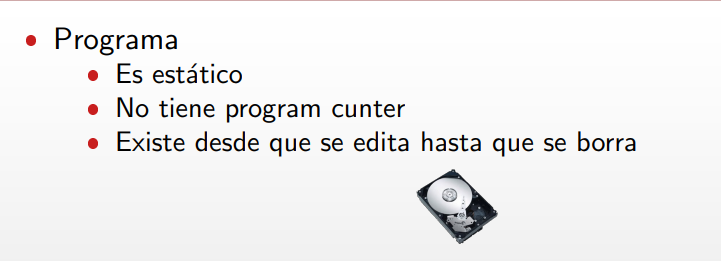
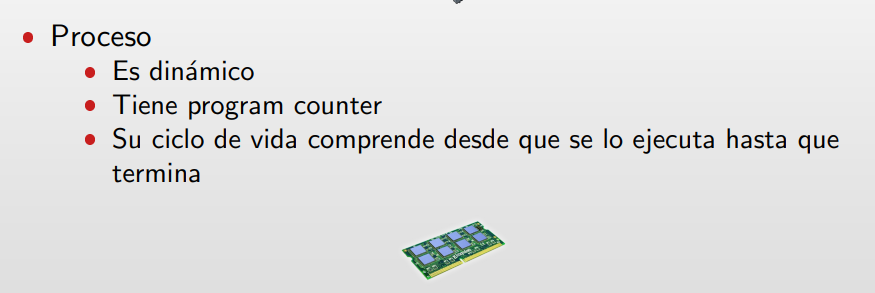
1)

a. Programa y Proceso





Hay una PCB por proceso, la cual contiene info del mismo. Esta es la primera que se cera y la ultima que se elimina al terminar un proceso

b. Defina Tiempo de retorno (TR) y Tiempo de espera (TE) para un Job

Tiempo de Retorno: tiempo que transcurre entre que el proceso llega al sistema hasta que completa su ejecución

Tiempo de Espera: tiempo que el proceso se encuentra en el sistema esperando, es decir el tiempo que pasa sin ejecutarse (TR - Tcpu)

c. Defina Tiempo Promedio de Retorno (TPR) y Tiempo promedio de espera (TPE) para un lote de JOBS

El TPR es el promedio de todos los tiempos de retorno de los procesos y el TPE es el promedio de todas las esperas de todos los procesos

d. ¿Qué es el Quantum?

Quantum (Q): medida que determina cuanto tiempo podria usar el procesador cada proceso:

Pequenio: overhead de context switch, cambia todo el rato

Grande: podriamos darle tiempo de más a un proceso

e. ¿Qué significa que un algoritmo de scheduling sea apropiativo o no apropiativo (Preemptive o Non-Preemptive)?

**Preemptive:** el proceso en ejecución puede ser interrumpido y llevado a la cola de listos:

* Mayor overhead, pero mejor servicio
* Un proceso no monopoliza el procesador

**Nonpreemptive:** una vez que un proceso está en estado de ejecución, continua hasta que termina o se bloquea por algún evento (e.j. I/O)

f. ¿Qué tareas realizan?

\* **Long term scheduler:** admite nuevos procesos a memoria (controla el grado de multiprogramación)

\* **Medium term scheduler:** realiza el swapping (intercambio) entre el disco y la memoria cuando el SO lo determina (puede disminuir el grado de multiprogramación)

\* **Short term scheduler:** determina que proceso pasara a ejecutarse

g. ¿Qué tareas realiza el Dispatcher?

Dispatcher es el componente encargado de gestionar la ejecución de los procesos en la CPU, realizando cambios de contexto, asignando la CPU a procesos, manejando interrupciones y excepciones, gestionando bloqueos y terminando procesos. Su objetivo es asegurar que los procesos se ejecuten de manera eficiente y efectiva de acuerdo con las decisiones tomadas por el planificador a corto plazo.

2)

a. Investigue y detalle para que sirve cada uno de los siguientes comandos. (Puede que algún comando no venga por defecto en su distribución por lo que deberá instalarlo):

**Top** muestra en tiempo real los procesos que se estan ejecutando asi como info del sistema y rednimiento.

Opciones comunes:

q: Sale de top.

Spacebar: Actualiza la pantalla en tiempo real.

k: Mata (finaliza) un proceso seleccionado.

u: Filtra los procesos por un usuario específico.

M: Ordena la lista de procesos por uso de memoria.

P: Ordena la lista de procesos por uso de CPU.

T: Muestra información sobre las CPU individuales si tienes múltiples CPU o núcleos.

**Htop** El comando htop es una herramienta de línea de comandos similar a top, pero ofrece una interfaz más amigable y visual para monitorear el rendimiento del sistema y gestionar procesos en sistemas Unix y Linux. htop presenta información en tiempo real en forma de una lista de procesos y gráficos que muestran el uso de CPU y memoria.

Opciones comunes:

F2 o S: Abre la pantalla de configuración, que te permite personalizar la vista de htop y seleccionar qué columnas se muestran.

F3 o /: Permite buscar procesos por nombre.

F4 o I: Filtra procesos por usuario.

F5 o T: Ordena la lista de procesos por diferentes criterios, como uso de CPU o uso de memoria.

F9 o k: Mata (finaliza) un proceso seleccionado.

F10 o q: Sale de htop.

**Ps** El comando ps se utiliza en sistemas Unix y Linux para mostrar información sobre los procesos en ejecución en el sistema. Proporciona una lista de procesos que se pueden filtrar y formatear de diversas maneras para obtener detalles específicos

* Puedes usar la opción -e para mostrar todos los procesos de todos los usuarios.
* Para ver los procesos de un usuario en particular, utiliza la opción -u seguida del nombre de usuario
* Puedes usar la opción -f para obtener información detallada de los procesos, que incluye la jerarquía de procesos, el estado, el PID (ID de proceso), el PPID (ID de proceso padre), el uso de CPU y más.
* Watch ps Si quieres ver la lista de procesos actualizada continuamente

**Pstree** El comando pstree se utiliza en sistemas Unix y Linux para mostrar una representación gráfica jerárquica de los procesos en ejecución en el sistema. Esta representación muestra las relaciones padre-hijo entre los procesos y es útil para visualizar la estructura de procesos en el sistema.

Mostrar un proceso específico y sus hijos: pstree -p 1234

Mostrar nombres de proceso en lugar de números de PID: pstree –a

Limitar la profundidad del árbol: pstree -l 2

**Kill** El comando kill se utiliza en sistemas Unix y Linux para enviar señales a procesos en ejecución. Estas señales pueden ser utilizadas para controlar y gestionar procesos de diversas formas, como finalizar un proceso, detenerlo, reiniciarlo o enviar otras notificaciones.

Finalizar un proceso por su número de PID: kill PID

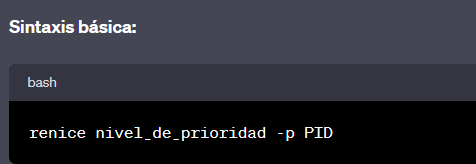
Enviar una señal específica: kill -s SIGHUP PID

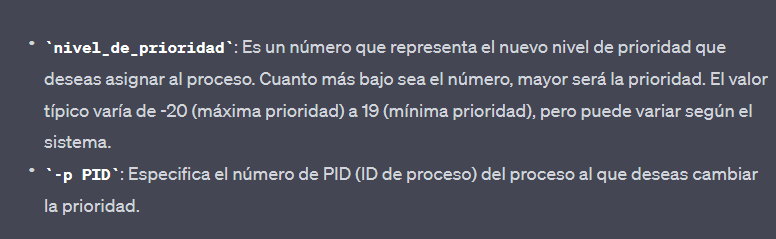
Finalizar un proceso por nombre: pkill myprocess

Finalizar un proceso con una señal específica por nombre: pkill -s SIGKILL myprocess

**Pgreppkillkillall** pgrep para buscar procesos, pkill para enviar señales a procesos y killall para enviar señales a procesos basados solo en sus nombres. A diferencia de los otros, **killall** no admite la búsqueda basada en propiedades, solo busca procesos por nombre.

**Renice** El comando renice se utiliza en sistemas Unix y Linux para cambiar la prioridad de ejecución de un proceso en el sistema. Con renice, puedes ajustar la prioridad de un proceso para asignar más o menos tiempo de CPU en función de sus necesidades. Los procesos con una prioridad más alta obtendrán más tiempo de CPU, mientras que los procesos con una prioridad más baja obtendrán menos tiempo de CPU.



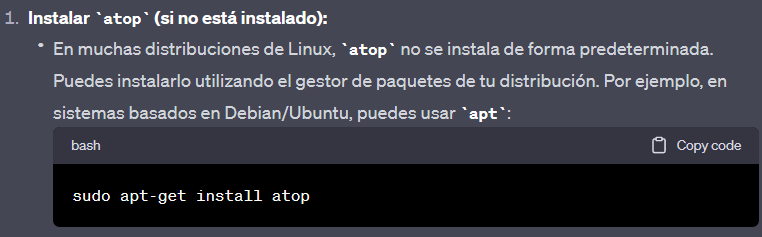


**Xkill** El comando xkill es una utilidad en entornos de escritorio basados en el sistema X Window System, que se utiliza para finalizar (matar) aplicaciones o ventanas gráficas que se han quedado bloqueadas o no responden. Es una herramienta útil para interactuar con aplicaciones gráficas que no se pueden cerrar de manera normal.

Ejecuta el comando xkill:

* El cursor se convertirá en una pequeña calavera o una cruz.
* Haz clic en la ventana o la aplicación que deseas cerrar de manera forzada. Al hacer clic en la ventana, xkill enviará una señal para finalizar la aplicación.
* La ventana o la aplicación seleccionada se cerrará y se liberarán los recursos asociados a ella.
* Cuando hayas terminado de usar xkill, puedes presionar Esc en el teclado para salir de xkill sin realizar ninguna acción

**Atop** El comando atop es una herramienta de monitorización del rendimiento en sistemas Unix y Linux que proporciona una vista detallada de los recursos del sistema y la actividad de los procesos en tiempo real. atop se utiliza para obtener información detallada sobre el uso de la CPU, la memoria, los discos, la red y otros recursos del sistema, lo que facilita el seguimiento y diagnóstico de problemas de rendimiento.



d. Comunicación entre procesos:

i. Investigue la forma de comunicación entre procesos a través de pipes.

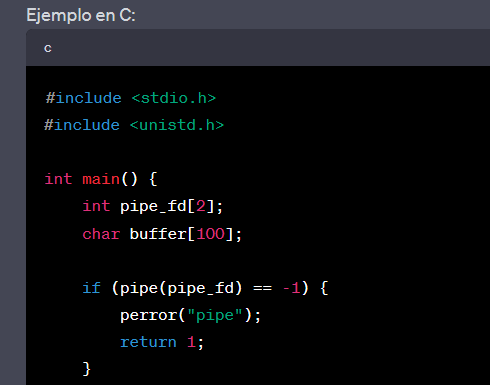
**Pipes Anónimos (Unnamed Pipes):**

Los pipes anónimos son la forma más simple de comunicación entre procesos.

Se utilizan para la comunicación entre un proceso padre y sus procesos hijos, ya que comparten el mismo espacio de direcciones de memoria.

Los pipes anónimos son unidireccionales, lo que significa que la comunicación fluye en una sola dirección, generalmente desde el proceso padre hacia el proceso hijo.

Pueden crearse usando la función pipe() en C.

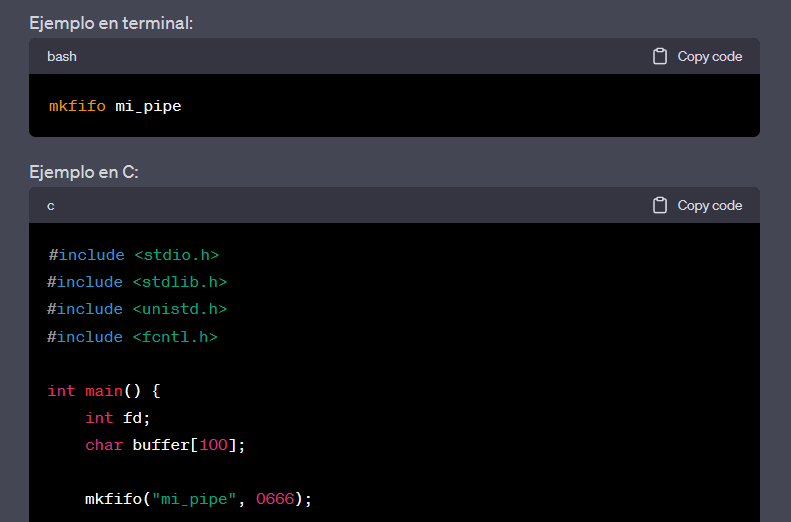


**Pipes con Nombre (Named Pipes o FIFOs):**

Los pipes con nombre son tuberías con un nombre en el sistema de archivos y se utilizan para la comunicación entre procesos que pueden estar en diferentes sesiones o incluso en máquinas diferentes.

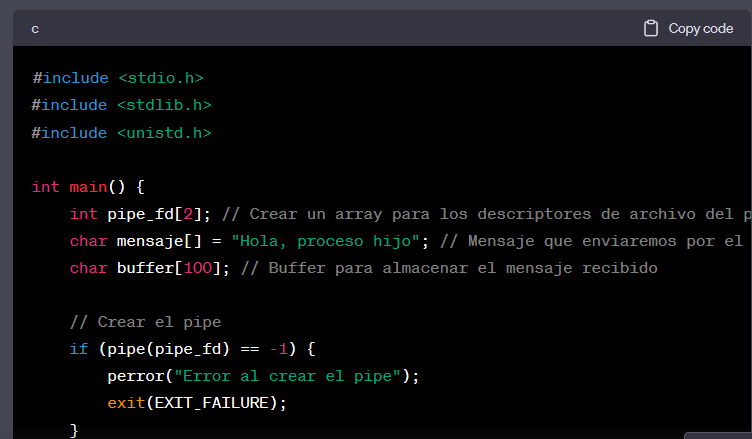
Los pipes con nombre son bidireccionales, lo que significa que se pueden usar para la comunicación en ambas direcciones.

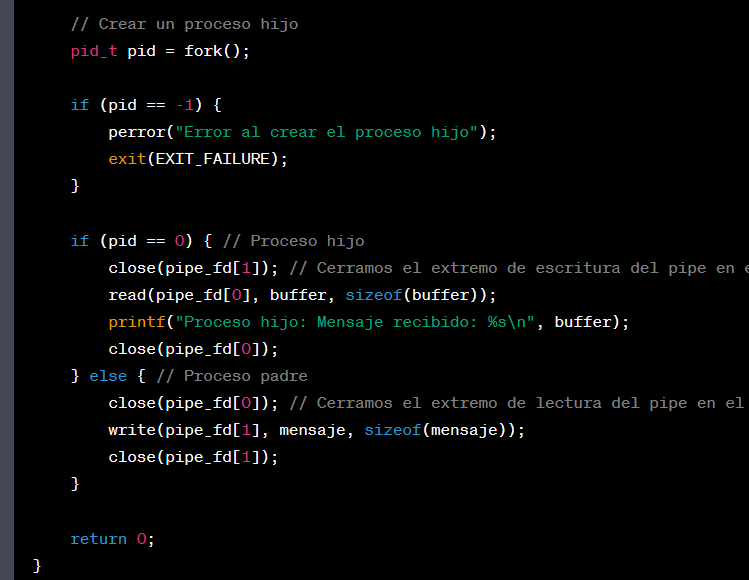
Pueden crearse utilizando el comando mkfifo en la terminal o la función mkfifo() en C.



ii. ¿Cómo se crea un pipe en C?.

En C, puedes crear un pipe (tubería) utilizando la función pipe(). Esta función está disponible en la biblioteca de C estándar (stdio.h). Un pipe se utiliza para establecer una comunicación unidireccional entre dos procesos, de modo que un proceso pueda escribir datos en el pipe y el otro proceso pueda leer esos datos desde el pipe. Aquí tienes un ejemplo de cómo crear y utilizar un pipe en C:





iii. ¿Qué parametro es necesario para la creación de un pipe?. Explique para que se utiliza.

La función pipe() en C no requiere ningún parámetro para su creación. La función pipe() crea un pipe (tubería) anónimo y devuelve dos descriptores de archivo (file descriptors) en un array de enteros. Estos descriptores de archivo se utilizan para acceder a los extremos del pipe. El array tendrá dos elementos:

pipe\_fd[0]: Este descriptor se utiliza para leer datos del pipe. Es el extremo de lectura (read end) del pipe.

pipe\_fd[1]: Este descriptor se utiliza para escribir datos en el pipe. Es el extremo de escritura (write end) del pipe.

Por lo tanto, al llamar a pipe(), se obtienen estos dos descriptores de archivo que permiten la comunicación unidireccional entre dos procesos. Uno de los procesos puede escribir datos en el extremo de escritura (pipe\_fd[1]), y el otro proceso puede leer esos datos desde el extremo de lectura (pipe\_fd[0]).

iV. ¿Qué tipo de comunicación es posible con pipes?

Con los pipes (tuberías) en sistemas Unix y Linux, es posible realizar una comunicación unidireccional entre dos procesos. La comunicación se puede establecer de la siguiente manera:

Comunicación de Padre a Hijo: Un proceso padre puede crear un pipe y luego bifurcar (fork) un proceso hijo. El padre escribe datos en el pipe, y el hijo los lee desde el pipe. Esto permite que el proceso padre comparta información con sus procesos hijos.

Comunicación de Hijo a Padre: De manera similar, un proceso hijo puede crear un pipe y comunicarse con su proceso padre escribiendo datos en el pipe y permitiendo que el padre los lea desde allí.

Comunicación entre Procesos Independientes: Los pipes también se pueden utilizar para la comunicación entre procesos que no son padres ni hijos. En este caso, dos procesos independientes pueden cooperar utilizando un pipe para intercambiar datos.

Comunicación en Cadena: Los pipes se pueden utilizar en una secuencia de procesos en cadena, donde la salida de un proceso se convierte en la entrada del siguiente. Esto es útil en la implementación de filtros o tuberías (pipes) de comandos en la línea de comandos de Unix, donde los procesos se comunican pasando datos a través de pipes.

Es importante destacar que los pipes son unidireccionales, lo que significa que la comunicación fluye en una sola dirección. Por lo tanto, si se necesita comunicación bidireccional, generalmente se utilizan dos pipes (uno para cada dirección) o se exploran otras formas de comunicación interproceso, como las colas de mensajes, la memoria compartida o los sockets, que ofrecen mayor flexibilidad en términos de comunicación bidireccional.

e. ¿Cuál es la información mínima que el SO debe tener sobre un proceso?¿En que estructura de datos asociada almacena dicha información?

El sistema operativo (SO) debe mantener información básica sobre cada proceso en ejecución para administrarlos adecuadamente. La información mínima que el SO debe tener sobre un proceso generalmente incluye:

Identificación del Proceso (PID - Process ID): Un número único que identifica de manera única cada proceso en el sistema. Los PID suelen ser enteros positivos.

Estado del Proceso: Indica si el proceso está en ejecución, suspendido, en espera, terminado, etc.

Prioridad del Proceso: Una prioridad que determina la asignación de tiempo de CPU y otros recursos del sistema al proceso.

Programa y Datos del Proceso: La ubicación en memoria del código ejecutable del programa, así como los datos y las estructuras de control asociadas.

Registros del Procesador: Información sobre los registros del procesador que se utilizan para mantener el contexto del proceso cuando se interrumpe su ejecución y se reanuda más tarde.

Información de Programa: Esto incluye el contador de programa (program counter) y otros registros relacionados con la ejecución del código.

Recursos Asignados: Información sobre los recursos asignados al proceso, como descriptores de archivos, espacio de memoria, ID de usuario, ID de grupo, etc.

Estado de la Memoria: Información sobre el espacio de direcciones de memoria del proceso, incluidas las áreas de código, datos, pila y montones (heap).

Cola de Procesos: En qué colas de planificación se encuentra el proceso, si es que está esperando su turno para ejecutarse.

La información sobre un proceso se almacena en una estructura de datos asociada conocida como un "Bloque de Control de Proceso" o "PCB" (Process Control Block). El PCB es una estructura de datos que mantiene un registro de toda la información necesaria para administrar un proceso, como la que se mencionó anteriormente. El sistema operativo crea y administra PCBs para cada proceso en ejecución y los utiliza para cambiar entre procesos y mantener la coherencia y el control del sistema.

Cuando un proceso se suspende o se interrumpe para dar tiempo de CPU a otro proceso, el SO guarda el contexto del proceso actual en su PCB y restaura el contexto del nuevo proceso a ejecutar. Esto permite que el SO administre múltiples procesos de manera eficiente y les brinde recursos de manera adecuada.

f. ¿Qué significa que un proceso sea “CPU Bound” y “I/O Bound”?

Los términos "CPU Bound" (ligado a la CPU) y "I/O Bound" (ligado a la E/S, donde E/S significa entrada/salida) son dos conceptos que describen el comportamiento de un proceso en un sistema informático en función de cómo utiliza los recursos disponibles. Estos términos son especialmente relevantes en el contexto de la planificación de procesos y el rendimiento del sistema. Aquí te explico qué significa cada uno:

CPU Bound (Ligado a la CPU):

Un proceso se considera "CPU Bound" cuando pasa la mayor parte de su tiempo utilizando la CPU para realizar cálculos o procesamiento intensivo.

Estos procesos generalmente no hacen muchas operaciones de entrada/salida (E/S). Pasan la mayor parte del tiempo realizando cálculos o procesamiento numérico.

Los procesos CPU Bound tienden a utilizar una cantidad significativa de tiempo de CPU y, a menudo, se ejecutan durante largos períodos de tiempo sin esperar E/S.

Ejemplos de procesos CPU Bound pueden incluir algoritmos matemáticos complejos, renderizado de gráficos, simulaciones numéricas, etc.

I/O Bound (Ligado a la E/S - Entrada/Salida):

Un proceso se considera "I/O Bound" cuando pasa la mayor parte de su tiempo esperando o realizando operaciones de entrada/salida, como lectura o escritura en disco, entrada de datos desde el teclado, acceso a la red, etc.

Estos procesos tienden a usar la CPU solo brevemente para realizar operaciones de E/S y luego esperan a que se complete la operación de E/S.

Los procesos I/O Bound a menudo no utilizan una gran cantidad de tiempo de CPU y, por lo tanto, pueden liberar la CPU para que otros procesos la utilicen mientras esperan E/S.

Ejemplos de procesos I/O Bound pueden incluir aplicaciones que leen o escriben archivos grandes de manera frecuente, servidores web que manejan solicitudes de clientes, procesos de bases de datos, etc.

Es importante comprender si un proceso es CPU Bound o I/O Bound, ya que esto puede influir en la planificación de procesos y la asignación de recursos en un sistema operativo. Por ejemplo, en un sistema multitarea, los procesos I/O Bound a menudo se benefician de una planificación de procesos que permite una conmutación rápida entre procesos para que no bloqueen la CPU mientras esperan E/S. En contraste, los procesos CPU Bound pueden beneficiarse de una asignación prolongada de tiempo de CPU para maximizar la eficiencia de cálculos intensivos.

g. ¿Cuáles son los estados posibles por los que puede atravesar un proceso?

Un proceso en un sistema operativo puede atravesar varios estados a medida que se crea, se ejecuta, se suspende y finalmente se completa. Los estados posibles por los que puede pasar un proceso son los siguientes:

Nuevo (New): En este estado, el proceso ha sido creado pero aún no ha sido admitido por el sistema operativo para su ejecución. El proceso se encuentra en este estado mientras se realizan las inicializaciones iniciales necesarias.

Listo (Ready): Un proceso en estado "Listo" está preparado para ejecutarse y espera ser asignado por el planificador de procesos para obtener tiempo de CPU. Los procesos en este estado pueden ejecutarse tan pronto como se les asigne tiempo de CPU.

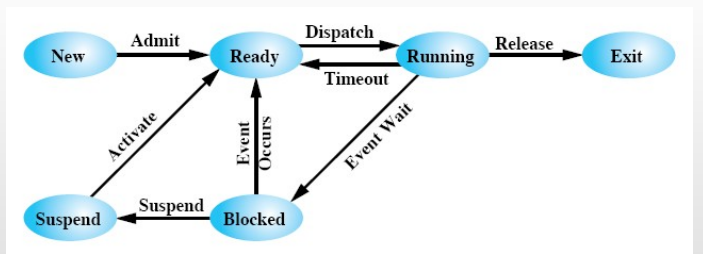
Ejecución (Running): Cuando un proceso recibe tiempo de CPU y se está ejecutando activamente en la CPU, se encuentra en el estado de "Ejecución". Solo un proceso puede estar en este estado en un momento dado en un sistema monoproceso o un solo núcleo.

Bloqueado (Blocked o Waiting): Un proceso entra en este estado cuando realiza una operación de entrada/salida (E/S) u otra operación que requiere esperar a que se complete una tarea externa, como la lectura de datos desde el disco o la espera de entrada del usuario. El proceso no puede continuar hasta que se complete la operación de E/S o la condición de espera se cumpla.

Terminado (Terminated o Exit): Cuando un proceso ha completado su ejecución o ha sido terminado por el sistema operativo debido a algún error, se encuentra en el estado "Terminado". En este estado, se liberan todos los recursos asociados con el proceso, y el sistema operativo realiza cualquier limpieza necesaria.

Estos son los estados típicos por los que puede pasar un proceso en un sistema operativo. El cambio entre estos estados se maneja mediante la planificación de procesos y eventos del sistema operativo, como la interrupción de E/S, la asignación de tiempo de CPU y la terminación de procesos. La transición de un estado a otro está determinada por las políticas de planificación y las operaciones realizadas por el proceso y el sistema operativo.

h.Explique mediante un diagrama las posibles transiciones entre los estados.



i. ¿Que scheduler de los mencionados en 1 f se encarga de las transiciones?

El planificador a corto plazo, también conocido como "Short-Term Scheduler," es el encargado de gestionar las transiciones entre los estados de los procesos en un sistema operativo. Este planificador toma decisiones en tiempo real sobre cuál de los procesos listos (en estado "Listo") se ejecutará a continuación en la CPU y cuánto tiempo de CPU se le asignará.

El Short-Term Scheduler se encarga de las transiciones entre los estados "Listo" y "Ejecución". Cuando un proceso está listo para ejecutarse, el planificador a corto plazo decide cuál de los procesos listos obtendrá el control de la CPU. También puede realizar cambios de contexto, es decir, guardar el estado del proceso actualmente en ejecución y cargar el estado del proceso seleccionado para su ejecución.

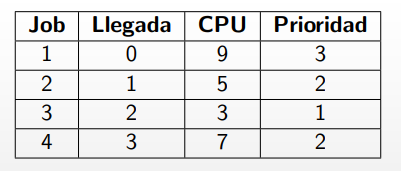
El objetivo principal del Short-Term Scheduler es maximizar la utilización de la CPU y minimizar el tiempo de respuesta, tomando decisiones rápidas y eficientes sobre la asignación de tiempo de CPU a los procesos en ejecución.

3)

**FCFS (FIFO):**

* First come first served
* Cuando hay que elegir un proceso para ejecutar, se selecciona el mas viejo
* No favorece a ningún tipo de procesos, pero en principio podíamos decir que los CPU Bound terminan al comenzar su primer r´afaga, mientras que los I/O Bound no

Se ejecuta los procedimientos en orden de llegada sin parar a evaluar si hay otros.



El job 1 tardara 9 en completarse, el 2 tardaría 5+el tiempo que espero en este caso 8 aprox xd, porque el job 1 tomo 8 tiempos desde que llego el job 2, y asi con los otros.

(b) Parámetros requeridos:

No.

(c) Adecuado para:

FCFS es adecuado para procesos no interactivos y no prioritarios, donde la duración de los procesos es conocida.

(d) Ventajas y desventajas:

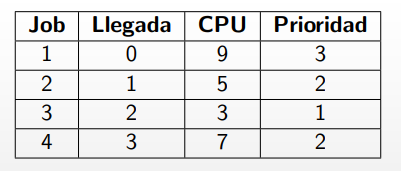
Ventajas: Fácil de entender e implementar. Garantiza una justa asignación de CPU.

Desventajas: No es adecuado para sistemas en tiempo real, puede provocar la inanición de procesos largos (procesos cortos son bloqueados detrás de procesos largos).

**SJF (Shortest Job First)**

* Shortest Job First
* Política nonpreemptive que selecciona el proceso con la ráfaga más corto
* Calculo basado en la ejecución previa
* Procesos cortos se colocan delante de procesos largos
* Los procesos largos pueden sufrir starvation (inanición)

Se ejecutan procesos hasta que terminen, pero si hay otros en cola, al momento de seleccionar el próximo proceso a ejecutarse, se selecciona el que sea más corto en el tiempo.



De esto ejecutaria el job 1, y cuando termine ejecutaria el job 3 porque es el mas corto, luego el job 2 y el job 4.

(b) Parámetros requeridos:

Requiere conocer la duración de los procesos de antemano o estimarla.

(c) Adecuado para:

SJF es adecuado para procesos no prioritarios en los que se conoce la duración de los procesos.

(d) Ventajas y desventajas:

Ventajas: Minimiza el tiempo de espera promedio y mejora la eficiencia.

Desventajas: Puede causar inanición si llegan procesos más cortos constantemente.

**Round Robin (RR)**

* Round Robin
* Politica basada en un reloj
* **Quantum (Q):** medida que determina cuanto tiempo podr´a usar el procesador cada proceso:
  + Pequeño: overhead de context switch
  + Grande: se le da tiempo de sobra a procesos y pueden desperdiciarse tiempo de cpu
* Cuando un proceso es expulsado de la CPU es colocado al final de la Ready Queue y se selecciona otro (FIFO circular)

Este tiene un Quantum, que es cuando se le asigna de tiempo a cada proceso e ir evualuando en cada Q que hacer.

Se hace una cola, por ejemplo empieza el 1 y la cola queda 2,3,4 1... este 1 porque termino el Q pero debe seguir en algun momento.

Si es de **timer variable**, por ejemplo si un proceso termina antes de los 4 tiempo de Q, por ejemplo 3, simplemente se termina ahí y se busca otro proceso para que haga su Q de 4. En cambio, si es **timer fijo** y termina en el Q 3, se busca al otro proceso que siga y se le da los Q que le falto al anterior, en este caso 1, lo ejecuta y si no termino se lo vuelve a mandar a la cola...

(b) Parámetros requeridos:

Requiere un valor de quantum como parámetro.

(c) Adecuado para:

Round Robin es adecuado para sistemas multitarea y procesos interactivos, ya que garantiza un tiempo justo de CPU para todos los procesos.

(d) Ventajas y desventajas:

Ventajas: Evita la inanición, proporciona una respuesta rápida, adecuado para sistemas interactivos.

Desventajas: Puede tener un alto tiempo de espera promedio si el quantum es demasiado grande.

**Prioridades**

* Cada proceso tiene un **valor que representa su prioridad** → menor valor, mayor prioridad
* Se selecciona el proceso de mayor prioridad de los que se encuentran en la Ready Queue
* Existe una Ready Queue por cada nivel de prioridad
* Procesos de baja prioridad pueden sufrir starvation (inanici´on)
  + Soluci´on: permitir a un proceso cambiar su prioridad durante su ciclo de vida → Aging o Penalty
* Puede ser un algoritmo preemptive o no

Los procesos se ejecutan según su prioridad, donde el proceso de mayor prioridad recibe la CPU antes que los de menor prioridad. **Saca PROCESOS** si llega otro de mayor prioridad

Ejemplo: Si tenemos tres procesos P1 (prioridad alta), P2 (prioridad media) y P3 (prioridad baja), se ejecutaría P1, luego P2 y finalmente P3.

Se genera una cola como en los otros, pero en este caso por cada una de las prioridades, si hay 3 prioridades hay 3 colas.

(b) Parámetros requeridos:

Requiere asignar prioridades a los procesos.

(c) Adecuado para:

Adecuado para sistemas donde es importante dar prioridad a ciertos procesos, como sistemas en tiempo real o sistemas críticos.

(d) Ventajas y desventajas:

Ventajas: Permite priorizar procesos críticos, adecuado para sistemas donde la priorización es fundamental.

Desventajas: Puede llevar a la inanición de procesos de baja prioridad y requiere una gestión cuidadosa de las prioridades.

9.Inanicion (starvation)

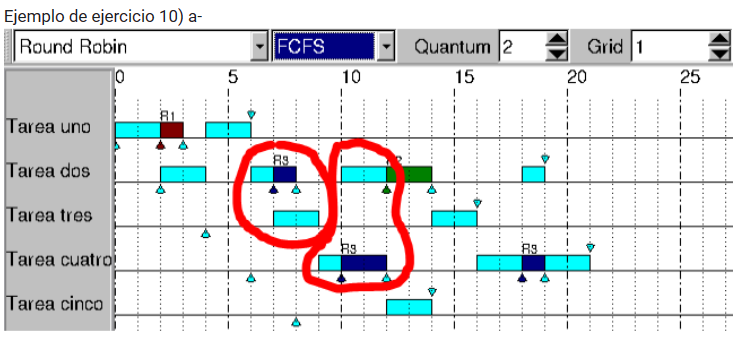
La inanición, en el contexto de la administración de procesos en un sistema operativo, se refiere a una situación en la que un proceso se queda atrapado y nunca recibe la oportunidad de ejecutarse o completar su trabajo, a pesar de que está listo para ejecutarse. En otras palabras, el proceso se "muere de hambre" de tiempo de CPU y queda bloqueado indefinidamente en la cola de procesos listos sin recibir ninguna asignación de CPU.

(b) ¿Cuál/es de los algoritmos vistos puede provocar la inanición?  
La inanición es más probable en algoritmos de planificación que no garantizan una asignación justa de tiempo de CPU a todos los procesos. En los algoritmos de planificación basados en prioridades, donde los procesos de alta prioridad pueden obtener la CPU antes que los de baja prioridad, existe el riesgo de que los procesos de baja prioridad queden inmovilizados.

El algoritmo de planificación Round Robin generalmente no causa inanición porque asigna un quantum de tiempo de CPU a cada proceso en la cola de listos, lo que garantiza que cada proceso obtenga un tiempo de ejecución equitativo. Sin embargo, si un proceso tiene un tiempo de CPU extremadamente largo y la duración del quantum es corta, podría experimentar una latencia excesiva en su ejecución.

Los algoritmos de planificación de prioridades son más susceptibles a la inanición, especialmente si no se toman medidas para evitarla. Para mitigar la inanición en algoritmos de planificación basados en prioridades, se pueden implementar técnicas como la asignación de límites de tiempo de CPU para procesos de baja prioridad o el uso de envejecimiento (aumento gradual de la prioridad de los procesos en espera). Estas técnicas ayudan a garantizar que incluso los procesos de baja prioridad eventualmente tengan la oportunidad de ejecutarse.

10. Los procesos, durante su ciclo de vida, pueden realizar operaciones de I/O como lecturas o escrituras a disco, cintas, uso de impresoras, etc. El SO mantiene para cada dispositivo, que se tiene en el equipo, una cola de procesos que espera por la utilización del mismo (al igual que ocurre con la Cola de Listos y la CPU, ya que la CPU es un dispositivo mas). Cuando un proceso en ejecución realiza una operación de I/O el mismo es expulsado de la CPU y colocado en la cola correspondiente a el dispositivo involucrado en la operación. El SO dispone también de un “I/O Scheduling” que administrada cada cola de dispositivo a través de algún algoritmo (FCFS, Prioridades, etc.). Si al colocarse un proceso en la cola del dispositivo, la misma se encuentra vacía el mismo será atendido de manera inmediata, caso contrario, deberá esperar a que el SO lo seleccione según el algoritmo de scheduling establecido. Los mecanismos de I/O utilizados hoy en día permiten que la CPU no sea utilizada durante la operación, por lo que el SO puede ejecutar otro proceso que se encuentre en espera una vez que el proceso bloqueado por la I/O se coloca en la cola correspondiente. Cuando el proceso finaliza la operación de I/O el mismo retorna a la cola de listos para competir nuevamente por la utilización de la CPU



Ejemplo de I/O mal echo en la trea 3, empieza en 4 cuando en realidad debe ser en momento el 3.

11. Algunos algoritmos pueden presentar ciertas desventajas cuando en el sistema se cuenta con procesos ligados a CPU y procesos ligados a entrada salida. Analice las mismas para los siguientes algoritmos: (a) Round Robin (b) SRTF (Shortest Remaining Time First)

(a) Round Robin:

Desventaja en procesos E/S: En el caso de procesos intensivos en E/S, el algoritmo Round Robin puede ser ineficiente. Debido a la asignación de un quantum fijo a cada proceso, un proceso de E/S podría no completar su trabajo dentro de un solo quantum y, como resultado, se realizarían cambios de contexto innecesarios. Esto puede llevar a una alta sobrecarga de cambio de contexto y un rendimiento ineficiente en procesos de E/S.

(b) SRTF (Shortest Remaining Time First):

Desventaja en procesos largos: El algoritmo SRTF tiende a dar prioridad a los procesos más cortos en términos de tiempo de CPU restante. Esto puede hacer que los procesos largos sean relegados y experimenten tiempos de espera significativos. Si hay un proceso intensivo en CPU en curso y llega un proceso más corto que necesita la CPU, el proceso intensivo en CPU podría verse interrumpido constantemente, lo que puede afectar su rendimiento. Esto es especialmente evidente si hay muchos procesos intensivos en E/S en espera, ya que los procesos largos podrían quedar en un estado de espera prolongada.

En sistemas donde coexisten procesos intensivos en CPU y procesos intensivos en E/S, es importante seleccionar un algoritmo de planificación que se adapte a las características de los procesos y los objetivos del sistema. Por ejemplo, en tales situaciones, podría ser útil utilizar una combinación de algoritmos o implementar políticas de planificación que prioricen procesos de E/S para reducir la sobrecarga en sistemas intensivos en E/S. La elección del algoritmo de planificación debe considerar el equilibrio entre la equidad, la eficiencia y los objetivos específicos del sistema.

**Algoritmo VRR (Virtual Round Robin):** Este algoritmo funciona igual que el Round Robin, con la diferencia que cuando un proceso regresa de una I/O se coloca en una cola auxiliar. Cuando se tiene que tomar el próximo proceso a ejecutar, los procesos que se encuentra en la cola auxiliar tienen prioridad sobre los otros. Cuando se elije un proceso de la cola auxiliar se le otorga el procesador por tantas unidades de tiempo como le falto ejecutar en su ráfaga de CPU anterior, esto es, se le otorga la CPU por un tiempo que surge entre la diferencia del quantum original y el tiempo usado en la última ráfaga de CPU.

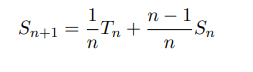
Es decir, que, si se ejecutó 2 veces y el Q era de 6, y fue a I/O. Cuando vuelva va a ir a la cola auxiliar con mayor prioridad y se le va a dar la CPU por Q-tiempo anterior. En este caso 6-2=4, se ejecutará en CPU esos 4 tiempos que le falto anteriormente.

13. Suponga que un SO utiliza un algoritmo de VRR con Timer Variable para el planificar sus procesos. Para ello, el quantum es representado por un contador, que es decrementado en 1 unidad cada vez que ocurre una interrupción de reloj. ¿Bajo este esquema, puede suceder que el quantum de un proceso nunca llegue a 0 (cero)? Justifique su respuesta

Bajo un esquema de planificación de VRR (Variable Round Robin) con un contador que se decrementa en 1 unidad cada vez que ocurre una interrupción de reloj, es posible que el quantum de un proceso nunca llegue a 0 (cero) si el contador nunca se agota debido a interrupciones frecuentes y periódicas.

En un sistema operativo con interrupciones de reloj regulares, el contador de quantum generalmente se reinicia con un valor completo cada vez que se produce una interrupción de reloj. Esto significa que, si el contador se inicializa con un valor "n" y se produce una interrupción de reloj antes de que el proceso haya consumido todo su quantum, el contador se restablecerá a "n". Esto permitiría que el proceso continúe ejecutándose sin que el contador llegue a cero.

Sin embargo, es importante tener en cuenta que esta situación de que el quantum nunca llegue a cero es poco común en la práctica y puede no ser eficiente. En un sistema real, generalmente se espera que el quantum se agote después de que un proceso haya tenido la oportunidad de ejecutarse durante un tiempo razonable. Si el contador de quantum no se agota con el tiempo, podría provocar una latencia innecesaria en la planificación de procesos y afectar negativamente el rendimiento del sistema. Por lo tanto, la implementación y el ajuste adecuado del contador de quantum son esenciales para garantizar una planificación eficiente y equitativa de procesos.

14.

Donde:

Ti = duración de la ráfaga de CPU i-ésima del proceso.

Si = valor estimado para el i-ésimo caso

Si = valor estimado para la primera ráfaga de CPU. No es calculado.

(a) Suponga un proceso cuyas ráfagas de CPU reales tienen como duración: 6, 4, 6, 4, 13, 13, 13 Calcule que valores se obtendrían como estimación para las ráfagas de CPU del proceso si se utiliza la formula 1, con un valor inicial estimado de S1=10.

S(i+1) = (1/n) \* T(i) + (n - 1)/n \* S(i)

Donde:

S(i) es el valor estimado para la i-ésima ráfaga de CPU.

T(i) es la duración de la i-ésima ráfaga de CPU del proceso.

n es el número de la ráfaga de CPU (n = 1 para la primera ráfaga).

Comenzando con **S1 = 10**, aquí están los cálculos para cada ráfaga de CPU:

S(2) = (1/**2**) \* 6 + **(2** - 1)/**2** \* **10** = 3 + 5 = 8

S(3) = (1/3) \* 4 + (3 - 1)/3 **\* 8** = 4/3 + 2/3 \* 8 = 4/3 + 16/3 = 20/3 ≈ 6.67

S(4) = (1/4) \* 6 + (4 - 1)/4 \* **20/3** ≈ 1.5 + 3 \* 20/3 ≈ 20

S(5) = (1/5) \* 4 + (5 - 1)/5 \* **20** ≈ 0.8 + 4 \* 20/5 = 16

S(6) = (1/6) \* 13 + (6 - 1)/6 \* **16 ≈** 2.17 + 5 \* 16/6 ≈ 18.83

S(7) = (1/7) \* 13 + (7 - 1)/7 \* **18.83** ≈ 1.86 + 6 \* 18.83/7 ≈ 25.07

Entonces, los valores estimados de las ráfagas de CPU del proceso, utilizando la fórmula proporcionada, serían aproximadamente:

S1 = 10 S2 ≈ 8 S3 ≈ 6.67 S4 ≈ 20 S5 ≈ 16 S6 ≈ 18.83 S7 ≈ 25.07

La formula anterior 1 le da el mismo peso a todos los casos (siempre calcula la media). Es posible reescribir la formula permitiendo darle un peso mayor a los casos mas recientes y menor a casos viejos (o viceversa). Se plantea la siguiente formula: Sn+1 = αTn + (1 − α)Sn



Sn+1 es el valor estimado en el período siguiente.

α es un factor de suavizado en el rango (0,1), que controla cuánto peso se le da a la nueva observación (Tn) en comparación con el valor anterior estimado (Sn).

Tn es el valor real observado en el período n.

Sn es el valor estimado en el período n.

Dado que tenemos una serie de observaciones de ráfagas de CPU: 6, 4, 6, 4, 13, 13, 13, podemos utilizar esta fórmula para estimar los valores de ráfagas de CPU en el futuro.

Vamos a calcular los valores estimados para esta serie de observaciones utilizando un valor de α específico. En este caso, puedes elegir un valor de α que se adapte a tus necesidades. Por ejemplo, si eliges α = 0.2, la fórmula se vería de la siguiente manera:

S1 = 6 (primer valor real)

S2 = α \* T1 + (1 - α) \* S1 = 0.2 \* 6 + 0.8 \* 6 = 6

S3 = α \* T2 + (1 - α) \* S2 = 0.2 \* 4 + 0.8 \* 6 = 5.2

S4 = α \* T3 + (1 - α) \* S3 = 0.2 \* 6 + 0.8 \* 5.2 = 5.36

S5 = α \* T4 + (1 - α) \* S4 = 0.2 \* 4 + 0.8 \* 5.36 = 4.768

S6 = α \* T5 + (1 - α) \* S5 = 0.2 \* 13 + 0.8 \* 4.768 = 5.5344

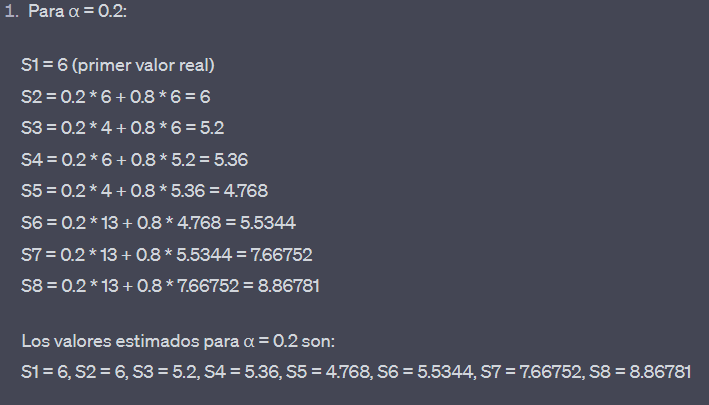
S7 = α \* T6 + (1 - α) \* S6 = 0.2 \* 13 + 0.8 \* 5.5344 = 7.66752

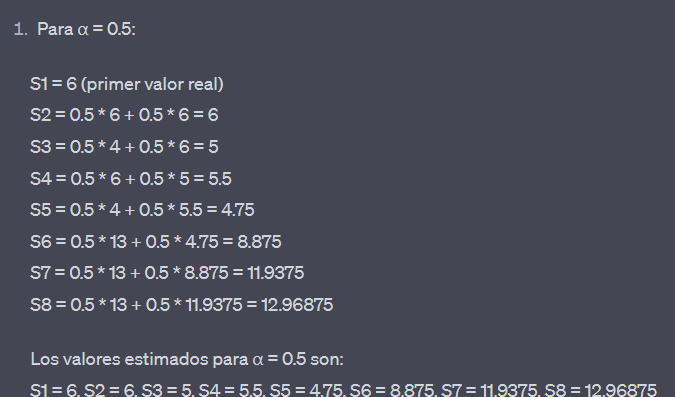
S8 = α \* T7 + (1 - α) \* S7 = 0.2 \* 13 + 0.8 \* 7.66752 = 8.86781

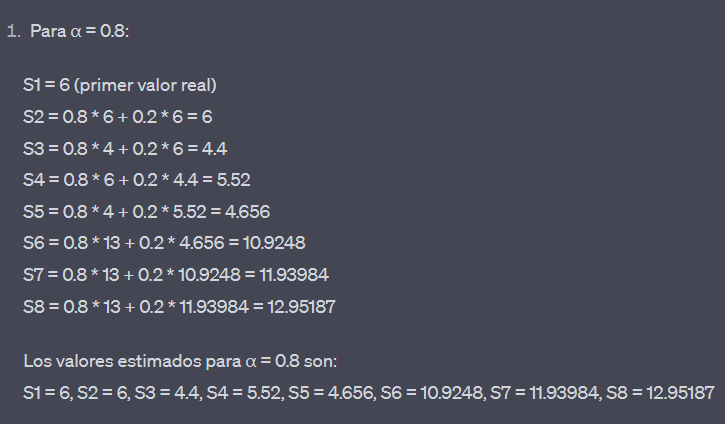
Así que, utilizando un valor de α = 0.2, los valores estimados para las ráfagas de CPU del proceso serían:

S1 = 6, S2 = 6, S3 = 5.2, S4 = 5.36, S5 = 4.768, S6 = 5.5344, S7 = 7.66752, S8 = 8.86781

Puedes ajustar el valor de α según tus necesidades para dar más o menos peso a las observaciones anteriores en las estimaciones. **Un valor más alto de α** dará más peso a las observaciones más recientes, mientras qu**e un valor más bajo** dará más peso a las observaciones pasadas.







Para todas las estimaciones realizadas en a y c ¿Cuál es la que mas se asemeja a las ráfagas de CPU reales del proceso?

El de α más alta.

15. Colas Multinivel

(a) Suponga que se tienen dos tipos de procesos: Interactivos y Batch. Cada uno de estos procesos se coloca en una cola según su tipo. ¿Qué algoritmo de los vistos utilizaría para administrar cada una de estas colas? A su vez, se utiliza un algoritmo para administrar cada cola que se crea. Así, por ejemplo, el algoritmo podría determinar mediante prioridades sobre que cola elegir un proceso.

Round Robin (RR):

El algoritmo Round Robin es adecuado para sistemas de tiempo compartido en los que se da prioridad a la equidad y la respuesta rápida a los procesos interactivos.

Cada proceso recibe una pequeña porción de tiempo de CPU (quantum) antes de pasar al siguiente proceso en la cola. Esto asegura que todos los procesos tengan la oportunidad de ejecutarse de manera justa.

Puede aplicarse a ambas colas de procesos interactivos y batch, pero es más adecuado para procesos interactivos debido a su respuesta rápida.

Prioridad (Priority Scheduling):

El algoritmo de prioridad asigna una prioridad a cada proceso en función de ciertos criterios (por ejemplo, importancia, nivel de servicio, etc.).

Puedes utilizar un esquema de prioridad para administrar las colas de procesos. Los procesos interactivos pueden recibir prioridades más altas que los procesos batch para garantizar una respuesta rápida a las interacciones del usuario.

FIFO (First-In-First-Out):

El algoritmo FIFO programa los procesos en el orden en que llegan a la cola. El primer proceso en entrar es el primero en ser ejecutado.

Este algoritmo puede ser adecuado para colas de procesos batch, ya que los procesos batch a menudo se ejecutan sin la necesidad de una respuesta inmediata y se pueden ejecutar en el orden en que se enviaron.

Shortest Job Next (SJN) o Shortest Job First (SJF):

Estos algoritmos programan el proceso con la menor duración estimada o tiempo de ejecución restante. Son ideales para minimizar el tiempo de respuesta, pero requieren información precisa sobre los tiempos de ejecución de los procesos.

Estos algoritmos pueden aplicarse a procesos interactivos y batch, pero son más efectivos cuando se dispone de información precisa sobre el tiempo de ejecución de los procesos.

24. Indique brevemente a que hacen referencia los siguientes conceptos: (a) Huella de un proceso en un procesador (b) Afinidad con un procesador (c) ¿Por qué podría ser mejor en algunos casos que un proceso se ejecute en el mismo procesador? (d) ¿Puede el usuario en Windows cambiar la afinidad de un proceso? ¿y en GNU/Linux? (e) Investigue el concepto de balanceo de carga (load balancing). (f) Compare los conceptos de afinidad y balanceo de carga y como uno afecta al otro

(a) Huella de un proceso en un procesador: La "huella de un proceso" se refiere a la cantidad de recursos de procesamiento, memoria y otros recursos del sistema que un proceso utiliza mientras se ejecuta en un procesador. Representa el impacto o carga que un proceso coloca en el procesador y en el sistema en su conjunto.

(b) Afinidad con un procesador: La "afinidad con un procesador" se refiere a la asignación de un proceso específico a un núcleo o procesador particular en un sistema multiprocesador. Esto significa que el proceso se ejecutará en ese núcleo específico en lugar de ser movido entre núcleos.

(c) ¿Por qué podría ser mejor en algunos casos que un proceso se ejecute en el mismo procesador? En algunos casos, puede ser beneficioso que un proceso se ejecute en el mismo procesador o núcleo debido a la reducción de la latencia de acceso a la memoria caché local y a la minimización de las interrupciones relacionadas con el cambio de contexto. Esto puede ser especialmente importante en aplicaciones de tiempo real y de alto rendimiento.

(d) ¿Puede el usuario en Windows cambiar la afinidad de un proceso? ¿Y en GNU/Linux? En Windows, los usuarios pueden cambiar la afinidad de un proceso. Esto se hace en el Administrador de Tareas de Windows. En GNU/Linux, los usuarios también pueden cambiar la afinidad de un proceso utilizando herramientas como "taskset" o "chrt".

(e) Balanceo de carga (Load Balancing): El balanceo de carga se refiere a la distribución equitativa de la carga de trabajo entre múltiples recursos de procesamiento, como núcleos de CPU o servidores, con el objetivo de optimizar la utilización de los recursos, mejorar la eficiencia y garantizar una respuesta rápida a las solicitudes. Se utiliza en sistemas con alta demanda para evitar la congestión y garantizar un uso equitativo de los recursos.

(f) Comparación entre afinidad y balanceo de carga: La afinidad se refiere a la asignación de un proceso a un núcleo específico, lo que puede mejorar el rendimiento en ciertos casos, pero también puede llevar a la falta de equilibrio en la carga si no se administra adecuadamente. El balanceo de carga se enfoca en distribuir la carga de manera equitativa entre los recursos de procesamiento y, a menudo, implica mover procesos entre núcleos para lograr un mejor rendimiento global y una mejor utilización de los recursos. A menudo, la asignación de afinidad y el balanceo de carga son consideraciones contrapuestas, y la elección depende de los requisitos específicos y objetivos del sistema.