentra desocupada por lo que pasa a ser atendido inmediatamente por esta. En el resumen historico podemos identificar el tiempo de arribo de algunos de los clientes, también podemos deducir el tiempo que se requirió para atender a cada cliente por ejemplo: los arribos se efectuaron como sigue: [a] al tiempo 0.0, [b] al tiempo 0.46, [f] al tiempo 3.12, [g] al tiempo 3.22, etc. Para los clientes [c], [d], [e], [h], no podemos establecer su tiempo de arribo ya que tal vez tuvieron que formar parte de la línea de espera de la ventanilla.

Para el cálculo de la duración de los tiempos de servicio para cada cliente unicamente se tiene que efectuar la diferencia del tiempo que el cliente termina de ser atendido y el tiempo en el cual comienza a ser atendido por la actividad. Por ejemplo para el cliente [f], este termina de ser atendido al tiempo 4.49 y a esta cantidad se le resta el tiempo al que comienza a ser atendido 3.12 por la actividad, dando como resultado el tiempo de servicio para el cliente [f] igual a 1.37 unidades de tiempo. De forma similar se calculan los tiempos para los demás clientes.

*** REPORTE GENERAL ***

*** PARAMETROS DEL SISTEMA ***

Num. de entidades terminadas: 14
Num. de entidades perdidas: 0
Tiempo de simulación: 8.10
Tiempo de arribo entre entidades: Exponencial 0.50 +- 0.00

*** REPORTE DE PARAMETROS POR ACTIVIDAD ***

Número de actividad		1		2
Tiempo de servicio		1.00		1.00
Entidades procesadas	+	7		7
Entidades pendientes	+	1		0
Long. media de cola		0.22		0.14
Long. máxima de cola	+	1		1
Tiempo medio de espera	+	0.22		0.16
Num. de veces bloqueada		0		0
% T. en estado de bloqueo		0.00		0.00
% T. en estado ocioso		17.71	+	30.75
Tiempo medio de residencia	+	1.20		0.91

Figura 44.

*** RESUMEN HISTORICO ***

#.- Bloqueo [a,z].- Ocupada

TIEMPO	ACTIVIDADES 1	2
0.00	a	
0.46	a	b
	a	b
1.35	c	b
	c	b
2.22	c	d
	c	d
2.73	e	d
2.87	e	
3.12	e	f
3.18		f
3.22	g	f
	g	f
4.06	h	f
4.49	h	
4.84	h	i
5.07	h	
5.13		
5.58	j	
	j	
6.23	j	k
6.57	j	
6.69	j	l
6.83		l
7.00		
7.06	m	
7.15	m	n
7.31		n
7.71		
8.01	o	
	o	

Figura 45.

Ejemplo No. 5. **SIMULACION DEL PROCESO DE OBTENCION DE LA RESINA ALKYD.**

La preparación de la resina alkyd es esencialmente un proceso de esterificación, las reacciones son llevadas a elevadas temperaturas en un reactor. La preparación del batch y de la mezcla requieren de 4 horas. Despues una corriente de gas inerte es introducida a través de una perforación debajo del nivel de la mezcla liquida. La principal razón de rozear el gas inerte (normalmente dióxidos de nitrógeno o carbono) es la prevención de reacciones secundarias por contaminación de oxígeno. El gas contenido en el reactor evacuado mediante una bomba de vacio.

Después de que la atmósfera inerte ha sido obtenida el anhidrido maleico es adicionado para la preparación de la esterificación. Después el pentaerytritol es agregado en tres pasos mientras se determina el avance de la reaccion probando el pH de la mezcla. Cuando la reacción de esterificación se termina la temperatura es elevada a 270 C y mantenida por un periodo de tiempo y se realiza un vacio dentro del reactor. Para garantizar la calidad del producto. muestras son llevadas a laboratorio hasta que las especificaciones finales son obtenidas. Finalmente la mezcla es enfriada y el producto es descargado. El diagrama de la secuecia de las actividades se muestra en la figura 46.

Los objetivos del estudio son:

1. Identificar aquellas actividades que retrasan el desarrollo de proceso.

2. Probar los efectos de la variación del tiempo de operación para las actividades que retrazan el proceso.

3. Determinar los parámetros del diseño del proceso que reducen el tiempo de duración de las actividades, para optimizar la productividad.

Mediante la operación de 24 hrs. de lunes a sabado, alrededor de 7 a 9 batches de resina son producicos semanalmente.

La siguiente figura 47, lista la ouración de las actividades del proceso que están descritas por una distribución probabilística. La siguientes tablas muestran las distribuciones de frecuencias para las otras actividades faltantes de especificar su tiempo de servicio.

Descripción del problema.

En este ejemplo los arribos se llevan en forma condicional, este tipo de proceso químico tiene la caracteristica de procesar un batch a la vez, razón por la cual es arribo de la entidades o batches se encuentra condicionado a la terminación del servicio de la actividad 14. La simulación se efectuara para el procesamiento de 5 batches.

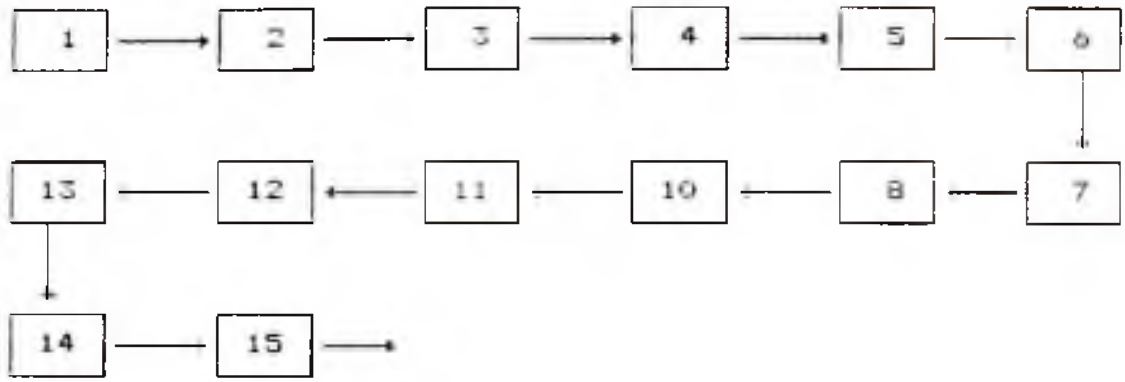


Figura 46.

Tiempos de servicio de las actividades

Actividad	*Identificacion*	*Duración*
Preparación del batch y mezclado	1	240 min.
Roceado del gas inerte	2	T. Frec.
Adición del anhidrido maleico	3	Uniforme 11 ± 7 min.
Obtención de la mezcla homogenea	4	60 min.
Primera parte del Pentaerytritol	5	15 min.
Segunda parte del Pentaerytritol	6	15 min.
Tercera parte del pentaerytritol	7	T. Frec.
Elevación a Temperatura de 270 C	8	Uniforme 268 ± 17 min.
Obtención de la mezcla homogenea	9	60 min.
Obtención de vacio en el reactor	10	T. Frec.
Primera muestra	11	60 min.
Segunda muestra	12	Normal 25± 20.
Enfriamiento del reactor	13	T. Frec.
Descargado del reactor	14	15 min.

Figura 47.

Distribuciones de Frecuencias Acumuladas

Actividad 2.

Tiempo (min)	55	60	65	70	75	80	85	90	125	130	190	230
Frecuencia	2	7	13	62	80	84	89	91	93	96	98	100

Actividad 7.

Tiempo (min)	5	10	15	20	25	30	35	40	45	60
Frecuencia	12	23	55	67	83	90	93	95	98	100

Actividad 10.

Tiempo (min)	30	45	50	60	65	70	75	80	90	105
Frecuencia	4	9	11	84	87	89	91	96	98	100

Actividad 13.

Tiempo (min)	5	10	15	20	25	26	30	35	40	55	60	80
Frecuencia		7	17	37	54	56	78	88	93	95	98	100

Figura 48

Los resultados estadisticos de la simulación se despliegan en la siguiente figura 49.

El histórico de la simulación se muestra en la figura 50.

Interpretación de resultados.

Los resultados obtenidos nos muestran la operación normal del proceso sin considerar cambios en él, con el reporte general podemos observar el tiempo se servicio real de aquellas actividades que su tiempo de servicio se encuentra descrito por una tabla de frecuencias como son las actividades: 2, 7, 10, 13. Tambien obtenemos el tiempo que se requiere para procesar los 5 batches que es de 4986.90 minutos.

El tipo de arribo de las entidades o batches ocasionan que no existan problemas de bloqueo entre las actividades, por lo que unicamente se puede observar el desarrollo de los 5 batches en el proceso, así como el tiempo que requirió cada batch en cada actividad.

La actividad que más duración tiene es la actividad 8. De acuerdo a esto se puede proponer para optimizar el sistema que los arribos sean condicionados al termino de servicio de esta actividad, pudiendo obtener un incremento en la producción de batches y evitando conflictos de bloqueos entre actividades, es decir por ejemplo; que la actividad 7 termina de atender a su batch y este requiere pasar a ser atendido por la actividad 8, pero esta todavia no termina de atender al batch que contiene, por lo que ocasiona que la actividad 7 adquiera el estado de bloqueo, debido a esto se condiciona el arribo al termino del servicio de esta actividad. Pudiendo obtener en menos cantidad de tiempo el lote de 5 batches.

*** REPORTE GENERAL ***

*** PARAMETROS GENERALES DEL SISTEMA ***

Num. de entidades terminadas:5
Num. de entidades perdidas:0
Tiempo de simulación:4986.90
Tiempo de arribo entre entidades: Condicionado por actividad 14.

*** REPORTE DE PARAMETROS POR ACTIVIDAD ***

Número de actividad	1	2	3
Tiempo de servicio	240.00	Frec. T.	11.00
Entidades procesadas	+ 5	5	5
Entidades pendientes	+ 0	0	0
Long. media de cola	0.00	0.00	0.00
Long. máxima de cola	0	0	0
Tiempo medio de espera	0.00	0.00	0.00
Num. de veces bloqueada	0	0	0
% T. en estado de bloqueo	0.00	0.00	0.00
% T. en estado ocioso	75.94	87.37	+ 98.85
Tiempo medio de residencia	+ 451.26	289.26	136.70

Figura 49.

*** REPORTE DE PARAMETROS POR ACTIVIDAD ***

Número de actividad	4	5	6
Tiempo de servicio	60.00	15.00	15.00
Entidades procesadas	5	5	5
Entidades pendientes	0	0	0
Long. media de cola	0.00	0.00	0.00
Long. máxima de cola	0	0	0
Tiempo medio de espera	0.00	0.00	0.00
Num. de veces bloqueada	0	0	0
% T. en estado de bloqueo	0.00	0.00	0.00
% T. en estado ocioso	93.98	98.50	98.50
Tiempo medio de residencia	181.96	124.96	121.96

*** REPORTE DE PARAMETROS POR ACTIVIDAD ***

Número de actividad	7	8	9
Tiempo de servicio	Frec. T.	268.00	60.00
Entidades procesadas	5	5	5
Entidades pendientes	0	0	0
Long. media de cola	0.00	0.00	0.00
Long. máxima de cola	0	0	0
Tiempo medio de espera	0.00	0.00	0.00
Num. de veces bloqueada	0	0	0
% T. en estado de bloqueo	0.00	0.00	0.00
% T. en estado ocioso	97.99	73.21	93.98
Tiempo medio de residencia	123.96	366.12	107.35

Figura 49. Continuación

*** REPORTE DE PARAMETROS POR ACTIVIDAD ***

Número de actividad	10	11	12
Tiempo de servicio	Frec. T.	60.00	25.00
Entidades procesadas	5	5	5
Entidades pendientes	0	0	0
Long. media de cola	0.00	0.00	0.00
Long. máxima de cola	0	0	0
Tiempo medio de espera	0.00	0.00	0.00
Num. de veces bloqueada	0	0	0
% T. en estado de bloqueo	0.00	0.00	0.00
% T. en estado ocioso	94.29	93.98	97.44
Tiempo medio de residencia	92.35	83.35	36.91

*** REPORTE DE PARAMETROS POR ACTIVIDAD ***

Número de actividad	13	14
Tiempo de servicio	Frec. T.	15.00
Entidades procesadas	5	5
Entidades pendientes	0	0
Long. media de cola	0.00	0.00
Long. máxima de cola	0	0
Tiempo medio de espera	0.00	0.00
Num. de veces bloqueada	0	0
% T. en estado de bloqueo	0.00	0.00
% T. en estado ocioso	97.47	98.50
Tiempo medio de residencia	31.20	18.00

Figura 49. Continuación

*** RESUMEN HISTORICO ***

#.- Bloqueo [a,z].- Ocupada

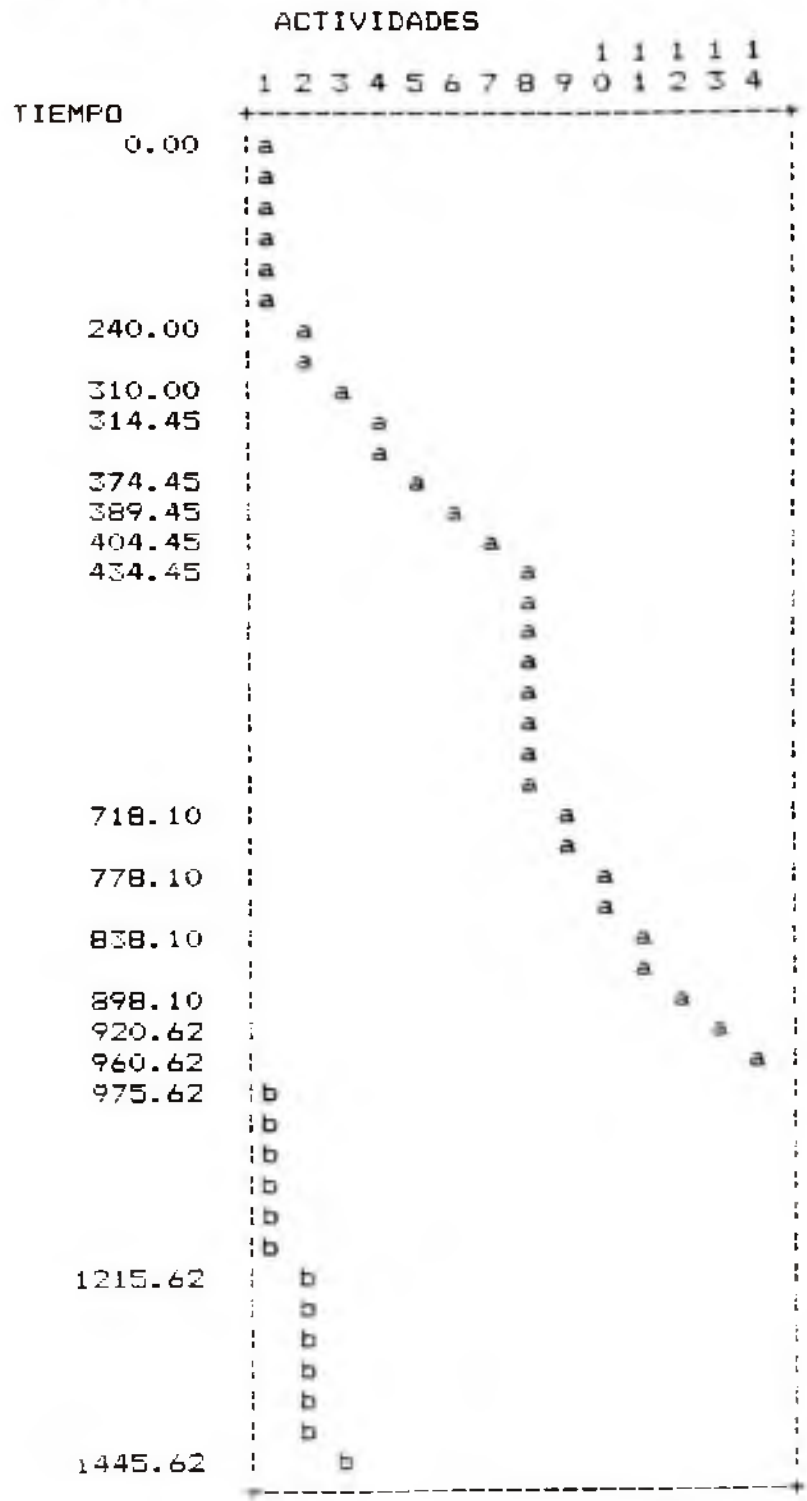


Figura 50.

*** RESUMEN HISTORICO ***

#.- Bloqueo [a,z].- Ocupada

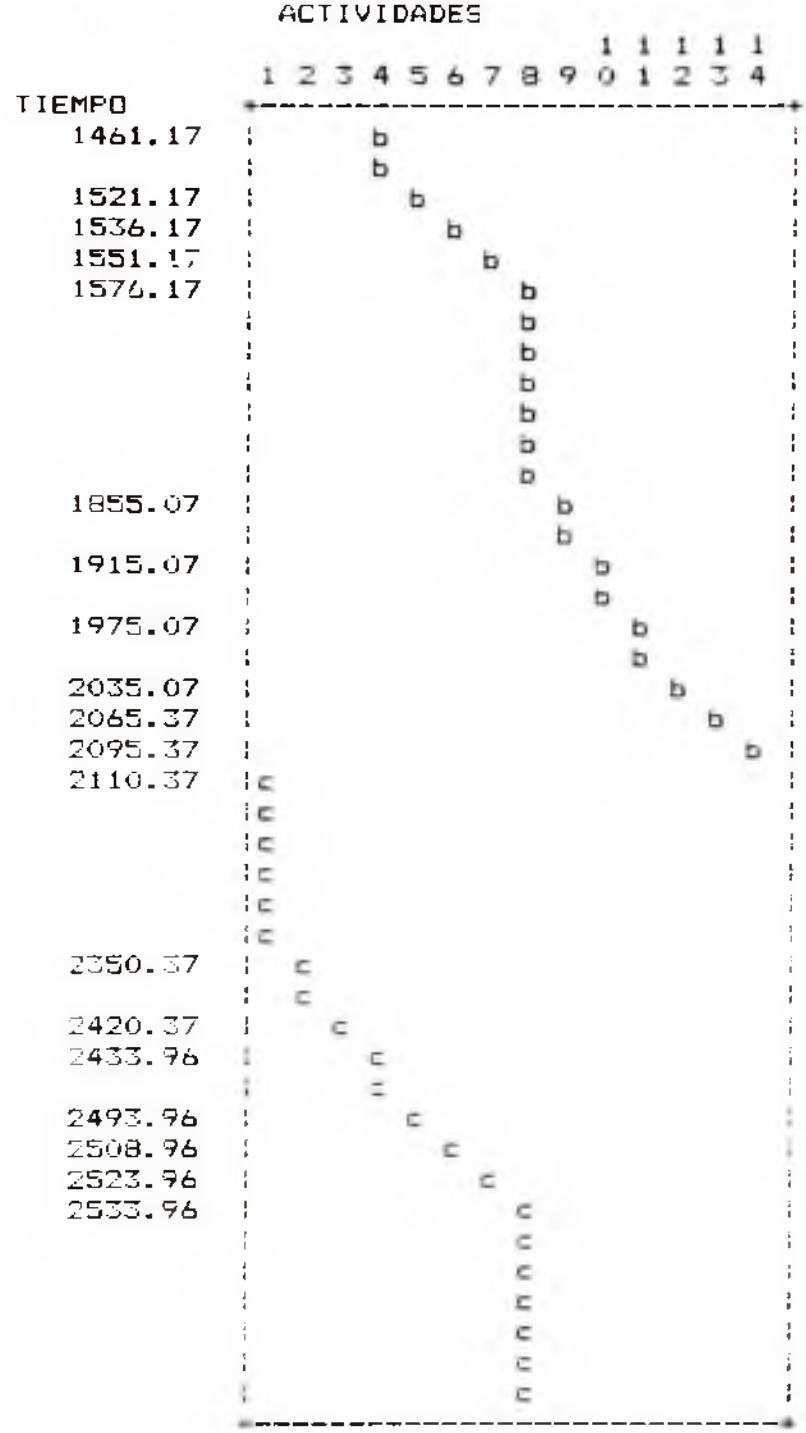


Figura 50. Continuación

*** RESUMEN HISTORICO ***

#.- Bloqueo [a,z].- Ocupada

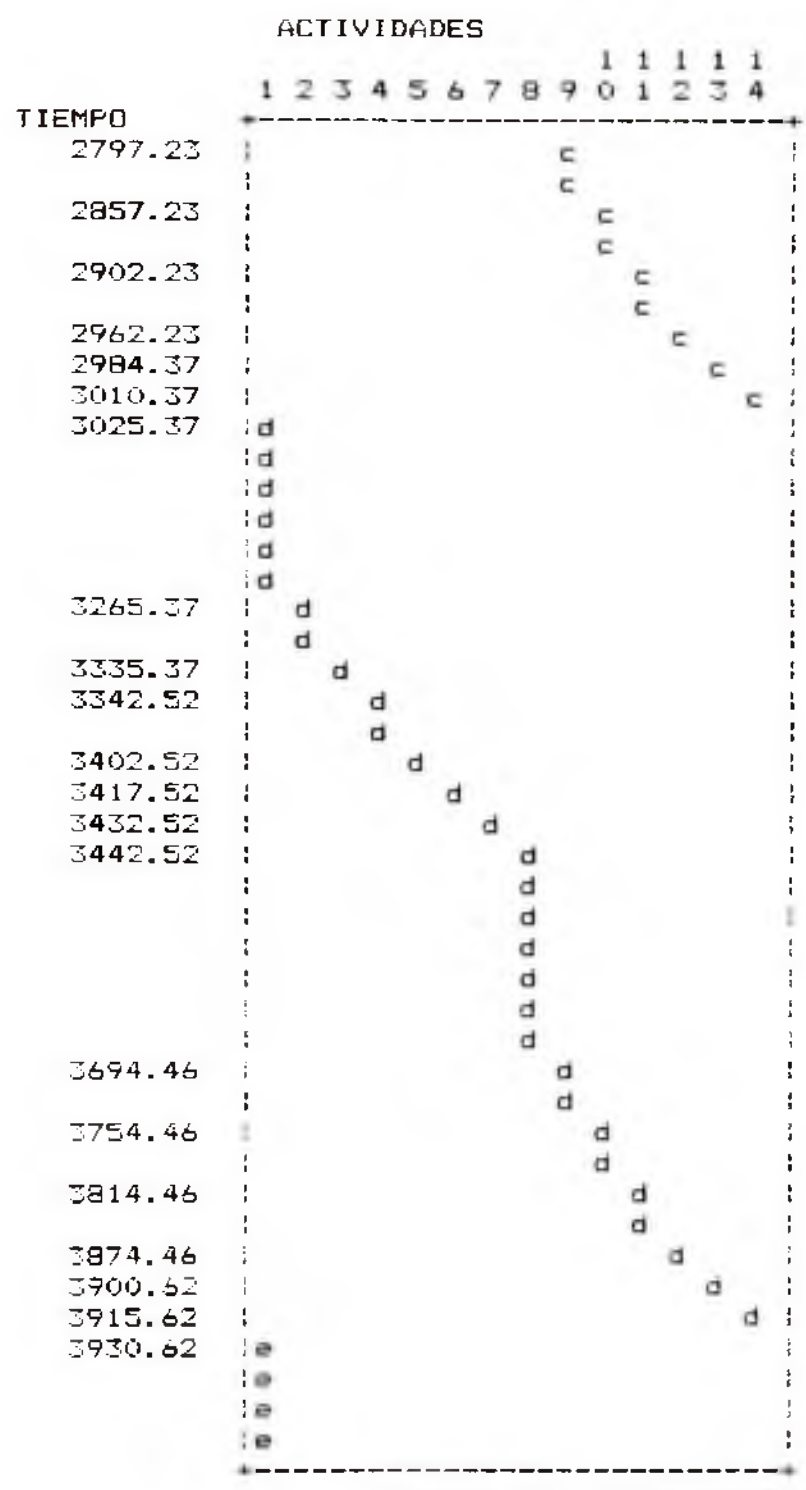


Figura 50. Continuación

```
*** RESUMEN HISTORICO ***

#.- Bloqueo        [a,z].- Ocupada

                ACTIVIDADES
                                    1 1 1 1 1
                1 2 3 4 5 6 7 8 9 0 1 2 3 4
TIEMPO      +-------------------------------+
            |e                              |
            |e                              |
  4170.62   |  e                            |
            |  e                            |
            |  e                            |
            |  e                            |
            |  e                            |
  4360.62   |    e                          |
  4377.11   |      e                        |
            |      e                        |
  4437.11   |        e                      |
  4452.11   |          e                    |
  4467.11   |            e                  |
  4492.11   |              e                |
            |              e                |
            |              e                |
            |              e                |
            |              e                |
            |              e                |
            |              e                |
  4750.17   |                e              |
            |                e              |
  4810.17   |                  e            |
            |                  e            |
  4870.17   |                    e          |
            |                    e          |
  4930.17   |                      e        |
  4956.90   |                        e      |
  4971.90   |                          e    |
  4986.90   |                               |
            +-------------------------------+
```

Figura 50. Continuación

Conclusiones

Con los ejemplos presentados en este capítulo, se puede observar la gran versatilidad con la que cuenta SIMDISC. Siendo de muy fácil manejo, para la construcción de modelos. En comparación con otros lenguajes (GPSS, SIMSCRIPT, FORTRAN, GASP, etc.) tomaría mucho más tiempo construir, el mismo modelo; que construirlo con SIMDISC.

Los reportes producidos por el simulador son de gran utilidad para el análisis del modelo. En el reporte general se puede observar el comportamiento de las actividades y de sus líneas de espera.

En el resumen histórico se observa el flujo de las entidades a través de las actividades del sistema, en éste reporte aparacen los tiempos en los cuales ocurren eventos importantes; por ejemplo, el bloqueo de una actividad y la duración en estado de bloqueo, el periodo de tiempo ocioso de una actividad, etc.

Se podría decir que el reporte general no es flexible, es decir; que siempre reportará los mismos parámetros estadísticos. Ya que podría darse el caso; que alguno de los parámetros no sea de gran importancia en algún sistema en particular, pero si lo sería otro parámetro no existente dentro del reporte.

Los parámetros seleccionados para presentarse en este reporte son los de mayor importancia para describir el comportamiento de las actividades y de sus líneas de espera. También pueden servir como datos para el cálculo de otros parámetros.

CONCLUSIONES GENERALES

- Este paquete ofrece algunas ventajas considerables sobre los lenguajes de simulación tales como GPSS, SLAM, SIMSCRIPT, etc. las ventajas son:

A) Ofrece una total interacción con el usuario, el cual no requiere de conocimientos de programación para la utilización del mismo, siendo una ventaja bastante significativa, ya que existe un gran rechazo por parte de los usuarios para el estudio de nuevos lenguajes de programación.

B) El simulador tiene la opción de poder observar en la pantalla el desarrollo de la simulación, cualidad recomendable para fines didácticos, diseño de sistemas y optimización de los mismos. Siendo también una ventaja sobre los lenguajes, ya que estos únicamente producen una serie de resultados al final de la simulación o en el mejor de los casos producen un listado de los eventos que se efectuaron durante la simulación, al final de cuentas siguen siendo números, que son difíciles de inter pretar.

Debido a que SIMDISC (Simulador de Sistemas Discretos) se encuentra orientado a la simulación de modelos estocásticos y siendo una herramienta interactiva con el usuario a nivel prototipo, cuenta también con algunas carencias con respecto a los lenguajes de simulación, las cuales son:

A) El simulador únicamente maneja un solo tipo de entidad. Ya que en muchos de los sistemas reales existen varios tipos de entidades. Involucrando la especificación de los tiempos de servicio para cada tipo de entidad dentro de una misma actividad y restricciones de atención a ciertos tipos de entidades por parte de las actividades.

B) Para el simulador todas entidades tienen la misma prioridad. Algunos sistemas manejan sus entidades por medio de prioridades, esta característica acarrea la necesidad que en el simulador se pueda especificar la diciplina que seguirán las entidades dentro de la línea de espera de una actividad.

C) SIMDISC no puede simular sistemas continuos o sistemas combinados (discretos y continuos), debido a que los sistemas continuos se comportan de acuerdo a una función que es bien particular para cada sistema.

Las carencias anteriores A) y B) son proyectos factibles de añadir a SIMDISC, sin la necesidad de hacer cambios muy significativos dentro del simulador. La carencia C) aumenta considerablemente la complejidad del simulador, ya que involucra un cambio en la administración de entidades, manejo del reloj y otras.

- Con los ejemplos presentados en el capítulo 3, se puede observar la gran versatilidad con la que cuenta SIMDISC. Siendo de muy fácil manejo, para la construcción de modelos. En comparación con otros lenguajes (GPSS, SIMSCRIPT, FORTRAN, GASP, etc.), ya que tomaría más tiempo la construcción del modelo; que construirlo con SIMDISC.

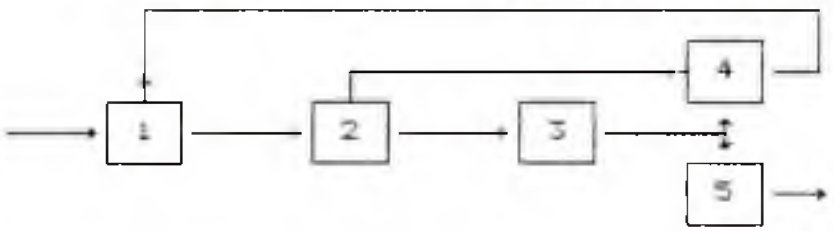
- El manejo del reloj de tiempo se lleva a cabo mediante el criterio de avance al próximo evento, el cual acelera la simulación, avanzando el reloj al tiempo en el cue se llevará a cabo el próximo evento. Haciendose uso de corrutinas con la finalidad de poder tener una mejor representación de las entidades que fluyen dentro del sistema, disminuyendo la complejidad del algoritmo, en comparación si éste estuviese construido de forma secuencial.

- Al inicio de la simulación se realiza la petición de toda la memoria para el desarrollo de la misma, optimizando la velocidad del algoritmo; evitando las operaciones de mantenimiento de bloques de memoria necesarios para la simulación y acelerándolo en lo que se refiere a la creación y terminación de entidades o corrutinas.

- Para aquellos sistemas en los cuales existan problemas de congestionamiento o "Cuellos de Botella", el algoritmo se encuentra diseñado para tratar de desbloquear a las actividades lo más rápido posible evitando que el congestionamiento se siga extendiendo a las actividades predecesoras. Se podría pensar en la asignación de niveles de profundidad a las actividades, para que cuando ocurran los bloqueos de las actividades se intente desbloquear primero a aquellas actividades que tengan el mayor nivel de profundidad, éste criterio es ideal para aquellos sistemas en los cuales no existan recirculaciones de entidades, ya que; con los sistemas que si cuentan con recirculaciónes aparece el problema de asignación de nivel a las actividades. Por ejemplo:

Figura A.

Figura B.



Estableciendo el siguiente criterio: el nivel de la actividad será igual al nivel de la actividad predecesora + 1, y para las actividades de entrada su nivel es de 0; entonces para la figura A, los niveles de las actividades 1, 2 y 3 son 0, 1 y 2 respectivamente, pero para la figura B, existe ambigüedad en las actividades 4 y 1, ya que ambas tienen dos actividades predecesoras de diferente nivel.

BIBLIOGRAFIA.

1. *ABEL, PETER.* **IBM PC Assembler Language and Programming,** Prentice Hall, 1987.

2. *BANKS, JERRY.* **Discrete Event Simulation,** Prentice Hall, 1984.

3. *ARCE MEDINA, ENRIQUE.* **A Resin Process Improvement trough Simulation,** International Conference of American Chemistry Society, Toronto Canada, 1987.

4. *BANKS, JERRY.* **Discret Event Simulation,** Prentice Hall, 1984.

5. *EDWIN G. LANDAUER.* **The Effect of Random Number Generators on an Application,** Comput. & Indus. Engng., Vol. 8, No. 1, pp. 65-72, 1984.

6. *J. KRIZ AND H. SANDMAYR.* **Extension of Pascal by Coroutines and its Application to Quasi-Parallel Programming and Simulation,** Software-Practice and Experience, vol 10, 773-789 (1980).

7. *N. D. FRANCIS.* **Generation of Random Numbers on Micros, A Simulation Study.,** Microprocessing and Microprogramming, Vol 15, 17-19 (1985).

8. *POLENTZ, LLOYD M.* **The Monte Carlo method of predicting production time,** Chemical Engineering, August 11, 157-161, 1980.

9. *PRITSKER, A. A. B.*, **The GASP IV Simulation language,** Wiley, New York 1974.

10. *SCHRIBER, THOMAS J.* **Simulation Using GPSS,** John Wiley & Sons, 1974.

11. *TIRADO RIOS URIEL.* **Implementación de Pascal Concurrente para Microcomputadoras IBM PC,** Informe Técnico No. 38, Departamento de Ingeniería Eléctrica, Cinvestav, julio 1987.

12. *TURBO PASCAL VER.* 4.0, Reference Manual, Borland International.

13. *ULLMAN JEFFREY D., AHO ALFRED V.*, **Data Structures and Algorithms,** Addison Wesley, 1983.

14. *VINARY K. VASUDEN., James M. Pruett.*, **MOSES: Manufacturing Organization Simulation and Evaluating System.,** Simulation, Enero, 37 - 45, 1990.

15. *W. H. KAUBISCH, R. H. PERROTT and C. A. R. HOARE.* **Quasiparallel Programming,** Software-Practice and Experience, Vol 6, 341-356 (1976).

APENDICE A.

CONSTRUCCION DEL RESUMEN HISTORICO

El resumen histórico reporta la forma en la que se desarrolló la simulación. Se observan los estados de las actividades, con el transcurso del tiempo, se distinguen los periodos de tiempo ocioso, de bloqueo, y el tiempo que la actividad permaneció ocupada.

Las entidades se distinguen por medio de una letra minúscula [a-z] que se les asigna de acuerdo al orden de llegada al sistema.

Cada estado de la actividad tiene un símbolo asociado, es decir cuando la actividad se encuentra en estado ocioso se representará por medio de un espacio en blanco, si se encuentra ocupada aparecerá la letra minúscula que representa la entidad y para el estado de bloqueo el símbolo es [▒] para el resumen desplegado a la pantalla, pero para la impresora el símbolo es [#].

La historia de la simulación se almacena en disco en el archivo *SIMDISC.SIM*. De la información de este archivo se contruye el resúmen histórico. La estructura del archivo es la siguiente:

<tiempo> <Número de actividad> <estado> <Número de entidad>

Cada registro en el archivo *events.sim* indica el tiempo en el que se efectuó un cambio de estado en alguna de las actividades de la red, indicando también cual de las entidades presentes en el sistema influyó en el cambio de estado de la actividad.

Los estados de una actividad son representados mediante digitos, en donde el 0 corresponde al estado ocioso, el 1 si la actividad se encuentra ocupada y el 2 para el estado de bloqueo.

El algoritmo utilizado para la construcción del histórico es el siguiente:

Paso 1. Se inicializa a todas la actividades en estado ocioso.

Paso 2. En seguida se realiza la lectura del registro que indica el cambio de estado y asigna el carácter correspondiente al estado que adquiere la actividad.

Paso 3. Efectúa la lectura de todos los cambios de estado que se efectuaron al mismo tiempo anteriormente leido. Asignando los caracteres correspondientes a los estados de las actividades.

Paso 4. Se despliega para el tiempo, todos los estados de las actividades.

Paso 5. Se avanza el tiempo con el incremento especificado para la construccion del histórico.

Paso 6. Se repiten los pasos 4 y 5 hasta que el tiempo llegue al siguiente cambio de estado, y regresa al paso 2. La lectura de registros continuará hasta que sea leído el último registro donde el número de actividad es igual a 0.

A continuación se muestra el contenido del archivo *events.sim* después de la simulación de la red del ejemplo 1 del capitulo III.

Archivo events.sim

```
 0.0000000000E+00 1 1 1
 9.2495799999E+00 1 0 1
 9.2495799999E+00 4 1 1
 1.0020968370E+01 4 0 1
 1.0020968370E+01 5 1 1
 1.7948797476E+01 1 1 2
 3.0261052729E+01 5 0 1
 3.2122877476E+01 1 0 2
 3.2122877476E+01 4 1 2
 3.5521176802E+01 1 1 3
 4.0415756802E+01 1 0 3
 4.0415756802E+01 1 2 3
 4.8365999357E+01 2 1 4
 5.9507555326E+01 4 0 2
 5.9507555326E+01 1 0 3
 5.9507555326E+01 4 1 3
 5.9507555326E+01 5 1 2
 6.1079633806E+01 1 1 5
 6.2677799357E+01 2 0 4
 6.2677799357E+01 2 2 4
 7.3289250732E+01 3 1 6
 7.7498213806E+01 1 0 5
 7.7498213806E+01 1 2 5
 7.8698028960E+01 5 0 2
0 0 0 0
```

El resumen histórico se encuentra organizado de la siguiente forma: los números de las actividades se encuentran en el reglón superior, el tiempo se encuentra del lado izquierdo el cual se incrementa de acuerdo al intervalo de tiempo que se haya especificado previamente para su construcción. La siguiente figura 51 muestra la estructura del resúmen histórico para el ejemplo anterior con un incremento de 5.

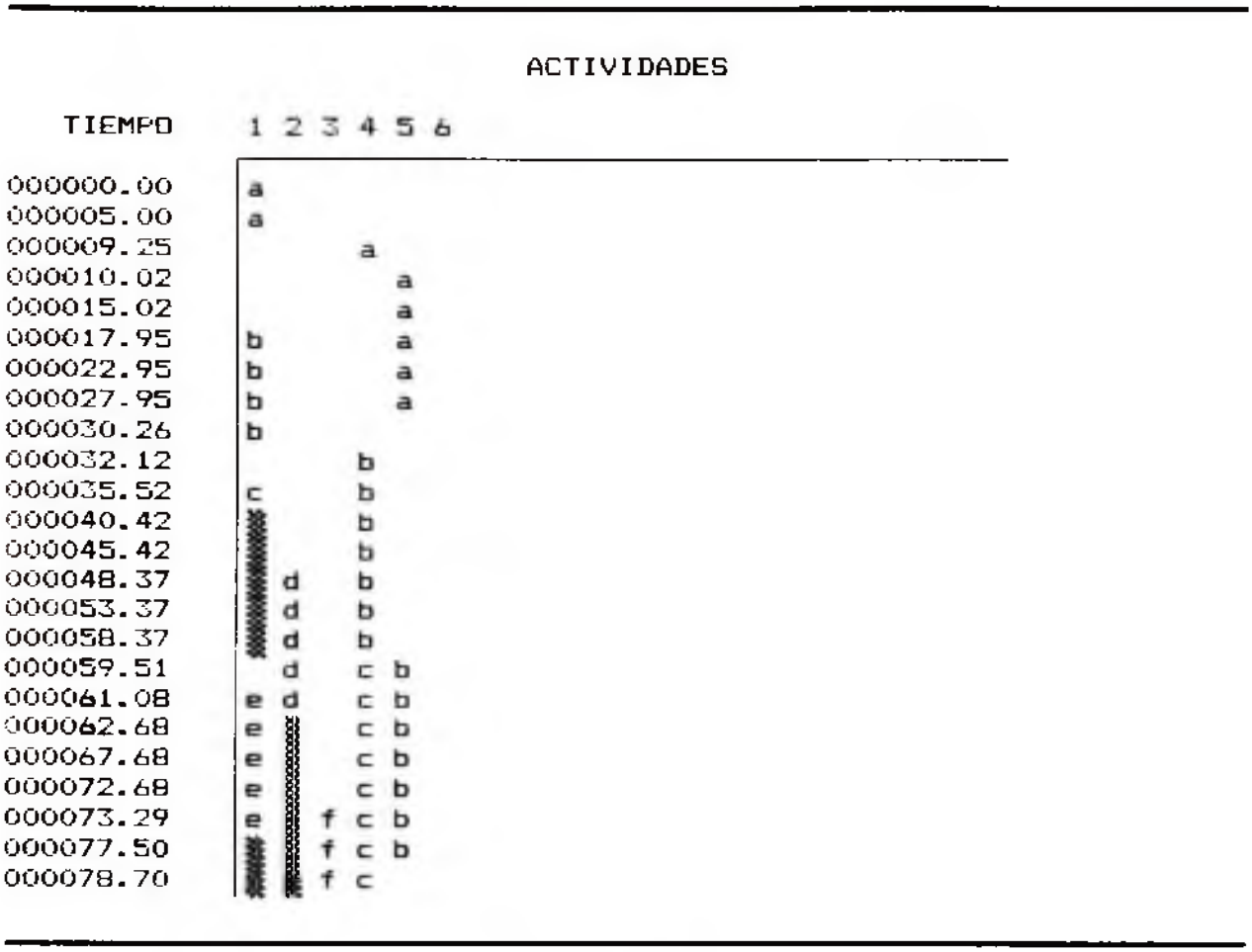

ACTIVIDADES
TIEMPO
1 2 3 4 5 6
000000.00
000005.00
000009.25
000010.02
000015.02
000017.95
000022.95
000027.95
000030.26
000032.12
000035.52
000040.42
000045.42
000048.37
000053.37
000058.37
000059.51
000061.08
000062.68
000067.68
000072.68
000073.29
000077.50
000078.70

Figura 51.

APENDICE B.

MANUAL DE USUARIO PARA EL SIMULADOR.

El simulador está construido con la finalidad de ser amigable para el usuario, para el manejo del simulador no se requieren conocimientos de programación, por lo que cualquier persona lo puede utilizar. Está construido a base de menus jerárquicos. Los cuales llevan de la mano al usuario para la construcción de la red de actividades, en caso de introducir un dato erróneo, el simulador desplegará ventanas de mensajes de error indicando el porqué no acepta dicho dato.

Las opciones dentro de los menús pueden ser seleccionadas mediante dos formas: por medio de las flechas del cursor, con las cuales se puede desplazar a través de las opciones, y una vez localizada la opción se presiona <enter> y el sistema ejecutará la acción correspondiente, o por medio de la selección de la letra que se encuentra de diferente color en cada opción, al teclear la letra la opción correspondiente se realiza automáticamente.

Se cuenta también en cada menú con una función de ayuda <F1>. La cual despliega una ventana, indicando para que sirven cada una de las opciones que se encuentran dentro del menú. Para desaparecer esta ventana oprimir la tecla <Esc>.

En la pantalla inicial del simulador se cuenta con tres secciones, la primera es la barra del menú principal, localizandose en el renglón superior, la segunda es el rectángulo que se encuentra a la mitad de la pantalla, y que lleva como nombre *Resumen de actividades*, lugar donde se listará la red de actividades, y por ultimo es el renglón de mensajes, el cual indica las posibles teclas a utilizar en un menú, o al desplegarse un mensaje de error.

Para arrancar el simulador se introduce el disco del sistema en la unidad de disco y escriba SIMDISC como se muestra a continuación:

A> SIMDISC <return>

Enseguida se desplegara el menú principal (vease figura 52).

MENU PRINCIPAL.

El menú principal cuenta con las opciones de gestión de Archivos:
Cargar, Nuevo, Guardar

Manipulación de Directorios:
Directorio y cAmbio de dir.

Dentro de la opción de *Archivo*.

La opción *Edición* construye la red de actividades y establece los parámetros de cada actividad.

La opción *eJecuta* efectúa la simulación de la red de actividades especificando el tiempo de simulación o el número de entidades a procesar y el tipo de desarrollo.

La opción *Reporte* despliega los resultados estadísticos de la simulación y por último *Salir* termina la ejecución, regresando al sistema operativo DOS.

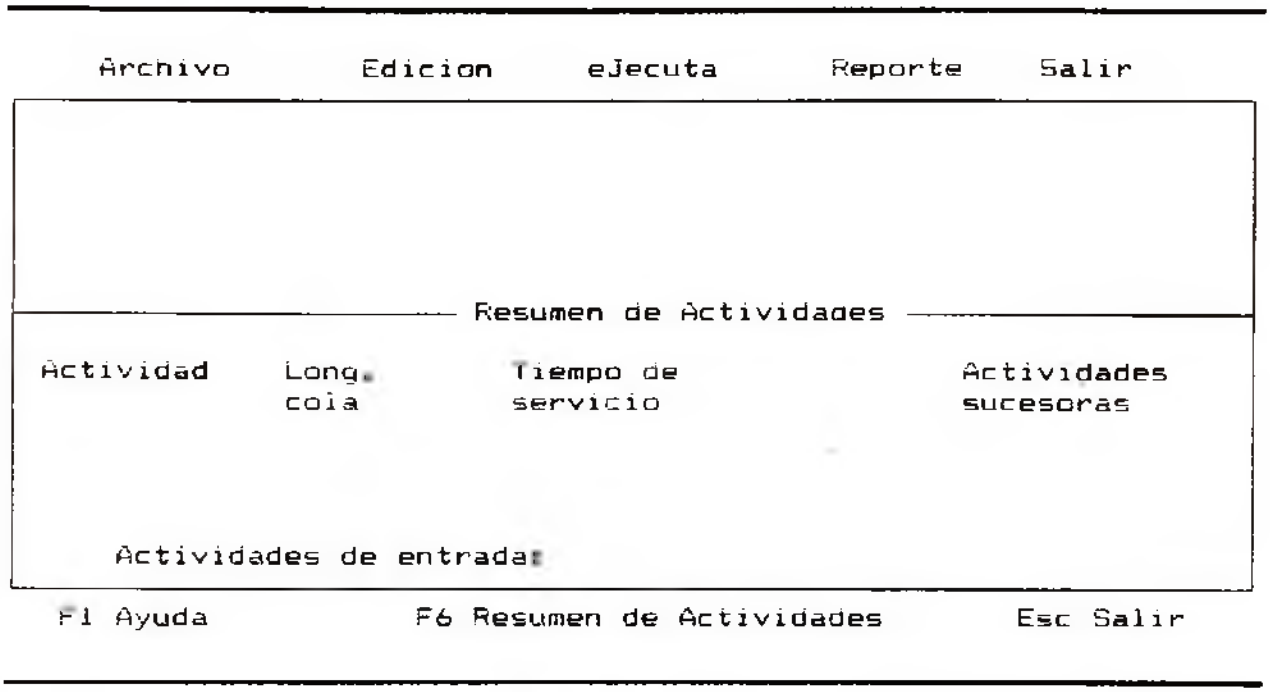


Figura 52.

MENU ARCHIVO

En este menú se ofrecen las operaciones de manejo de archivos, así como la manipulación de unidades de disco y de los directorios.El menú correspondiente se observa en la figura 53.

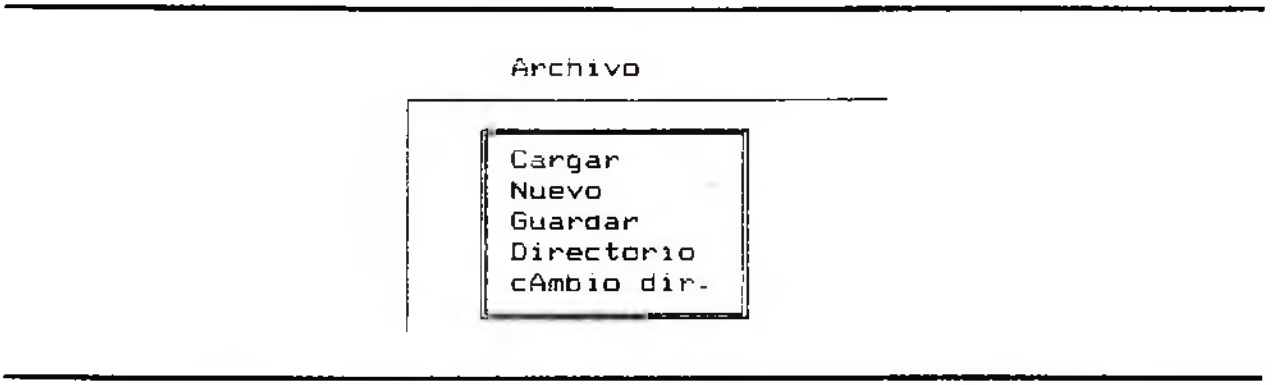


Figura 53. Menú de Archivo.

Cargar.

En esta opción se carga el archivo de trabajo. La información del proceso que se desea simular se encuentra dividida en dos archivos, con las extension .dat, y .prc respectivamente. El primer archivo guarda las caracteristicas e interconexiones entre las actividades del proceso. Y en el segundo se guarda el dibujo de como se encuentran localizadas las actividades en la pantalla.

Para cargar el archivo introducir unicamente el nombre del archivo sin ninguna extensión, ya que el simulador interpretará automáticamente y cargará en memoria los archivos necesarios.

Si se desea escoger un archivo en particular de un grupo de archivos de trabajo existentes en la unidad de disco, unicamente presione la tecla <space> una sola vez y presione <return>. En donde aparecerá una ventana con nombres de archivos disponibles para el simulador. Unicamente seleccione el archivo deseado y presione <return> (figura 54). Los archivos que describen a la red de actividades se cargarán automáticamente.

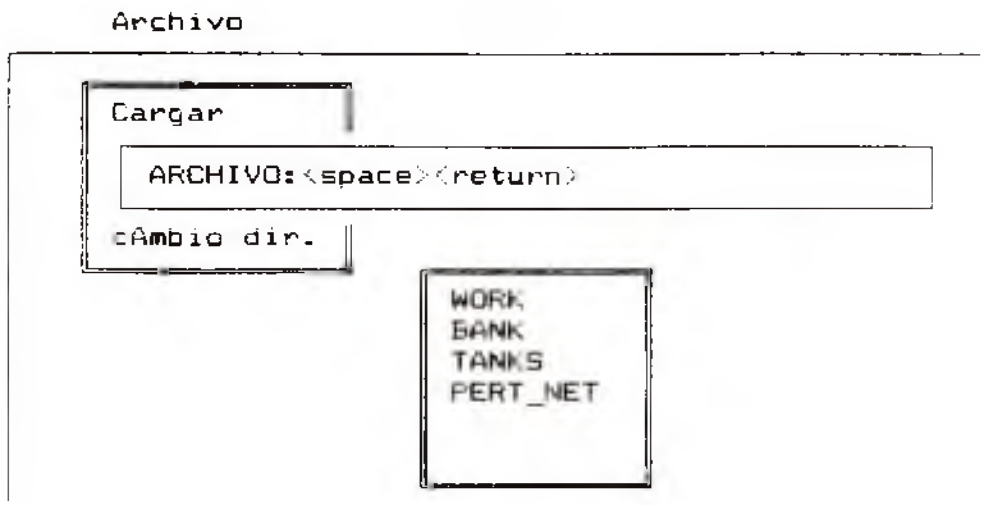


Figura 54. Selección de un archivo de trabajo.

Si el nombre del proceso seleccionado unicamente cuenta con el archivo de extension .dat se cargará el archivo de trabajo, pero hay que señalar que la información del dibujo del proceso se pierde por lo que hay que volver a generarlo, mediante la opción edita.

Pero si unicamente se encuentra el archivo .prc en la unidad de disco, el nombre del proceso no aparecerá en la ventana de archivos disponibles para el simulador.

Si la operación de cargado del proceso se lleva a cabo satisfactoriamente, la red de actividades se desplegará en la sección de Resumen de actividades en donde se mostrarán las caracteristicas de cada actividad y su interconexión con las demas actividades.

Nuevo.

En esta opción se limpia la memoria del simulador, es decir que si se encuentra un proceso en memoria, y se selecciona esta opción, se inicializa el sistema y por consiguiente se puede empezar a construir una nueva red de actividades o proceso.

Al llevarse a cabo esta opción se observará que el proceso existente se borra automaticamente de la sección de Resumen de actividades. El nombre que se le asigna al nuevo proceso es WORK, el nombre se puede cambiar al seleccionar la opción Guardar en donde se preguntará el nuevo nombre que recibirá el proceso.

Guardar.

Una vez construida la red de actividades mediante la opción **Edición** si se desea guardar en disco seleccione esta opción. Esta opción crea dos archivos con el mismo nombre, pero con dos extensiones diferentes, debido a que cada archivo contiene información del proceso necesaria para su simulación. (ver opción Cargar).

El nombre del archivo a seleccionar puede contar con un máximo de ocho caracteres. El nombre se debe de introducir sin ninguna extensión.

Los caracteres válidos para el nombre del archivo son "a" ... "z", "0" ... "9" y "_", cualquier combinación de estos caracteres son aceptados para el nombre del archivo.

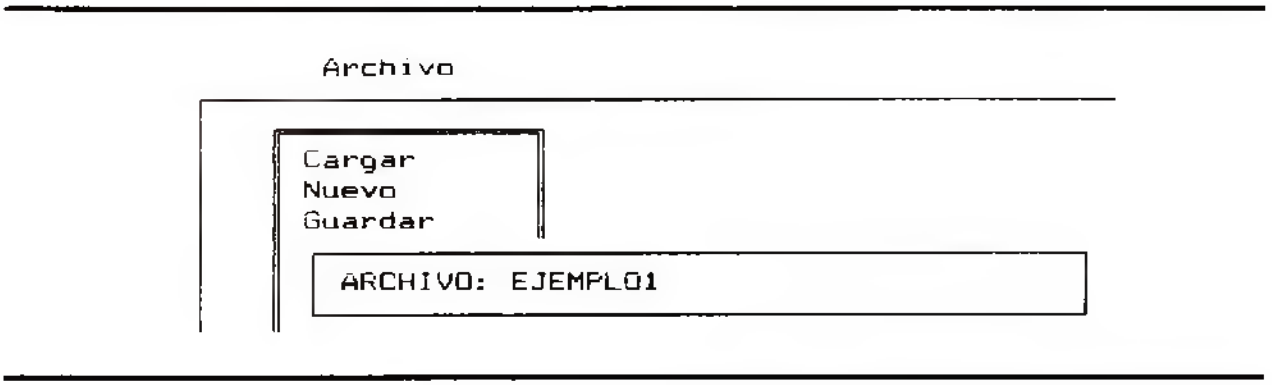


Figura 55. Guardar un Archivo.

NOTA: El simulador no verificará la existencia previa de un archivo con el mismo nombre, por lo que hay que tener cuidado de no repetir nombres de archivos, para evitar pérdida de otros procesos.

Directorio.

Con esta opción se puede revisar el contenido existente en el disco de trabajo, previamente al aparecer el directorio se preguntará por el patrón de búsqueda (figura 56). En donde se puede introducir los caracteres de substitución del DOS [*] y [?], para la búsqueda de archivos específicos en el directorio. El directorio se desplegará en una ventana en el centro de la pantalla, en caso de existir demasiados archivos para el espacio de la ventana, presione una tecla cualquiera y aparecerá la segunda parte del directorio.

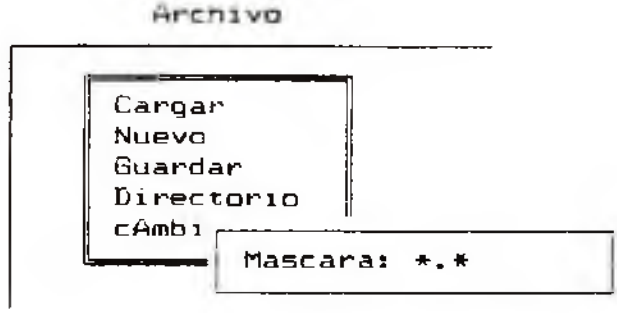


Figura 56. Introducción del patrón de busqueda para el directorio.

Cambio dir.

Debido a que un disco se puede encontrar organizado mediante directorios, con esta opción se puede selecionar el directorio del disco. Apareciendo el disco y el directorio con el cual se encuentra trabajando el simulador. Si al introducir el nuevo directorio, este no se encuentra el simulador mandará un mensaje de error, especificando la no existencia del directorio.

Con esta opción tambien se puede cambiar de unidad de disco de trabajo. Con la finalidad de guardar y cargar archivos en una unidad de disco diferente, a la unidad en la cual se encuentra el simulador.

Se recomienda cuando se vaya a trabajar con el simulador, usar dos discos, un disco para el simulador y el otro disco para almacenar y cargar los archivos de trabajo. Esto es debido a que el disco del simulador necesita espacio, porque cuando se efectúa una simulación, se va dejando en el disco la información de como se va llevando a cabo la simulación del proceso.

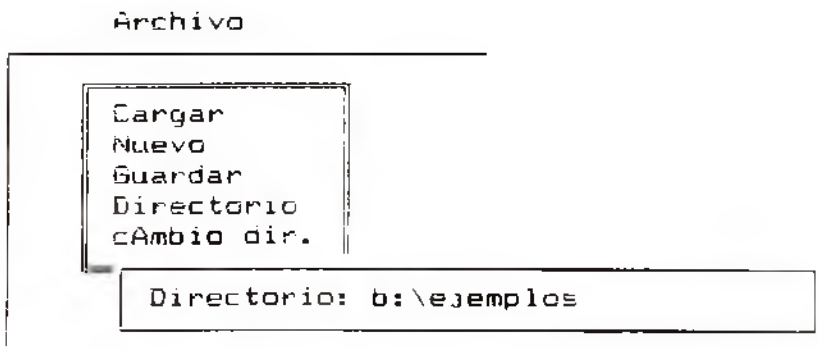


Figura 57. Cambio de Directorio.

MENU EDICION.

Con este menú se contruye la red del proceso. por lo que involucra operaciones de mantenimiento de actividades, en el proceso, asi como de especificación de parametros de cada actividad. Se puede editar una sola actividad a la vez, especificando el numero de actividad. El menú ofrece las siguientes opciones.

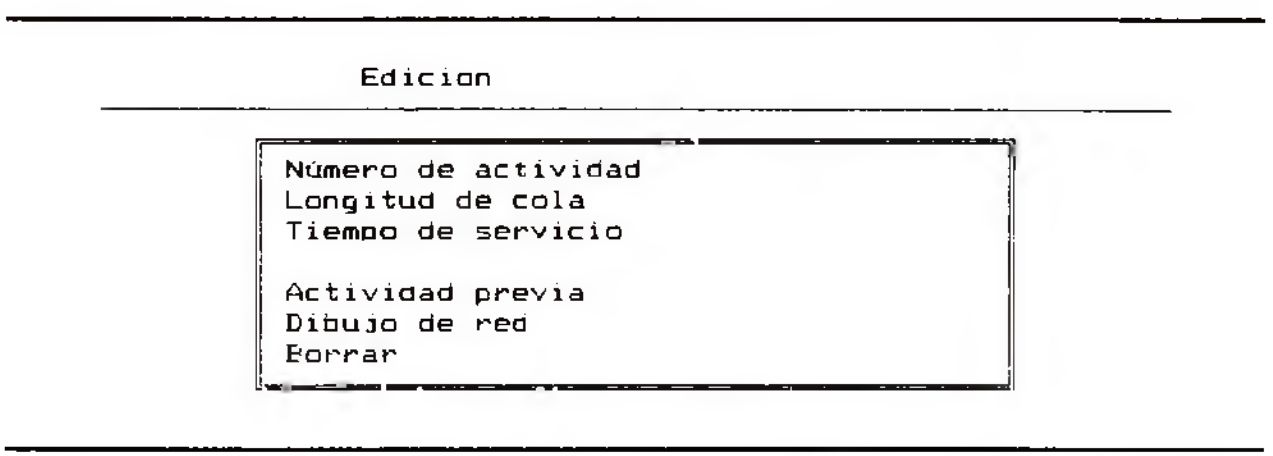


Figura 58. Menú de edición.

Número de actividad.

En esta opción se le asigna a una actividad un número de identificación, y si la actividad existe se edita, mostrando los parámetros de la actividad.

Longitud de cola.

Es la capacidad máxima de la línea de espera de la actividad. Es decir la cantidad máxima de entidades en espera de servicio que puede contener la actividad.

Tiempo de servicio.

Es el tiempo que le toma a la actividad realizar una orden de trabajo. El tiempo se puede especificar mediante dos formas, mediante una distribución probabilística o mediante una tabla de distribución de frecuencias.

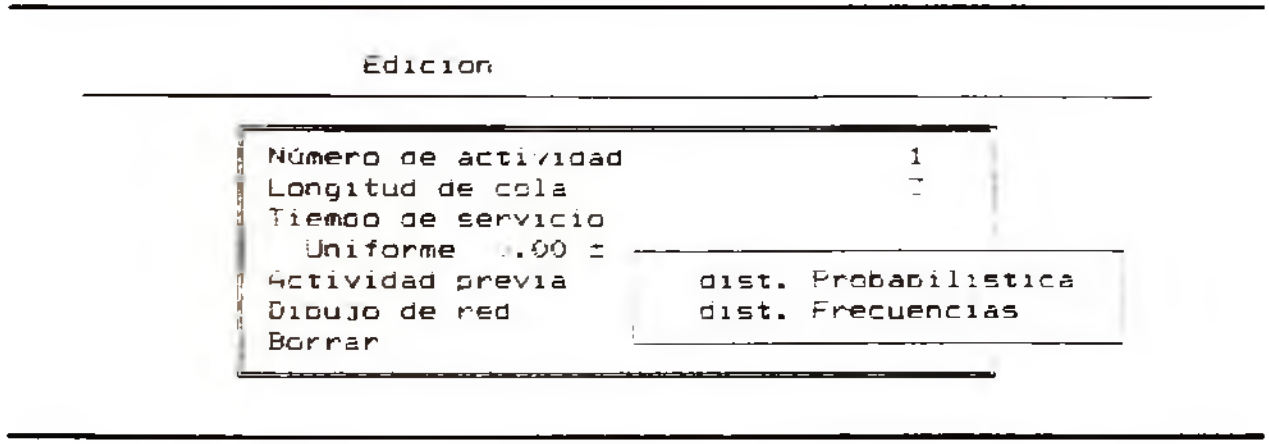


Figura 59. Selección del tipo de tiempo de servicio.

Distribución Probabilistica.

Cuando la duración del tiempo de servicio varía de acuerdo a una función especifica. Y teniendo los datos de la media y la varianza (figura 61), se puede establecer la función. Las funciones probabilísticas que se pueden seleccionar son las siguientes:

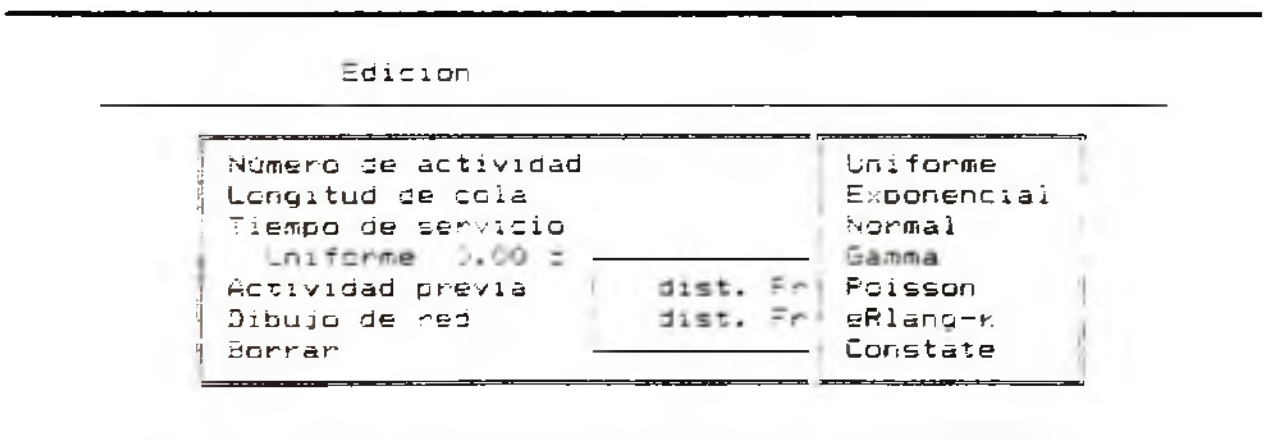


Figura 60. Selección de la función probabilistica.

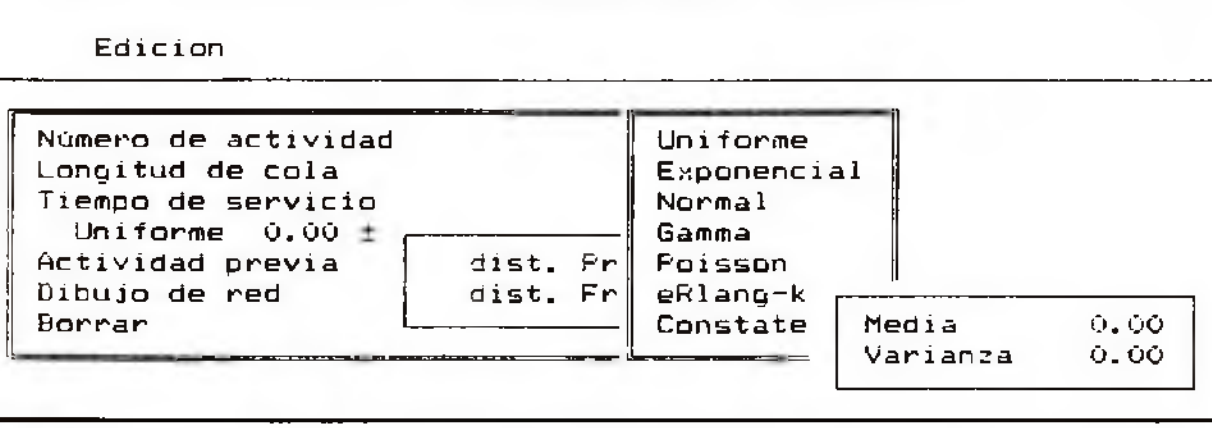


Figura 61. Especificación de los parametros de la distribución.

Distribución de frecuencias.

Cuando la variación de los tiempos de servicio no cumple con una función probabilistica, entonces se especifica una tabla, que nos indica el número de eventos que tomaron una cierta cantidad de tiempo para realizar una orden de trabajo, por ejemplo:

tiempo	*frecuencia*
20	2
25	5
32	9
38	18

Este tipo de tabla no se puede introducir tal cual al simulador, sino que debe transformarse la segunda columna a frecuencia acumulada, según el método de Monte Carlo [3]. En la siguiente forma:

tiempo	*frecuencia %*	*frecuencia acumulada %*
20	3.53	3.53
25	8.47	12.00
32	15.25	27.25
38	30.51	57.76
45	42.37	100.00

Y finalmente obtenemos la tabla que debemos introducir tiempo vs. frecuencia acumulada.

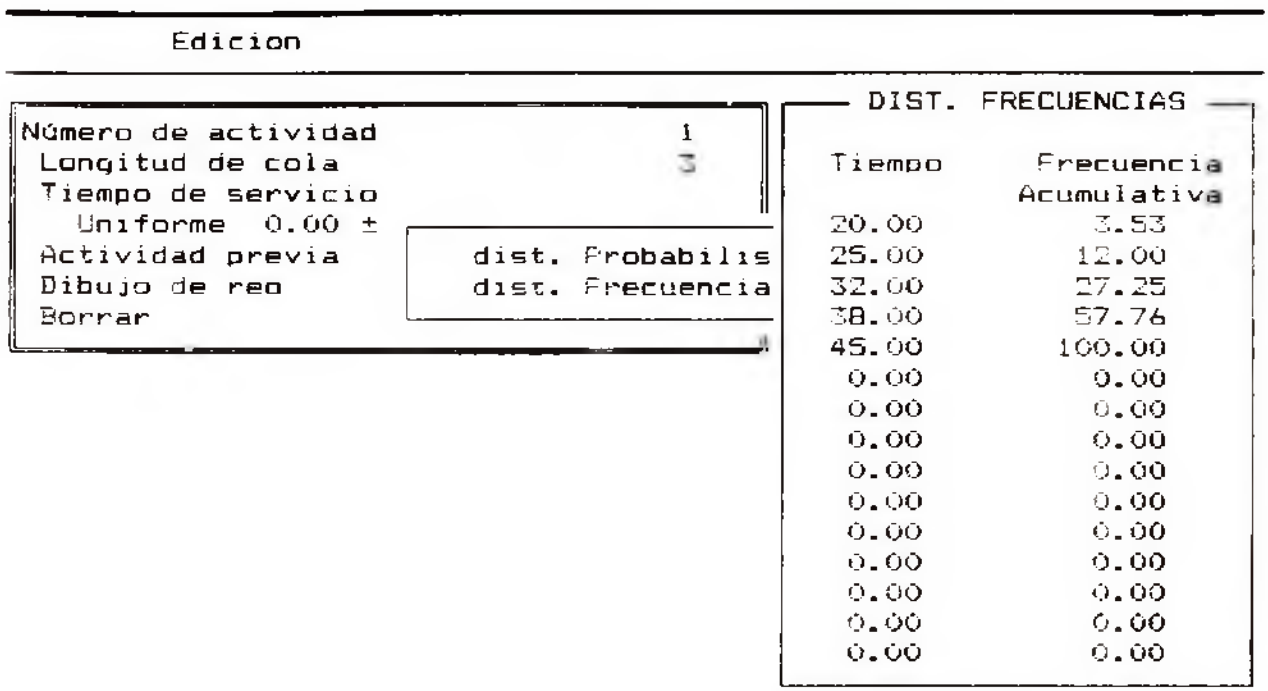


Figura 62. Introducción de la distribución de frecuencias.

Actividad previa.

Con esta opción se construye la red de actividades, si no se especifica la actividad que precede a la actividad que se encuentra editando, esta se perderá y no se visualizará en la sección de Resumen de actividades.

Hay actividades que no tienen actividad previa, en estas actividades unicamente tecleese cero, y quedarán como actividades de entrada, y aparecerá el número de identificación en el renglón de Actividades de Entrada dentro de las sección de Resumen de actividades. Y enseguida se desplegarán los parámetros de dicha actividad.

Para introducir la actividad previa es necesario que la actividad exista, en caso contrario, el simulador mandará el siguiente mensaje << Error No exite la actividad previa >>.

Ya que esta opción es la que relaciona la actividad a la red del proceso, la actividad no estará en la red si no aparece en la sección de Resumen de actividades.

En la sección de Resumen de actividades se puede observar las relaciones entre las actividades. En donde la actividad predecesora es la actividad que se especifica en la columna de actividad, y las actividades sucesoras a esta son las que se encuentran en la columna actividades sucesoras. Una actividad puede tener de 0 hasta 15 actividades sucesoras.

Dibujo de red.

Con esta opción se construye el dibujo de la red de actividades en la pantalla. Al seleccionar la opción la pantalla de menus se borra, presentando la pantalla en la cual se guardará el dibujo del proceso. La actividad que se encuentre editando aparecerá de color verde en la esquina superior izquierda de la pantalla. Representando a una actividad se muestra un cubo, con el número de actividad correspondiente.

El cubo se puede localizar en la pantalla, utilizando las teclas con flechas. El simulador cuenta con más de una pantalla para la localización del cubo, es decir, la red se dibuja sobre una sábana, y el monitor unicamente muestra una parte de ella. El modo de operación es Modo de Localización y la pantalla se encuentra como la siguiente figura 63.

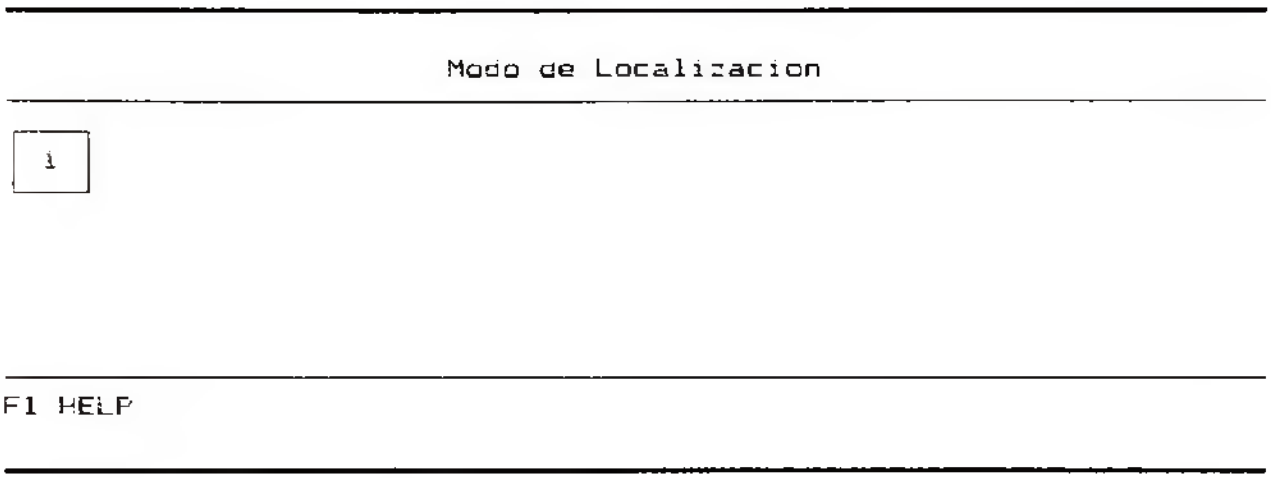


Figura 63. Modo de localización de actividades.

Para dibujar líneas se necesita estar en Modo de Línea, para accesar este modo tecleese <F10>, y el mensaje correspondiente al Modo se desplegará en la parte superior de la pantalla. Aparecerá un cursor en forma de cruz en el centro de la pantalla. El cual se puede desplazar con las teclas previamente especificadas.

El cursor puede variar de velocidad de desplazamiento, mediante las teclas <F7> y <F8>, aumentando y disminuyendo la velocidad respectivamente.

La línea se traza esoecificando dos puntos. La parte central del cursor indica los extremos de la línea, para marcar un extremo de la línea presione la tecla <enter>.

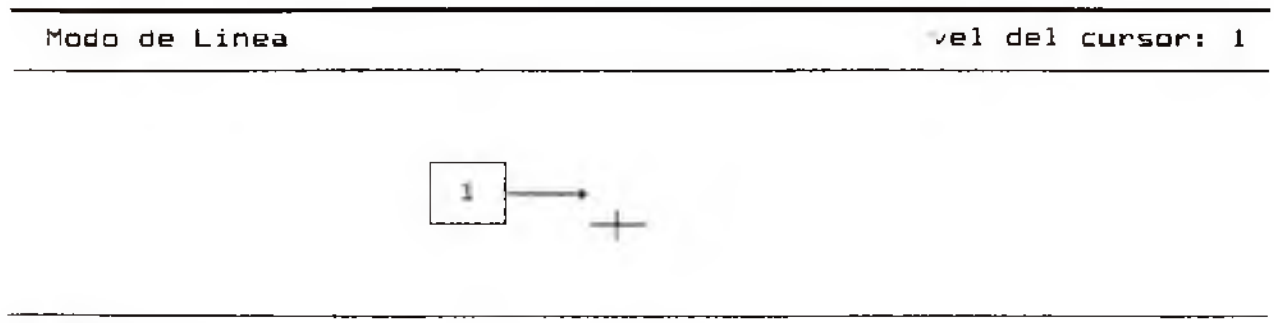


Figura 64. Pantalla del Modo de línea.

En el modo de linea tambien se pueden generar las flechas que indican la dirección del flujo entre las actividades, para obtener las flechas presione conjuntamente las teclas <Alt> y la tecla de flecha correspondiente a la dirección de la flecha que se desee obtener. Por ejemplo:

<Alt><→> produce → <Alt><←> produce ←
<Alt><↑> ptoduce ↑ <Alt><↓> produce ↓

Si se desea borrar una línea o una área de la pantalla, se realiza con el modo de Borrar, el cual se accesa con la tecla <F9>. Apareciendo en la parte superior de la pantalla el mensaje correspondiente al modo. Para borrar una área se debe marcar los extremos de ésta, es decir especificando la esquina superior izquierda y enseguida la esquina inferior derecha o viceversa. Una vez marcada el área, ésta desaparecerá automáticamente.

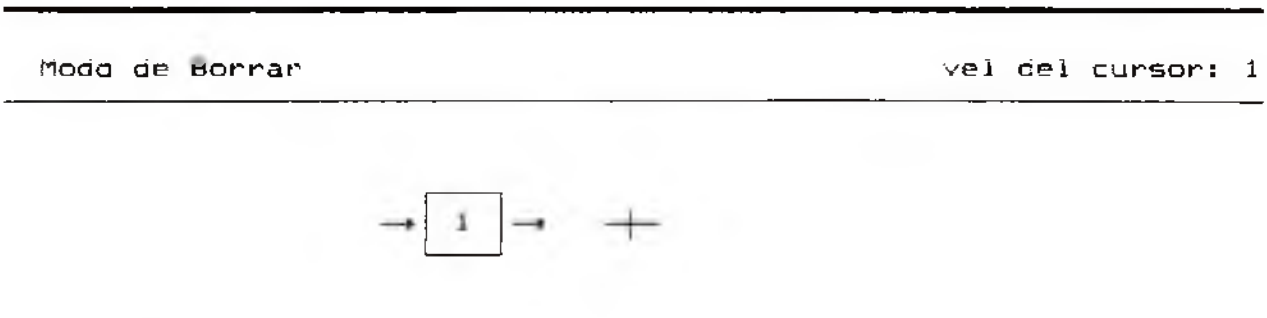


Figura 65. Pantalla del Modo de Borrar.

Dentro de los modos para linea y borrar se recomienda no realizar las operaciones con desplazamiento de la pantalla, ya que si se desea dibujar una linea y se recorre la pantalla ésta aparecerá truncada, lo mismo ocurrirá con el modo de borrar areas.

Para salir de los modos previamente mencionados presione la tecla <Esc> para regresar al modo de Localización. Y para regresar al menú de edición teclee nuevamente la misma tecla para dejar el modo. Todos los modos cuentan con una ventana de ayuda, y ésta se activa al presionar la tecla <F1>, desplegando las teclas que se pueden utilizar dentro del modo.

Para guardar el dibujo de la red de actividades tecleese <F2>, se almacenará el dibujo, así como la especificación de las actividades de la red. El nombre del archivo debe de ser previamente establecido mediante la opción Guardar del menú de Archivo. En caso contrario, la red se guardará con el nombre de Work.

Borrar.

En esta opción se puede eliminar las interconexiones entre las actividades, así como una actividad dentro de la red. Desplegando un menú ofreciendo dos opciones Actividad o Liga.

Si se desea eliminar una actividad seleccione la opción Actividad, enseguida se preguntará por la verificación de la operación, en caso de contestar <S>, desaparecerá la actividad en la sección de Resumen de Actividades, así como todas las ocurrencias de ésta en las interconexiones que se tengan con las demás actividades. Borra automáticamente del dibujo de la red de actividades el cubo que representa a dicha actividad.

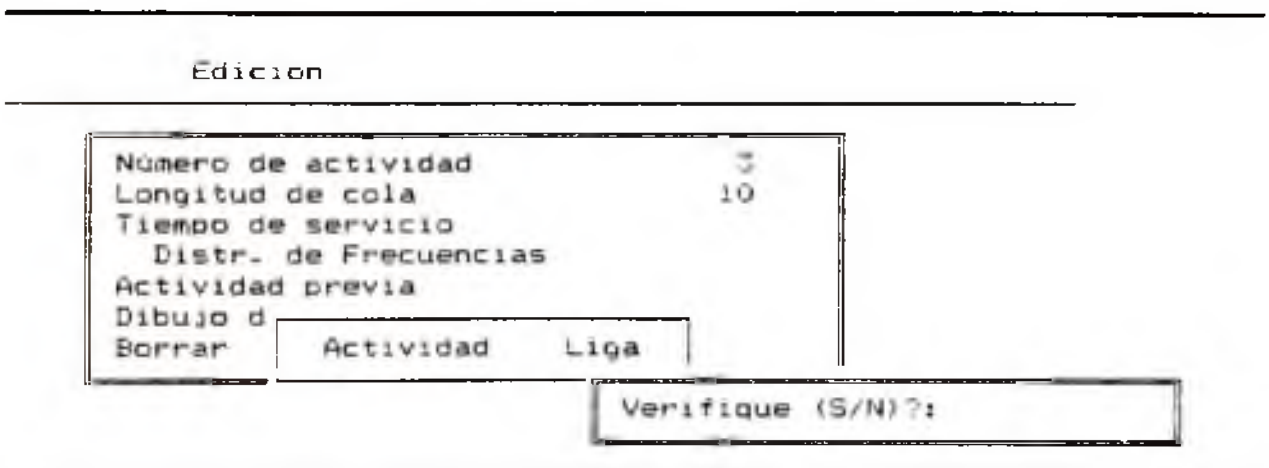


Figura 66. Borrado de una actividad.

En caso de seleccionar la opción Liga, se preguntará por la actividad que precede a la actividad que se está editando, y automáticamente desaparecerá la interconexión entre ambas actividades. Si en el dibujo de la red se encuentra una interconexión mediante una o varias líneas éstas no desaparecerán del dibujo por lo que hay que borrarlas. El resultado de la operación se puede observar en la sección de Resumen de actividades.

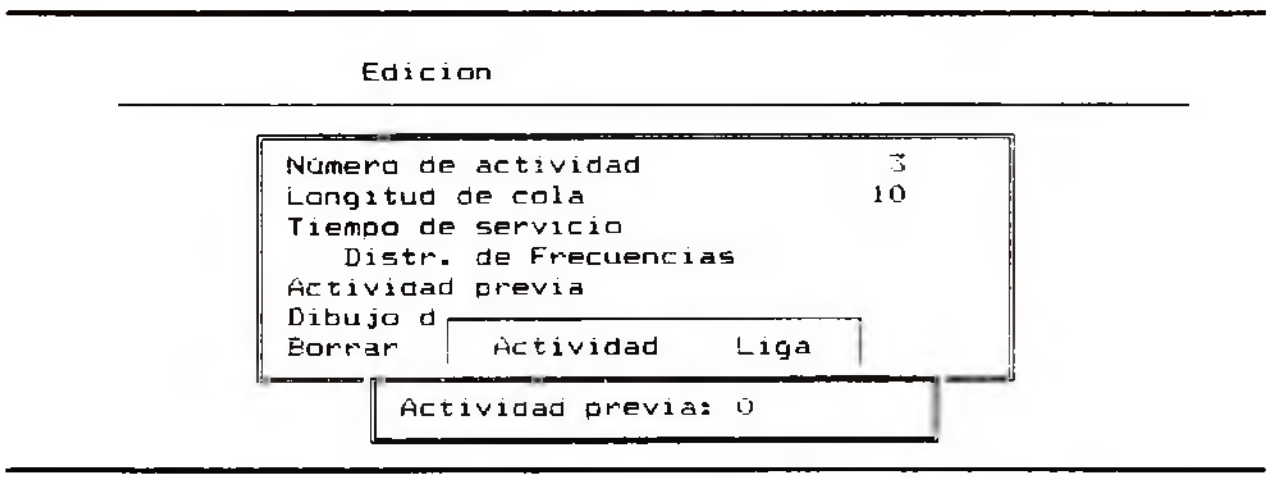


Figura 67. Borrado de una liga entre actividades.

MENU EJECUTA.

Con este menú se establecen los parámetros de la simulación, como son el tiempo de arribo entre las ordenes de trabajo u entidades, tiempo de simulación, cantidad de entidades a simular, y el tipo de ejecución en que se deseé efectuar la simulación.

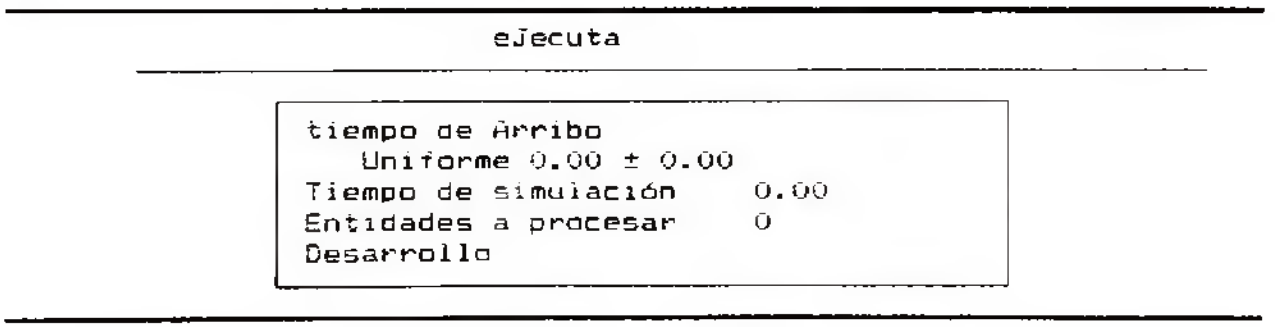


Figura 68. Menú de Ejecución.

Tiempo de arribo

En esta opción se especifica el tiempo de arribo entre las entidades. Este tiempo se puede especificar mediante una función probabilística, o mediante una tabla de distribución de frecuencias y por arribo condicionado.

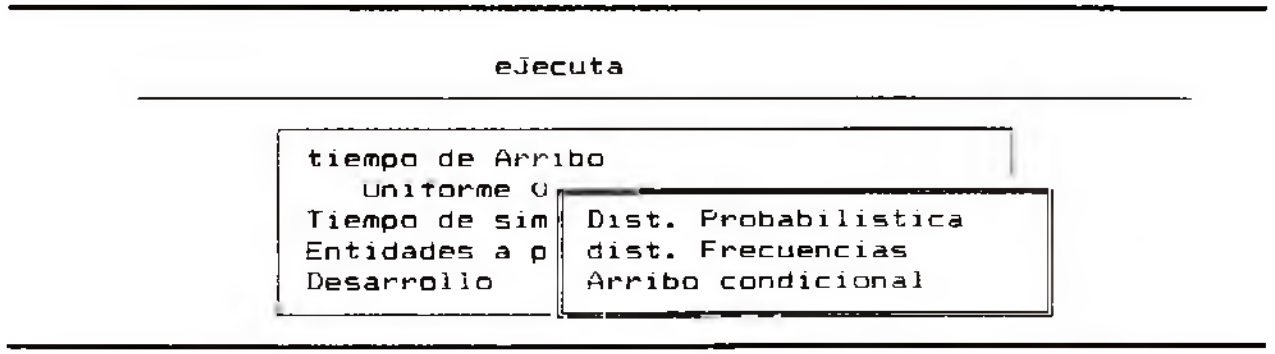


Figura 69. Especificación del tipo de arribo de las entidades.

Dist. Probabilistica.

Con esta opción se especifica la función probabilística, así como sus parámetros como son la media y la varianza (figura 70).

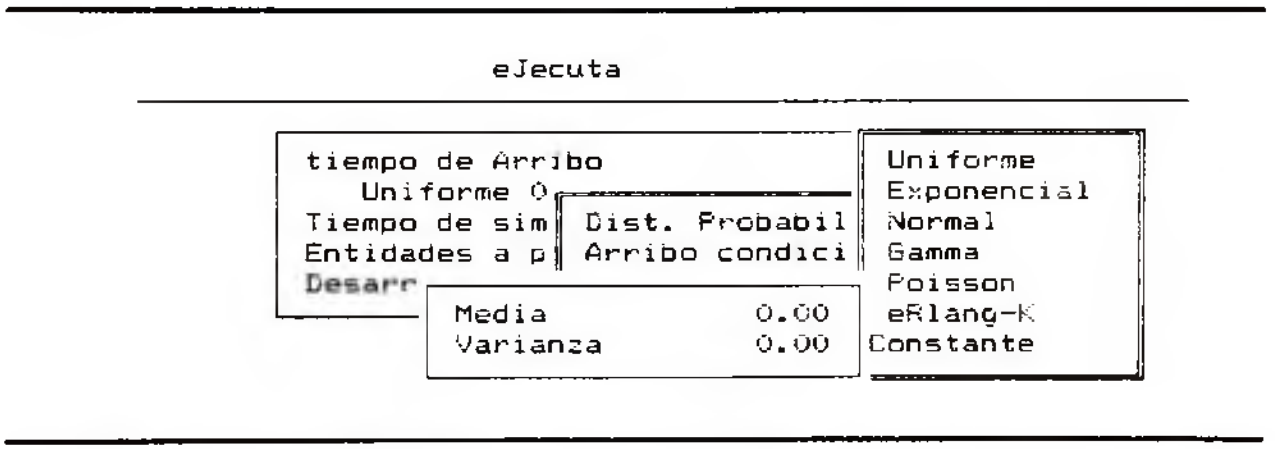


Figura 70. Arribo descrito por una distribución probabilística.

Ditribucion de Frecuencias.

Con esta opción se introduce la tabla de frecuencias que describe el comportamiento de los tiempos de arribo entre las entidades.

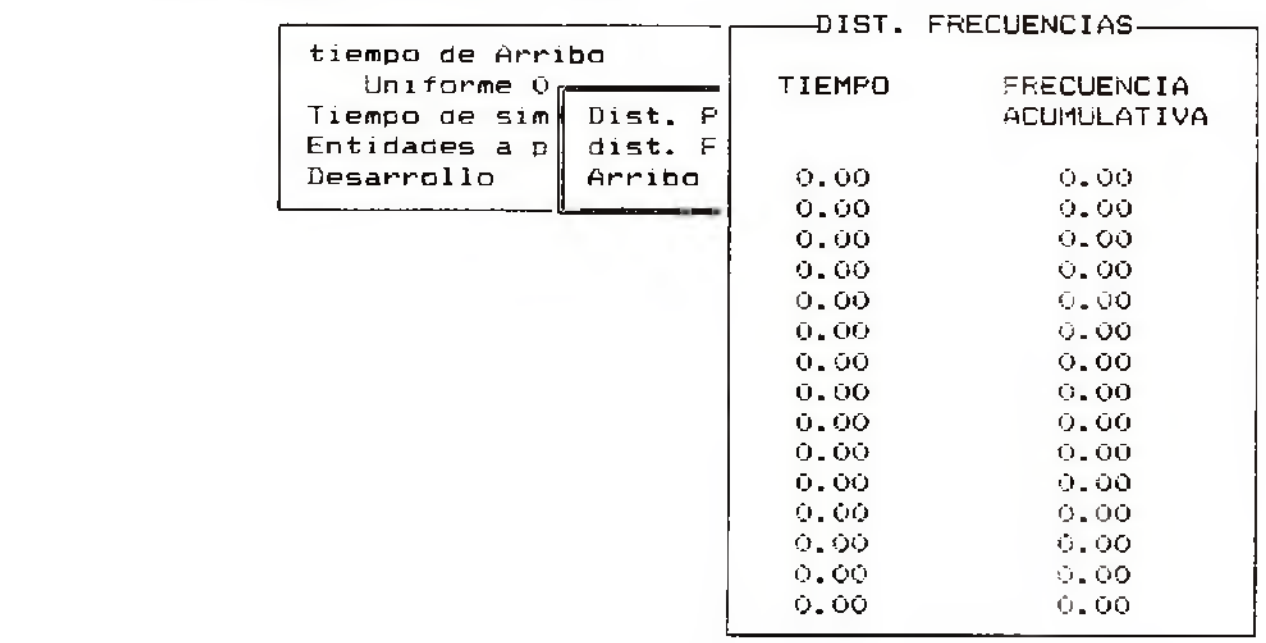


Figura 71. Especificación del tipo de arribo de las entidades.

Arribo Condicionado.

Se condiciona la llegada de una orden de trabajo a la red, con la terminación del servicio de una actividad. En esta opción se pregunta por la actividad que condicionará el arribo de las ordenes de trabajo. La actividad que se especifique debe de existir en la red, en caso contrario no se llevará a cabo la ejecución de la simulación por arribo condicional, el simulador desplegará una ventana especificando la no existencia de la actividad en la red.

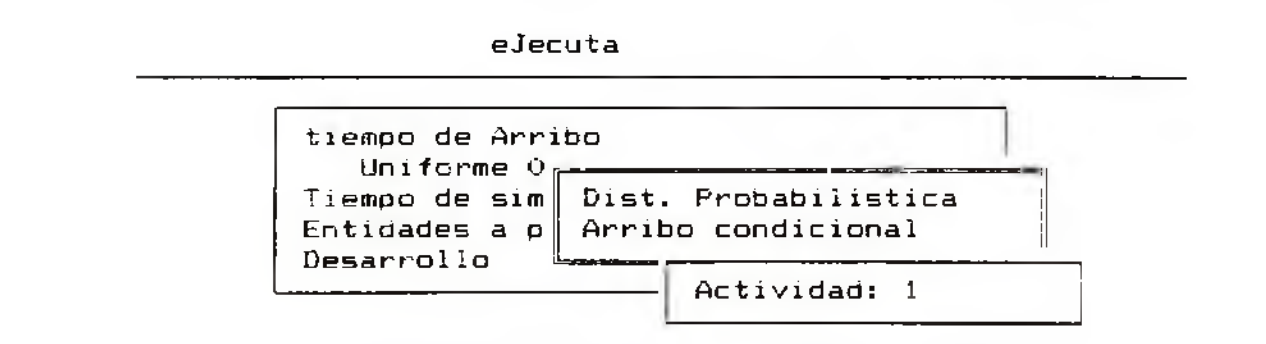


Figura 72. Arribo condicionado.

Tiempo de simulación

Se establece el tiempo que durará la simulación. El tiempo que se establezca debe de ser relativo a las unidades de tiempo que plantea el problema.

Entidades a procesar.

Con esta opción se establece la cantidad de ordenes de trabajo que se deben realizar completamente por la red. La simulación no se detendrá hasta cuando se termine la última orden de trabajo especificada.

Desarrollo.

Se establece el tipo de ejecución que se desea para la simulación. Se cuenta con tres tipos de desarrollo que son: Paso por paso, Rápido y Extra Rápido.

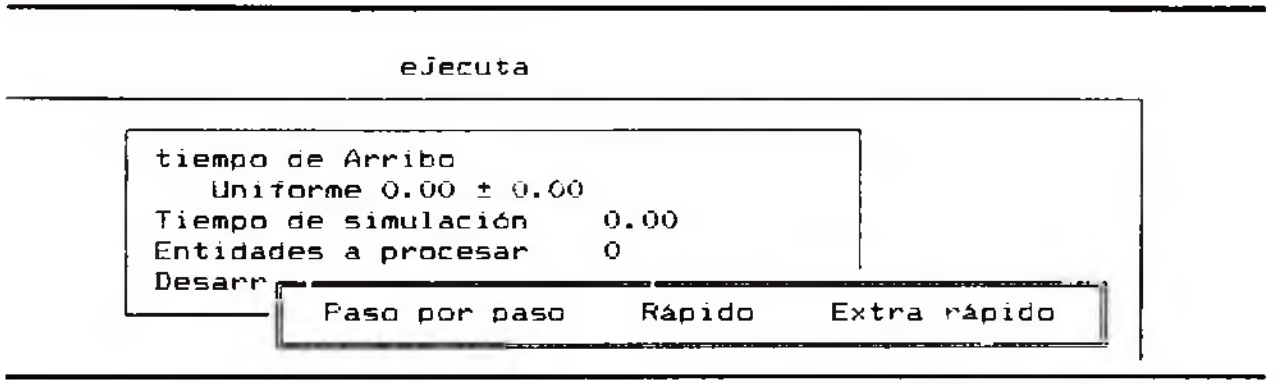


Figura 73. Menú de Desarrollo.

Paso por Paso.

Con esta opción se ejecuta la simulación paso por paso, mostrando un cambio en la simulación a la vez. Para continuar con el siguiente cambio en el sistema presione cualquier tecla. En este tipo de desarrollo se puede desplegar la ventana de reporte, mostrando un reporte estadístico de una actividad especifica. Los parámetros que se reportan será explicados en la siguiente sección de Menú Reporte.

Se puede interrumpir la simulación en cualquier momento con la tecla <F10>. Para las redes que son más grandes que una pantalla, la red se puede desplazar con las teclas de flechas, para observar los cambios en otros sectores de la red.

La ventana de reporte se puede activar con la tecla <F1>. Listando las estadísticas de la primera actividad, con las teclas <PgUp> y <PgDn> se puede retroceder o avanzar respectivamente de actividad (figura 74). Para desactivar la ventana de reporte tecleese <Esc>.

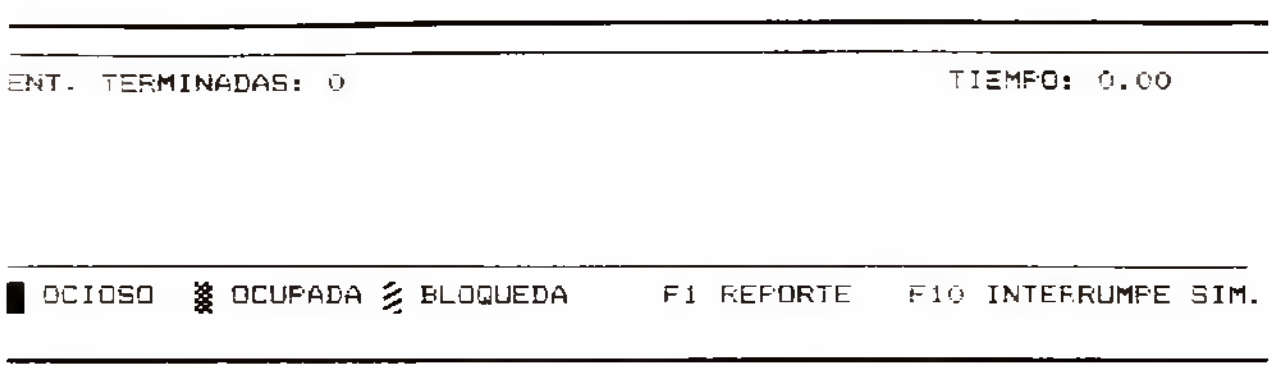


Figura 74. Pantalla de simulación.

Rápido.

Con este tipo de desarrollo unicamente se pueden observar los cambios de la simulación rápidamente. No se puede desplegar la ventana de reporte. Si la red de actividades ocupa más de una pantalla, la red se puede mover en la pantalla con las teclas de flechas. También se puede interrumpir la simulación con la tecla ‹F10›. Una vez interrumpida la simulación, esta no puede reestablecer.

Extra Rapido.

Con esta opción se ejecuta la simulación internamente, sin mostrar el desarrollo de la simulación en la red de actividades. Unicamente se observa como avanza el tiempo en la ventana que aparece al seleccionar esta opción. La simulación se puede interrumpir con la tecla ‹F10›.

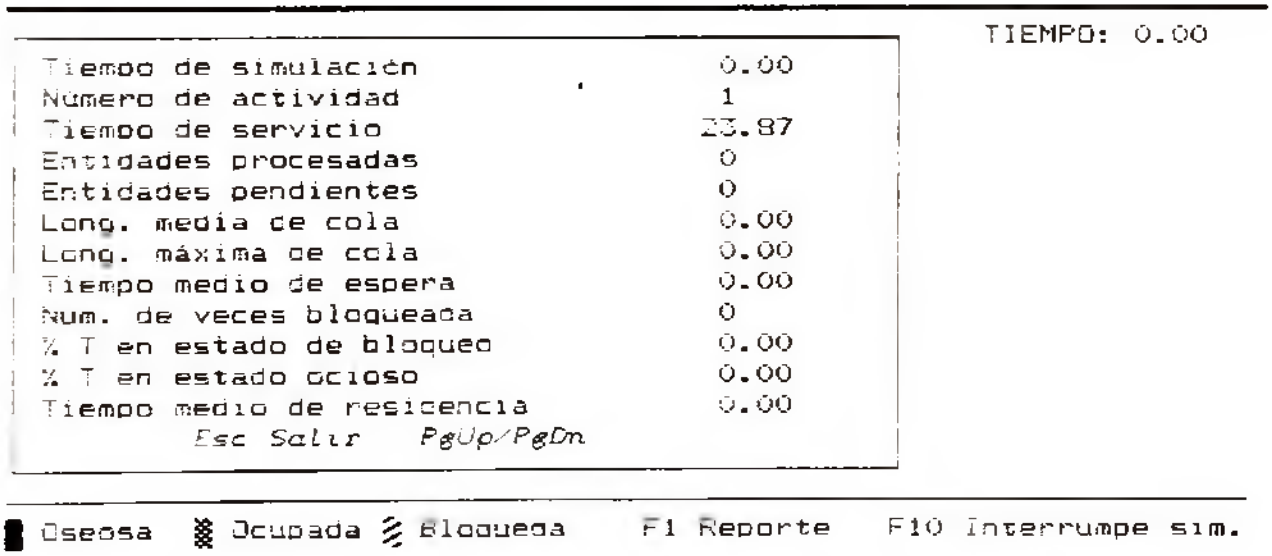


Figura 75. Reporte en la pantalla de simulacion.

MENU REPORTE.

Con este menú se pueden observar los resultados de la simulación tanto en su desarrollo como en resultados estadísticos. Ofreciendo las siguientes dos opciones Reporte general y Resumen histórico. Ambos reportes pueden ser desplegados en pantalla o ser enviados a la impresora.

Figura 76. Menú de reporte.

Reporte General.

En este tipo de reporte se muestran las estadísticas de cada actividad en la simulación, así como el tiempo de simulación y la cantidad de ordenes realizadas por la red. Los resultados que sean máximos en el parámetro estadístico serán señalados con el signo [+].

Se pueden mostrar los resultados de hasta cinco actividades a la vez, para redes mayores a cinco actividades se pueden observar los resultados de las démas actividades tecleando <→> o <←>, para desplazar las actividades de derecha a izquierda o viceversa. Para salir del reporte tecleese <Esc>.

Los parámetros estadísticos que se reportan para cada actividad son los mismos de la figura 75.

Resumen histórico.

En este tipo de reporte se muestra como se llevó a cabo la simulación con respecto a la variable tiempo. Mostrando por medio de barras en un diagrama la cantidad de tiempo que una actividad permaneció en los tres estados distintos; ocioso, trabajando o en estado de bloqueo, cada estado tiene un símbolo asociado; por ejemplo, si la actividad se encuentra trabajando ésta se representará por media de una barra blanca, con la letra de identificación de la entidad, si se encuentra bloqueada se representará por una barra punteada, pero si se encuentra ociosa no aparecerá ningún símbolo, solamente aparecerá un espacio en blanco.

Cada entidad cuenta con una letra minúscula que la identifica, estas letras son [a .. z], dependiendo del orden de llegada de las entidades es la letra que se le asigna, por ejemplo: a la primera entidad que entre al sistema se le asigna la letra [a], a la segunda entidad la letra [b], etc. Pero si la simulación es bastante larga y en el sistema arriban más de 26 entidades se reiniciará la asignación de letras a las entidades, es decir que la entidad número 27 llevará también como letra de identificación la [a], la entidad 28 la letra [b] y así sucesivamente.

El histórico puede ser definido por medio de intervalos de tiempo, es decir que se puede variar el intervalo de incrementos en el tiempo para poder observar mejor los cambios en las actividades. Es recomendable establecer un incremento de tiempo adecuado a la duración de la simulación, ya que un intervalo pequeño ocaciona bastante consumo de memoria, y por consiguiente su análisis puede llegar a ser tedioso, en cambio para incrementos bastante grandes, ocacionan poco consumo de memoria, pero con el inconveniente de no poder observar las diferencias de duración en tiempo adecuadas para cada una de las actividades.

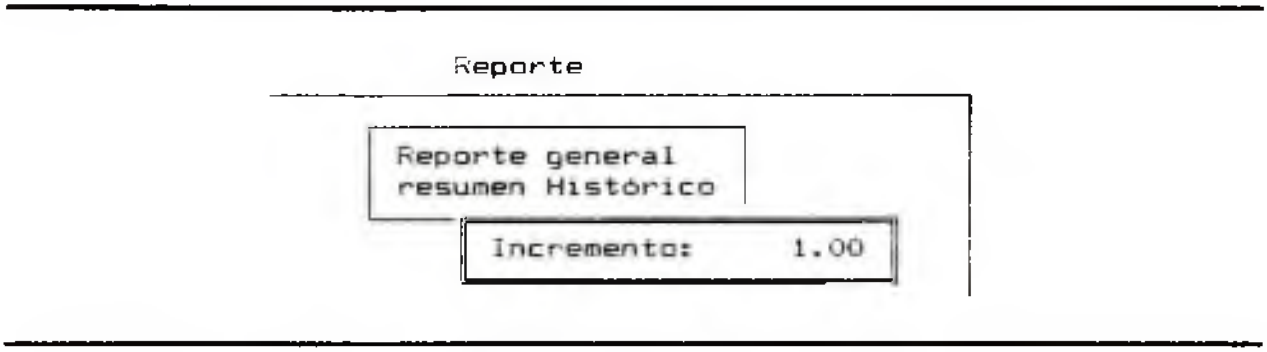


Figura 77. Menú del resumen histórico.

Aunque el algoritmo con el cual está diseñado este reporte, despliega el tiempo en el cual ocurrió un cambio en el sistema, aunque el tiempo no vaya acorde con el incremento de tiempo que previamente se haya establecido. Por lo que el incremento de tiempo que se pide se utiliza unicamente para disminuir el intervalo de tiempo cuando no existen cambios en el sistema durante un largo periodo.

El histórico se despliega desde el tiempo de inicio, y se puede avanzar a través del tiempo untilizando las teclas de flechas <Up> para avanzar y <Dn> para retroceder, cada vez que presiona una de estas dos flechas el histórico avanza o retrocede una unidad de tiempo según sea la tecla presionada.

Para salir del resumen histórico tecleese <Esc>, y regresará al menú de reporte. Una vez dejado el histórico este se pierde de la memoria, por lo que si se desea hacer otra consulta, se vuelve a generar otra vez, especificando el incremento de tiempo que se desee para el nuevo histórico.

El histórico es guardado en un archivo en disco de nombre *EVENTS.SIM*, el cual contiene la información del histórico en forma de numeros. Este archivo es actualizado para cada corrida de simulación que se efectúe, perdiéndose totalmente el histórico de la simulación anterior.

Debido a que el histórico se encuentra en disco, su única limitante es el espacio en este. La unidad de disco en el cual aparecerá el archivo, será aquel en el que se encuentre el simulador. Ya que el disco del sistema cuenta con suficiente espacio para el archivo. Por lo que es recomendable tener los archivos de datos o redes de actividades en una unidad de disco aparte, para evitar la saturación del disco del sistema.

RESUMEN DE ACTIVIDADES

Esta es la sección que nos muestra como está construida la red de actividades. Mostrando las caracteristicas de cada actividad, así como las interconexiones entre las actividades. La sección contiene las siguientes columnas: Actividad, Long. de cola, Distribución, Media, Varianza, y Actividades Sucesoras. Cada renglón corresponde a la definición de una actividad.

Para poder recorrer las actividades dentro de esta sección, es necesario accesar dicha sección, y se accesa presionando la tecla <F6>, desde cualquier menú del simulador. Para salir de la sección y regresar al menu previo, presione nuevamente la misma tecla.

Resumen de Actividades

EJEMPLO1

Actividad	Long. cola	Tiempo de servicio	Actividades sucesoras
1	3	Exponencial 5.98 ± 3.56	3
2	4	Uniforme 7.65 ± 4.79	3
3	0	Distr. de Frecuencia	4
4	2	Poisson 10.56 ± 4.21	5 6
5	0	Constante 14.78 ± 0.00	

Actividades de entrada: 1 2

↑ Retrocede ↓ Avanza F6 Menu previo F9 Imprime Resumen

Figura 78. Sección de Resumen de Actividades.

En esta sección únicamente se puede listar la red de actividades, no se puede cambiar la información, si se desea cambiar algún parametro de la actividad, se debe de hacer por medio del menú de edición. Debido a que la sección muestra hasta seis actividades, se pueden observar las demás actividades presionando las teclas <Up> o <Dn>, mostrándose las seis actividades previas o subsecuentes respectivamente.

Cuando una orden de trabajo llega a la red, ésta puede entrar al sistema mediante las actividades de entrada. Pueden existir de una hasta quince actividades de entrada. Las actividades de entrada son listadas dentro de la sección por el letrero *Actividades de Entrada*.

La columna Actividades sucesoras nos indica, las actividades que son subsecuentes a la actividad. Una actividad puede tener como máximo quince actividades sucesoras.

La red de actividades se puede imprimir cuando se accesa a la sección de Resumen de Actividades mediante la tecla <F9>, debe de estar encendida la impresora ya que el simulador no detecta si la impresora se encuentra preparada para imprimir. En caso de que la impresora no se encuentre conectada o encendida el simulador terminará su ejecución con un error.

A P E N D I C E C

PROGRAMA FUENTE DE SIMDISC

```
program simdisc;
uses graph, globals, nucleo, menus, auxproc;
{$i graphics.pas}
{$i initvars.pas}

procedure editmanag;
var
 flag, flag1, flag2: boolean;
 option, opt, col: byte;
 nodenum: word;
 edit_node: ptr_node;
begin
  act_wind (14, 3, 50, 11, DOUBLE);  { display the window area }
  flag2 := false;           { flag for machine number especification }
  col := 23;
  repeat
    flag := false;
    option := runmenu (menu3, VERTMENU);  { manage the edit menu }
    if option = $ff then helpmanag (3)     { if user press F1 }
    else if not (flag2) and (option > 1) then begin
       note_mess;
       option := 10;          { the node number }
    end;
    case option of
     0: flag := true;
     1: begin                { read the node to edit }
        nodenum := 0;
        readint (col, 1, nodenum, flag1);
        if flag1 and (nodenum <> 0) then begin
         flag_save := false;
         edit_node := find (nodenum, lineprod.nextn);
         if edit_node = nil then createnode (edit_node, nodenum);
         impnode (edit_node);     { display the activity parameters }
         flag2 := true;
        end;
        end;
     2: begin                              { read max queue capacity }
              nodenum := edit_node^.locap;
              readint (col, 2, nodenum, flag1);
              edit_node^.locap := nodenum;
              disp1_aux (edit_node);
         end;
     3: begin                           { service time especification }
         act_wind (35, 7, 58, 10, SINGLE);
         opt := runmenu (menu9, VERTMENU); { display the prob/frec }
         case opt of
           1: read_prob (edit_node);
           2: begin
               read_frec (edit_node^.timef, edit_node^.frec);
               edit_node^.flag_frec := true;
              end;
         end;
```

```
          kill_wind;          { close the window }
          printc (3, 4, '                                ', NORM and $f0);
          if edit_node^.flag_frec then begin
             printc (3, 4, 'Dist. de frecuencias', NORM and $f0);
          end
          else begin
               s2 := distribution_type [edit_node^.distr_type];
               s2 := s2 + '  :
               str (edit_node^.serv_time:1:2, s1);
               s2 := s2 + ' ' + s1 + '    + chr (241) + '  :
               str (edit_node^.variance:1:2, s1);
               s2 := s2 + s1;
               printc (3, 4, s2, NORM and $f0);
          end;
          displ_aux (edit_node);
        end;
      4: begin                        { read the previos activity }
               nodenum := 0;
               readint (col, 5, nodenum, flag1);
               if flag1 then begin
                  make_graph (edit_node, nodenum);
                  display_new_node;
               end;
        end;
      5: begin          { locates the activity in the screen graphics }
           move (ptr  ($b800,  0)^,  menuscr^,  4000);
           str (edit_node^.identif, s2);
           with edit_node^ do locate (x, y, s2);
           move (menuscr^, ptr ($b800, 0)^, 4000);
        end;
      6: begin          { delete activity or link }
           act_wind (24, 9, 45, 11, SINGLE);
           opt := runmenu (menu12, HORMENU);
           case opt of
                1: del_node (edit_node, flag2);
                2: del_link (edit_node);
           end;
           kill_wind;
        end;
  end;
  until flag;
  kill_wind;
end;

procedure runmanag;                { manage the simulation menu }
var
  flag, flag1: boolean;
  col, option: byte;
begin
  col := 24;
  flag := false;
  cond_gen_perf := false;
  act_wind (30, 3, 64, 9, SINGLE);
```

```
repeat
 if not (flag_jobs) then processjobs := 0;
 case type_arrival of    { display the arrival type interarrival }
    0: begin                    { Probabilistic distr. }
        s2 := distribution_type (dtiat);
        str (iat:1:2, s1);
        s2 := s2 + '  ' + s1 + '  ' + chr (241) + '  ';
        str (viat:1:2, s1);
        s2 := s2 + s1;
       end;
    1: s2 := 'Dist. de frecuencias';
    2: begin                          { conditional arrive }
        s2 := 'Arribo cond. por actividad ';
        str (cond_mach, s1); s2 := s2 + s1;
       end;
 end;
 printc (4, 2,                                  , NORM);
 printc (4, 3, s2, NORM and $f0);
 printc (col, 3,              , NORM);
 printc (col, 4,              , NORM);
 printfloat (col, 3, simduration, NORM and $f0);
 printint (col, 4, processjobs, NORM and $f0);
 option := runmenu (menu4, VERTMENU);
 case option of
      0: flag := true;
      1: read_simparam;              { read arrival time parameters }
      2: begin                       { read simulation time }
          readfloat (col, 3, simduration, flag1);
          if flag1 then flag_jobs := false;
         end;
      3: begin                    { read entities number to process}
           readint (col, 4, processjobs, flag1);
           if flag1 then flag_jobs := true;
         end;
      4: begin                    { type to performance }
           act_wind (38, 8, 78, 10, DOUBLE);
           option := runmenu (menu5, NORMENU);
           kill_wind;
           if option <> 0 then begin
             type_perform := option;
             if lineprod.sons [0] = 0 then begin
                 act_wind (30, 11, 69, 13, SINGLE);
                 s1 := 'Error No existe linea de produccion. ';
                 printc (2, 1, s1, INVERT or $80);
              print_note (55, NORM,  'presione cualquier tecla ');
                 kill_wind;
                 kill_wind;
                 exit;
              end;
              move (ptr ($b800, 0)^, menuscr^, 4000);
              if flag_jobs then simduration := $ffff;
```

```
                    if type_perform = 3 then begin { extra fast perform }
                      clr_mess;
                      print_mess (5, CAP, 'F10');
                      print_mess (10, NORM, 'Interrumpe simulacion');
                      act_wind (42, 9, 65, 11, DOUBLE);
                      printc (6, 0, '  SIMULANDO  ', INVERT or $80);
                      printc (2, 1, 'Tiempo:', NORM);
                      initialize;
                      kill_wind;
                      clr_mess; messline;
                    end
                    else run_process;          { slow and fast perform }
                    move (menuscr^, ptr ($b800, 0)^, 4000);
                  end;
                end;
          $ff: helpmanag (4);
        end;
    until flag;
    kill_wind;
end;

procedure mainmanag;
var
  option: byte;
begin
  repeat
     option := runmenu (menu1, HORMENU);      { manage the main menu }
      case option of
       1: filemanag;                          { call file window managment }
       2: editmanag;                          { call edit window managment }
       3: runmanag;                           { call run window managment }
       4: reportmanag;                    { call report window managment }
       5: begin                           { Quit }
            save_test;                    { check if the graph has been }
            exit;                         { saved }
          end;
     $ff: helpmanag (1);                  { user request help window }
     end;
  until false;
end;

begin
 init_globals;              { initialize global variables }
 getdir (0, drive);         { set system drive }
 if not ((drive = 'A:\') or (drive = 'B:\')) then drive := drive + '\';
 drive_system := drive;
 main_menu;                 { draw the main menu screen }
 mainmanag;                 { call the principal menu }
 dispose_globals;           { deallocates the global variables }
 fillchar (ptr ($b800, 0)^, 4000, 0);     { clears the screen }
 printc (1, 1, 'Teclee CLS <enter> para obtener el cursor. ', $09);
 drive_system := copy (drive_system, 1, length (drive_system) - 1);
 chdir (drive_system);
end.
```

```
unit auxproc;

interface

uses globals, menus, dos, printer;

type

  ptr_hist = ^hist_descr;

  hist_descr = record                    { descriptor for history resume }
                 prevh: ptr_hist;        { each node corresponds a one row

                 nexth: ptr_hist;        { in the history }
                 clock: array [0..17] of byte;
                 buffer: array [0..129] of byte;
               end;

var
    list_hist, tail_hist: ptr_hist;      { list for history rows }
    flag_save: boolean;                  { flag for file protect }
    flag_name: boolean;                  { flag for name especification

    drive: string;                       { directory status }
    namearch: text;

procedure impnode (p:ptr_node);
procedure make_graph (p: ptr_node; n: byte);
procedure del_link (p: ptr_node);
procedure del_node (p: ptr_node; var flag2: boolean);
procedure note_mess;
procedure read_prob (edit_node: ptr_node);
procedure read_frec (var time, frec: frec_table);
procedure save_files;
procedure save_test;
procedure createnode (var p: ptr_node; num: byte);
procedure helpmanag (index: byte);
procedure filemanag;
procedure reportmanag;
procedure main_menu;

implementation

procedure filemanag;                     { display the file window and }
var                                      { manages the file operations }
  option: byte;
  flag: boolean;
begin
  act_wind (3, 3, 21, 9, DOUBLE);
  repeat
    flag := false;
    option := runmenu (menu2, VERTMENU);
```

```
    case option of
      0: flag := true;
      1: loadarcn;              { load activities net }
      2: newarcn;               { clear the simulator memory }
      3: savearcn;              { save activities net }
      4: directory;             { display disk directory }
      5: changedir;             { change work directory }
    $ff: helpmanag (2);         { user press F1 }
    end;
  until flag;
  kill_wind;
end;

procedure reportmanag;          { manage the report menu }
var
  option: byte;
  flag, flag1, flag2: boolean;
  report_type, device_type: byte;
  step_time: real;
  s : string;
begin
  act_wind (40, 2, 60, 6, SINGLE);
  repeat
    flag := true;
    option := runmenu (menu6, VERTMENU);
    if lineproc.sons [0] = 0 then begin
       act_wind (30, 11, 69, 13, SINGLE);
       s := 'Error: No existe linea de produccion. ';
       printc (2, 1, s, INVERT or $80);
       print_note (55, NORM,  presione cualquier tecla');
       flag := false;
       kill_wind;
       kill_wind;
       exit;
    end;
    flag2 := true;
    case option of
       0: flag := false;                              { EXIT }
       1: report_type := 1;                           { General Report }
       2: begin                                       { history resume }
            report_type := 2;
            step_time := 10.0;
            act_wind (45, 5, 75, 8, DOUBLE);
            printc (2, 1,  Incremento: , NORM); { read step time }
            readfloat (18, 1, step_time, flag2);
            kill_wind;
          end;
     $ff: begin                                       { user press F1 }
            helpmanag  3);
            flag2 := false;
          end;
    end;
```

```
      if flag and flag2 then begin
       act_wind (45, 6, 70, 8, DOUBLE);
       repeat
         option := runmenu (menu7, HORMENU);    { select device output }
         flag1 := true;
         case option of
           0: flag1 := false;
           1: device_type := 1;                  { select screen output }
           2: device_type := 2;                  { select printer output}
         $ff: device_type := 0;
         end;
         if flag1 then begin
           if (device_type = 1) and (report_type = 1) then
                report_screen;
           if (device_type = 1) and (report_type = 2) then
                display_history (step_time);
           if (device_type = 2) then begin
             act_wind (50, 3, 65, 10, SINGLE);
             printc (2, 1, ' IMPRIMIENDO ', INVERT or $80);
             if report_type = 1 then report_printer;
             if report_type = 2 then print_history (step_time);
             kill_wind;
           end;
         end;
       until not (flag1);
       kill_wind;
      end;
   until not (flag);
   kill_wind;
end;

procedure report_printer;
var
  p, pinit, plast, q: ptr_node;
  counter, cr_line: byte;
  flag: boolean;

procedure print (q, plast: ptr_node; index: byte);
var
  symbol: string;
begin
   while q <> plast do begin
     gen_report (q);
     symbol := '  ';
     if q^.identif = max [index] then symbol :=    + ;
     write (lst, symbol, statment[index], ' ');
     q := q^.nextn;
   end;
   writeln (lst);
end;
```

```
begin
 p := lineprod.nextn;
 maximum (p);
 pinit := p;
 writeln (lst,                                ***'+ rep_statment [0] +  ***');
 writeln (lst);
 writeln (lst,    *** PARAMETROS GENERALES DEL SISTEMA *** );
 writeln (lst);
 writeln (lst,      + rep_statment [12], endedjobs);
 writeln (lst,      + rep_statment [14], lose_job);
 writeln (lst,      + rep_statment [13], time:7:2);
 if cond_gen_perf then begin
     str (cond_mach, s2);
     s2 :=    Arribo cond. por actividad:  + s2;
 end
 else begin
     s2 := distribution_type (dtiat);
     str (iat:1:2, s1);
     s2 := s2 +        + s1 +  --  ;
     str (viat:1:2, s1);
     s2 :=    Tiempo de arribo entre entidades:  + s2 + s1;
 end;
 writeln (lst, s2);
 writeln (lst);
 writeln (lst);
 flag := true;
 while pinit <> nil do begin
   plast := pinit;
   counter := 0;
   s2 := '+------------------------------+';
   while (counter <= 2) and (plast <> nil) do begin
      s2 := s2 + '-----------+';
      inc (counter, 1);
      plast := plast^.nextn;
   end;
   writeln (lst,    *** REPORTE DE PARAMETROS POR ACTIVIDAD ***);
   writeln (lst, s2);
   for counter := 1 to 11 do begin
      write (lst, '|', rep_statment [counter], '|');
      print (pinit, plast, counter);
      writeln (lst, s2);
   end;
   cr_line := 8;
   if flag then begin
      cr_line := 3;
      flag := false;
   end;
   for counter := 1 to cr_line do
      writeln (lst);
   pinit := plast;
 end;
end;
```

```
procedure print_history (step_time: real);
var
  p: ptr_node;
  mach, st, symbol, job, nrows: byte;
  taux, time: real;
  flag: boolean;
  linehorz, row, timestr: string;

procedure make_rows (var row1, row2, horz: string);
var
  s, s3: byte;
  p: ptr_node;
begin
  p := lineprod.nextn;
  row1 := '';  row2 := '';
  horz := '+';
  while p <> nil do begin
    s := p^.identif div 10 + 48;
    s3 := p^.identif mod 10 + 48;
    if s = 48 then s := 32;
    row1 := row1 + chr (s) +    ;
    row2 := row2 + chr (s3) + ' ';
    horz := horz + '--';
    p^.state := 0;
    p := p^.nextn;
  end;
  horz := horz + '+';
end;

procedure history_head;
begin
  writeln (lst, '          *** RESUMEN HISTORICO ***');
  writeln (lst);
  writeln (lst, '        #.- Bloqueo       [a,z].- Ocupada ');
  writeln (lst);
  writeln (lst,                    ACTIVIDADES ');
  writeln (lst,                    ' + s1);
  writeln (lst, '                  ' + s2);
  writeln (lst, '   TIEMPO         ' + linehorz);
end;

begin
  assign (file_event, drive_system + 'events.sim');
  reset (file_event);  make_rows (s1, s2, linehorz);
  history_head;  nrows := 9;
  readln (file_event, time, mach, st, job);
  flag := false;
```

```
  repeat
    taux := time;  str (time:9:2, timestr);
    row := '      ' + timestr +     |';
    repeat
      p := find (mach, lineprod.nextn);
      p^.state := st;
      p^.lq := job;
      readln (file_event, time, mach, st, job);
      if mach = 0 then flag := true;
    until (taux <> time) or flag;
    if flag then time := taux + 1;
    while taux < time do begin
      p := lineprod.nextn;
      while p <> nil do begin
        case p^.state of
           0: symbol := 32;
           1: symbol := p^.lq mod 26 + 96;
           2: symbol := 35;
        end;
        row := row + chr (symbol) +  ';
        p := p^.nextn;
      end;
      taux := taux + step_time;
      writeln (lst, row + '|');
      inc (nrows, 1);
      if nrows > 60 then begin
        writeln (lst,'                    ' + linehorz);
        for nrows := 0 to 4 do writeln (lst);
        history_head;
        nrows := 9;
      end;
      if taux < time then row := '                    |';
    end;
  until flag;
  writeln (lst, '                    + linehorz);
  close (file_event);
end;
```

```
{ ***************************************************************
                GRAPHICS MANAGMENT AND SIMULATION PRIMITIVIES

                              FILE GRAPHICS.PAS
  ***************************************************************

const
STACK_SIZE_COROUT = 3000;
       MAX_COROUT = 49;

type
  sprite = record                    { sprite descriptor and especificati
n }
             xsize, ysize: byte;
             image: array [0..77] of byte;  { 6 bytes x 13 bytes }
           end;

var
    blocklist: ptr_descr:                { list of block's jobs
  }
    jobs_descr: array [0..MAX_COROUT] of descr:{ array for jobs manipul
tion}
    main_descr: descr;
    corout_avail: byte;                    { coroutines avail }
    reporti, backrepi: ptr_word;         { buffer for report and help win
ows }
    blockw, blockb, blocki: sprite;    { block state sprite }
    jobsprite, backjob: sprite;          { job representation sprite }
    running, main, timelist: ptr_descr;  { vars for simulation }
    simduration: real;
    screenx, screeny:integer;
    xreal, yreal: word;
    t: byte;
    next_gen, prev_t: real;
    gen_job: boolean;

function graphinit: boolean;      { initialize the graph mode 320 x 200


var
 error, grdrv, gm: integer;
 s: string;
begin
  grdrv := detect;
  initgraph (grdrv, gm, drive_system);
  error := graphresult;
  if error = grok then begin
      graphinit := true;
      setgraphmode (cgac2);
  end
  else begin                                  { graphics error found }
    graphinit := false;
    str (error, s);
    s := 'Error de graficos... error numero: ' + s;
    s := s + '   presione cualquier tecla ';
    print_note (10, NORM or $80, s);
  end;
end;
```

```
{ ************************************************************************
                  procedures for simulation primitives
  ************************************************************************
}

procedure insertf (pd: ptr_descr; var l: ptr_descr);
begin
  pd^.next := l;
  l := pd;
end;

procedure append (pd: ptr_descr; var l: ptr_descr);
var
 p: ptr_descr;
begin
 if l = nil then l := pd
 else begin
    p := l;
    while p^.next <> nil do
        p := p^.next;
    p^.next := pd;
 end;
end;

procedure remove (var l, pd:ptr_descr);
var
 p: ptr_descr;
begin
 if l <> nil then begin
       pd := l;
       l := l^.next;
       pd^.next := nil;
 end
 else pd := nil;
end;

procedure act_state (p: ptr_node);
var
  snum: string;
  xaux, yaux: integer;
begin
 if type_perform = 3 then exit;
 xaux := p^.x - xreal; yaux := p^.y - yreal;
 if (xaux < 0) or (yaux < 9) then exit;
 if (xaux >= 310) or (yaux >= 170) then exit;
 case p^.state of
           0: putsprite (xaux, yaux, blocki);
           1: putsprite (xaux, yaux, blockw);
           2: putsprite (xaux, yaux, blockb);
 end;
 setcolor (0); str (p^.identif, snum);
 outtextxy (xaux + 5, yaux + 5, snum);
end;
```

```
procedure act_queue (p: ptr_node);
var
  yaux, xaux, x1, y1: integer;
  i: byte;
begin
  if type_perform = 3 then exit;
 xaux := p^.x - xreal;
 yaux := p^.y - yreal;
 if (xaux < 0) or (yaux < 9) then exit;
 if (xaux >= 310) or (yaux >= 170) then exit;
 x1 := xaux + 14;
 y1 := yaux - 5;
 for i := 1 to p^.lq + 1 do begin
     putsprite (x1, y1, jobsprite);
     if i > p^.lq then putsprite (x1, y1, backjob);
     dec (x1, 8);
     if x1 <= xaux - 4 then begin
       x1 := xaux + 14;
       dec (y1, 6);
       if y1 < 9 then exit;
     end;  { if }
 end;  { for }
end;

procedure disp_time;
var
  i: byte;
  offset: word;
  snum: string;
begin
 for i := 0 to 6 do begin
     offset := $2000 * (i and 1) + 80 * (i shr 1);
     fillchar (ptr ($b800, offset + 63)^, 17, 0);
     fillchar (ptr ($b800, offset + 31)^, 10, 0);
 end;
 setcolor (1);
 str (time:1:2, snum);
 outtextxy (255, 0, snum);
 str (endedjobs, snum);
 outtextxy (125, 0, snum);
end;

procedure  delete_zone (x1, y1: word; color: byte);
var
  i, offset: word;
begin
    for i := y1 to y1 + 6 do begin
       offset := $2000 * (i and 1) + 80 * (i shr 1) + (x1 shr 2);
       fillchar (ptr ($b800, offset)^, 13, color);
    end;
end;
```

```
procedure control_table;    { display the report graphics window }
var
  i: byte;
  offset, j, ofsb, segb, ofsr, segr: word;
  p, curr: ptr_node;
  s: string;
  f: file;
begin
  segr := seg (report1^);
  ofsr := ofs (report1^);
  segb := seg (backrep1^);
  ofsb := ofs (backrep1^);
  assign (f, drive_system +  arch5.sim');
  reset (f, 1);
  blockread (f, report1^, 8680);
  close (f);
  j := 0;
  for i := 0 to 123 do begin              { display the report window }
      offset := $2000 * (i and 1) + 80 * (i shr 1);
      move (ptr ($b800, offset)^, ptr (segb, ofsb + j)^, 70);
      move (ptr (segr, ofsr + j)^, ptr ($b800, offset)^, 70);
      inc (j, 70);
  end;
  setcolor (0); str (time:8:2, s); outtextxy (210, 4, s);
  curr := lineprod.nextn;
  repeat
      for i := 1 to 11 do
         delete_zone (220, 4 + i * 9, $55);
      delete_zone (205, 22, $55);
      gen_report (curr);
      for i := 0 to 10 do
          outtextxy (210, 13 + i * 9, statment [i + 1]);
      getchar (carac);
      if ord (carac) = 0 then getchar (carac);
      case ord (carac) of
         PGDN: begin
                 p := lineprod.nextn;
                 if p <> curr then
                   while p^.nextn <> curr do
                     p := p^.nextn;
                 curr := p;
               end;
         PGUP: if curr^.nextn <> nil then curr := curr^.nextn;
      end;
   until ord (carac) = ESC;
   j := 0;                                { turn_off the window report }
   for i := 0 to 123 do begin
          offset := $2000 * (i and 1) + 80 * (i shr 1);
          move (ptr (segb, ofsb + j)^, ptr ($b800, offset)^, 70);
          inc (j, 70);
   end;
end;
```

```
procedure control_process_screen (i, op: byte);
var                 { move the screen graphics, if the user request in }
 p: ptr_node;       { simulation mode }
begin
 case i of
    UP_ARROW: begin
                dec (screeny, 9); dec (yreal, 18);
                if screeny < 0 then begin
                   screeny := 0; yreal := 0;
                end;
                move_screen (screenx, screeny);
              end;
 RIGHT_ARROW: begin
                inc (screenx, 10);
                inc (xreal, 40);
                if screenx > 40 then begin
                    screenx := 40;
                    xreal := 160;
                end;
                move_screen (screenx, screeny);
              end;
  DOWN_ARROW: begin
                inc (screeny, 9);
                inc (yreal, 18);
                if screeny > 45 then begin
                   screeny := 45; yreal := 90;
                end;
                move_screen (screenx, screeny);
              end;
  LEFT_ARROW: begin
                dec (screenx, 10);
                dec (xreal, 40);
                if screenx < 0 then begin
                 screenx := 0; xreal := 0;
                end;
                move_screen (screenx, screeny);
              end;
          F1: if op = 1 then control_table;
         F10: begin
                gotoxy (1, 22);
                writeln ('Simulacion Interrumpida. Presione Esc. ');
                getchar (caracter);
                transfer (main^.corout);
              end;
 end;  {case}
 p := lineprod.nextn;          { draw the activities state and queues }
 while p <> nil do begin       { after screen has been moved }
    act_state (p);
    act_queue (p);
    p := p^.nextn;
 end;
end;
```

```
procedure debuger;          { manage the animation in simulation mode }
begin
     if type_perform <> 3 then diso_time
     else begin
       gotoxy (51, 9); write (time:1:2);
     end;
     case type_perform of
       1: begin                              { step by step mode }
            getchar (carac);
            if ord (carac) = 0 then begin
               getchar (carac);
               control_process_screen (ord (carac), 1);
            end;
          end;
       2: if keypressed then begin          { fast mode }
             getchar (carac);
             if ord (carac) = 0 then begin
                getchar (carac);
                control_process_screen (ord (carac), 2);
             end;
          end;   { 2 }
       3: if keypressed then begin           { extra fast mode }
             getchar (carac);
             if ord (carac) = 0 then getchar (carac);
             if ord (carac) = F10 then transfer (main^.corout);
          end;
     end;    { case }
end;

procedure schedule;                  { find the next event to transfer }
begin
  remove (timelist, running);
  if running = nil then transfer (main^.corout);
  prev_t := time; time := running^.key; debuger;
end;

procedure hold (dt: real);            { put the event in the time list }
 var l: ptr_descr;                   { and locates the event in the next}
begin                                 { time to reactivate it }
  running^.key := time + dt;
  if timelist = nil then begin       { insert running in the timelist }
     timelist := running;            { with time ascending order }
     running^.next := nil;
  end;
  if running^.key < timelist^.key then begin  { if its the lowest }
      running^.next  := timelist;             { reactivation time }
      timelist := running;
  end
  else begin                           { insert in time order }
     l := timelist;
     while (l^.next <> nil) and (running^.key >= l^.next^.key) do
             l := l^.next;
```

```
      running^.next := l^.next;
      l^.next := running;
   end;
   schedule;
   transfer (running^.corout);
end;

procedure wait (var l: ptr_descr);  { insert the event in wait list }
begin
   append (running, l);
   schedule;
   transfer (running^.corout);
end;

procedure signal (var l: ptr_descr);       { substract from the list }
var                                    { the next event to activate }
   pd: ptr_descr;
begin
   remove (l, pd);
   if pd <> nil then begin
        insertf (running, timelist);
        running := pd;
        running^.key := time;
        transfer (running^.corout);
    end;
 end;

procedure activate (p: ptr_word; s:string);  { initialize the event }
var
   index: byte;
begin
   index := 0;    { find the next posible stack in descriptors array }
   while (index <= corout_avail) and (jobs_descr [index].flag_busy) do
        inc (index, 1);
   if index > corout_avail then begin
         s := 'No existe suficiente memoria para la simulacion ;
         if type_perform <> 3 then begin
           writeln (s);
           writeln ('La simulacion es interrumpida.');
         end
         else begin
            gotoxy (2, 24);
            write (s + ' Presione Esc');
            getchar (carac);
         end;
         transfer (main^.corout);
   end;
   with jobs_descr [index] do begin
      corout := newprocess (p, init_ptr, STACK_SIZE_COROUT);
      flag_busy := true;
      sym := s;
   end;
```

```
    insertf (running, timelist);
    running := @jobs_descr [index];
    transfer (running^.corout);
    end;

  procedure terminate;                          { terminate the event life }
  begin
    running^.flag_busy := false;
    schedule;
    transfer (running^.corout);
  end;

  procedure startsimulation;                     { initialize the global varia
bles }
  begin                                          { for simulation }
    main := @main_descr;
    main^.next := nil; main^.sym := 'm'; running := main;
    initnucleus (running^.corout);
    time := 0; timelist := nil; blocklist := nil;
    main^.key := simduration + 0.1; endedjobs := 0;
  end;

  procedure order (var next: range_entrys);
  var     { order the next posibles activities in ascending order }
    n, i, j: byte;                                { state + length_queue }
    q1, q2: ptr_node; exp1, exp2: shortint;
  begin
    n := 0;
    while next [n] <> 0 do
        inc (n, 1);
    if n > 0 then begin                { if the node has next machines }
     dec (n, 1);
     for i := 0 to n - 1 do begin
         j := n;
         while j > i do begin
            q1 := find (next [j], lineprod.nextn);
            q2 := find (next [j - 1], lineprod.nextn);
            exp1 := q1^.lq + q1^.state ;
            if exp1 < exp2 then
                   swapbyte (next [j - 1], next [j]);
            dec (j, 1);
         end;
     end;
    end;
  end;
```

```
function gatefree (p: node): ptr_node;  { locate the entry activity }
var                                     { to accept the new event }
  i: byte; flag: boolean: q: ptr_node;
  nextmachs: range_entrys;
begin
  if p.sons[0] = 0 then q := nil
  else begin
    nextmachs := p.sons;
    order (nextmachs);
    flag := true;
    i := 0;          { keeps the index for the next posible machine }
    while (nextmachs [i] <> 0) and flag do begin
      q := find (nextmachs [i], lineprod.nextn);
      if (q^.state = 0) or ((q^.lqcap - q^.lq) > 0) then
         flag := false;
      inc (i, 1);
    end;
    if not (flag) then
       q := find (nextmachs [i - 1], lineprod.nextn) { gate found }
    else q := nil;                  { all gates posibles are busy and }
  end;                              { buffer length full }
  gatefree := q;
end;

procedure releasej (var curr: ptr_node; jid: byte);
var                       { release the event from the activity and }
  next: ptr_node;       { find the next posible activity and finally }
  flag: boolean;        { check to reactivate the blocked events     }
  p: ptr_descr;
  n, i: byte;
begin
  curr^.state := 0;                  { leave the current machine }
  write (file_event, time, ' ', curr^.identif);
  writeln (file_event, ' ', curr^.state, ' ', jid);
  act_state (curr);
  inc (curr^.jobout, 1);
  curr^.tlastidle := time;
  if curr^.sons[0] <> 0 then begin  { test if its an ended machine }
    next := gatefree (curr^);       { looks for the next machines }
    if next = nil then begin        { the machine start blocked }
        inc (curr^.n_block, 1);
        curr^.state := 2;
        act_state (curr);
        write (file_event, time, ' ', curr^.identif);
        writeln (file_event, ' ', curr^.state, ' ', jid);
        curr^.tlastblock := time;
        repeat                  { blocked until next posible machine }
           wait (blocklist);  { be idle or can enter in buffer list }
           next := gatefree (curr^);
        until (next <> nil) and not (next^.prev_hold);
```

```
        next^.prev_hold := true;
        curr^.tblock := curr^.tblock + (time - curr^.tlastblock);
        curr^.tidle := curr^.tidle + (time - curr^.tlastblock);
        curr^.state := 0;
        write (file_event, time, '  ', curr^.identif);
        writeln (file_event, '  ', curr^.state, '  ', jid);
        act_state (curr);
      end;
    end
    else next := nil;
    curr^.twait := curr^.twait + (time - curr^.tinput);
    with curr^ do
      if lq > 0 then begin
        timelq [lq] := timelq [lq] + (time - tprevlq);
        tprevlq := time;
        dec (lq, 1);
        act_queue (curr);
        signal (queue);
      end;
    p := blocklist;
    n := 0;                          { n keep the length of the list }
    while p <> nil do begin
      inc (n , 1); p := p^.next;
    end;
    i := 1;                           { test if jobs can continue }
    while i <= n do begin
      signal (blocklist); inc (i, 1);
    end;
    curr := next;      { curr points to the next activity }
end;

procedure request (curr: ptr_node; jid: byte);
begin                        { the event request service to the activity }
    curr^.prev_hold := false; inc (curr^.jobin, 1);
    if curr^.state > 0 then begin
       with curr^ do
         timelq [lq] := timelq [lq] + (time - tprevlq);
       inc (curr^.lq, 1); act_queue (curr); curr^.tprevlq := time;
       if curr^.lq > curr^.lqmax then      {keeps the max queue length}
         curr^.lqmax := curr^.lq;
       wait (curr^.queue);
    end                             { if machine is idle go to work the job}
    else curr^.tidle := curr^.tidle + (time - curr^.tlastidle);
    curr^.state := 1;
    write (file_event, time, '  ', curr^.identif, '  ');
    writeln (file_event, curr^.state, '  ', jid);
    act_state (curr);
end;
```

```
function rnd_frec (var time, frec: frec_table): real;
var                              { calculates the activity service time }
  i: byte;                       { from frecuency table }
  r: real;
begin
  r := rnd (0, 50, 50);  i := 0;
  while r > frec [i] do
    inc (i, 1);
  rnd_frec := time [i];
end;

procedure job;                          { event algoritm }
var
  jid: byte;  st: real;  curr: ptr_node;
begin
  jid := t;
  curr := gatefree (lineprod);    { verify if there are an activity }
                                  { to get in to the system }
  if curr = nil then begin
     terminate;                   { if not disponible activity then }
     inc (lose_job, 1);          { lose the job }
    end;
  while curr <> nil do begin
     curr^.tinput := time; request (curr, jid);
     curr^.twq := curr^.twq + (time - curr^.tinput);
     inc (curr^.nwq, 1);
     with curr^ do
       if flag_frec then st := rnd_frec (timef, frec)
       else st := rnd (distr_type, serv_time, variance);
     next_gen := st; hold (st);
     if type_arrival = 2 then
       if curr^.identif = cond_mach then gen_job := true
       else gen_job := false;
     releasej (curr, jid);
  end;
  inc (endedjobs, 1);  terminate;
end;

procedure generate;          { generates the events for frecuency and }
var                  { probabilistic distribution inter arrival time }
  dt: real;
  njob: word;
  sjob: string;
begin
  njob := 0;
  repeat
     njob := njob + 1; t := njob; str (njob, sjob);
     activate (@job,  j:' + sjob);
     if type_arrival = 0 then dt := rnd (dtiat, iat, viat)
     else dt := rnd_frec (arr_time, arr_frec);
     if flag_jobs and (njob + 1 > processjobs) then terminate;
     hold (dt);
  until false; end;
```

```
procedure cond_gen;    { generates the events for conditional arrive }
var
  dt: real;
  njob: byte;
  sjob: string;
begin
  njob := 0;
  repeat
     if gen_job then begin
       njob := njob + 1;
       t := njob;
       str (njob, sjob);
       activate (@job, 'j:'+ sjob);
     end;
     dt := next_gen;
     if flag_jobs and (njob + 1 > processjobs) then terminate;
     hold (dt);
  until false;
end;

procedure initialize;          { initialize the activitie's variables}
var                            { for simulation and request the memory}
  q: ptr_node;                 { for coroutines}
  index: byte;  sum: real;  ptr_mem: ptr_word;
begin
 gen_job := true; lose_job := 0;
 q := lineprod.nextn;
 while q <> nil do begin
   with q^ do begin
     queue:= nil; lq := 0; lqmax := 0; state := 0; tblock := 0;
     tlastblock := 0; twait := 0; tidle := 0; tlastidle := 0;
     n_block := 0; jobout := 0; jobin := 0; nwq := 0; twq := 0;
     prev_hold := false; tprevlq := 0;
     for index := 0 to 15 do timelq [index] := 0;
   end;
   q := q^.nextn;
 end;
 mark (ptr_mem);
 index := 0;
 while (index <= MAX_COROUT) and (memavail - 150 > STACK_SIZE_COROUT) d
1 begin
   with jobs_descr [index] do begin
      next := nil;
      key := 0;
      flag_busy := false;
      getmem (init_ptr, STACK_SIZE_COROUT);
   end;
   corout_avail := index;
   inc (index, 1);
 end;
 assign (file_event, drive_system + 'events.sim );
 rewrite (file_event);
 startsimulation;
```

```
  case type_arrival of
    0: activate (@generate, 'g');
    1: activate (@generate, 'g');
    2: activate (@cond_gen, 'g');
  end;
  if flag_jobs then begin
    time := prev_t;
    simduration := prev_t;
  end;
  timelist := nil;
  blocklist := nil;
  running := nil;
  writeln (file_event, 0,' ', 0, ' ', 0, ' ', 0);
  close (file_event);
  release (ptr_mem);       { deallocates all the memory of coroutines }
  index := 0;
  while index <= corout_avail do begin
    with jobs_descr[index] do begin
      init_ptr := nil;
    end;
    inc (index, 1);
  end;
  q := lineprod.nextn;          { calculates the time spent in queue }
  while q <> nil do begin
    sum := 0;
    for index := 0 to q^.lqmax do
        sum := sum + index * q^.timelq [index];
    q^.timelq [0] := sum / time; { timelq[0] contents the real value }
    if q^.state = 0 then       { if the activity terminate in idle st }
          q^.tidle := q^.tidle + (time - q^.tlastidle);
    q^.twait := q^.twait + (time - q^.tinput);
    q := q^.nextn;
  end;
end;

procedure run_process;  {initialize the simulation in graphics mode }
begin
  if graphinit then begin
     move_area ('arch4.sim');
     screenx := 0;
     screeny := 0;
     xreal := 0;
     yreal := 0;
     move_screen (screenx, screeny);
     settextstyle (defaultfont, horizdir, 1);
     initialize;
     gotoxy (1, 23);
     writeln ('Fin de simulacion. Presione ESC');
     getchar (carac);
     closegraph;
  end;
end;
```

```
unit nucleo;

interface

uses globals;

const
  DIVI = 10000;
var
       seed, mult: word;

   procedure initnucleus (var p: ptr_word);
   procedure transfer (var p:ptr_word);
   function newprocess (p, stack: ptr_word; size: word): ptr_word;
   procedure initheap (ofsheaporg, ofsheapptr: word);
   procedure initprocess (size: word; ptr_stack, p: ptr_word);
   function rnd (dist: byte; mean, variance: real): real;

implementation

{$l nucleo}

procedure initnucleus (var p: ptr_word); external;
procedure transfer (var p:ptr_word); external;
procedure initheap (ofsheaporg, ofsheapptr: word); external;
procedure initprocess (size: word; ptr_stack, p: ptr_word); external;

function newprocess (p, stack: ptr_word; size: word): ptr_word;
begin
  initprocess (size, stack, p);
  newprocess := ptr (seg (stack^), ofs(stack^) + size - 26);
end;

function randomize (var seed: word; mult: word): word;
begin
  inline ($06/                          { push   es           }
          $C4/ $7E/ $06/                { les    di, [bp + 6] }
          $8B/ $46/ $04/                { mov    ax, [bp + 4] }
          $26/ $8B/ $0D/                { mov    cx, es:[di]  }
          $F7/ $E1/                     { mul    cx           }
          $8B/ $D8/                     { mov    bx, ax       }
          $81/ $E2/ $ff/ $00/           { and    dx, 00ffh    }
          $B1/ $08/                     { mov    cl, 08       }
          $D3/ $E2/                     { shl    dx, cl       }
          $25/ $00/ $ff/                { and    ax, 0ff00h   }
          $B1/ $08/                     { mov    cl, 08       }
          $D3/ $E8/                     { shr    ax, cl       }
          $0B/ $C2/                     { or     ax, dx       }
          $89/ $46/ $fe/                { mov    [bp - 2], ax }
          $26/ $89/ $1D/                { mov    es:[di], bx  }
          $07                           { pop    es           }
  );
end;
```

```
{ the functions to generate random variates were sustracted from }
{ the book DISCRETE-EVENT SYSTEM SYMULATION. Jerry Banks,
John S. Carson. cap. 8 }

function uniform (m, v: real): real;    { uniform distribution }
var
  a, b, r: real;
begin
 a :=  m - v;
 b :=  m + v;
 r := frac (randomize (seed, mult) / DIVI);
 uniform := a + (b - a) * r;
end;

function exponential (m, v: real): real; { exponencial distribution }
var
  r: real;
begin
  r := frac (randomize (seed, mult) / DIVI);
  exponential := - 1 * m * ln (r);
end;

function normal (m, v: real): real;       { normal distribution }
var
  sum, r: real;
  i, k: byte;
begin
  sum := 0;
  for i := 1 to 12 do begin
    r := frac (randomize (seed, mult) / DIVI);
    sum := sum + r;
  end;
  normal := m + sqrt (v) * (sum - 6);
end;

function poisson (m, v: real): real;     { poisson distribution }
var
 p, factor, r: real;
 flag: boolean;
 n: word;
 nr: real;
 i: byte;
begin
 if m < 15 then begin
   p := 1; n := 0; factor := exp (-1 * m);
   flag := false;
   repeat
     r := frac (randomize (seed, mult) / DIVI); p := p * r;
     if p < factor then flag := true
     else inc (n, 1);
   until flag;
   poisson := n;
 end
```

```
 else begin
   factor := 0;
   for i := 1 to 12 do begin
     r := frac (randomize (seed, mult) / DIVI);
     factor := factor + r;
   end;
   poisson := (factor - 6) * sqrt (m) + m - 0.5;
 end;
end;

function erlang (m, v: real): real;        { erlang distribution }
var
 k, i: byte;
 p: real;
begin
 p := 1;
 for i := 1 to 5 do
   p := p * frac (randomize (seed, mult) / DIVI);
 erlang := - m * ln (p);
end;

function gamma (m, v: real): real;       { gamma distribution }
var
  teta, beta, a, b. factor, r1, r2, x: real;
begin
 teta := 1 / m; beta := 1 / v / (teta * teta);
 a := sqrt (2 * beta - 1); b := 2 * beta - ln (4) + (1 / a);
 repeat
   repeat
      r1 := frac (randomize (seed, mult) / DIVI);
   until r1 > 0.40;
   r2 := frac (randomize (seed, mult) / DIVI);
   x := beta * a * exp (ln (r1  / (1 - r1)));
   factor := b - ln (r1 * r1 * r2);
 until x <= factor;
 gamma := x/(beta * teta);
end;

function rnd;              { manages the random variables }
var
  l: real;
begin
  case dist of
       0: l := uniform (mean, variance);
       1: l := exponential (mean, variance);
       2: l := normal (mean, variance);
       3: l := gamma (mean, variance);
       4: l := poisson (mean, variance);
       5: l := erlang (mean, variance);
       6: l := mean;
  end;
  rnd := l;
end;
```

```
begin
  seed := 10123;            { initialize the seed and mult variable }
  mult := 7743;             { for pseudo random  number generator }
  initheap (ofs (heaporg), ofs (heapptr)); { initialize the nucleus ptr

end.

;  ***************************************************************
;            PRIMITIVES FUNCTIONS FOR CORUTINES
;
;                      FILE NUCLEO.ASM
;  ***************************************************************

   code  segment  byte  public
         assume  cs: code
      public initnucleus, transfer, initprocess, initheap

; ***************************************************************************


        addr_current    dd    0
        addr_heaporg    dd    0
        addr_heapptr    dd    0
        cursor          dw    0
; ***************************************************************************

;   procedure initnucleus (var p: ptr_word);
 initnucleus  proc  far
              push  bp
              mov   bp, sp
              push  es
              les   bx, [bp + 6]
              mov   word ptr addr_current + 2, es
              mov   word ptr addr_current, bx
              pop   es
              pop   bp
              ret   4
  initnucleus endp
; ***************************************************************************

;   procedure initheap (ofsheaporg, ofsheapptr: word);

    ofsheaporg  equ  [bp + 8]
    ofsheapptr  equ  [bp + 6]

   initheap  proc  far
             push  bp
             mov   bp, sp
             mov   ax, ofsheaporg
             mov   word ptr addr_heaporg + 2, ds
             mov   word ptr addr_heaporg, ax
             mov   ax, ofsheapptr
             mov   word ptr addr_heapptr + 2, ds
             mov   word ptr addr_heapptr, ax
             pop   bp
             ret   4
             initheap  endp
; ***************************************************************************
```

```
; procedure initprocess (size: word; stack, p: ptr_word);
    flags  equ  [bp - 2]
    ofs_p  equ  [bp + 6]
    seg_p  equ  [bp + 8]
ofs_stack  equ  [bp + 10]
seg_stack  equ  [bp + 12]
     size  equ  [bp + 14]

initprocess  proc  far
             push  bp
             mov   bp, sp
             pushf
             push  es
             les   si, ofs_stack
             add   si, size                ; finds the bottom of stack
             sub   si, 6
             mov   ax, seg_p               ; segment of procedure p
             mov   es:[si], ax
             mov   ax, ofs_p               ; offset of procedure p
             mov   es:[si - 2], ax
             mov   ax, flags               ; flags
             mov   es:[si - 4], ax
             sub   si, 4
             mov   es:[si - 2], si         ; bp
             mov   es:[si - 4], ds         ; data segment
             mov   bx, seg_stack
             mov   ax, ofs_stack
             mov   es:[si - 6], bx         ; initial value of heaporg
             mov   es:[si - 8], ax
             mov   es:[si - 10], bx        ; initial value of heapptr
             mov   es:[si - 12], ax
             xor   ax, ax
             mov   es:[si - 14], ax        ; cursor
             lea   dx, firstcall
             mov   es:[si - 16], dx        ; keep offset of firstcall
             pop   es
             popf
             pop   bp
             ret   10
  initprocess endp
; ***************************************************************************
```

```
; procedure transfer (var p: ptr_word);
     transfer proc far
              pushf
              push  bp
              mov   bp, sp
              push  ds
              les   si, addr_heaporg
              push  es:[si + 2]
              push  es:[si]
              les   si, addr_heapptr
              push  es:[si + 2]
              push  es:[si]
              mov   ax, cursor
              push  ax
              lea   ax, restart
              push  ax
              les   si, addr_current
              mov   es:[si + 2], ss
              mov   es:[si], sp
              les   si, [bp + 8]
              mov   word ptr addr_current + 2, es
              mov   word ptr addr_current, si
              mov   ax, es:[si + 2]
              mov   bx, es:[si]
              mov   ss, ax
              mov   sp, bx
              pop   bx
              jmp   bx
    firstcall: pop  bx
              mov   cursor, bx
              les   si, addr_heapptr
              pop   es:[si]
              pop   es:[si + 2]
              les   si, addr_heaporg
              pop   es:[si]
              pop   es:[si + 2]
              pop   ds
              pop   bp
              popf
              add   sp, 4
              call  dword ptr [bp + 2]
    restart:  pop   bx
              mov   cursor, bx
              les   si, addr_heapptr
              pop   es:[si]
              pop   es:[si + 2]
              les   si, addr_heaporg
              pop   es:[si]
              pop   es:[si + 2]
              pop   ds
              pop   bp
              popf
              ret   2
     transfer endp
```

```
unit globals;

interface

{  ************************ global variables ***************************


const                          { constants definition }

  LEFT_ARROW = 75;
  DOWN_ARROW = 80;
 RIGHT_ARROW = 77;
    UP_ARROW = 72;
        PGDN = 73;
        PGUP = 81;
      RETURN = 13;
         ESC = 27;
          F1 = $3b;
          F2 = $3c;
          F7 = $41;
          F8 = $42;
          F9 = $43;
         F10 = $44;
  MAX_ENTRYS = 14;

type

     ptr_word = ^word;

    ptr_descr = ^descr;

        descr = record                    { coroutine descriptor }
                  corout: ptr_word;       { points to ss:sp }
                  init_ptr: ptr_word;     { points to the top of stack }
                  next: ptr_descr;
                  key: real;              { process organization }
                  sym: string;            { process identification }
                  flag_busy: boolean;     { flag for busy stack }
                end;

     ptr_node = ^ node;     { this structure descript the operation    }
                            { of a single machine into the production }
                            { line }

  range_entrys = array [0..15] of byte;  { up to 15 succesive mach.}
                                         {can hold a previos machine}
    frec_table = array [0..14] of real;  {up to 14 frec. data can }
                                         {be defined in the frec. T.}
          node = record              { **** activity descriptor *** }
                   identif: byte;        { number of identification }
                      sons: range_entrys;{ succesors activities }
                     queue: ptr_descr;
                     lqcap: byte;        { queue capacity }
                        lq: byte;        { queue length at time T }
                     lqmax: byte;        { max. queue length }
```

```
                distr_type: byte;          { type of distribution prob.}
                 serv_time: real;          { machine's service time }
                  variance: real;          { machine's variance of time}
                     state: byte;          { 0:Idle, 1:work, 2:block}
                    tblock: real;          { time in blocked state }
               tlastblock: real;           { time of blocked state start}
                  n_block: byte;           { counter of blocked state }
                    twait: real;           { residence time in activity}
                    tidle: real;           { idle time }
                 tlastidle: real;          { time idle state start }
                    tinput: real;          { start residence time }
                    jobout: word;          { number of jobs processed }
                     jobin: word;          { number of arrival jobs }
                   tprevlq: real;          { time that lq start }
                    timelq: frec_table;    {time of lq}
                       nwq: byte;          { # jobs that leave the queue}
                       twq: real;          { time spent in queue }
             timef, frec: frec_table;
                         x: integer;       { x's coordenate of the block}
                         y: integer;       { y's coordenate of the block}
                 prev_hold: boolean;       { previos hold }
                 flag_frec: boolean;       { frec. t. especification }
                     nextn: ptr_node
                end;

var
 lineprod: node;                  { node that keep the lineprod list }
 last_node: ptr_node;             { points to the end of lineprod list}
 buffodds, buffeven: ptr_word;       { buffer for graph screen }
 menuscr: ptr_word;                  { buffer for text screen }
 endedjobs: word;                 { number of jobs totaly processed }
 viat, iat: real;                    { interarrival jobs parameters }
 dtiat: byte;                        { interarrival jobs distribution }
 processjobs: word;                  { number of jobs to process }
 file_event: text;                 { text file for process parameters }
 type_perform: byte;
 flag_jobs: boolean;               { flag especification for number of}
                                     { jobs to process }
 type_arrival: byte;     { arrival 0: prob. 1: frec. 2: condicional }
 arr_time, arr_frec: frec_table;     { frec. dist. table arrivals }
 cond_gen_perf: boolean;           { variable for conditional arrive}
 cond_mach: word;           { mach. num. that conditionalize arrive }
 lose_job: word;                     { number of losed jobs }
 time: real;                         { clock for simulation }
 drive_system: string;               { system loadeo drive }
 filename: string;                 { file name in simulator's memory }
 statment: array [1..11] of string; { variables in the report }
 rep_statment: array [0..14] of string;  { var label in the report }
 tot: array [3..11] of real;
 max: array [3..11] of real;         { contents the maximum value of }
 carac: char;                        { each parameter in the report  }
 s1, s2: string;                     { auxiliar variables }
```

```
procedure getchar (var car: char);   { read a char from the keyboard }
                                     { with out echo }
function keypressed: boolean;        { if Keyboard has been pressed }
                                     { then return true }
procedure gotoxy (x, y: byte);       { position the cursor in x, y }
                                     { coordenates }
function find (num: byte; ptr: ptr_node): ptr_node;
function distribution_type (option: byte): string;
procedure maximum (p: ptr_node);
procedure gen_report (p: ptr_node);

implementation

procedure getchar (var car: char);
begin
  inline ($50/                  { push ax }
          $06/                  { push es }
          $56/                  { push si }
          $b4/ $08/             { mov ah, 8 }
          $cd/ $21/             { int 21h }
          $c4/ $76/ <car/       { les si, car }
          $26/ $88/ $04/        { mov es:[si], al }
          $5e/                  { pop si }
          $07/                  { pop es }
          $58                   { pop ax }
  );
end;

function keypressed: boolean;
begin
  inline ($53/                  {     push bx                    }
          $8a/ $fc/             {     mov  bh, ah                }
          $b4/ $0b/             {     mov  ah, 0bh               }
          $cd/ $21/             {     int  21h                   }
          $3c/ $00/             {     cmp  al, 00                }
          $74/ $02/             {     je   @2                    }
          $b0/ $01/             {     mov  al, 01                }
          $8a/ $e7/             { @2: mov  ah, bh                }
          $88/ $46/ $FF/        {     mov [bp - 1], al           }
          $5b                   {     pop  bx                    }
  );
end;

procedure gotoxy (x, y: byte);
begin
  inline ($50/                { push ax     }
          $52/                { push dx     }
          $53/                { push bx     }
          $8a/ $76/ <y/       { mov  dh, y  }
          $8a/ $56/ <x/       { mov  dl, x  }
          $b7/ $00/           { mov  bh, 0  }
          $b4/ $02/           { mov  ah, 02 }
          $cd/ $10/           { int  10h    }
          $5b/                { pop  bx     }
```

```
          $5a/                  { pop   dx      }
          $58                   { pop   ax      }
  );
end;
function find (num: byte; ptr: ptr_node): ptr_node;
var                             { find the node with the num ident. in the}
  p: ptr_node;                  { node list and if not found return nil }
begin
 while (ptr <> nil) and (ptr^.identif <> num) do
                ptr := ptr^.nextn;
 find := ptr;
end;

function distribution_type (option: byte): string;
  var                           { return the distr. label for the correspondig }
    res: string;                 { distr. type }
begin
  case option of
     0: res := 'Uniforme';
     1: res := 'Exponencial';
     2: res := 'Normal';
     3: res := 'Gamma';
     4: res := 'Poisson ';
     5: res := 'eRlang-k';
     6: res := 'Constante';
  end;
  distribution_type := res;
end;

procedure maximum (p: ptr_node); { founds the maximum value of each }
var                              { report parameter in the general }
  i: byte;                       { report }
begin
  for i := 3 to 11 do begin
     max[i] := 0;  tot[i] := 0;
  end;
  while p <> nil do begin
    with p^ do begin
      if tot[3] < jobout then begin
             tot[3] := jobout;
             max[3] := identif;
      end;
      if tot[4] < jobin then begin
             tot[4] := jobin;
             max[4] := identif;
      end;
      if tot [5] < lq then begin
             tot[5] := timelq [0];
             max[5] := identif;
      end;
      if tot [6] < lqmax then begin
             tot[6] := lqmax;
             max[6] := identif;
      end;
```

```
      if nwq > 0 then
        if tot [7] < twq / nwq then begin
             tot[7] := twq / nwq;
             max[7] := identif;
      end;
      if tot [8] < n_block then begin
             tot[8] := n_block;  max[8] := identif;
      end;
      if tot [9] < tblock then begin
             tot[9] := tblock;
             max[9] := identif;
        end;
      if tot [10] < tidle then begin
             tot[10] := tidle;
             max[10] := identif;
      end;
      if jobout > 0 then
        if tot [11] < twait / jobout then begin
             tot[11] := twait / jobout;
             max[11] := identif;
        end;
    end;
    p := p^.nextn;
  end;
end;

procedure gen_report (p: ptr_node);  { generates the formated output}
begin                                { for each report parameter }
 with p^ do begin
  str (identif:3, statment[1]);
  statment [1] := '   '+ statment [1] + '    ';
  if flag_frec then statment [2] := 'Frec. T.'
  else str (serv_time:8:2, statment[2]);
  str (jobout:3, statment [3]);
  statment [3] := '   '+ statment [3] +      ';
  str (jobin - jobout:3, statment[4]);
  statment [4] := '   '+ statment [4] + '    ';
  str (timelq[0]:8:2, statment [5]);
  str (lqmax:3, statment [6]);
  statment [6] := '   '+ statment [6] +      ';
  if nwq > 0 then str (twq / nwq:8:2, statment [7])
  else statment [7] := '     0.00';
  str (n_block:3, statment [8]);
  statment [8] := '   '+ statment [8] + '     ';
  if time > 0 then str (tblock / time * 100:8:2, statment [9])
  else statment [9] := '     0.00';
  if time > 0 then str (tidle / time * 100:8:2, statment [10])
  else statment [10] := '     0.00';
  if jobin > 0 then str (twait / jobin:8:2, statment [11])
  else statment [11] :=      0.00';
 end;
end;
end.
```

```
{ ********************** init global variables ********************** }

procedure make_cube (var s: sprite; color: byte);
var                              { make the cube sprite for the activity }
  j: byte;                            { representation on graphics }
  s1: string;
begin
with s do begin
  fillchar (image, 78, color);
  for j := 0 to 17 do
     image [j] := $aa;
  image [0] := $aa shr 4; { make black the first two pixel of sprite}
  image [6] := $aa shr 2; { make black the fist pixel of second line }
  j := 23;
  while j <= 78 do begin
     image [j] := (image [j] and $c0) or $2a;
     inc (j, 6);
  end;
  image [77] := image [77] and $f0;
  image [71] := image [71] and $fc;
end;
end;

procedure init_globals;     { initialize the globals variables }
begin
  iat := 0;
  type_perform := 0;
  viat := 0;
  dtiat := 0;
  simduration := 0;
  time := 0;
  flag_jobs := false;
  processjobs := 0;
  endedjobs := 0;
  screenx := 0;
  screeny := 0;
  flag_save := true;
  flag_name := false;
  lineprod.sons[0] := 0;
  lineprod.nextn := nil;
  last_node := nil;
  filename := 'WORK';
  xreal := 0;
  yreal := 0;
  rep_statment [0] := ' REPORTE GENERAL ';
  rep_statment [1] := ' Numero de actividad          ';
  rep_statment [2] := ' Tiempo de servicio           ';
  rep_statment [3] :=  Entidades procesadas          ';
  rep_statment [4] := ' Entidades pendientes         ';
  rep_statment [5] := ' Long. media de cola          ';
  rep_statment [6] := ' Long. maxima de cola         ';
  rep_statment [7] := ' Tiempo medio de espera       ';
  rep_statment [8] := ' Num. de veces bloqueada      ';
```

```
  rep_statment [9] := ' % T. en estado de bloqueo    ';
  rep_statment [10] := ' % T. en estado ocioso        ';
  rep_statment [11] := ' Tiempo medio de residencia   ';
  rep_statment [12] := 'Num. de entidades terminadas: ';
  rep_statment [13] := 'Tiempo de simulacion: ';
  rep_statment [14] := 'Num. de entidades perdidas:';
  with blockw do begin
    xsize := 5;
    ysize := 12;
  end;
  make_cube (blockw, $55);
  with blocki do begin
    xsize := 5;
    ysize := 12;
  end;
  make_cube (blocki, $ff);
  with blockb do begin
    xsize := 5;
    ysize := 12;
  end;
  make_cube (blockb, $5f);
  with jobsprite do begin
    xsize := 0;
    ysize := 3;
    image [0] := $3c;
    image [1] := $c3;
    image [2] := $c3;
    image [3] := $3c;
  end;
  with backjob do begin
    xsize := 0;
    ysize := 3;
    fillchar (image, 4, 0);
  end;
  xir := 1;
  yir := 1;
  last_wind := 0;
  getmem (buffodds, 16200);
  getmem (buffeven, 16200);
  getmem (menuscr, 4000);
  getmem (report1, 8680);
  getmem (backrep1, 8680);
  fillchar (buffodds^, 16200, 0);
  fillchar (buffeven^, 16200, 0);
  fillchar (menuscr^, 4000, 0);
  fillchar (report1^, 8680, 0);
  fillchar (backrep1^, 8680, 0);
{ the coordenates of any option are relatives to the current window }
{ ********************* main menu ******************************}
 with menu1[1] do begin      { file }
  x := 8; y := 1; car := $41; nc := 0;
  mess := 'Archivo ';
 end;
```

```
with menu1[2] do begin      { edit }
  x := 20; y := 1; car := $45; nc := 0;
  mess := 'Edicion';
end;
with menu1[3] do begin      { run }
  x := 32; y := 1; car := $4a; nc := 1;
  mess := 'eJecuta';
end;
with menu1[4] do begin    { report }
  x := 44; y := 1; car := $52; nc := 0;
  mess := 'Reporte';
end;
with menu1[5] do begin
   x := 56; y := 1; car := $53; nc := 0;
   mess := 'Salir';
end;
with menu1[6] do begin
   x := 0; y := 1;
end;
{ ************************** File menu ***************************** }
with menu2[1] do begin
  x := 2; y := 1; car := $43; nc := 0;
  mess := 'Cargar';
end;
with menu2[2] do begin
  x := 2; y := 2; car := $4e; nc := 0;
  mess := 'Nuevo';
end;
with menu2[3] do begin
  x := 2; y := 3; car := $47; nc := 0;
  mess := 'Guardar';
end;
with menu2[4] do begin
   x := 2; y := 4; car := $44; nc := 0;
   mess := 'Directorio';
end;
with menu2[5] do begin
   x := 2; y := 5; car := $41; nc := 1;
   mess := 'cAmbio dir ;
end;
with menu2[6] do begin
  x := 0; y := 1;
end;
{ ************************ Edit menu ******************************** }
with menu3[1] do begin
 x := 2; y := 1; car := $4e; nc := 0;
 mess := 'Numero de actividad';
end;
with menu3[2] do begin
  x := 2; y := 2; car := $4c; nc := 0;
  mess := 'Longitud de cola';
end;
```

```
with menu3[3] do begin
  x := 2; y := 3; car := $54; nc := 0;
  mess := 'Tiempo de servicio ';
end;
with menu3[4] do begin
 x := 2; y := 5; car := $41; nc := 0;
 mess := 'Actividad previa ';
end;
with menu3[5] do begin
  x := 2; y := 6; car := $44; nc := 0;
  mess := 'Dibujo de red';
end;
with menu3[6] do begin
  x := 2; y := 7; car := $42; nc := 0;
  mess := 'Borrar';
end;
with menu3[7] do begin
  x := 0; y := 1;
end;
{ ************************ Run menu ********************************* }
with menu4[1] do begin
  x := 2; y := 1; car := $41; nc := 10;
  mess := 'tiempo de Arribo';
end;
with menu4[2] do begin
  x := 2; y := 3; car := $54; nc := 0;
  mess := 'Tiempo de simulacion';
end;
with menu4[3] do begin
  x := 2; y := 4; car := $45; nc := 0;
  mess := 'Entidades a procesar';
end;
with menu4[4] do begin
  x := 2; y := 5; car := $44; nc := 0;
  mess := 'Desarrollo';
end;
with menu4[5] do begin
  x := 0; y := 1;
end;
{ ************************ Performance menu ************************* }
with menu5[1] do begin
  x := 2; y := 1; car := $50; nc := 0;
  mess := 'Paso por paso';
end;
with menu5[2] do begin
  x := 18; y := 1; car := $52; nc := 0;
  mess := 'Rapido';
end;
with menu5[3] do begin
  x := 27; y := 1; car := $45; nc := 0;
  mess := 'Extra rapido';
end;
```

```
with menu5[4] do begin
  x := 0; y := 1;
end;
{ ***************************** report menu ************************}
with menu6[1] do begin
  x := 2; y := 1; car := $52; nc := 0;
  mess := 'Reporte general';
end;
with menu6[2] do begin
  x := 2; y := 2; car := $48; nc := 8;
  mess := 'resumen Historico';
end;
with menu6[3] do begin
  x := 0; y := 1;
end;
{ ***************************** device menu ************************}
with menu7[1] do begin
  x := 3; y := 1; car := $50; nc := 0;
  mess := 'Pantalla';
end;
with menu7[2] do begin
  x := 13; y := 1; car := $49; nc := 0;
  mess := 'Impresora ';
end;
with menu7[3] do begin
  x := 0; y := 1;
end;
{ ********************** dist. type menu ***************************}
 with menu8[1] do begin
   x := 2; y := 1; car := $55; nc := 0;
   mess := 'Uniforme';
 end;
 with menu8[2] do begin
    x := 2; y := 2; car := $45; nc := 0;
    mess := 'Exponencial';
  end;
 with menu8[3] do begin
    x := 2; y := 3; car := $4e; nc := 0;
    mess := 'Normal';
  end;
  with menu8[4] do begin
     x := 2; y := 4; car := $47; nc := 0;
     mess := 'Gamma';
  end;
  with menu8[5] do begin
    x := 2; y := 5; car := $50; nc := 0;
    mess := 'Poisson ';
  end;
  with menu8[6] do begin
     x := 2; y := 6; car := $52; nc := 1;
     mess := 'eRlang-k';
  end;
```

```
  with menu8[7] do begin
    x := 2; y := 7; car := $43; nc := 0;
    mess := 'Constante ';
  end;
  with menu8[8] do begin
    x := 0; y := 1;
  end;
{ ********************************* distribution menu ********************}
  with menu9[1] do begin
    x := 2; y := 1; car := $50; nc := 6;
    mess := 'dist. Probabilistica';
  end;
  with menu9[2] do begin
    x := 2; y := 2; car := $46; nc := 6;
    mess := 'dist. Frecuencias';
  end;
  with menu9[3] do begin
    x := 0; y := 1;
  end;
{ ******************** service time parameters menu ********************}
  with menu10[1] do begin
   x := 2; y := 1; car := $4d; nc := 0;
   mess := 'Media';
  end;
  with menu10[2] do begin
   x := 2; y := 2; car := $56; nc := 0;
   mess := 'Varianza ';
  end;
  with menu10[3] do begin
   x := 0; y := 1;
  end;
{ ****************************** arrival menu ***************************}
  with menu11[1] do begin
    x := 2; y := 1; car := $44; nc := 0;
    mess := 'Dist. probabilistica';
  end;
  with menu11[2] do begin
    x := 2; y := 2; car := $46; nc := 6;
    mess := 'dist. Frecuencias';
  end;
  with menu11[3] do begin
    x := 2; y := 3; car := $41; nc := 0;
    mess := 'Arribo condicional';
  end;
  with menu11[4] do begin
    x := 0; y := 1;
  end;
{ ************************  Delete menu  ******************************}
  with menu12[1] do begin
    x := 2; y := 1; car := $41; nc := 0;
    mess := 'Actividad';
  end;
```

```
    with menu12[2] do begin
      x := 15; y := 1; car := $4c; nc := 0;
      mess := 'Liga';
    end;
    with menu12[3] do begin
      x := 0; y := 1;
    end;
end;

procedure dispose_globals;     { deallocates the dinamics globals }
begin                          { variables }
  freemem (buffodds, 16200); buffodds := nil;
  freemem (buffeven, 16200); buffeven := nil;
  freemem (menuscr, 4000); menuscr := nil;
  freemem (report1, 7910); report1 := nil;
  freemem (backrep1, 7910); backrep1 := nil;
end;
```

El jurado designado por la Sección de Computación del Departamento de Ingeniería Eléctrica del Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional, aprobó esta tesis el 4 de febrero de 1991.

Ana María Antonia Martínez Enriquez

José Oscar Olmedo Aguirre

Enrique Arce Medina