UNIDAD 2 Administración de procesos y del procesador.

La gestión de procesos es la tarea fundamental de cualquier sistema operativo moderno. El sistema operativo debe asignar recursos a los procesos, permitir el intercambio de información entre los mismos, proteger los recursos de un proceso del resto y facilitar la sincronización de procesos. Para alcanzar estos objetivos, el sistema operativo mantiene una estructura de datos para cada proceso que describe su estado y los recursos que posee y que permite al sistema operativo imponer un control sobre los procesos (PCB Bloque de Control de Proceso).

En un monoprocesador multiprogramado, debe intercalarse en el tiempo, la ejecución de múltiples procesos. En un multiprocesador, no sólo puede intercalarse la ejecución sino que los procesos se pueden ejecutar simultáneamente. Ambos, intercalación y ejecución simultánea, son formas de concurrencia y llevan a una multitud de problemas complejos, tanto para el programador de aplicaciones como para el sistema operativo.

El sistema operativo debe llevar a cabo la función de planificar y ofrecer mecanismos para compartir y sincronizar procesos.

2.1 Concepto de proceso.

Programas.- Colección de instrucciones que el procesador interpreta y ejecuta, se almacenan en sistemas no volátiles necesitando ser cargados en memoria principal para poder ser ejecutados, se considera un ente estático.

Procesos.-Programa en ejecución, el sistema operativo les asigna recursos, Se consideran un ente dinámico.

El proceso es una abstracción creada por el SO, que se compone de:

- Código de Programa: **sección texto**
- Contexto de Ejecución: **PC**, **registros del procesador** y una **pila** para invocación de procedimientos
- **Sección de Datos**, que contiene variables globales
- Recursos del sistema.

Características

- Permite modularizar y aislar errores de programas durante su ejecución
- Soporta concurrencia de actividades, lo que permite un mejor aprovechamiento de los recursos
- Denominaremos como procesos tanto a los trabajos (jobs) en sistemas de lotes, como a las tareas (task) en sistemas de tiempo compartido

Modelo

La diferencia entre un programa (conjunto de instrucciones) y un proceso (instrucciones ejecutándose) es obvia y crucial para entender el funcionamiento de los SO.

Imaginemos un mecánico de autos en un taller donde se reparan carros con averías complejas en las que se hace necesario consultar el manual de cada modelo, que contiene las instrucciones para reparar cada posible avería. Además, se permiten reparaciones rápidas a las que se les da mayor prioridad. Existe un almacén de refacciones y herramientas suficientes para las reparaciones. Comparando esta situación con un sistema de cómputo se pueden establecer las siguientes analogías:

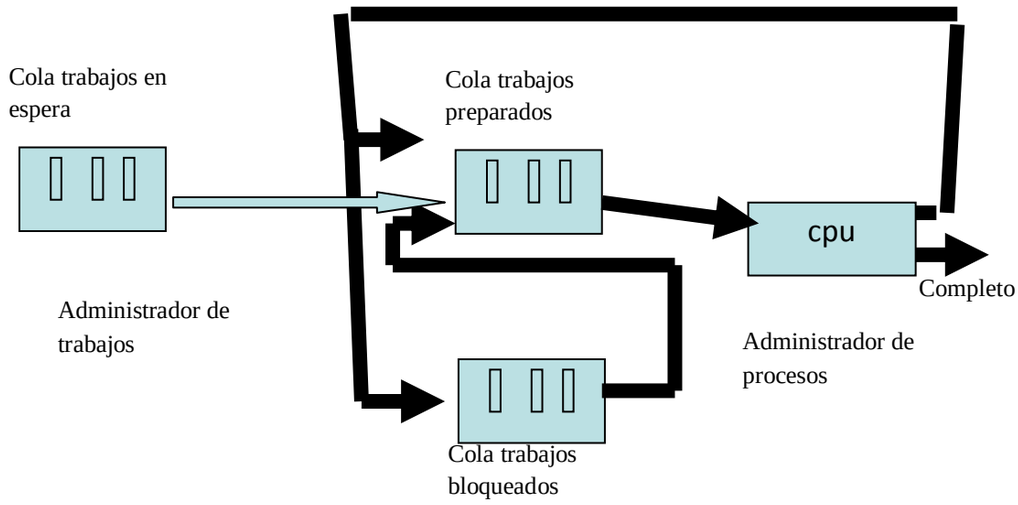
- El mecánico sería el CPU (procesador) que realizará el trabajo.
- El manual de cada reparación sería el programa.
- Las herramientas serían los recursos disponibles
- Las refacciones serían los datos.
- La actividad de usar las herramientas para desmontar las piezas defectuosas sustituyéndolas por otras nuevas siguiendo las instrucciones del manual equivaldría al proceso.

Suponiendo que en un momento dado el mecánico está realizando una reparación compleja (de las que llevan tiempo) y aparece un carro que solicita una reparación de las rápidas (ha aparecido una interrupción). El mecánico suspende momentáneamente la reparación compleja anotando en qué situación se queda dicha reparación y qué operación estaba realizando en ese momento (guarda el estado del proceso). Asimismo, sustituye el manual que estaba realizando por el de la nueva reparación que se dispone a realizar (cambio de programa). Comienza la nueva reparación (cambio de proceso), en la que las herramientas no serán las mismas que antes (distintos recursos); las indicaciones del usuario, las refacciones (datos) y las indicaciones del manual (programa) llevarán a feliz término la reparación para que el mecánico regrese a la reparación inicial.

Con este ejemplo se resalta que un proceso es una actividad que se apoya en datos, recursos, un estado en cada momento y un programa.

2.2 Estados y transiciones de los procesos

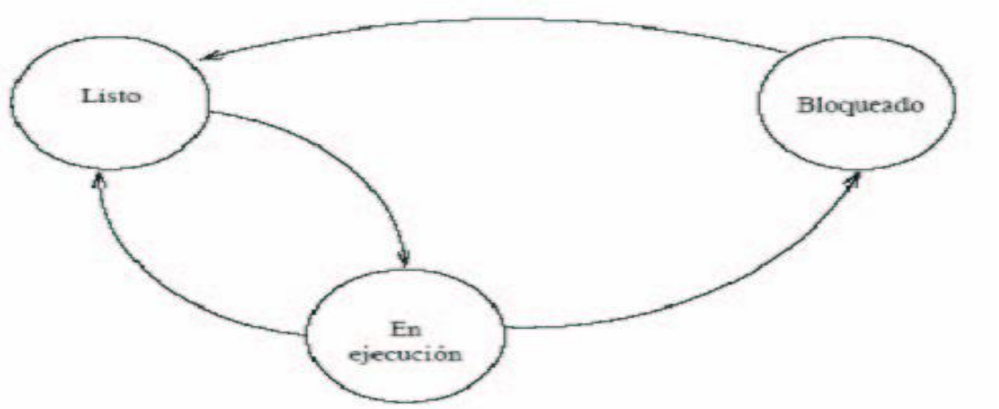
Los PCB’s se almacenan en listas, una para cada posible estado:



Los estados se pueden dividir en: Activos e inactivos

Activos los que compiten por el procesador. Tipos:

- **Ejecución**.- Cuando el proceso tiene el control del cpu
- **Preparado** (Listo).- Tienen las condiciones para ser ejecutados pero no están en ejecución por alguna causa.
- **Bloqueado**.- No pueden ejecutarse porque necesitan algún recurso no disponible



Inactivos.- No pueden competir por el cpu

Los 3 estados principales pueden no ser suficientes

Justificación:

Si todos los procesos están en bloqueados esperando un suceso y no hay memoria disponible para nuevos procesos, el procesador estará desocupado, sin uso. La Solución: procesos suspendidos:

• Permitir la ejecución de más procesos

• Ampliar la memoria principal

• Intercambio de procesos entre memoria y disco (swapping)

Surgen 2 nuevos estados de un proceso:

- **suspendido listo**: el proceso está suspendido, pero se encuentra listo para ejecutarse
- **suspendido bloqueado**: el proceso está suspendido y además está esperando que suceda un evento

• El sistema operativo puede poner en suspendido un proceso y transferirlo a disco

• El espacio liberado en la memoria principal es usado para traer otro proceso

¿Qué proceso elegir para cargar en memoria?

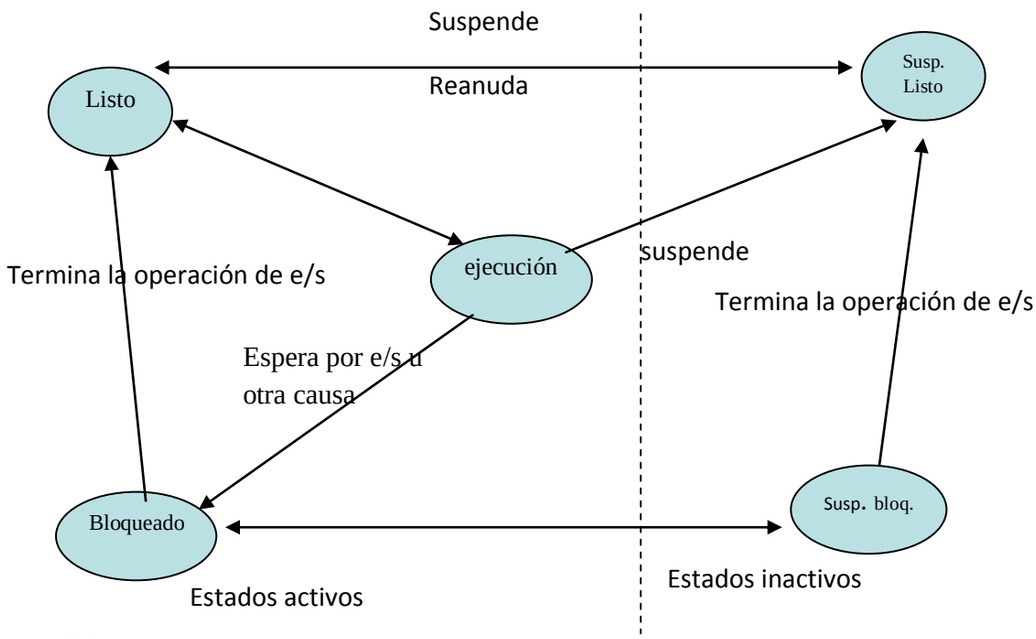
- Uno nuevo
- Uno previamente suspendido (debemos elegir los que se encuentran en suspendido listo y no en suspendido bloqueado)

Otras razones por las que un proceso puede pasar a estado suspendido:

• El sistema está en riesgo de fallo. El sistema suspende todos los procesos activos para poder corregir errores y volver a activarlos cuando el sistema funcione correctamente

• Un proceso sospechoso de mal funcionamiento puede ser suspendido hasta verificar su correcto funcionamiento

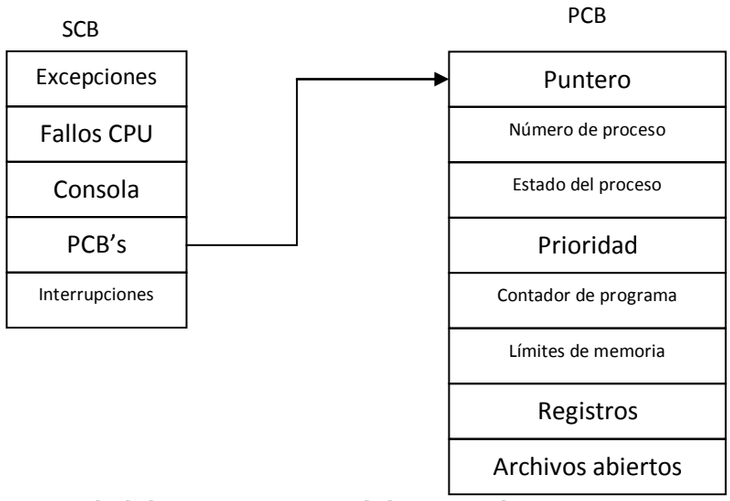
• El planificador puede suspender los procesos de baja prioridad en momento de carga excesiva del sistema



Control de procesos

PCB (Bloque de control de procesos)

Conjunto de datos donde se incluye el estado de un proceso en cada momento, los recursos usados, registros, etc.



Objetivos

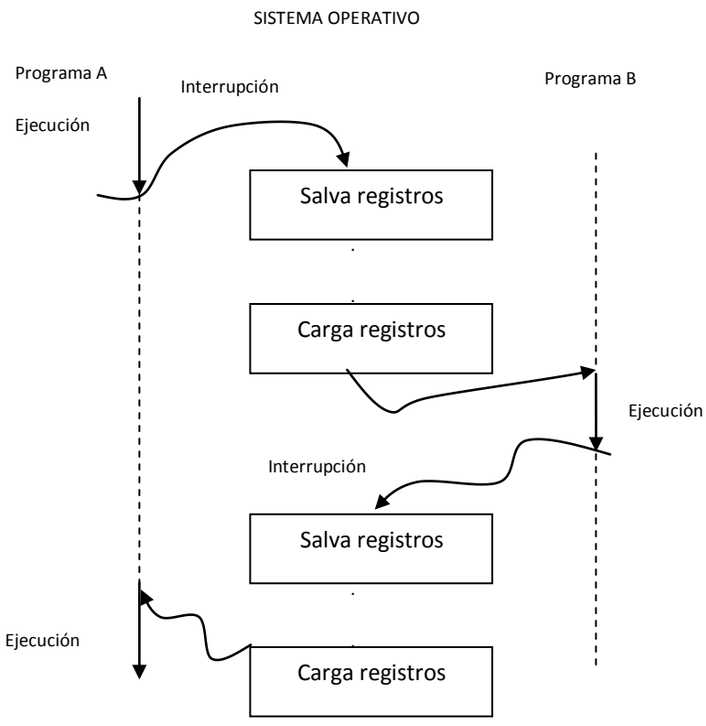
- Que el sistema operativo localice la información sobre el proceso
- Mantener registrados los datos del proceso en caso de suspensión o reanudación de la ejecución

Información contenida:

- **Estado del proceso**.- Contenido del contador de programa, estado del cpu en cuanto a prioridad del proceso, modo de ejecución, etc. y estado de los registros internos de la computadora.
- **Estadísticas de tiempo y ocupación de recursos**.- Planificación del CPU
- **Ocupación de memoria interna y externa**.- Swapping
- **Recursos en uso**
- **Archivos en uso**
- **Privilegios**

SCB (Bloque de control del sistema).- Objetivos globales similares al PCB, además enlaza los PCB's de los procesos existentes

Cambio de un proceso a otro suponiendo un solo cpu, una tarea en ejecución a la vez y 2 procesos activos (A y B) compitiendo por el cpu, ambos en memoria principal



Cambio de A por B

1. Deja de ejecutar el proceso en curso (A)
2. Se salva el estado del proceso A para su regreso al punto de interrupción
3. Cede el control al kernel
4. Cambio de contexto (de modo usuario a modo supervisor)
5. El kernel estudia si el proceso B está

preparado para su ejecución
6. Si si, cambio de contexto (de modo supervisor a modo usuario)
7. Repone el estado de B (si es que se había interrumpido antes)
8. Pone en ejecución a B

El cambio de contexto se da en caso de: ejecución de instrucción privilegiada, llamada al sistema operativo o interrupción

TRANSICIONES DE ESTADO.- Cambio de un estado a otro

Comienzo (Creación).- Un proceso comienza al ser dada la orden de ejecución insertándose en la lista de listos

Ejecución.- Cuando el CPU está inactivo y en la cola de listos haya un proceso en espera de ser ejecutado. (Pasa al CPU)

Bloqueado.- Un proceso en ejecución solicita una operación a un dispositivo, durante la espera el proceso se bloqueará. Su PCB se insertará en la lista de bloqueados.

Preparado.- 4 causas:

1 Orden de ejecución de un programa en espera
2 Si un proceso está bloqueado esperando una operación de entrada/salida y ésta termina.
3 Si un proceso está en ejecución y aparece una interrupción que fuerza al sistema operativo a ejecutar otro proceso, el 1º pasará a listo y su PCB se inserta en la lista de listos (Quantum, proceso de mayor prioridad)
4 **Activación**.- Un proceso suspendido listo pasará a listo al ser reactivado

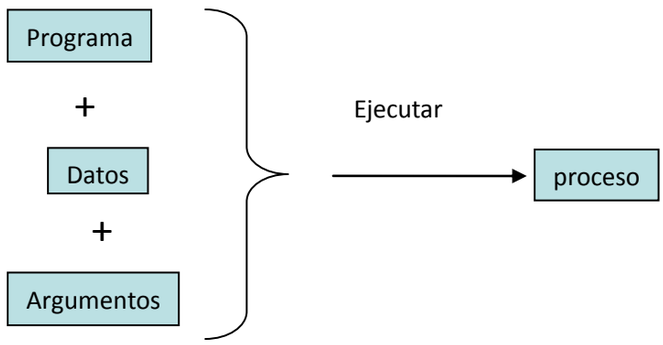
Suspendido bloqueado.- Un proceso está bloqueado y el sistema operativo lo suspende

Suspendido listo.- 3 causas:

1 Suspensión de un proceso listo
2 Suspensión de un proceso en ejecución
3 Desbloqueo de un proceso suspendido bloqueado al desaparecer la causa de su bloqueo

Operaciones sobre procesos

1.- Creación.- Se produce con la orden de ejecución del programa, usa argumentos (nombre, prioridad). En este momento aparece el PCB y es insertado en la lista de listos



Pasos que sigue el S.O.

1. Asignarle un PCB
2. Establecer espacio de direcciones de memoria
3. Cargar imagen (ejecutable) en memoria
4. Marcar la tarea como ejecutable

Tipos de creación:

Jerárquica.- cada proceso que se crea es hijo del proceso creador y hereda el entorno de ejecución del padre. El primer proceso que ejecuta un usuario será hijo del intérprete de comandos. Un proceso durante su ejecución puede crear varios procesos hijos a través de llamadas al sistema para creación de procesos. Al restringirse un proceso hijo a un subconjunto de recursos del padre, se evita que éste sature al sistema creando demasiados procesos hijos. Al crear procesos hijos, el padre continúa ejecutando concurrentemente con sus hijos o espera a que todos sus hijos hayan terminado y luego continúa él.

No jerárquica.- Cada proceso creado por otro se ejecuta independiente de su creador con un entorno independiente.

2.- Destrucción.- Orden de eliminación del proceso, el sistema operativo destruye el PCB y libera los recursos empleados

– Normalmente lo hace un antepasado directo (p.e. el proceso padre)
– Significa la terminación de toda su *"descendencia" (*terminación en cascada)
– **Pasos que sigue el S.O.**

1. Envío de datos del proceso finalizado al creador. (p.e. Código de finalización)
2. El SO desasigna los recursos que tiene

3.- Suspensión.- Paralización de un proceso que puede ser reanudado posteriormente, se aplica en ocasiones de mal funcionamiento o sobrecarga de trabajo (Guarda su PCB en disco)

4.- Reanudación.- Activación de un proceso suspendido (Reinserta el PCB en memoria)

5.- Cambio de prioridad.- Reasignación de un nuevo rango de prioridad
6.- Temporizar la ejecución.- Un proceso se ejecuta cada cierto periodo de tiempo, por etapas o de una vez
7.- Despertar un proceso.- Desbloqueo de un proceso
8.- Bloqueo.- puesta en espera de un proceso

Prioridades

Todo proceso por su importancia tiene necesidades de ejecución en cuanto a urgencia de recursos. No todos acceden de igual forma y con igual frecuencia al CPU debido a su prioridad

Tipos según quién las asigna:

1. **Asignadas por el sistema operativo**.- Se asigna al iniciar la ejecución y depende de los privilegios de su propietario
2. **Asignadas por el propietario**.- El usuario la asigna.

Tipos según su posibilidad de variación:

1. **Estáticas**.- No pueden ser modificadas durante la ejecución del proceso
2. **Dinámicas**.- Pueden ser modificadas en la ejecución respondiendo a eventos

Tipos de procesos

Según el uso:

1. **Reutilizables**.- Pueden cambiar los datos que usan, si vuelven a ejecutarse comienzan desde su estado inicial y procesar nuevos datos (programas de usuario) Están escritos con variables genéricas y cada uno introducirá los valores de las variables genéricas que precise.
2. **Reentrantes**.- No tienen asociados datos, sólo código puro. Los datos que usan están en los registros internos y no se modifican durante su uso. (programas del sistema operativo)

Según la utilización de memoria:

1. **Procesos residentes.-** Durante su vida activa tiene que estar cargado en memoria.
2. **Procesos intercambiables**.- Pueden ser llevados de memoria principal a disco mientras estén bloqueados. La memoria liberada puede ser reasignada

Según el acceso a los recursos

1. **Apropiativos**.- Al tener asignado un recurso no permiten que otro proceso lo use hasta que ellos lo liberen
2. **No apropiativos**.- Permiten a otros procesos acceder a recursos usados por ellos

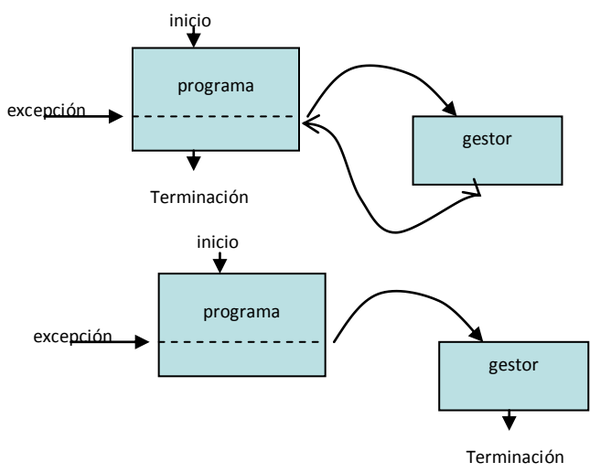
Excepciones

Durante la ejecución puede haber fallos que el sistema operativo debe controlar: hardware, de software, datos incorrectos, anomalías, etc.

Gestor de excepciones.- parte del sistema operativo que controla las excepciones

Tipos según la gravedad:

- **Catastróficos**.- Imposibilitan el funcionamiento del sistema y no hay modo de recuperarlo (apagón)
- **No recuperables**.- Sin afectar al sistema, hacen que el proceso no pueda continuar (div/0)
- **Recuperables**.- Con algunos ajustes el proceso puede continuar (datos incorrectos)



Tipos de tratamiento:

- Tratamiento de la excepción y continuación del proceso
- Tratamiento de la excepción y terminación del proceso

2.3 Procesos ligeros (Hilos o hebras).

Un **hilo de ejecución**, en sistemas operativos, es similar a un proceso en que ambos representan una secuencia simple de instrucciones ejecutada en paralelo con otras secuencias. Los hilos permiten dividir un programa en dos o más tareas que corren simultáneamente, por medio de la multiprogramación. En realidad, este método permite incrementar el rendimiento de un procesador de manera considerable. En todos los sistemas de hoy en día los hilos son utilizados para simplificar la estructura de un programa que lleva a cabo diferentes funciones. Todos los hilos de un proceso comparten los recursos del proceso. Residen en el mismo espacio de direcciones y tienen acceso a los mismos datos. Cuando un hilo modifica un dato en la memoria, los otros hilos utilizan el resultado cuando acceden al dato. Cada hilo tiene su propio estado, su propio contador, su propia pila y su propia copia de los registros de la CPU. Los valores comunes se guardan en el bloque de control de proceso (PCB), y los valores propios en el bloque de control de hilo (TCB).

Un ejemplo de la utilización de hilos es tener un hilo atento a la interfaz gráfica (iconos, botones, ventanas), mientras otro hilo hace una larga operación internamente. De esta manera el programa responde más ágilmente a la interacción con el usuario.

Diferencias entre hilos y procesos

Los hilos se distinguen de los tradicionales procesos en que los procesos son generalmente independientes, llevan bastante información de estados, e interactúan sólo a través de mecanismos de comunicación dados por

el sistema. Por otra parte, muchos hilos generalmente comparten otros recursos directamente. En sistemas operativos que proveen facilidades para los hilos, es más rápido cambiar de un hilo a otro dentro del mismo proceso, que cambiar de un proceso a otro. Este fenómeno se debe a que los hilos comparten datos y espacios de direcciones, mientras que los procesos al ser independientes no lo hacen. Al cambiar de un proceso a otro el sistema operativo (mediante el dispatcher) genera lo que se conoce como overhead, que es tiempo desperdiciado por el procesador para realizar un cambio de modo (mode switch), en este caso pasar del estado de Running al estado de Waiting o Bloqueado y colocar el nuevo proceso en Running. En los hilos como pertenecen a un mismo proceso al realizar un cambio de hilo este overhead es casi despreciable.

2.4 Concurrencia y secuenciabilidad.

Concurrencia y secuenciabilidad

La concurrencia comprende un gran número de cuestiones de diseño, incluyendo la comunicación entre procesos, competencia por los recursos, sincronización de la ejecución de varios procesos y asignación del tiempo de procesador a los procesos y es fundamental para que existan diseños como *Multiprogramación, Multiproceso y Proceso distribuido*

Los procesos son concurrentes si existen simultáneamente 2 o más y llegan al mismo tiempo a ejecutarse.

La concurrencia puede presentarse en tres contextos diferentes:

• Varias aplicaciones: (multiprogramación) para permitir que el cpu sea compartido entre varios trabajos

• Aplicaciones estructuradas: Como ampliación del diseño modular y la programación estructurada, algunas aplicaciones pueden implementarse eficazmente como un conjunto de procesos concurrentes.

• Estructura del sistema operativo: Las mismas ventajas de estructuración son aplicables a los sistemas operativos que están implementados como un conjunto de procesos.

Tipos de computadora en los que puede haber concurrencia:

• Multiprogramación con un CPU. El sistema operativo se encarga de repartir el CPU entre los procesos, intercalando su ejecución para dar una apariencia de ejecución simultánea.

• Multiprocesador. Máquina formada por más de un CPU que comparten memoria principal. Los procesos no sólo intercalan su ejecución sino también la superponer.

• Multicomputadora. Es una máquina de memoria distribuida formada por una serie de computadoras, es posible la ejecución simultánea de los procesos en los diferentes CPU's.

La concurrencia será aparente siempre que el número de procesos sea mayor que el de procesadores disponibles y será real cuando haya un proceso por procesador (Paralelismo).

Pros:

• Facilita la programación de aplicaciones: permite que se estructuren como un conjunto de procesos que cooperan entre sí para alcanzar un objetivo común.

•Acelera los cálculos: Dividiendo una tarea en varios procesos, ejecutándolos en "paralelo".

• Posibilita el uso interactivo a múltiples usuarios que trabajan de forma simultánea.

• Permite un mejor aprovechamiento de los recursos, en especial del CPU, ya que pueden aprovechar las fases de entrada-salida de unos procesos para realizar las fases de procesamiento de otros.

Contras:

• Inanición e interrupción de procesos

• Ocurrencia de bloqueos

• Que 2 o más procesos requieran el mismo recurso

Tipos de procesos concurrentes:

Proceso independiente: El que se ejecuta sin cooperación de otros. Ejemplo: varias ventanas de una misma aplicación de forma simultánea.

Procesos cooperantes: Los que están diseñados para trabajar conjuntamente, deben comunicarse e interactuar. (Aplicaciones en red)

Tipos de interacción:

• Motivada porque los procesos comparten o compiten por el acceso a recursos. Ejemplo: dos procesos independientes compiten por el acceso a disco o para modificar una base de datos.

• Motivada porque los procesos se comunican y sincronizan entre sí para alcanzar un objetivo común. Ejemplo: compilador con varios procesos que trabajan conjuntamente para obtener un solo archivo de salida.

Aspectos de un sistema operativo para gestionar la concurrencia.

1. Debe seguir la pista de los distintos procesos activos, por medio de PBC's

2. Debe asignar y quitar recursos a cada proceso activo:

• Tiempo de procesador.

• Memoria: (virtual, swapping)

• Archivos

• E/S:

3. Debe proteger datos y recursos de cada proceso contra injerencias no intencionadas de otros procesos.

2.4.1 Exclusión mutua de secciones criticas.

Regiones críticas

Parte de un programa, en la cual se intenta el acceso a recursos compartidos

Tipos de procesos

• Los procesos no conocen a los demás, son independientes, no trabajan juntos.

• Los procesos conocen indirectamente a otros: no los conocen necesariamente, pero comparten algunos

recursos.

• Los procesos conocen a otros: se comunican y trabajan conjuntamente en una misma actividad.

Competencia de procesos por los recursos

Hay conflicto cuando los procesos compiten por el mismo recurso al mismo tiempo y el sistema operativo asignará el recurso a uno y el resto tendrá que esperar, el que quede esperando se retrasará, se bloqueará y en el peor de los casos nunca terminará bien

Ejemplo: Un Sistema Operativo debe asignar un identificador de proceso (PID) a dos procesos en un sistema multiprocesador. Si se realiza esta acción en dos procesadores a la vez sin control, se puede asignar el mismo PID a dos procesos distintos. Este problema se debe a que la asignación de PID es una sección crítica que debe ejecutarse en forma atómica, de forma completa e indivisible y ningún otro proceso podrá ejecutar dicho código mientras el primero no haya acabado su sección.

Debe haber sincronización que permita a los procesos cooperar entre ellos sin problemas protegiendo el código de la región crítica:

• Cada proceso debe solicitar permiso para entrar en la sección crítica mediante código.

• Cuando un proceso sale de la sección crítica debe indicarlo mediante código. Esto permitirá que otros procesos entren a ejecutar la sección crítica.

Requisitos:

• Exclusión mutua: Si un proceso está ejecutando código de la sección crítica, ningún otro proceso lo podrá hacer.

• Progreso: Si ningún proceso está ejecutando dentro de la sección crítica, se elegirá alguno de los que desean entrar.

• Espera acotada: Debe haber un límite en el número de veces que se permite que los demás procesos entren a ejecutar código de la sección crítica después de que un proceso haya efectuado una solicitud de entrada y antes de que se conceda la suya.

Exclusión mutua

Operación de control que permite la coordinación de procesos concurrentes, prohibiendo a otros procesos realizar una acción cuando un proceso haya obtenido el permiso.

Involucra al sistema operativo (quien asigna recursos), y a procesos, que deben expresar los requisitos de exclusión mutua, como puede ser bloqueando los recursos antes de usarlos.

Problemas:

• Interbloqueo. Se tienen dos procesos 1 y 2 y dos recursos críticos, R1 y R2. Cada proceso necesita acceder a ambos recursos para llevar a cabo una parte de su función, es posible que: el sistema operativo asigna R1 a 1 y R2 a 2. Cada proceso está esperando uno de los dos recursos. Ninguno liberará el recurso que ya tiene hasta que adquiera el otro y ejecute su sección crítica. Ambos procesos están interbloqueados.

• Inanición. Tres procesos, 1, 2 y 3, necesitan acceder periódicamente al recurso R. 1 tiene el recurso, 2 y 3 espera. Cuando 1 deja su sección crítica, 2 y 3 pueden acceder a R. Se concede acceso a 3 y antes que termine su sección crítica, 1 solicita acceso de nuevo. Se concede el acceso a 1 después de que 3 termine y si 1 y 3 se conceden el acceso repetidamente el uno al otro, se puede negar indefinidamente a 2 el acceso.

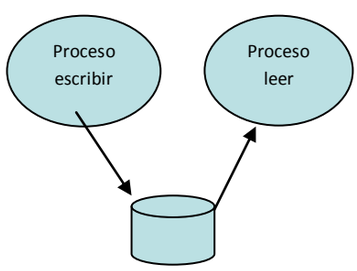
Requisitos para la exclusión mutua.

1. Solo un proceso, de los que tienen regiones críticas por el mismo recurso, debe tener permiso para entrar en ella en un instante dado.
2. Un proceso no debe poder solicitar acceso a una sección crítica para después ser demorado indefinidamente; no puede permitirse el interbloqueo o la inanición.
3. Cuando ningún proceso está en su sección crítica, cualquier proceso que solicite entrar en la suya debe poder hacerlo sin dilación.
4. Un proceso permanece en su sección crítica solo por un tiempo finito.

Ejemplo de exclusión mutua:

Registro:

Clave	Nombre	Paterno	Materno	Dirección



Para que un registro sea válido debe estar totalmente actualizado (consistente)

2 o más procesos pueden efectuar operaciones de lectura, pero escritura solo uno debe hacerlo y ninguna otra operación.

2 situaciones:

Sin sincronización.- Puede ser que Escribir esté actualizando un registro y sin terminar le sorprenda el cambio de proceso, terminará en el siguiente turno de CPU. Con el cambio le tocará turno a Leer, que leerá el registro obteniendo datos inconsistentes

Con sincronización.- Mediante un mecanismo que impida la lectura de un registro (bloqueo) a un proceso mientras Escribir realice operaciones. Leer al usar el CPU y solicitar lectura sobre el registro lo encontrará bloqueado y quedará en espera de que Escribir termine para hacer su lectura

2.4.2 Sincronización de procesos en S.C.
2.4.2.1 Mecanismo de semáforos. 2.4.2.2 Mecanismo de monitores.

Sincronización

Coordinación para llevar a cabo el trabajo de un grupo de procesos cooperantes asegurando el acceso a recursos compartidos. Previene y/o corrige errores debidos a estos accesos.

Para que los procesos puedan sincronizarse debe disponerse de servicios que permitan bloquear o suspender la ejecución de un proceso.

Algoritmos:

- **Espera Activa**:
  Establecen la espera de entrada a la RC con un ciclo que será roto en el momento en que se cumpla una condición. El proceso no queda bloqueado y el sistema se sobrecarga.
  - **Espera con mutex:** Usa una variable switch con dos operaciones atómicas:
    • lock: bloqueo. Si el switch ya está bloqueado por otro proceso, el proceso que realiza la operación esperará. En caso contrario se bloquea el mutex sin poner en espera al proceso.
    • unlock: Desbloquea el switch. Si existen procesos esperando, activará a uno de ellos que será el nuevo proceso que adquiera el switch. La operación unlock sobre un mutex debe ejecutarla el proceso que adquirió con anterioridad el mutex mediante lock.
    (Ejemplo de cabina telefónica)
    3 Alternancia.- Usa una variable de turnos (lista de espera)
- **Espera no activa**.- Establecen la espera para entrar a una RC bloqueando el proceso.
  - **Semáforos**: Variable entera usada como contador de peticiones de entrada a RC compartida por todos los procesos, gestiona el tráfico de procesos. Cuando un proceso intenta entrar en una RC mientras otro accede a los recursos compartidos, se bloqueará igual que cuando un proceso accede a un recurso ocupado. Se usa en sistemas con memoria compartida. Se le puede asignar un valor inicial 1 y sólo se puede acceder utilizando dos operaciones atómicas: wait (-1) y signal (+1). El algoritmo de uso es el siguiente:
    Si el semáforo es menor o igual que cero, cualquier operación wait que se realice sobre el semáforo bloqueará al proceso. Si el semáforo es positivo, cualquier proceso que ejecute una operación wait no se bloqueará.
    El valor que tiene que tomar el semáforo inicialmente es 1, de esta forma solo se permite a un único proceso acceder a la sección crítica. Si el valor inicial del semáforo fuera, por ejemplo, 2, entonces dos procesos podrían ejecutar la llamada wait sin bloquearse y por tanto se permitiría que ambos ejecutaran de forma simultánea dentro de la sección crítica
  - **Monitores**.- En los métodos anteriores el programador debe proporcionar de modo explícito el modo de sincronización. El monitor termina con esto, pero debe ser soportado por el lenguaje correspondiente. Encapsula el código relativo a un recurso compartido en un solo módulo, así todos los accesos estarán forzados a usar dichas funciones: mantenimiento más simple, menos errores de programa.
    Los monitores son estructuras de datos utilizadas en lenguajes de programación para sincronizar dos o más procesos o hilos de ejecución que usan recursos compartidos. En el estudio y uso de los semáforos se puede ver que las llamadas a las funciones necesarias para utilizarlos quedan repartidas en el código del programa, haciendo difícil corregir errores y asegurar el buen funcionamiento de los algoritmos. Para evitar estos inconvenientes se desarrollaron los monitores. El concepto de monitor fue definido por primera vez por Charles Antony Richard Hoare en un artículo del año 1974. La estructura de los monitores se ha implementado en varios lenguajes de programación, incluido Pascal concurrente, Modula-2, Modula-3 y Java, y como biblioteca de programas.
    Componentes.-
    - Inicialización: contiene el código a ser ejecutado cuando el monitor es creado
    - Datos privados: contiene los procedimientos privados, que sólo pueden ser usados desde dentro del monitor y no son visibles desde fuera
    - Procedimientos del monitor: son los procedimientos que pueden ser llamados desde fuera del monitor.
    - Cola de entrada: contiene a los threads que han llamado a algún procedimiento del monitor pero no han podido adquirir permiso para ejecutarlos aún.

    Exclusión mutua en un monitor
    Los monitores están pensados para ser usados en entornos multiproceso o multihilo, y por lo tanto muchos procesos o threads pueden llamar a la vez a un procedimiento del monitor. Los monitores garantizan que en cualquier momento, a lo sumo un thread puede estarse ejecutando dentro de un monitor. Ejecutar dentro de un monitor significa que sólo un thread estará en estado de ejecución mientras dura la llamada a un procedimiento del monitor. El monitor hace cumplir la exclusión mutua implícitamente, de modo que sólo un procedimiento esté siendo ejecutado a la vez. De esta forma, si un thread llama a un procedimiento mientras otro thread está dentro del monitor, se bloqueará y esperará en la cola de entrada hasta que el monitor quede nuevamente libre.
    Para que resulten útiles en un entorno de concurrencia, los monitores deben incluir algún tipo de forma de sincronización. Por ejemplo, supóngase un thread que está dentro del monitor y necesita que se cumpla una condición para poder continuar la ejecución. En ese caso, se debe contar con un mecanismo de bloqueo del thread, a la vez que se debe liberar el monitor para ser usado por otro hilo. Más tarde, cuando la condición permita al thread bloqueado continuar ejecutando, debe poder ingresar en el monitor en el mismo lugar donde fue suspendido. Para esto los monitores poseen variables de condición que son accesibles sólo desde adentro

2.4.3 Interbloqueo (DeadLock).

Análisis

En un conjunto de procesos, cada uno está esperando un evento que sólo otro proceso del conjunto puede causar. Puesto que todos los procesos están esperando, ninguno de ellos puede causar ninguno de los eventos que podrían despertar a cualquiera de los demás miembros del conjunto, y todos los procesos continúan esperando indefinidamente.

Ejemplo: Una carretera en 2 direcciones tiene un puente que sólo deja pasar vehículos en un sentido, con las siguientes situaciones:

Un auto llega al puente y en sentido contrario no hay nadie y puede cruzar

Si el paso es controlado por un semáforo en cada lado y 100 m antes de cada uno hay detectores de autos para encender el semáforo opuesto en rojo, pude pasar que 2 autos lleguen al mismo tiempo en ambos sentidos y ambos estén detenidos por el semáforo (interbloqueo)

Si no hay semáforos, un conductor cede el paso, pero antes de que termine el otro, aparece un 3º y así sucesivamente, puede hacer que el conductor educado no cruce mientras haya carros en sentido contrario (postergación indefinida)

RECURSOS.- elemento que un programa o proceso puede usar en la computadora donde se ejecuta, y puede ser usado por un solo proceso en un momento dado.

Tipos

Reutilizables.- El que puede ser usado con seguridad por un proceso y no se agota con el uso, son liberados para que otros los reusen: CPU, canales de E/S, memoria y estructuras de datos archivos, bases de datos y semáforos.

Consumibles.- El que puede ser creado (producido) y destruido (consumido). No hay límite en la cantidad. Un proceso productor que no está bloqueado puede liberar cualquier número de recursos consumibles. Cuando un proceso adquiere un recurso, éste deja de existir: señales, mensajes, e información en buffers.

Operaciones

- Solicitud
- Uso
- Liberación

Condiciones para que se dé un interbloqueo (deben estar todas)

1. Condición de exclusión mutua. Cada recurso está asignado únicamente a un solo proceso o está disponible.

2. Condición de retener y esperar. Los procesos que actualmente tienen recursos que les fueron otorgados previamente pueden solicitar nuevos recursos.

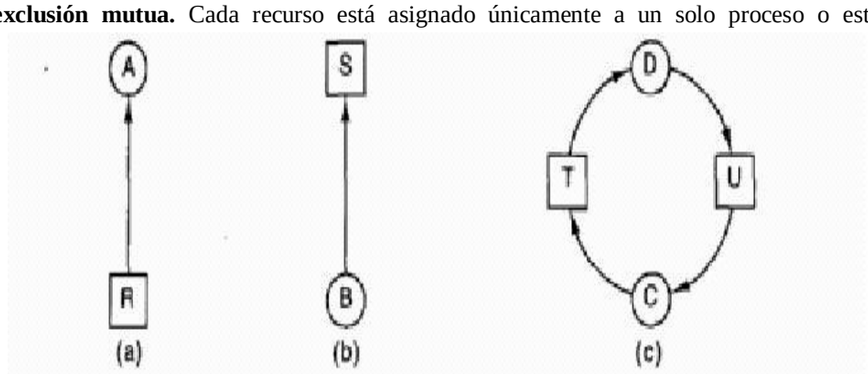


Figura 3-7. Grafos de asignación de recursos. (a) Retención de un recurso. (b) Petición de un recurso. (c) Bloqueo mutuo.

3. Condición de no expropiación. No es posible quitarle por la fuerza a un proceso los recursos que le fueron otorgados previamente. El proceso que los tiene debe liberarlos

4. Condición de espera circular. Debe haber una cadena circular de dos o más procesos, cada uno de los cuales está esperando un recurso retenido por el siguiente miembro de la cadena.

Además:

Número finito de recursos y procesos.

Un proceso puede pedir tantos recursos como necesite siempre que no exceda los existentes

2.4.3.1 Prevención. 2.4.3.2 Detección. 2.4.3.3 Recuperación.

Prevención

A grandes rasgos, se debe diseñar un sistema de manera que esté excluida, a priori, la posibilidad de interbloqueo. Los métodos para prevenir el interbloqueo son de dos tipos.

Indirectos.- Consisten en impedir la aparición de alguna de las tres primeras condiciones necesarias

Directos.- Consisten en evitar la aparición de la condición 4.

Exclusión Mutua.- En general, no puede anularse. Si el acceso a un recurso necesita exclusión mutua, el sistema operativo debe soportar la exclusión mutua.

Retención y Espera.- Se previene exigiendo que todos los procesos soliciten todos los recursos que necesiten a un mismo tiempo y bloqueando el proceso hasta que todos los recursos puedan concederse simultáneamente. Es ineficiente porque:

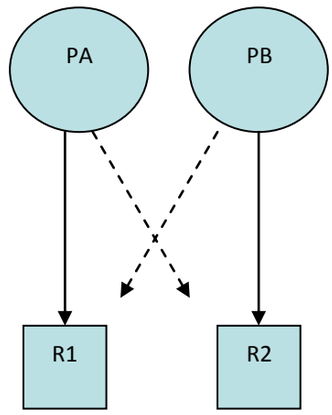
- Un proceso puede estar detenido durante mucho tiempo, esperando que se concedan todas sus solicitudes de recursos, cuando de hecho podría haber avanzado con sólo algunos de los recursos.

- Los recursos asignados a un proceso pueden permanecer sin usarse durante mucho tiempo, tiempo durante el cual se priva del acceso a otros procesos.

No expropiación.- Se previene:

1 Si a un proceso que retiene ciertos recursos se le deniega una nueva solicitud, dicho proceso deberá liberar sus recursos anteriores y solicitarlos de nuevo, cuando sea necesario, junto con el recurso adicional.

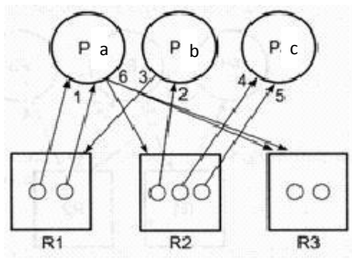
2 Si un proceso solicita un recurso que actualmente está retenido por otro proceso, el sistema operativo puede expulsar al segundo proceso y exigirle que libere sus recursos. Este último esquema evitará el interbloqueo sólo si no hay dos procesos que posean la misma prioridad, es práctico sólo cuando se aplica a recursos cuyo estado puede salvarse y restaurarse más tarde de una forma fácil, como es el caso de un procesador.



Círculo Vicioso de Espera.- Puede prevenirse definiendo una ordenación lineal de los tipos de recursos. Si a un proceso se le han asignado recursos de tipo R, entonces sólo podrá realizar peticiones posteriores sobre los recursos de los tipos siguientes a R en la ordenación. Para comprobar el funcionamiento de esta estrategia, se asocia un índice a cada tipo de recurso. En tal caso, el recurso R1, antecede a R2, en la ordenación si i < j. Suponga que dos procesos A y B se interbloquean, porque A ha adquirido R1, y solicitado R2, mientras que B ha adquirido R2; y solicitado R1. Esta situación es imposible porque implica que í < j y j < i. Puede ser ineficiente, retardando procesos y denegando accesos a recursos innecesariamente.

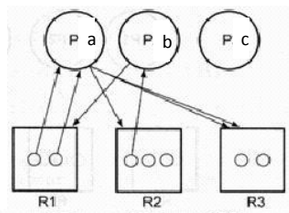
Detección y Recuperación

La detección no limita ni restringe como la prevención. Se concederán recursos que los procesos necesiten cuando sea posible. El sistema operativo ejecuta un algoritmo que permite detectar la condición de espera circular: Cuando un recurso se solicita o libera, se determina si contiene algún ciclo. Si se encuentra uno, se termina uno de los procesos del ciclo. Si esto no rompe el bloqueo mutuo, se termina otro proceso, continuando así hasta romper el ciclo. Ejemplo:



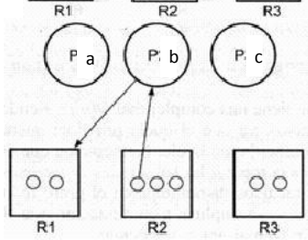
En este diagrama el proceso Pa tiene asignados los 2 recursos de R1, Pb tiene asignado un recurso de R2 y Pc tiene asignados 2 recursos de R2. Después Pa pide un recurso de R2 y 2 de R3, y Pb pide un recurso de R1, provocando así un interbloqueo.

Aplicando el algoritmo se hace una reducción en Pc quedando entonces:



A diferencia de la figura anterior se puede apreciar que Pc fue "reducido", se le retiran los recursos que estaba usando para ver si se elimina el interbloqueo. Pero como podemos ver, aún no es eliminado.

Se aplica de nuevo el algoritmo.



Con esta última iteración el interbloqueo es eliminado, dejando solamente a Pb usar los recursos que necesite y así poder liberarlos para que los procesos interrumpidos puedan tomarlos.

La revisión puede hacerse en cada solicitud de recursos o según la probabilidad de que haya un ciclo

Ventajas de verificación frecuente:

Pronta detección

Desventajas:

Consume un tiempo de procesador considerable.

Formas de recuperación:

1. Abandonar todos los procesos bloqueados. Esta es una de las soluciones más comunes adoptadas en un sistema operativo.

2. Retroceder cada proceso interbloqueado hasta algún punto de control definido previamente y volver a ejecutar todos los procesos. Puede repetirse el interbloqueo original e implica mecanismos de retroceso y reinicio

3. Abandonar sucesivamente los procesos bloqueados hasta que deje de haber interbloqueo. Seleccionando procesos según algún criterio. Después de abandonar cada proceso, se debe ejecutar de nuevo el algoritmo de detección para ver si todavía existe interbloqueo.

4. Apropiarse de recursos sucesivamente hasta que deje de haber interbloqueo. Seleccionando procesos según un criterio y ejecutando el algoritmo de detección después de cada apropiación. Un proceso que pierde un recurso por apropiación debe retroceder hasta un momento anterior a la adquisición de ese recurso. **Criterios de selección**.

Escoger el proceso con:
• La menor cantidad de tiempo de procesador consumido
• El mayor tiempo restante estimado
• El menor número total de recursos asignados
• La prioridad más baja
Algunas de estas cantidades son más fáciles de medir que otras. Además de las medidas de prioridad, no existe otra indicación del "costo" para el usuario frente al costo para el sistema en conjunto.
Mecanismos para evitarlo
El interbloqueo se evita analizando con detenimiento cada petición de recurso para ver si se puede satisfacer sin peligro, hay algún algoritmo que siempre pueda evitar el interbloqueo tomando la decisión correcta en todos los casos? **SI**, pero sólo si se cuenta con información por adelantado. (Algoritmo del banquero)
Estado seguro.- Un estado es seguro si existe una secuencia de otros estados que conduzca a una situación en la que todos los clientes obtienen préstamos hasta sus límites de crédito (todos los procesos obtienen todos sus recursos y finalizan).
Algoritmo del banquero para recursos múltiples.- Este modelo es difícil de aplicar al caso general de un número arbitrario de procesos y un número arbitrario de clases de recursos, cada una con múltiples ejemplares (p. Ej., 2 cd-rom, 3 unidades de cinta, etc.). No obstante, el algoritmo del banquero puede generalizarse para este fin:

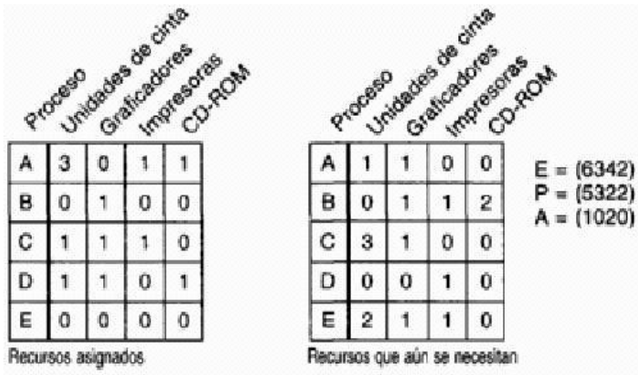


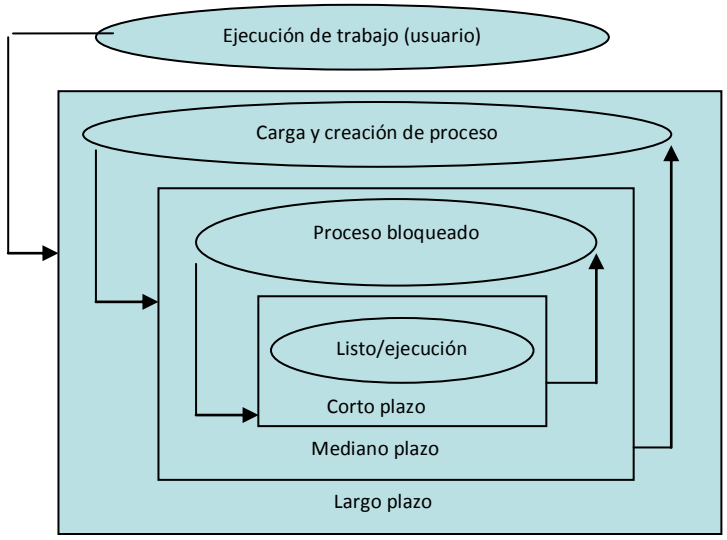
Figura 3-13. El algoritmo del banquero con múltiples recursos.

La matriz izquierda muestra cuántos ejemplares de cada recurso se han asignado actualmente a cada uno de los cinco procesos. La derecha muestra cuántos recursos necesita cada proceso para poder finalizar. Los procesos deben expresar sus necesidades de recursos totales antes de ejecutarse, a fin de que el sistema pueda calcular la matriz de la derecha en cada paso. Los vectores muestran los recursos existentes: E, los entregados: P y los disponibles: A. Por E podemos ver que el sistema tiene 6 unidades de cinta, 3 graficadores, 4 impresoras y 2 unidades CD-ROM. De éstos, 5 cintas, 3 graficadores, 2 impresoras y 2 CD-ROM están asignados. Esto puede verse sumando las 4 columnas de recursos en la matriz izquierda. El vector de recursos disponibles A es la diferencia entre lo que el sistema tiene y lo que se está usando.
Algoritmo para verificar si un estado es seguro o no.
1. Busque un renglón R, cuyas peticiones de recursos no se hayan otorgado y sean en todos los casos menores o iguales que A. Si no existe tal renglón, el sistema tarde o temprano llegará a un bloqueo mutuo porque ningún proceso podrá ejecutarse hasta finalizar.
2. Suponga que el proceso de la fila elegida solicita todos los recursos que necesita (lo cual se garantiza que es posible) y finaliza. Marque ese proceso como terminado y agregue todos sus recursos al vector A.
3. Repita los pasos 1 y 2 hasta que todos los procesos se marquen como terminados, en cuyo caso el estado inicial era seguro, o hasta que ocurra un bloqueo mutuo, en cuyo caso no lo era. Si hay varios procesos que pueden escogerse en el paso 1, no importa cuál se seleccione: la reserva de recursos aumentará o, en el peor de los casos, seguirá igual.

TABLA 5.1 Resumen de los enfoques de detección, prevención y predicción del interbloqueo en los sistemas operativos [ISLO80]

Principio	Política de Reparto de Recursos	Esquemas	Ventajas Principales	Desventajas Principales
Prevención	Conservadora; los recursos están poco ocupados.	Solicitud de todos los recursos a la vez	• Funciona bien con procesos que realizan una sola ráfaga de actividad. • No es necesaria la apropiación.	• Ineficiencia. • Retrasos en el inicio de los procesos.
		Apropiación	• Conveniente cuando se aplica a recursos cuyo estado se puede salvar y restaurar fácilmente.	• Apropia más habitualmente de lo necesario. • Sujeto a reinicios cíclicos.
		Ordenación de recursos	• Aplicación factible por medio de chequeos durante la compilación. • No hace falta procesamiento durante la ejecución, pues el problema se resuelve durante el diseño del sistema.	• No permite las solicitudes incrementales de recursos. • Poco uso de la apropiación.
Detección	Muy liberal; los recursos solicitados se conceden cuando es posible.	Comprobación periódica del interbloqueo.	• Nunca retrasa el inicio de un proceso • Facilita el manejo en línea	• Pérdidas inherentes a la apropiación.
Predicción	A medio camino entre la detección y la prevención.	Manipulación para encontrar al menos un camino seguro.	• No es necesaria la apropiación.	• Deben conocerse las demandas futuras de recursos. • Los procesos pueden

2.5 Niveles, objetivos y criterios de planificación.
Niveles de planificación.



- **Planificación de largo plazo**.- (Planificación de trabajos), determina a qué trabajos se les permite entrar al sistema, cuál es el próximo trabajo que se va a ejecutar. Existe en los sistemas por lotes donde la decisión se basa en las necesidades de recursos y su disponibilidad. En los sistemas de tiempo compartido tiene como misión cargar los programas que se desea ejecutar en memoria, es por tanto el encargado de crear los procesos.
- **Planificación de mediano plazo**.- (Planificación de Swapping), determina a qué proceso se le permite competir por el CPU. Suspende y/o activa temporalmente procesos para mantener una operación uniforme en el sistema y ayuda a realizar algunas funciones para optimizar el rendimiento del sistema.
- **Planificación de corto plazo**.- (Planificación de CPU), determina a qué proceso deberá asignarse el CPU (despachar). Esta operación se realiza muchas veces por segundo, por lo que el despachador debe estar permanente en memoria.

Objetivos de la planificación

- **Justicia.-** Sin favorecer o perjudicar procesos
- **Máxima capacidad de ejecución**.- Realizar los trabajos lo más rápido posible. Minimizar los cambios de procesos
- **Máximo número de usuarios interactivos.-** Simultáneos
- **Predecibilidad**.- Saber en todo momento cómo será la ejecución.
- **Mínima sobrecarga.-** A menor sobrecarga mayor velocidad. Minimizar los cambios de contexto
- **Equilibrio en el uso de recursos**.- Que estén ocupados equitativamente el mayor tiempo posible
- **Seguridad de las prioridades.-** Ejecutar más pronto los de más alta prioridad
- Muchos de estos objetivos están en conflicto unos con otros, esto hace que la planificación sea un problema complejo.

Criterios de planificación.

- **Tiempo de respuesta**.- Velocidad con que la computadora responde a una petición, depende mucho de la velocidad de los dispositivos E/S.
- **Tiempo de servicio**.- Tiempo que tarda en ejecutarse un proceso, desde su carga en memoria, espera en la lista de listos, ejecución en CPU y operaciones e/s.
- **Tiempo de ejecución**.- Tiempo de servicio menos la espera en la lista de listos, o sea, el tiempo teórico que necesitaría el proceso para ejecutarse si fuera el único.
- **Tiempo de CPU**.- Tiempo que un proceso usa el CPU sin contar el tiempo de bloqueado.
- **Tiempo de espera**.- Tiempo en que el proceso está activo pero sin ser ejecutado (listas)

- **Eficiencia**.- Que el cpu siempre esté ocupado para lograr un buen rendimiento.
- **Rendimiento**.- Número de procesos realizados por unidad de tiempo, mientras mayor, mejor.

Medidas

- **t**.- Tiempo que un proceso P necesita estar en ejecución para hacer su tarea.
- **ti**.- Instante en que el usuario da la orden de ejecución del proceso
- **tf**.- Instante en que el proceso termina la ejecución
- **Tiempo de servicio** (T): T=tf-ti **Tiempo de espera** (E): E=T-t
- t=20 ti=1 tf=30 T=30-1=29 E=29-20=9
- **Índice de servicio**.- relación para evaluar la actuación de la política establecida. Representa la relación de tiempo de ejecución y tiempo de vida
- I=t/T I=20/29=0.68
- Si es un solo proceso, cuando I tiende a 1 el proceso está limitado por cpu, si tiende a 0 está limitado por e/s. Si son varios procesos deben sacarse valores promedio:
- Tiempo medio de servicio, de espera y eficiencia (índice medio de servicio)
- **Tiempo del núcleo**.- Tiempo consumido por el kernel para tomar decisiones del planificador de procesos, incluido el tiempo de cambio de contexto y proceso

Tiempo de inactividad (Idle).- Tiempo consumido cuando la lista de listos está vacía.

2.6 Técnicas de administración del planificador.

Planificación del CPU

Ciclo de ráfaga del CPU y de E/S.

El éxito de la planificación del CPU depende de la siguiente prioridad observada de los procesos: la ejecución de un proceso consiste en un ciclo de ejecución del CPU y de E/S, y los procesos se alternan entre estos dos estados. La ejecución del proceso se hace alternando una ráfaga de CPU y una ráfaga de E/S. La última ráfaga de CPU terminará con una solicitud al sistema para que concluya la ejecución.

Planificador del CPU.

Siempre que el CPU queda inactivo, el sistema operativo debe seleccionar para su ejecución uno de sus procesos de la lista de listos. La selección es revisada por el planificador a corto plazo,

Estructura de planificación.

Las decisiones de planificación del CPU pueden efectuarse cuando un proceso cambia:

- De ejecución a bloqueado
- De ejecución a listo
- De bloqueado a listo
- Cuando termina.

Algoritmos de planificación

Los procesos que se asignan al cpu son tomados de la lista de listos. Esta lista se alimenta de 2 puntos:

- Cuando un usuario inicia la ejecución de un programa, el planificador a largo plazo recibe la orden de ejecución, crea el proceso y lo pasa al planificador a corto plazo.
- Cuando un proceso deja de estar en ejecución y no hay causas de bloqueo, o deja de estar bloqueado.

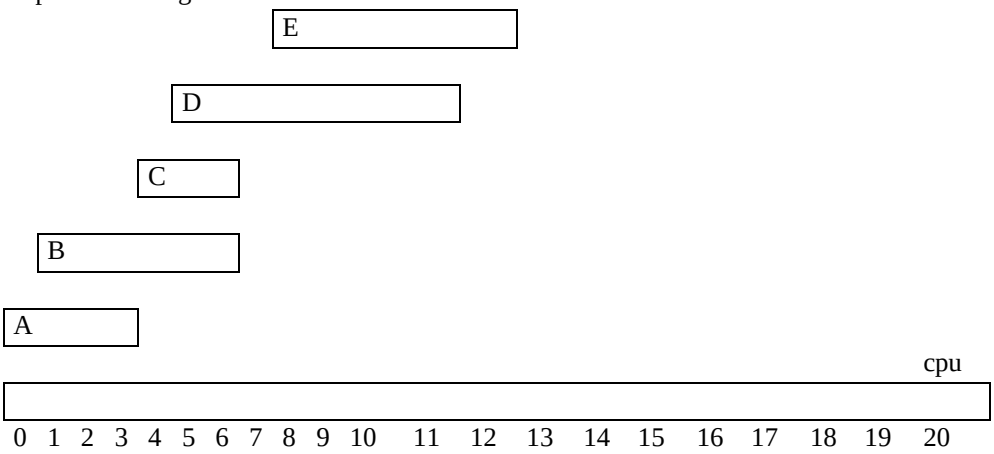
Políticas de planificación:

- Apropiativas.- Producen un cambio de proceso con cada cambio de contexto; el proceso que usa el cpu puede ser suspendido y permitir el acceso al cpu a otro proceso. (Tiempo compartido y tiempo real)
- No apropiativas.- Un proceso no abandona nunca el procesador desde su comienzo hasta su fin. (Por lotes).

Para las diferentes políticas hay diversos algoritmos (ninguno perfecto), para su estudio nos basaremos en los siguientes procesos:

Proceso	Instante de llegada	Tiempo de ejecución	Prioridad
A	0	3	0
B	1	5	1
C	4	2	0
D	5	6	2
E	8	4	1

Representación gráfica

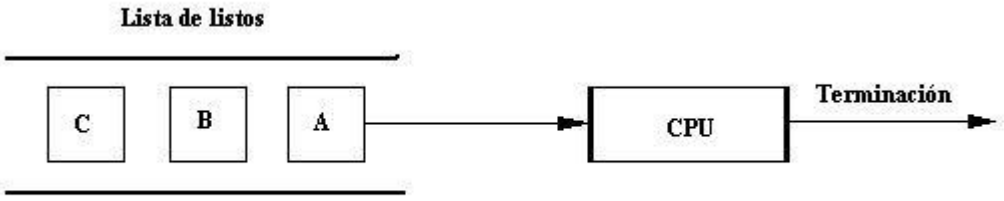


Se da el tiempo de entrada al sistema, el tiempo que cada uno necesita para ejecutarse si fuera el único, se supone que no tienen operaciones e/s para facilitar su estudio

2.6.1 FIFO

Primero en llegar primero en ser servido (FCFS)

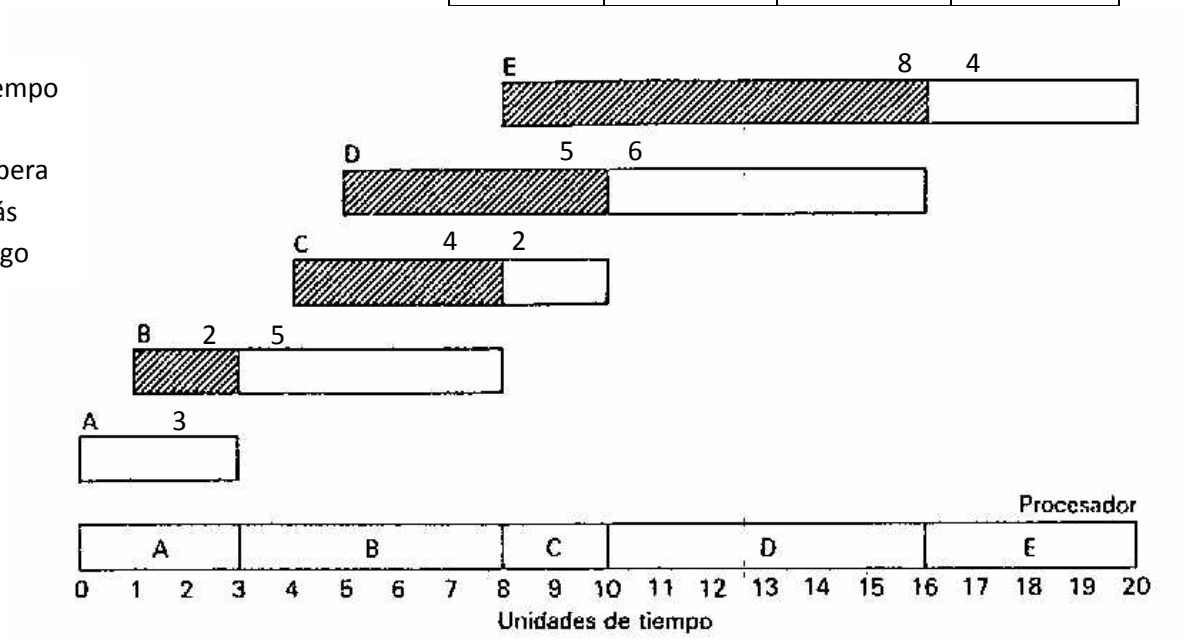
EL cpu ejecuta cada proceso hasta que termina, los procesos que entran a la lista de listos estarán formados en el orden en que llegan hasta que les toque turno (FIFO).



Es una política muy simple y muy pobre.

proceso	instante de llegada (ti)	tiempo de ejecución (t)	instante de fin (tf)	T (tiempo de servicio tf-ti)	E (tiempo de (espera T-t)	I (índice de (servicio t/T)
A	0	3	3	3	0	1.00
B	1	5	8	7	2	0.71
C	4	2	10	6	4	0.33
D	5	6	16	11	5	0.55
E	8	4	20	12	8	0.33
			Promedios	7.80	3.80	0.59

Tiempo de espera más largo



El tiempo de espera de cada proceso depende del número de procesos que están en la lista de listos al momento de su ejecución y del tiempo que cada uno tenga en uso el cpu, y es independiente de las necesidades de ejecución del propio proceso

Características: No apropiativa, Justa y Predecible.

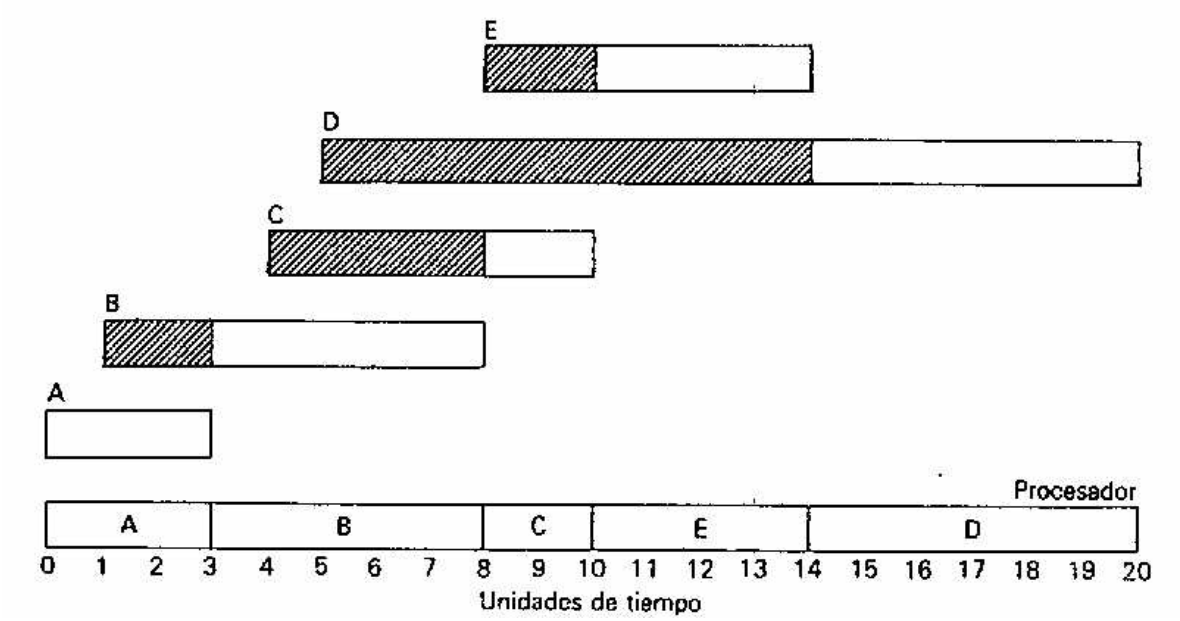
2.6.2 SJF

Siguiente proceso el más corto (SJN)

Toma de la lista de listos el proceso que necesite menos tiempo de ejecución para realizar su trabajo. Debe saberse el tiempo de cpu que necesita cada proceso por medio de: información suministrada por el usuario, por el programa, por experiencia.

El tiempo de servicio T es bueno para procesos cortos y malo para procesos largos.

Proceso	**instante de llegada (ti)**	**tiempo de ejecución (t)**	**instante de fin (tf)**	**T (tiempo de servicio tf-ti)**	**E (tiempo de espera T-t)**	**I (índice de servicio t/T)**
A	0	3	3	3	0	1.00
B	1	5	8	7	2	0.71
C	4	2	10	6	4	0.33
D	5	6	20	15	9	0.40
E	8	4	14	6	2	0.67
			Promedios	7.4	3.4	0.62

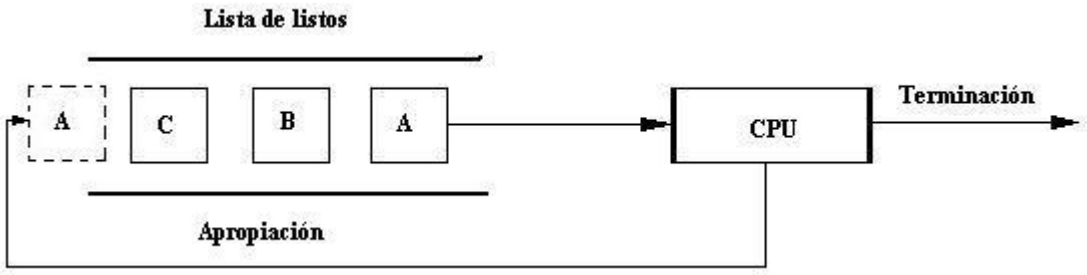


Características: No apropiativa, El tiempo de espera aumenta según la longitud de los procesos, Es poco predecible, No es justa con los procesos largos, Buen tiempo de servicio, Es difícil de implementar por los datos necesarios para la planificación

2.6.3 RR

Round Robin (RR)

Asignación cíclica, es una mejora de FCFS. A cada proceso se le da un determinado tiempo q (quantum), si no termina, se forma en la lista de listos, el cpu es para el siguiente proceso y así hasta que termine la ejecución.

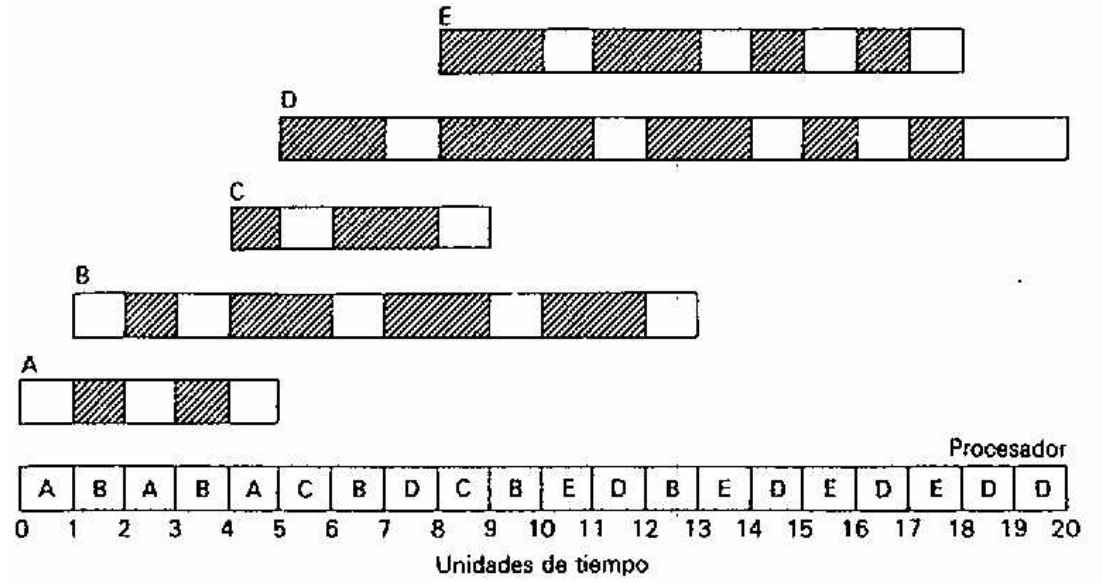


La administración de la lista de listos puede ser FIFO o por prioridades. Variando q se tienen diferentes comportamientos. Si q>tiempo de ejecución se convertiría en FCFS, si q tiende a 0, la sobrecarga sería muy grande, la mayor parte del tiempo se usaría en cambios de contexto.

proceso	instante de llegada (ti)	tiempo de ejecución (t)	instante de fin (tf)	T (tiempo de servicio tf-ti)	E (tiempo de (espera T-t)	I (índice de (servicio t/T)
A	0	3	5	5	2	0.60
B	1	5	13	12	7	0.42
C	4	2	9	5	3	0.40
D	5	6	20	15	9	0.40
E	8	4	18	10	6	0.40
			Promedios	9.40	5.40	0.44

Condiciones:

- q=1
- Si un proceso termina antes que su q, se le concede el cpu a otro proceso con q completo
- Al crearse un proceso se forma al final de listos
- Si un proceso se crea en el mismo momento que un q termina, se supone que dicho proceso se formó en listos antes que la expiración de q



Características:

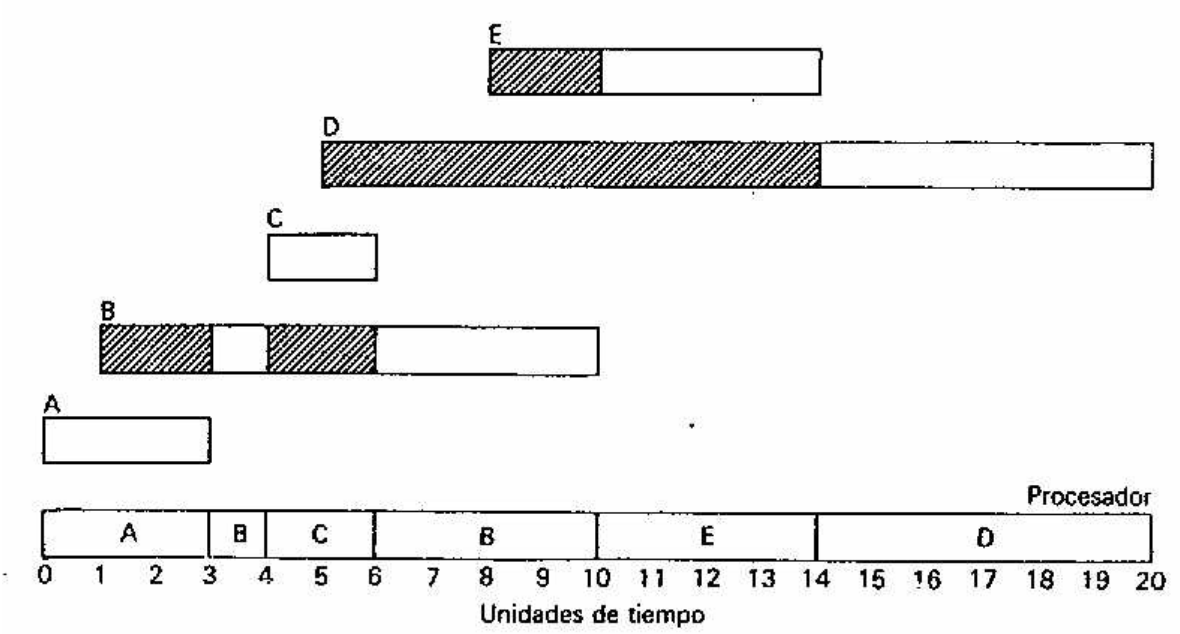
- Baja sobrecarga si el cambio de contexto es eficiente y los procesos siempre están en memoria principal
- El tamaño óptimo de q depende de: Las cargas que vaya a soportar el sistema y La cantidad de procesos
- Es la más usada en sistemas de tiempo compartido
- Es apropiativa

Opcionales

Siguiente proceso el de tiempo restante más corto (SRT)

Mezcla de RR y SJN. Cambia el proceso que está en el cpu por el proceso con una exigencia de tiempo total menor. El tiempo de respuesta para los procesos largos mejora. El tiempo de espera es corto para la mayoría de los procesos.

proceso	instante de llegada (ti)	tiempo de ejecución (t)	instante de fin (tf)	T (tiempo de servicio tf-ti)	E (tiempo de (espera T-t)	I (índice de (servicio t/T)
A	0	3	3	3	0	1.00
B	1	5	10	9	4	0.56
C	4	2	6	2	0	1.00
D	5	6	20	15	9	0.40
E	8	4	14	6	2	0.67
			Promedios	7.0	3.0	0.72



Características: Apropiativa, Puede ser injusta porque un proceso corto puede quitar a uno largo, Tiene un buen tiempo medio de servicio, Eficiente

Highest Response Ratio Next (HRN)

Siguiente el de más alto índice de respuesta (HRN)

Trata de corregir las injusticias de SJN con los procesos largos y de FCFS con los procesos cortos. Se hace variable la prioridad de los procesos calculándola así: P=(w+t)/t

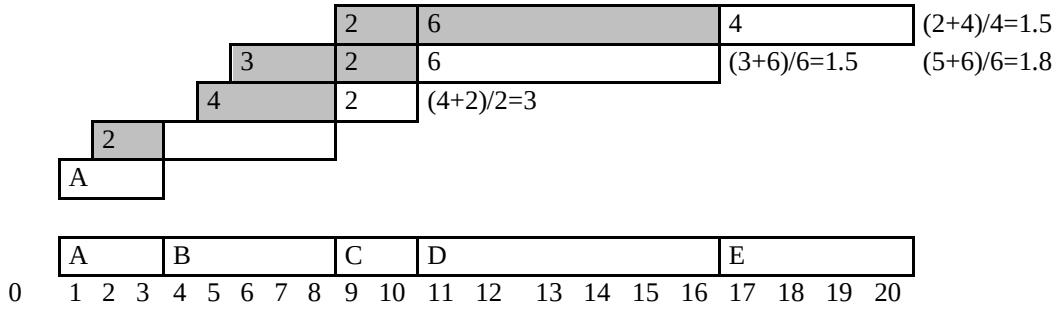
P=prioridad w=tiempo de espera en listos t=tiempo de ejecución

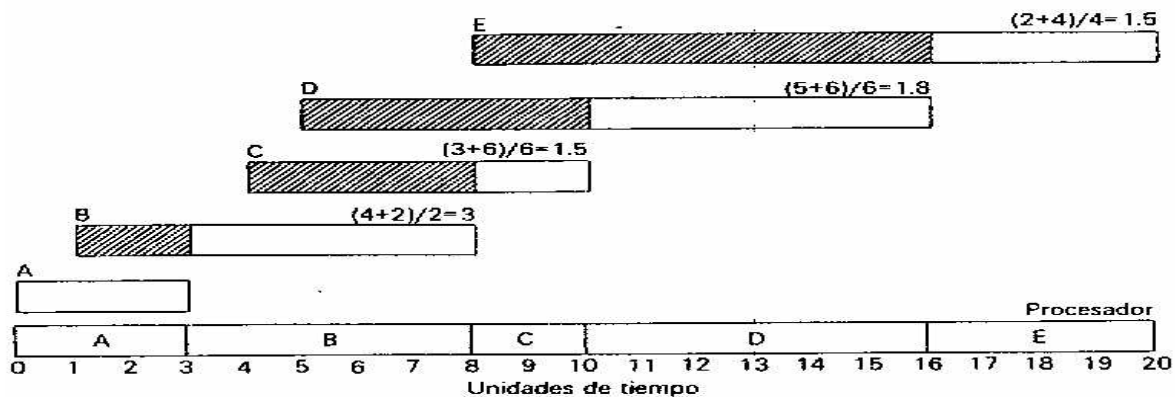
Al principio P=1 e irá subiendo su valor a medida que el proceso esté en listos (w favorece a procesos largos), e irá disminuyendo cuanto más tiempo esté en ejecución (t favorece a los cortos). Cuando un proceso abandona el cpu, por haber terminado o por una petición, el proceso listo con mayor prioridad (el que haya esperado más) será el seleccionado para el cpu.

Desventajas:

- Si se ejecuta un proceso corto inmediatamente después de que uno largo haya comenzado a usar el CPU deberá esperar mucho tiempo
- Es muy costosa de implementar, la prioridad debe calcularse para todos los procesos en espera cada vez que termine el proceso que usa el cpu y se sobrecarga el sistema, pero es justa y no apropiativa

proceso	**instante de llegada (ti)**	**tiempo de ejecución (t)**	**instante de fin (tf)**	**T (tiempo de servicio tf-ti)**	**E (tiempo de (espera T-t)**	**I (índice de (servicio t/T)**
A	0	3	3	3	0	1.00
B	1	5	8	7	2	0.71
C	4	2	10	6	4	0.33
D	5	6	16	11	5	0.55
E	8	4	20	12	8	0.33
			Promedios	7.8	3.8	0.59



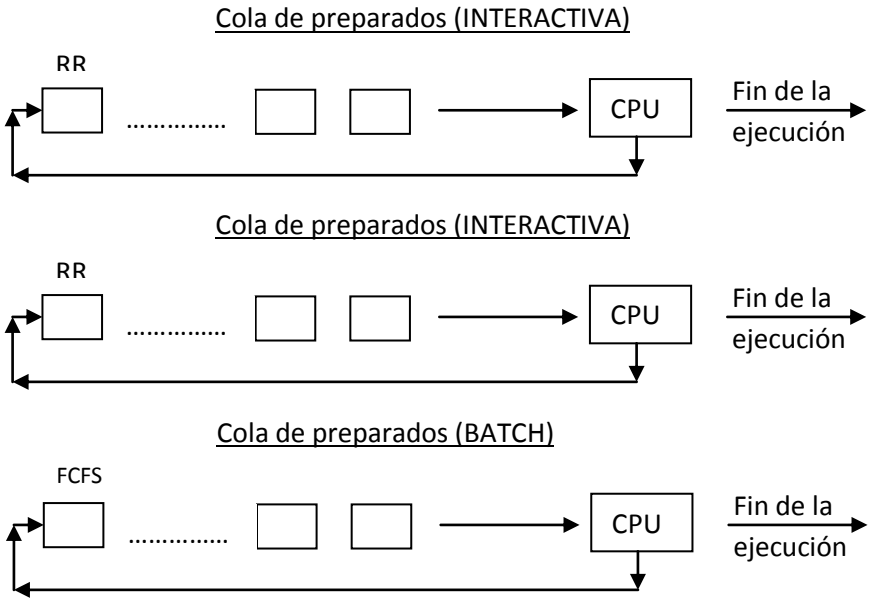


2.6.4 Queves multi-level. Colas múltiples

Cuando los procesos que van a ser ejecutados se pueden agrupar en distintos grupos, podemos asignarlos a diferentes colas, cada una con distinta planificación, para darle la que realmente necesite.

Esta política divide la cola de procesos preparados (listos) en varias colas separadas, de forma que los procesos se asignan a una cola específica según sus necesidades y tipo.

Para determinar en cada caso qué cola es la que suministrará un proceso para que acceda al cpu cuando éste deje a otro anterior, será controlada por un algoritmo de planificación entre las colas, que normalmente es apropiativo de prioridad fija.

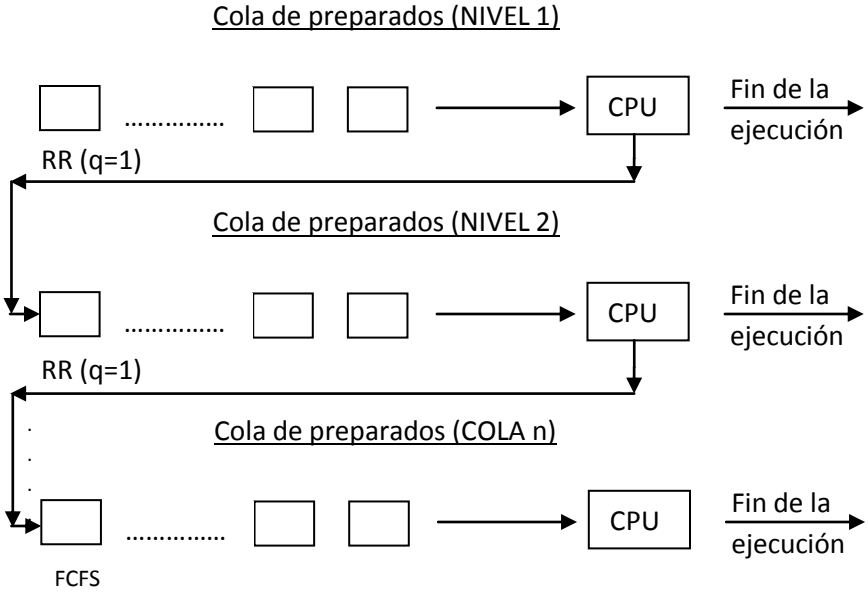


2.6.5 Multi-level feedback queves. (Colas múltiples con retroalimentación)

Para dar un trato justo a los procesos, es necesario conocer previamente todos sus parámetros característicos: longitud, si son limitados por e/s o por cpu, memoria requerida, etc. Como estos datos no suelen ser conocidos, es difícil determinar el trato que debe recibir cada proceso. Después de analizar las políticas anteriores, es fácil concluir que se deben adoptar las siguientes cuestiones:

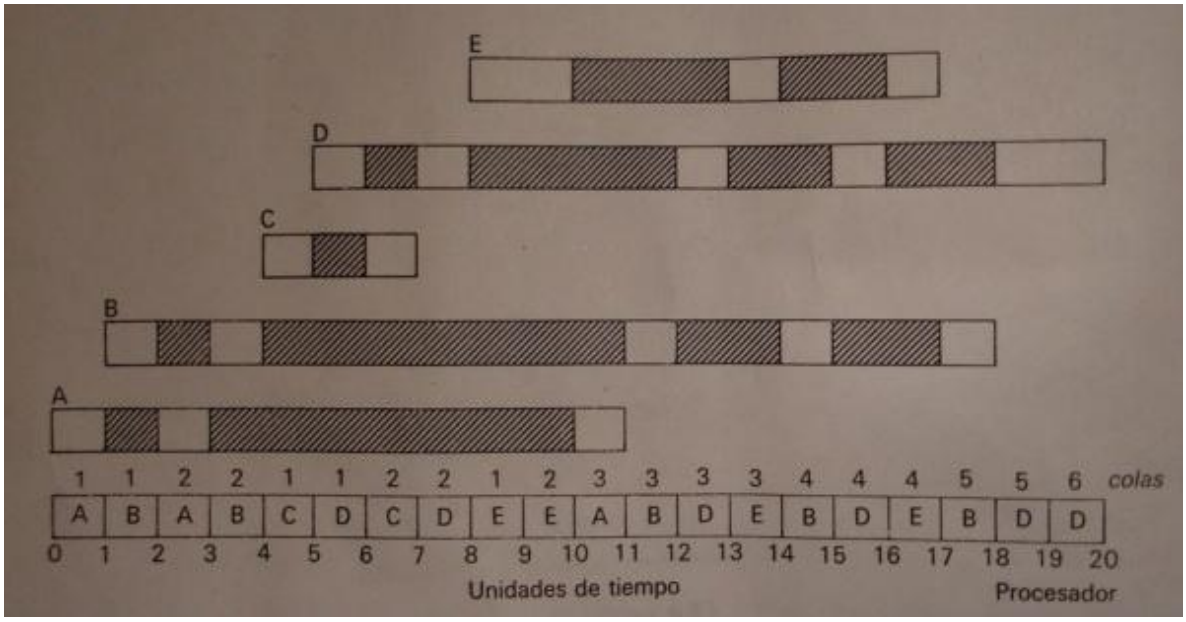
- Favorecer los procesos cortos
- Favorecer los procesos limitados por e/s
- Determinar la naturaleza del trabajo a realizar

El método de colas múltiples con retroalimentación divide los procesos en varias colas de procesos preparados (listos): cola 0, cola 1, cola 2, y así sucesivamente, de forma que las de numeración más baja tendrán una mayor prioridad.



Cuando el proceso que usa el cpu termina su quantum, se selecciona un nuevo proceso del principio de la cola del nivel más bajo que tenga algún proceso. Una vez que haya consumido el quantum de su cola un determinado número de veces, sin finalizar su ejecución, será colocado al final de a de nivel inmediatamente superior al anterior.
La política FB intenta dar un trato justo a los procesos por medio de separación en categorías, para así darles el servicio que necesitan. Los procesos limitados por cpu irán a las colas de menor prioridad (nivel más alto), mientras los de mayor prioridad serán aquellos procesos muy interactivos.

proceso	**instante de llegada (ti)**	**tiempo de ejecución (t)**	**instante de fin (tf)**	**T (tiempo de servicio tf-ti)**	**E (tiempo de (espera T-t)**	**I (índice de (servicio t/T)**
A	0	3	11	11	8	0.27
B	1	5	18	17	12	0.29
C	4	2	7	3	1	0.67
D	5	6	20	15	9	0.67
E	8	4	17	9	5	0.80
			Promedios	11.0	7.0	0.54



Soporta bien la sobrecarga, apropiativa, adaptable a las necesidades del sistema, y que cada cola puede ser gestionada de forma diferente