**U1. INTRODUCCION A SISTEMAS OPERATIVOS**

Un SO es una capa de SW o programa que hace utilizable al HW. Actúa como intermediario entre el usuario y el HW y es similar a un gobierno. Proporciona un modelo de computadora más simple.

Es un programa o proceso con las rutinas de control necesarias para mantener en continua operación a los programas que se están ejecutando. El HW proporciona la capacidad bruta de procesamiento y el SO pone dicha capacidad al alcance de los usuarios y administra el HW para lograr un buen rendimiento.

Niveles de complejidad de las componentes de una PC

<

|  |
| --- |
| Calculadora – Navegación – Reproductor – etc.  *Programas de aplicación* |
| Compilador – Interprete – Editor – Debugger  *Programas de sistema* |
| Sistema operativo |
| Lenguaje máquina  *Hardware* |
| Microprogramación |
| Hardware o dispositivos |

>

*Hardware*

El nivel de microprogramación y lenguaje maquina son instrucciones pregrabadas, simples y potentes. El lenguaje maquina reúne varias de las instrucciones de microprogramación en una sola para facilitar el trabajo del SO. El SO facilita aún más el lenguaje de programación, por lo que es un conjunto de programas instalados en el software o firmware que hacen utilizable el HW.

*Programas de sistema*

Constituido por el SO, compiladores, interpretes, editores y debuggeadores que gobiernan el HW y permiten abstraer la complejidad de este. Actúa de base para todo el SW que se utilice sobre él.

Los compiladores, interpretes, editores y debuggeadores están en otro nivel ya que estos pueden ser escritos y modificados con libertad, mientras que el núcleo del SO no puede cambiarse. Los programadores se encuentran en este nivel, mientras que en el nivel del SO se encuentran los diseñadores de SO.

*Programas de aplicación*

Aquí se encuentran los usuarios de alto nivel. Contiene el resto de los programas que ejecutan tareas particulares y persiguen un propósito específico para estos usuarios.

**Objetivos**

- Ofrecer ciertos servicios a los programas y a los usuarios de estos programas con el fin de facilitar la programación. Estos servicios difieren de un SO a otro.

- Controlar y coordinar el uso del HW entre los diversos programas de aplicación.

- Actuar como gestor de recursos, asignándolos a programas y usuarios según sus necesidades en forma eficiente y justa.

- Controlar la ejecución de los programas a fin de evitar errores y el uso incorrecto del HW. Principalmente controla la operación de los D-E/S.

**Funciones**

Para cumplir estos objetivos el SO consta de dos funciones:

Funciones para facilitar la interface (7): Define una serie de rutinas para permitirle al usuario llamarlas y así hacer más fácil la comunicación usuario-PC.

- *Creación de programa*: Ofrece al programador un conjunto de rutinas para ayudarlo en la creación del programa por medio de compiladores, intérpretes, editores y debuggeadores.

- *Ejecución del programa*: Facilita la transferencia del programa a MP para su ejecución y se encarga de preparar los recursos que este programa va a utilizar.

- *Acceso al sistema*: Ofrece la posibilidad de autenticar los usuarios que actúan en el SO, lo cual se aplica para sistemas multiusuarios. Dentro de las funciones de autentificación se incluyen el logueo y todas las operaciones que se realizan sobre los recursos.

- *Acceso a D-E/S*: Traduce las operaciones sobre estos dispositivos al lenguaje entendido por los mismos. Además gestiona la transmisión de datos desde y hacia el dispositivo y todos los eventos que este quiere comunicar.

- *Acceso a archivos*: Regula las operaciones de escritura y lectura de los archivos, impidiendo que surjan resultados no deseados al ser accedidos por dos o más procesos.

- *Control y detección de errores*: Permite detectar errores de procesamiento y de HW. Ante un error, el SO debe dar una respuesta al usurario e intentar minimizar el impacto de dicha falla. Ejemplos en procesamientos: desbordamiento aritmético, acceso prohibido a una posición de memoria, incapacidad para asignar un recurso, etc. Ejemplos en HW: fallos en los D-E/S, errores de memoria, lecturas incorrectas de bloques defectuosos de discos, etc.

- *Contabilidad*: Recolecta estadísticas de la utilización de los recursos y supervisa parámetros tales como tiempos de respuesta, de servicios, de lectura, etc. Estas estadísticas o heurísticas le permiten al SO mejorar y preparar de mejor manera las futuras ejecuciones.

Funciones para administrar los recursos:

Esta función que tiene el SO es para asignar los recursos en forma equitativa a todos los procesos. La particularidad radica en que el SO también debe asignarse recursos a el mismo, compitiendo con los demás procesos. A fin de lograr esto, el SO se va a ejecutar en un modo diferente que el resto de los programas, llamado modo Kernel o modo supervisor. Este modo implica que sus instrucciones son protegidas por el HW para que no puedan ser interrumpidas por otros procesos. En este modo el SO tiene acceso completo a todo el HW y puede ejecutar cualquier instrucción que la maquina sea capaz de ejecutar. No todo el SO se ejecuta en modo Kernel, solo su núcleo.

El SO temporalmente sede el procesador para la ejecución de los procesos en tiempos o lapsos definidos.

Los compiladores, intérpretes, debuggeadores, editores y el resto de SW de aplicación se ejecuta en modo programa o modo usuario.

**Evolución histórica**

• Generación cero (década del 40)

Ausencia de SO, solo tenían lenguaje máquina por lo que había acceso total a este.

Se trabajaba por medio del procesamiento en *serie*, es decir un procesamiento secuencial de todos los procesos. Era lento y poco práctico. Los usuarios modificaban sus programas con las tarjetas perforadas, las introducían a la computadora y los corregían y depuraban en su mismo tiempo de ejecución. Para usuarios avanzados.

• Primera generación (1945 - 1955)

Carencia de SO. Surgen los primeros bulbos y conexiones eléctricas. En el ’50 nace el concepto de trasmisión o lotes de trabajo.

El trabajo en *lotes* es muy similar al procesamiento en serie y consistía en reunir en bloques o lotes todos los trabajos similares para realizarlos al mismo tiempo, evitando la espera de dos o más trabajos como sucede en el procesamiento en serie. Se podían procesar los trabajos de rutina sin que haya un usuario interactivo presente en todo momento. Existía un código llamado monitor que avisaba cuando un lote terminaba para que ingresen el siguiente.

• Segunda generación (1955 - 1965)

Se afianza el procesamiento por lotes y aparecen los transistores. En el ’60 surge el SO de tiempo compartido con dos conceptos: multiprogramación y multiprocesamiento.

El *tiempo compartido* permiten que varios usuarios remotos ejecuten trabajos en la computadora al mismo tiempo. Le da la sensación al usuario de que es para el solo. Cada tarea recibe la atención del microprocesador durante una fracción de segundo. Para mantener el sistema en orden, cada tarea recibe un nivel de prioridad o se procesa en orden secuencial. Se asignan tiempos de procesamiento y existe una protección en la memoria entre cada proceso. Da la sensación al usuario de que todos los procesos se ejecutan en simultáneo.

La *multiprogramación* consiste en que varios procesos o programas de usuarios, que se encuentran al mismo tiempo en MP, utilicen un solo procesador, cambiando este rápidamente de uno a otro. Se aprovecha al máximo la CPU. (Particionamiento de memoria)

El *multiprocesamiento* permite que varios procesadores se puedan utilizar al mismo tiempo en un mismo sistema, a fin de incrementar el poder de procesamiento.

También surge el concepto de independencia de dispositivos, lo que implica que el programa usuario especifica las características generales del dispositivo a utilizar para que, posteriormente, el SO asigne los recursos según requerimiento y disponibilidad. Esto permite que el programador ya no se preocupe en saber con qué dispositivo debe interactuar.

• Tercera generación (1965 - 1980)

Se difunde masivamente el concepto de *multiprogramación*, es decir el procesamiento de varias tareas al mismo tiempo sobre uno o más procesadores. Aparecen los circuitos integrados. Se empieza a utilizar el particionamiento de memoria para permitir que más procesos puedan estar en ejecución. También se desarrollan las técnicas de aprovechamiento de los tiempos de espera de las operaciones de E/S.

Se protege el SO en memoria para impedir que los procesos de usuarios accedan a posiciones del SO. (Kernel)

Se empieza a utilizar la técnica de Spooling para aprovechar tiempos de espera. Consiste en introducir procesos en un buffer (un área especial en memoria o en un disco), de manera que el dispositivo pueda acceder a ellos cuando desee y elegir cual ejecutar.

Aparece el SO de *tiempo real*, el cual requiere una respuesta inmediata. Se utiliza para atender los procesos críticos o de alta prioridad que han de resolverse en tiempo muy pequeño. Generalmente asociados a usos militares o industriales. Deben proveer garantías absolutas de que cierta acción ocurrirá en un instante determinado. El objetivo es proporcionar rápidos tiempos de respuesta. La gestión de memoria es menos exigente que en tiempo compartido, usualmente los procesos residen permanentes en la memoria.

• Cuarta generación (1980 - 1990)

Se difunde la computación en los hogares con la aparición de las *computadoras personales* para usuarios no profesionales, con interface amigable y bajo costo en HW. Estas soportan la multiprogramación. Su trabajo es proporcionar buen soporte para un solo usuario. En este SO el código es más pequeño, tiene un conjunto de las funciones más importantes del SO de mainframes. Aparece el SW, un SO de interface amigable destinado a usuarios no profesionales con una interface gráfica muy desarrollada.

Se implementan los SO en *red*, el cual permite la interconexión de diferentes computadoras, que reportan a un SO central, para poder acceder a los servicios y recursos, y controlar el acceso a estos.

En base a esto, surge el SO *distribuido* que utiliza la red para conectar varios procesadores, que tienen una copia de micro Kernel, a un núcleo central para permitir distribuir los procesos entre ellos. Tiene mayor tolerancia a fallos en comparación con mainframes ya que en mainframes si se rompe el servidor todos los usuarios se perjudican, en cambio con distribuido si falla uno de los servidores existen otros para reemplazarlo, permitiendo servicio 24/7 ya que todos los servidores tienen el mismo núcleo y se puede modificar el uso de cada uno.

Se avanza en el control de acceso y seguridad y gran uso de sistemas de base de datos transaccionales a través de redes.

Surge el SO en *paralelo*. Consiste en conectar por bus varias CPU a un solo SO, es decir multiprocesadores. Se pretende que cuando existan dos o más procesos que compitan por algún recurso se puedan realizar o ejecutar al mismo tiempo. Puede ser simétrico o asimétrico: En el simétrico cada procesador ejecuta una copia idéntica del SO y que se comunican si es necesario, y en asimétrico se asigna a cada procesador una tarea particular (esquema maestro esclavo donde el maestro planifica y el esclavo ejecuta).

• Quinta generación (1990 - Ahora)

Con respecto a la *nanotecnología* el rol del SO está dado por la portabilidad y la microcodificación. Se enfocan a ser multiplataforma y regidos por esquemas virtuales a través de internet.

En la *robótica* apuntan a acompañar paradigmas de acción y reacción y avanzar en inteligencia artificial.

Los SO se adecuan a fin que la interface con el usuario sea capaz de capturar entradas no convencionales como la vista, los gestos, y la mente.

**Clasificación por tipo**

Serie, lotes, tiempo compartido, multiprogramación, tiempo real, PC, distribuidos, en paralelo.

**Clasificación por estructura**

- *Monolíticos*: Carecen de estructura jerárquica. Todas las rutinas o códigos están en un mismo nivel, por lo que si se modifica una hay que compilar el todo nuevamente.

- *Capas*: Para evitar lo anterior se estructura el código en capas independientes que pueden ser compiladas por separado. Cada capa obtiene servicios de la capa inferior y los brinda a la capa superior. Cada una define su interface, entendiendo como interface la definición de parámetros de ingreso y variables de salida que genera. Es decir que la capa inferior debe brindarle a la capa superior las salidas que representan sus entradas. Esto se denomina caja negra.

- *Máquinas virtuales*: Es una capa superior al HW con características virtuales, que permite instalar diferentes SO sobre esta, los cuales van a suponer tener la totalidad del HW.

- *Red*: Posee un servidor principal y un conjunto de terminales conectadas vía red. Estas terminales pueden tener características particulares y no tener SO (llamadas bobas) o tener SO. En el caso de que la terminal tenga una copia del SO, recurren al servidor principal para diferentes servicios como autentificación, file serve, recursos compartidos entre red, DHCP, DNS.

- *Distribuido*: Presenta diferentes servidores, los cuales tienen una copia de microKernel y por sobre este implementan una capa de servicios. Cada servidor puede brindar servicios particulares. Los servidores que ofrezcan el mismo servicio deberán estar sincronizados.

**Clasificación por servicio**

Un SO cuenta con una característica de cada una de las siguientes clasificaciones:

- Por número de usuario: Monousuario (multiprocesamiento) o multiusuario (multiprogramación).

- Por número de tareas: Monotarea (multiprocesamiento) o multitarea (multiprogramación).

- Por número de procesadores: Uniproceso (multiprogramación) o multiproceso.

**U2. ADMINISTRACION DE PROCESOS**

**Procesos**

Un proceso es un programa en ejecución. El programa por sí solo no es un proceso. Se convierte en proceso cuando se carga un archivo ejecutable (lista de instrucciones) en MP. Por lo que el proceso es la entidad activa de un programa, el cual contiene en un número de elementos. Los elementos esenciales son el conjunto de instrucciones y el conjunto de datos asociados a dicho código.

El SO gestiona e identifica a todos los procesos que se encuentran en ejecución a través de una estructura de datos conocida como Bloque de Control de Proceso (BCP). Este bloque se genera al momento de la creación de cada proceso, por lo que cada proceso tiene un único BCP que lo describe.

**Entradas del BCP** (8)

- ID: Identificación de cada proceso que lo distingue de los demás, por lo que es único. Ese ID dura lo que dura el proceso.

- Estado: Describe si el proceso se encuentra en ejecución o no, es decir que indica en qué punto de su ciclo de vida está. Un proceso puede estar en algunos estados más de una vez. Como también es posible que el proceso no esté en alguno de los estados del ciclo de vida.

- Prioridad: Cada proceso tiene un nivel de importancia, el cual puede ser variable o dinámico, es decir que puede ser cambiado por el SO según sus necesidades. En algunos casos algunas prioridades pueden ser seteadas por el programador.

- Contador de programa: Identifica la siguiente instrucción a ejecutar. Es importante tener esta entrada para continuar con la ejecución tras una interrupción por cambio de proceso o por operaciones de E/S. El SO siempre deja al proceso en ‘estado seguro’ antes de realizar un intercambio con otro proceso.

- Apuntadores de memoria: Delimitan el espacio que ha sido asignado al proceso. Al momento de la creación del proceso el SO le otorga un espacio en memoria donde es alojado.

- Información de D-E/S: Registra todos los dispositivos que están asignados al proceso. Esta información le permite al SO no reasignar un recurso ya asignado o darle prioridad al proceso para que libere un recurso crítico. Algunas implementaciones del SO pueden llevar el estado del recurso asignado en forma separada o incluido.

- Datos de contexto: Muestran la información contenida en los registros del procesador cuando el proceso está en ejecución. Si el proceso es interrumpido, estos datos deben almacenarse en el espacio de MP asignado a este proceso. Esta operación se lleva a cabo cuando se pasa el proceso a ‘modo seguro’.

- Información de contabilidad: Almacena las métricas del rendimiento del proceso. Por ejemplo: tiempo de ejecución, tiempo de espera, uso de un determinado recurso, etc. Estas variables le permiten al SO configurarse para darle a los procesos mejores tiempos de respuesta. Generalmente el SO se customiza en aspectos tales como planificación de procesador, técnicas de administración de memoria, esquemas de manejo de E/S, políticas de resolución de conflictos de interbloqueo, etc.

**Estados**

Modelo de 2 estados: Un proceso puede estar en *ejecución* o *no ejecución*. En *ejecución* se usa el procesador.

En las implementaciones más primitivas de los SO, el proceso se creaba y se ubicaba en estado *no* *ejecución*. Cuando el SO garantizaba que este proceso podría ejecutarse le asignaba el procesador colocándolo en estado *ejecución*. Posteriormente el proceso solo podría pasar a la salida. Esta salida se daba por causas normales o anormales.

Los SO evolucionados permitieron interrumpir al proceso en ejecución para encolarlo en una lista de procesos en no ejecución y asignar el procesador a otro proceso. Esta posibilidad se representa con la transición *interrumpir* y evidencia el concepto de multiprogramación.

Modelo de 5 estados: Se implementó para minimizar los tiempos desperdiciados del procesador por los procesos que se encuentran a la espera de un determinado suceso, el cual puede ser una operación de E/S, un resultado de otro proceso, la liberación de un determinado recurso, etc.

Inicialmente los procesos se crean y colocan en estado *nuevo*. En este punto se crea el BCP y el SO verifica si es posible admitirlo. La variable más importante analizada para la admisión es el espacio en MP para alojarlo. Una vez ubicado en la cola de *listos* participa por el recurso o por la asignación del recurso del procesador. El SO, por medio de un algoritmo, asigna el procesador a un proceso. Si el proceso termina se lo lleva a estado *terminado* a fin de registrar las métricas de ejecución y liberar los recursos que le fueron asignados. Si el proceso está en la espera de un determinado suceso, consumiría tiempos valiosos del procesador, por lo que se aparta al proceso de los productivos llevándolo al estado *bloqueado*, en donde se almacena con los datos de contexto. El proceso permanecerá allí hasta que suceda el evento que estaba esperando. Luego pasara al estado *listo* nuevamente para continuar con la ejecución cuando se le asigne el procesador nuevamente.

Modelo de 7 estados: Se le agrega al modelo anterior los estados *listo suspendido* y *bloqueado* *suspendido*, con el fin de liberar MP de los procesos no productivos ya que es frecuente que los sistemas se vean congestionados en almacenamiento principal.

Cuando el SO detecta una posible congestión, selecciona procesos bloqueados para transferirlos a memoria secundaria, colocándolos en estado *bloqueado suspendido*. Una vez que se dé el suceso que el proceso estaba esperando, se lo lleva a *listo suspendido*. Cuando el SO determina que la MP no está congestionada admite un nuevo proceso llevándolo a *listo*.

Este modelo tiene la desventaja de que se tarda mucho en pasar de MP a MS.

**Condiciones de creación de procesos**

Se crea un proceso en el arranque del sistema, en la ejecución de una llamada al sistema desde un proceso, cuando el usuario hace alguna petición para crearlo, o cuando se inicia un trabajo por lote.

**Condiciones normales y anormales de terminación de procesos**

Un proceso se termina cuando termino de ejecutarse, cuando hay una interrupción en su ejecución, si se descubre un error en el programa, o si el proceso ejecuta una llamada al sistema que indica al SO que elimine otros procesos.

**Hilos**

El proceso se puede dividir en hilo y proceso, siendo hilo la unidad que se ejecuta y proceso la unidad de propiedad de recursos.

Todo proceso tiene un hilo. Cuando se crea un proceso también se crea un hilo. Luego el hilo puede crear otros hilos dentro de este proceso.

Multihilo es la capacidad de un SO a dar soporte a múltiples hilos de ejecución en un solo proceso. En un entorno multihilo se define una unidad de asignación de recursos y una de protección. Sigue habiendo un único BCP y un espacio de direcciones de usuario asociado al proceso, pero hay varias pilas separadas para cada hilo y también un bloque de control para cada hilo.

En un proceso puede haber uno o más hilos. Cada uno cuenta con: un estado de ejecución, un contexto de hilo (contador de programa independiente), una pila de ejecución, almacenamiento estático para las variables locales, y acceso a la memoria y a los recursos de su proceso.

Todos los hilos de un proceso comparten el estado, los recursos, el espacio de direcciones y los datos. Se pueden sincronizar entre ellos para que no interfieran o corrompan estructuras.

Los estados de los hilos son ejecución, listo y bloqueado.

Los beneficios de trabajar con hilos están en el rendimiento: lleva menos tiempo crear un nuevo hilo, finalizarlo y cambiar de contexto. Se pueden sincronizar. También mejoran la eficiencia de la comunicación entre diferentes programas.

Las desventajas son que dificultan la programación y se necesita un HW que lo soporte.

En un uniprocesador, la multiprogramación permite el intercalado de múltiples hilos con múltiples procesos.

**U3. PLANIFICACION DE PROCESOS**

Planificación es la gestión el procesador realizada por los SO a través de distintas políticas y mecanismos. El objetivo de esto es dar un buen servicio a todos los procesos que existan en un determinado momento en el sistema. Es algo fundamental para los SO multiprogramas.

Al conmutar el procesador entre los procesos, el SO tiene mayor productividad del recurso informático.

Se utilizan 3 planificadores:

- Planificador de largo plazo (PLP): Se ocupa de las operaciones más generales y se encarga de cargar, gestionar y destruir los procesos y el BCP.

*Estados relacionados*: Nuevo y terminado.

- Planificador de corto plazo (PCP): Asigna los procesos al procesador. Se basa en un conjunto de algoritmos, técnicas y mecanismos de orden de cola. Este planificador se encuentra siempre en MP y debe proveer una selección justa y rápida del próximo proceso a ejecutar.

*Estados relacionados*: Listo, ejecución y bloqueado.

- Planificador de mediano plazo (PMP): Este planificador establece que procesos bloqueados serán suspendidos, por lo que determina cuando el almacenamiento principal se encuentra liberado para poder subir los procesos a MP nuevamente.

*Estados relacionados*: Bloqueado, bloqueado suspendido y listo suspendido.

**Métricas para el análisis de algoritmos de PCP**

Tiempo de servicio: Tiempo total de permanencia.

Tiempo de espera

Índice de servicio

Varía entre Cero y Uno. Cuando tiende a cero, nuestro sistema está condicionado por entradas y salidas. Cuando tiende a uno, decimos que el sistema está condicionado por el proceso.

**Políticas de trabajo**

No apropiativa: Los algoritmos no permiten intercambio entre procesos, es decir que cuando un proceso comienza la ejecución no libera al procesador hasta que termina en forma correcta o anormal.

Apropiativa: Determina que un proceso debe ser cambiado por otro, es decir que un nuevo proceso se apropia del procesador. Esto se llama cambio de contexto, el cual es regulado por el algoritmo y por el SO.

**Algoritmos para PCP**

• FIFO

Es de política no apropiativa.

Los procesos se ponen en cola a medida que llegan, por lo que el procesador ejecuta cada proceso en orden de llegada.

*Ventajas*:

- Simple y sencillo de implementar.

- Es predecible.

- Resulta justo aunque los procesos largos hacen esperar mucho a los cortos.

*Desventajas*:

- Pobre en su desempeño.

- El tiempo medio de espera suele ser largo.

- No sirve para SO de tiempo compartido.

• Round Robin

Es de política apropiativa.

El mecanismo de selección se basa en asignar un lapso de tiempo del procesador a cada proceso, el cual se conoce como quantum (q). Si este quantum toma el valor 1 se sobrecarga el SO y consume muchos recursos debido a los numerosos cambios de contexto. Y si es muy grande, el algoritmo se convierte en FIFO.

Tiene un manejo de cola FIFO. Si un proceso no concluye su ejecución en el quantum dado es removido al final de la cola y reemplazado por el siguiente proceso en cola. Si ingresa un proceso nuevo mientras hay otro, tiene prioridad el nuevo.

*Ventajas*:

- Se utiliza en tiempo compartido.

- El tiempo de espera crece de acuerdo al tiempo de ejecución de cada proceso.

- Ofrece un índice de servicio uniforme, sencillo, justo y de uso amplio.

*Desventajas*:

- Determinación del quantum más óptimo. Se debe tener en cuenta el tipo de SO, las cargas o procesos y su tipo.

- El tiempo de espera promedio suele resultar alto.

• SJN

Tiene características no apropiativas.

Este algoritmo elige de la lista de listos el proceso con menor cantidad de tiempo de ejecución, es decir que la cola se va ordenando de menor a mayor a medida que llegan los procesos.

*Ventajas*:

- El tiempo de espera aumenta en función a la longitud de los procesos, pero el tiempo medio de espera es óptimo.

- Presenta un buen tiempo de servicio.

- Apropiado para tareas por lote.

*Desventajas*:

- Poco predecible ya que es poco común poder determinar el tiempo de ejecución requerido del proceso.

• SRT

Tiene política apropiativa.

Compara el tiempo que le queda a cada proceso por ejecutar para determinar el que sigue. El manejo de cola consiste en ubicar a los procesos en orden ascendente en función al tiempo requerido y comparando el primero con lo restante del proceso en ejecución.

*Ventajas*:

- Mejora el tiempo de respuesta medio de los procesos largos.

- Excelente índice de servicio.

- Tiempo de espera bastante corto para todos los procesos.

- Muy eficiente.

*Desventajas*:

- Mayor sobrecarga.

- Es injusto si un proceso corto puede sacar a uno largo que se encuentre terminando.

• Prioridad

Puede ser apropiativo o no.

Resuelve ordenando los procesos en función a una prioridad asignada. Esta prioridad puede ser establecida por el SO y cada SO presenta su propia escala de prioridad. El SO define la prioridad en función a la formación interna del proceso (necesidad de memoria, numero de archivos abiertos, tiempo necesitado del procesador) o externamente en función al tipo de proceso o a las relaciones que tenga con otros procesos.

En la cola los procesos se ordenan de mayor a menor prioridad.

*Ventajas*:

- Fuertemente aplicable a situaciones particulares en las cuales se requiere una jerarquía de procesos. (SO de tiempo real).

*Desventajas*:

- Un proceso de baja prioridad sufre inanición, es decir que no existe otro proceso que lo bloquee, sino que por sus características se encuentra en desventaja con el resto de los procesos. Se mejora esto con la técnica de aumento o envejecimiento de prioridad, la cual aumenta la prioridad del proceso victima cuando cumple una cierta cantidad de tiempo de espera.

• HRN

Puede tener política apropiativa o no apropiativa. Si se resuelve no apropiativo, se debe evaluar la prioridad solo cuando termina el proceso que se está ejecutando. En apropiativo se evalúa cuando llega un nuevo proceso o cuando termina el que se está ejecutando.

Utiliza una fórmula de cálculo de prioridad que aumenta en función a la espera de los procesos y disminuye cuanto más tiempo en ejecución este el proceso.

Debido a que el cálculo de prioridad es variable no se implementa ningún manejo de cola.

*Ventajas*:

- Corrige las injusticias de SJN con los procesos largos y las de FIFO con los procesos cortos.

- Utiliza dos variables en la definición del próximo proceso a ejecutar.

*Desventajas*:

- Costoso y produce gran sobrecarga por los cálculos.

**Colas múltiples**

Divide la cola de procesos en múltiples colas. Los procesos serán asignados a cada cola dependiendo de su tipo y de sus necesidades. También se evaluaran sus prioridades y el tamaño requerido en memoria.

Cada cola presenta un algoritmo de planificación diferente que resulte conveniente al tipo de proceso que alberga.

Existe un algoritmo entre colas que determina la cola a utilizar en el procesador. Este algoritmo es apropiativo de prioridad fija.

**Colas múltiples con retroalimentación**

Divide a los procesos en varias colas jerárquicamente ubicadas por prioridad. Las colas de enumeración más baja son las de mayor prioridad.

El planificador multinivel estará definido por el número de colas, por el algoritmo de planificación para cada cola, el método o algoritmo para promover a un proceso de una cola a otra de menor prioridad, y podría considerarse un método para promover al proceso a una cola de mayor prioridad.

Generalmente el algoritmo entre colas es Round Robin asignando un determinado quantum que una vez cumplido indicara la promoción de un proceso a una cola de menor nivel. Es de política apropiativa.

**U4. CONCURRENCIA**

La condición de concurrencia o competencia se da entre dos procesos que acceden a un recurso compartido y el resultado depende del orden en que se ejecuten sus procesos. Un SO robusto y consistente no debería preocuparse por el orden de ejecución de los procesos.

Para analizar una condición de competencia, recurrimos al ejemplo del *Spool de impresión*. Este chequea periódicamente el spool para ver o comprobar la existencia de documentos para imprimir. El spool contiene 2 variables compartidas por todos los procesos del sistema: ‘In’ Indica la posición en la cual se debe incorporar un nuevo documento y ‘Out’ indica la posición del documento a imprimir.

*Análisis de la situación de concurrencia en el spool de impresión entre dos procesos A y B:*

Suponiendo que la ejecución de A requiere introducir un documento, lo hará leyendo y almacenando en una variable propia el valor de In = 7. Posteriormente coloca el documento en la posición 7. Como ha incorporado el documento, actualiza su variable local haciéndola igual a 8 (7+1). Al intentar sobrescribir el valor de in (Variable compartida) con su variable local, se genera un cambio de contexto. El control ahora es concedido al proceso B, que también requiere colocar un documento para impresión. B lee el valor de la variable compartida In, y la almacena en su variable local. Incorpora el documento y actualiza su variable local al valor de 8 (7+1). Posteriormente escribe el valor 8 en la variable compartida In. De esta manera el proceso B pisó el documento del proceso A. Cuando el control de ejecución retorna a A, continúa con la operación incompleta, escribiendo el valor 8 en la variable compartida In (que ya lo poseía). De esta manera, el sistema queda consistente.

**Sección critica**

Es la parte del código que utiliza un recurso compartido, el cual no puede ser ejecutado mientras otro proceso se encuentre en su SC por el mismo recurso. Esta limitante por la cual otros procesos no pueden ejecutar sus SCs se llama *exclusión mutua*, la cual se da solo cuando los procesos quieren acceder a un mismo recurso compartido.

Para garantizar la exclusión mutua, el SO implementa un conjunto de mecanismos o algoritmos de concurrencia, los cuales deben garantizar las siguientes condiciones:

1) Nunca dos procesos pueden encontrarse simultáneamente dentro de sus SCs.

2) No se debe realizar suposiciones sobre la velocidad de ejecución de los procesos ni del número de los procesadores.

3) Ningún proceso fuera de su SC debe impedir o bloquear a otros procesos ingresar en la SC.

4) Ningún proceso puede acceder a su SC en forma arbitraria o por sí solo.

**Mecanismos de concurrencia**

Los algoritmos de solución a analizar serán clasificados según la forma en que están construidos:

- De SW o HW

- De espera activa o espera no activa.

Los de *espera activa* son aquellos en los cuales los procesos que no pueden acceder a su SC esperan ciclando en nulo. Le corresponde el modelo de estado 2 con retroalimentación, 5 o 7.

Los de *espera no activa* son aquellos en los que los procesos que no pueden continuar con su ejecución son bloqueados a la espera de que el recurso quede liberado. Utiliza el modelo de estado 5 o 7.

• Deshabilitar interrupciones

Solución vía HW. Le permite al desarrollador apagar las interrupciones del SO. Las interrupciones son utilizadas para notificar eventos al SO y al procesador. Sin la concurrencia de interrupciones el proceso quedara como dueño del procesador hasta que ejecute una instrucción que las habilite o las prenda.

Entre otros usos, las interrupciones marcan las finalizaciones del tiempo de ejecución del proceso en el procesador.

*Fallas*:

- Se le concede los privilegios de administración al programador del proceso. Si este omite habilitar las instrucciones, no existirá evento por el cual el SO y el procesador se notifiquen de la culminación del proceso.

- La solución inhabilita las interrupciones de un solo procesador.

• Variables de candado

De espera activa. Utiliza una variable *candado* que es binaria para indicar si un proceso puede avanzar e ingresar a la SC. Si la variable se encuentra en 1, el proceso quedará ciclando en nulo hasta que esta cambie de valor.

El algoritmo presenta inicialmente una consulta sobre el valor de la variable de cierre. Si es 0, el proceso la hace valer 1 (cierra candado) y comienza la ejecución de su SC. Cuando el otro proceso llega a la condición de la variable de cierre, consulta si es igual a 0. Como esta puesta en 1 el procesador cicla en nulo. Cuando el control de ejecución retorna al primer proceso, este continúa con su ejecución hasta concluir con la SC y habilita el acceso de otro proceso colocando la variable en 0.

*Fallas*:

- Si se produce un cambio de contexto luego de evaluar la condición de la variable candado. Para el primer proceso la variable vale 0, pero justo antes de actualizarle el valor a 1 se da un cambio de contexto y el control pasa al otro proceso. Cuando este verifica el valor de la variable, como es 0 lo actualiza a 1 e ingresa a su SC. Supongamos que antes de concluir la SC se da un cambio de contexto y el control retorna al primer proceso. Este actualiza el valor de la variable a 1 ya que había verificado que estaba en 0, por lo que la sobrescribe y empieza la ejecución de su SC violando la primera condición de exclusión mutua.

• Alternancia estricta (o variable de turno)

De espera activa. La variable de *turno* inicialmente esta en 0, lo que indica que el turno le corresponde al proceso A. Este comienza la ejecución de su SC y cuando concluye permite el acceso del proceso B igualando la variable a 1. En este momento el proceso B puede acceder a su SC.

Debido a que las condiciones y asignaciones se encuentran cruzadas, las ejecuciones serán alternadas. El avance hacia la SC de un proceso depende del otro.

*Fallas*:

- Supongamos ahora que el proceso B concluye rápidamente su SC de modo de que ambos procesos se encuentren en su SNC. En este instante la variable de turno es 0. El proceso B concluye su ejecución de SNC antes que A y retorna en la condición de turno. En este punto viola la condición número 3 ya que el proceso A aún se encuentra ejecutando la SNC e impide que el proceso B ingrese a la suya.

• Solución de Peterson

Algoritmo de espera activa. Se basa en un código compartido para los procesos. Tanto el proceso 0 como el 1 deben invocar a la función enter\_region justo antes de ingresar a la SC, y deben invocar a la función leave\_region justo después de concluirla. Utiliza las variables compartidas turno y candado que toma de los algoritmos anteriores. El acceso a la SC está controlado por estas dos variables, en la que una depende del proceso y la otra de su adversario.

Esta solución es válida ya que si ambos procesos invocan en forma simultánea a enter\_region, uno de los dos pisará el valor de turno en última instancia y la doble condición impedirá su avance a la SC.

• Dormir y despertar

De espera no activa. El acceso a la variable *count* es inestricto y podría presentarse la siguiente situación:

Si el buffer está vacío y el consumidor acaba de leer el count para ver si es 0, en ese instante se planifica un cambio de contexto y se ejecuta el productor. Este coloca un elemento en el buffer, incrementa el valor del count y observa que ahora vale 1. Esto implicaba que antes valía 0, por ende el consumidor debería estar durmiendo así que el productor invoca un wakeup del consumidor. Desafortunadamente el consumidor aún no estaba dormido, por lo que la señal de despertar se pierde.

Cuando el control retorna al consumidor y este reanuda su ejecución se duerme ya que previamente había leído que el contador estaba en 0. El control retorna al productor, el cual llenara el buffer y también se dormirá. En este instante ambos procesos quedaran dormidos indefinidamente. En el caso contario también se da la falla.

*Falla*:

- Pierde las señales de dormir y despertar. Esto se resuelve con la técnica siguiente.

• Semáforos

Solución empleada por los SO modernos. Es de espera no activa.

Utiliza una variable *semáforo* que acumula las señales que desbloquean a un proceso dormido. Solo puede tomar valores desde cero al más infinito. Los semáforos se van a operar por medio de dos llamadas al SO que son down y up (instrucciones invisibles). La llamada de up incrementa el valor de la variable, mientras que down la decrementa. Si se intenta hacer un down de la variable con el valor cero, se bloquea al proceso que la intenta ejecutar.

La ejecución de esta variable es atómica, es decir que si asignamos en nuestro código una de las dos instrucciones, ejecutara el análisis del valor actual de la variable semáforo y la suma o resta del valor sin interrupción alguna por cambio de contexto ya que se hace en una sola acción invisible. De esta manera se garantiza siempre el valor correcto de la variable semáforo. Esta condición de atomicidad es indispensable para la solución de este tipo de problemas de condición de competencia.

La solución expuesta en el productor-consumidor utiliza dos semáforos para administrar la forma en que el buffer se va llenando. Al realizar un down(empty), siendo empty igual a 0, haría que el productor se bloquee a la espera de un up(empty) del consumidor. Esta condición se dará cuando el consumidor retire algún elemento del buffer. Por su parte, el consumidor no podrá realizar un down(full), siendo full igual a 0, hasta que el productor realice un up(full) luego de colocar un elemento.

El código también implementa un semáforo binario, el cual es un semáforo convencional pero una adaptación controlada por el código para que solo tome los valores 0 y 1. Este semáforo garantiza la exclusión mutua, realizando un down justo antes de ingresar a la SC y un up después de esta. Suponiendo que el proceso productor realiza un down(mutex), como este vale 1 lo coloca en 0 e ingresa a la SC. Si hay un cambio de contexto y se ejecuta el consumidor, al intentar realizar el down(mutex) se bloqueará y no accederá a la SC. Será despertado cuando el productor deje la SC y ejecute la señal up.

• Monitores

Utilizado en los lenguajes de programación de alto nivel, que con ayuda de semáforos garantizan la exclusión mutua.

Los monitores son contenedores de SCs. El código del programador contendrá instrucciones de SNCs y llamadas a código de SCs que se encuentran alojadas en el monitor. Al compilarse el programa, el compilador coloca semáforos para proteger la llamada del código de SC que está en el monitor.

• Instrucción TSL

Solución de SW con ayuda de HW. Válida para esquemas de múltiples procesadores. Bloquea el acceso de un procesador a un espacio de memoria. Implementado en SO paralelo y distribuido. Tiene instrucciones atómicas.

**U5. INTERBLOQUEO**

Se da ante la condición de competencia por un recurso en la cual ninguno de los procesos tiene suficientes criterios o atributos para continuar con la ejecución. Esta situación de espera entre los procesos suele extenderse en el tiempo haciendo que el sistema quede improductivo. En esta situación ninguno de los procesos puede concluir con su ejecución y los SO pueden llegar a implementar algún tipo de solución, aunque la mayoría la ignora.

No obstante, el SO si contabiliza la utilización de los recursos. Esta contabilidad la realiza por medio de 3 acciones: asignar recurso, utilizarlo, y liberarlo.

Para que ocurra el interbloqueo se tienen que cumplir 4 condiciones. El incumplimiento de al menos una concluiría en la no existencia de interbloqueo.

- *Exclusión mutua*: Existencia de al menos un recurso compartido por los procesos al cual solo uno puede acceder simultáneamente.

- *Posesión y espera*: Deben existir procesos que posean un recurso y se encuentren a la espera de otro, el cual debe estar en uso por otro proceso.

- *No apropiación*: Un recurso no puede ser expropiado de un proceso por otro en forma arbitraria.

- *Espera circular*: Cuando un proceso se encuentra a la espera de un recurso que tiene un siguiente proceso y este espera un recurso del siguiente proceso y así sucesivamente, hasta que el último proceso espera un recurso del primero.