

Práctica 4 – ISO

1. Responda en forma sintética sobre los siguientes conceptos:

(a) Programa y Proceso.

Programa: conjunto de instrucciones o código escrito en un lenguaje de programación que especifica una serie de acciones a realizar por una computadora para llevar a cabo una tarea específica.

- Es estático.
- No tiene program counter.
- Existe desde que se edita hasta que se borra.

Proceso: instancia en ejecución de un programa en un sistema operativo.

- Es dinámico.
- Tiene program counter.
- Su ciclo de vida comprende desde que se lo ejecuta hasta que termina.
- Poseen estados y en todo momento están en uno de ellos.

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

Tiempo de retorno (TR): tiempo que transcurre entre que el proceso llega al sistema hasta que completa su ejecución.

Tiempo de espera (TE): tiempo que el proceso se encuentra en el sistema esperando, es decir, el tiempo que pasa sin ejecutarse ($TR - \text{Tiempo de CPU "Tcpu"}$).

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

Tiempo Promedio de Retorno (TPR): es el promedio de los tiempos de retorno de todos los procesos del lote (suma de TR de cada proceso del lote / cantidad de procesos del lote).

Tiempo Promedio de Espera (TPE): Es el promedio de los tiempos de espera de todos los procesos del lote (suma de TE de cada proceso del lote / cantidad de procesos del lote).

(d) ¿Qué es el Quantum?

Es la medida que determina cuanto tiempo podrá usar el procesador cada proceso.

Pueden ser de dos tamaños:

- Pequeño: Genera un alto costo de context switch.
- Grande: Un proceso puede bloquear la CPU por mucho tiempo, pudiendo llegar a reducir la capacidad de respuesta del sistema ante otras tareas.

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

Algoritmos Apropiativos (Preemptive): el proceso que esté en ejecución en la CPU puede ser interrumpido/expulsado y llevado a la cola de listos dependiendo los criterios del algoritmo.

- Mayor overhead (costo adicional) pero mejor servicio.
- Los procesos no monopolizan el procesador.

Algoritmos No Apropiativos (Non-Preemptive): una vez que un proceso alcanza el estado "running", continua hasta que termina o se bloquea por algún evento (por ejemplo, I/O) y no puede ser expulsado/interrumpido.

(f) ¿Qué tareas realizan?

- i. Short Term Scheduler: selecciona/determina que proceso pasará a ejecutarse, es decir, se encarga de la selección de los procesos que están en estado “ready to run” que pasarán a estado “running”.
- ii. Long Term Scheduler: admite nuevos procesos a memoria controlando el grado de multiprogramación, es decir, es el encargado de seleccionar los procesos en estado “new” que serán admitidos pasando a estado “ready to run”.
- iii. Medium Term Scheduler: encargado de realizar el swapping, es decir, el intercambio entre el disco y la memoria (swapon) y viceversa (swapout) cuando el sistema operativo lo determina, esta acción puede disminuir el grado de multiprogramación. Utiliza la partición SWAP.

(g) ¿Qué tareas realiza el Dispatcher?

Se encarga de realizar el context switch (guardar el estado del proceso actual que se está ejecutando y cargar el estado de los próximos procesos a ejecutarse) y cambio del modo de ejecución. Es utilizado por el Short Term Scheduler ya que este se encarga de seleccionar los procesos, pero quien realiza el cambio del proceso de “ready to run” a “running” es el Dispatcher.

2. Procesos:

(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: herramienta interactiva que proporciona una actualización en tiempo real de los procesos que están usando la CPU y la memoria.
- htop: herramienta que muestra información detallada y en tiempo real sobre los procesos en ejecución y la utilización de recursos del sistema de una forma más amigable que el comando top. Podremos ver los procesos en lista, el árbol de procesos, utilizar funciones de búsqueda y filtrado, información sobre el sistema y la utilización de los recursos, todo esto dentro de una interfaz interactiva.
- ps: muestra información sobre los procesos en ejecución. (PID, TTY “Terminal”, TIME “tiempo de ejecución” y CMD “comando”).
- pstree: muestra una representación gráfica (en forma de árbol) de la jerarquía de procesos.
- kill: se utiliza para enviar señales a procesos en ejecución, permite la interacción con los procesos, como terminarlos de manera controlada o enviarles señales específicas. Requiere el uso del PID.
- pgrep: busca y lista los PID de los procesos que coinciden con un criterio de búsqueda específico. “pgrep [opciones] [patrón]”
- pkill: similar a pgrep, aplica la búsqueda de procesos, pero a estos les hace las funcionalidades del comando kill. “pkill [opciones] nombre_del_proceso”.
- pkillall: igual a pkill pero este permite enviar señales a procesos por su nombre o por su PID.
- killall: permite terminar procesos basados en sus nombres, a diferencia de kill que requiere del PID.
- renice: herramienta utilizada para ajustar la prioridad de ejecución de un proceso en tiempo real.
- xkill: Utilidad que se utiliza para terminar de manera forzada una aplicación gráfica que no responde o está causando problemas.
- atop: Herramienta interactiva que funciona de manera similar a top pero que muestra la carga de trabajo dentro del sistema. Indica la ocupación de los recursos más importantes del hardware a nivel del sistema y qué procesos son responsables de la carga indicada de CPU y memoria.

(b) Observe detenidamente el siguiente código. Intente entender lo que hace sin necesidad de ejecutarlo.

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main(void) {
    int c;
    pid_t pid;
    printf("Comienzo.\n");
    for (c = 0; c < 3 ; c++ )
    {
        pid = fork();
    }
    printf("Proceso\n");
    return 0;
}
```

i. ¿Cuántas líneas con la palabra “Proceso” aparecen al final de la ejecución de este programa?

Al final de la ejecución de este programa aparecen 8 líneas con la palabra “Proceso”.

ii. ¿El número de líneas es el número de procesos que han estado en ejecución? Ejecute el programa y compruebe si su respuesta es correcta, modifique el valor del bucle for y compruebe los nuevos resultados.

Si, el número de líneas que imprimen la palabra “Proceso” son la cantidad de procesos que se crean. Si ejecutamos el programa modificando el valor del bucle for vamos a ver que la cantidad de procesos crece exponencialmente en potencias de 2

(c) Vamos a tomar una variante del programa anterior. Ahora, además de un mensaje, vamos a añadir una variable y, al final del programa vamos a mostrar su valor. El nuevo código del programa se muestra a continuación.

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main(void) {
    int c;
    int p=0;
    pid_t pid;
    for (c = 0; c < 3 ; c++ )
    {
        pid = fork();
    }
    p++;
    printf("Proceso %d\n", p);
    return 0;
}
```

i. ¿Qué valores se muestran por consola?

Por consola se muestran 8 líneas con la palabra “Proceso1” ya que el valor de la variable “p” cada vez que se hace un “fork()” es 0.

ii. ¿Todas las líneas tendrán el mismo valor o algunas líneas tendrán valores distintos?

Todas tendrán el mismo valor.

iii. ¿Cuál es el valor (o valores) que aparece? Ejecute el programa y compruebe si su respuesta es correcta, Modifique el valor del bucle for y el lugar dónde se incrementa la variable p y compruebe los nuevos resultados.

Ya vimos lo que pasa si modificamos el valor del bucle for, ahora, si hacemos que el valor de “p” se incremente antes o después del bucle for, el resultado va a ser el mismo, pero si hacemos que la variable “p” se incremente dentro del bloque for el resultado será 8 líneas con “Proceso3”.

(d) Comunicación entre procesos:

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

Un Pipe (|) permite la comunicación entre procesos al redirigir la salida estándar de un proceso hacia la entrada estándar de otro. Permite el procesamiento en tiempo real de la salida de un comando antes de ser utilizada por otro.

ii. ¿Cómo se crea un pipe en C?

Para crear un Pipe en C lo podemos hacer mediante la función “pipe()” que se encuentra en la biblioteca de manejo de archivos “unistd.h”, además es necesario manejar un arreglo de enteros de dos posiciones para almacenar los descriptores de archivo (números que el sistema operativo usa para identificar y manipular archivos y canales de comunicación como los pipes) donde en la primera se usará para hacer referencia al extremo de lectura del pipe y la segunda posición hará referencia al extremo de escritura. “pipe(fd[N]) siendo N 0 ó 1 y fd un arreglo de enteros”.

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

La función “pipe()” necesita como parámetro un arreglo de enteros de dos posiciones, este es importante ya que es la forma en que se comunica al programa dónde se encuentran los extremos del pipe para comunicar los procesos.

Es necesario también combinar el uso de pipes con la función “close()” que se encarga de cerrar descriptores de archivos haciendo que se liberen recursos de memoria

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

Con los Pipes es posible lograr la comunicación entre procesos, estos son unidireccionales, es decir, un proceso puede escribir o leer un pipe, pero no ambas a la vez, y además, solo podría hacerlo un proceso padre a su hijo por ejemplo. Aunque sean unidireccionales, se puede combinar el uso de dos pipes para crear un canal bidireccional.

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

La información mínima que el sistema operativo debe tener sobre un proceso es el PID, el PPID, la Ubicación del mismo en memoria, los Recursos Asignados (espacio de memoria, descriptores de archivos, etc.), el Accounting, la Entrada y Salida, el Contexto de Ejecución (valores de los registros de la CPU), y la Planificación del mismo (prioridad, estado, tiempo consumido, etc.).

Toda esta información se almacena en una estructura llamada PCB (Process Control Block).

- Existe una por proceso.
- Es lo primero que se crea cuando un proceso es creado y también es lo último que se borra cuando el proceso termina.
- Tiene referencias a memoria.

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

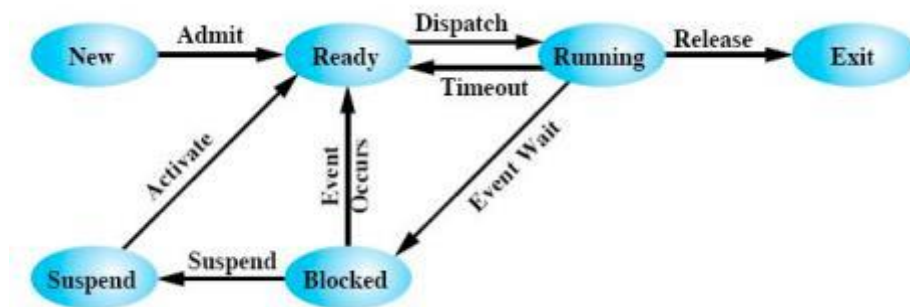
CPU Bound: un proceso es CPU Bound cuando su tiempo de ejecución está ligado y limitado por la CPU, normalmente son procesos que realizan tareas intensivas y consumen una gran cantidad de ciclos de CPU.

I/O Bound: un proceso es I/O Bound cuando su tiempo de ejecución está ligado y limitado por la entrada/salida, normalmente pasan la mayor parte de su tiempo esperando a que las operaciones de I/O se completen en vez de realizar tareas intensivas sin consumir una gran cantidad de ciclos de CPU.

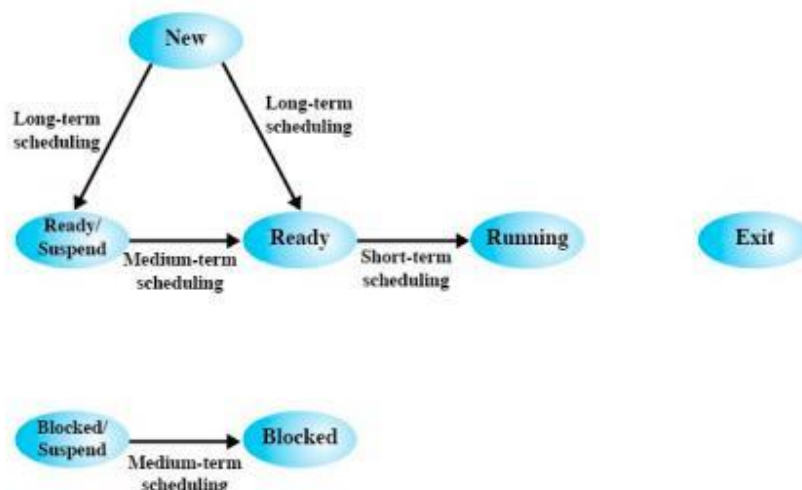
(g) ¿Cuáles son los estados posibles por los que puede atravesar un proceso?

- “New”: en este estado se crea la PCB y se carga al proceso en memoria dejando al proceso en un estado de inicialización.
- “Ready to run”: El proceso está listo para cargar su Program Counter y empezar a ejecutarse, pero no lo está haciendo, se encuentra almacenado en la cola de listos compitiendo por la CPU.
- “Running”: Se genera un cambio de contexto y el proceso empieza su ejecución teniendo la CPU a su disposición hasta que se termine su Quantum, se necesite una operación de I/O, hasta que termine o hasta que sea bloqueado. Puede tener tres salidas: “Exit”, “Waiting” o “Ready to run”.
- “Exit”: Se libera la CPU y se eliminan todas las estructuras del proceso en la memoria.
- “Waiting”: El proceso pasa a este estado cuando necesita esperar que se cumpla un evento para seguir ejecutándose (System calls, I/O, etc.). Queda en un estado de bloqueo hasta que la espera termina.

(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?



3. Para los siguientes algoritmos de scheduling:

- FCFS (First Come First Served)
- SJF (Shortest Job First)
- Round Robin
- Prioridades

(a) Explique su funcionamiento mediante un ejemplo.

Job	Llegada	CPU	Prioridad
1	0	9	3
2	1	5	2
3	2	3	1
4	3	7	2

FCFS:

- Cuando hay que elegir un proceso para ejecutar, se selecciona el más viejo.
- No favorece a ningún tipo de procesos.
- Los CPU Bound terminan al comenzar su primera ráfaga mientras que los I/O Bound no.
- Es no apropiativo.
- Orden de ejecución: Job 1, Job 2, Job 3, Job 4.

SJF:

- Selecciona al proceso con la ráfaga más corta.
- Los procesos cortos y los I/O Bound se ven beneficiados.
- Los procesos largos pueden sufrir starvation (inanición) ya que pueden pasar mucho tiempo sin ser seleccionados.
- Es no apropiativo.
- Orden de ejecución: Job 1, Job 3, Job 2, Job 4.

Round Robin:

- Política basada en el uso de un reloj (Quantum).
- Cuando un proceso es expulsado de la CPU es colocado al final de la cola de listos y se selecciona otro generando un FIFO circular.
- Existe un “contador” que indica las unidades de CPU en las que el proceso se ejecutó. Cuando este llega a 0 el proceso es expulsado.
- El “contador” puede ser Global o Local (PCB).
- Se puede tener 2 variantes con respecto al valor inicial del “contador” cuando un proceso es asignado a la CPU: Timer Variable o Timer Fijo.
- Es apropiativo.

RR	LLEGADA	CPU	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
1	0	9	>1	2	3	4												5	6	7	8					9<
2	1	5		>			1	2	3	4												5<				
3	2	3			>					1	2	3<														
4	3	7				>							1	2	3	4						5	6	7<		

Prioridades:

- Cada proceso tiene un valor que representa su prioridad, mientras menor sea su valor, mayor prioridad.
- Se selecciona al proceso con mayor prioridad de la cola de listos.
- Puede ser apropiativo o no.
- Existe una cola de listos para cada nivel de prioridad.
- Los procesos de baja prioridad pueden sufrir starvation (inanición).
- Orden de ejecución No Apropiativo: Job 1, Job 3, Job 2, Job 4

(b) ¿Alguno de ellos requiere algún parámetro para su funcionamiento?

El algoritmo de Round Robin requiere la definición del tamaño del Quantum.

(c) Cual es el más adecuado según los tipos de procesos y/o SO.

- FIFO/FCFS: No favorece a ningún tipo de procesos.
- SJF: Favorece a los I/O Bound y a los procesos cortos.
- Round Robin: Favorece más a los I/O Bound.
- Prioridades: Favorece a los de mayor prioridad.

(d) Cite ventajas y desventajas de su uso.

FIFO es sencillo de implementar y asegura que los procesos se ejecuten en el orden en que llegan, lo cual es justo para los primeros en la cola. Sin embargo, su principal desventaja es el problema del convoy, donde procesos cortos pueden quedar esperando mucho tiempo si están detrás de procesos largos.

SJF minimiza el tiempo promedio de espera al priorizar los procesos más cortos, lo que lo hace eficiente en ciertas cargas de trabajo. No obstante, es difícil predecir el tiempo de ejecución de un proceso, y este enfoque puede ser injusto al favorecer procesos cortos mientras discrimina los largos.

Round Robin es equitativo, ya que distribuye el tiempo de CPU de manera uniforme entre los procesos. Es ideal para sistemas interactivos o de tiempo compartido. Sin embargo, tiene como desventaja el aumento del overhead debido al cambio de contexto, y su eficiencia depende de un ajuste adecuado del quantum.

La planificación por prioridades permite dar mayor relevancia a los procesos más críticos, lo que la hace flexible y útil en diversos escenarios. Por otro lado, puede generar inanición, donde procesos de baja prioridad no se ejecutan, y su implementación es más compleja en comparación con otros algoritmos.

4. Para el algoritmo Round Robin, existen 2 variantes:

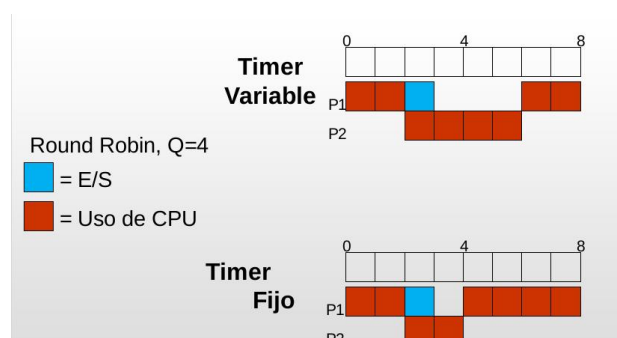
- *Timer Fijo*
- *Timer Variable*

(a) ¿Qué significan estas 2 variantes?

Timer Variable: el "contador" se inicializa en Q (contador = Q) cada vez que un proceso es asignado a la CPU, es decir, si un proceso no llega a usar todo su Quantum, el sobrante se tira a la basura y no se reutiliza.

Timer Fijo: El "contador" se inicializa en Q (contador = Q) cuando su valor es cero, es decir, si un proceso no llega a usar todo su Quantum, el sobrante lo usará el siguiente proceso a ejecutarse y cuando el Quantum llegue a cero, se reasignará su valor.

(b) Explique mediante un ejemplo sus diferencias.



(c) En cada variante ¿Dónde debería residir la información del Quantum?

Para el Timer Variable, cada proceso debería almacenar la información de su Quantum en su PCB, mientras que con Timer Fijo, la información se debe almacenar en una estructura o espacio que sea accesible directamente por el sistema operativo.

5. Se tiene el siguiente lote de procesos que arriban al sistema en el instante 0 (cero):

Job	Unidades de CPU
1	7
2	15
3	12
4	4
5	9

(a) Realice los diagramas de Gantt según los siguientes algoritmos de scheduling:

- i. FCFS (First Come, First Served)
- ii. SJF (Shortest Job First)
- iii. Round Robin con quantum = 4 y Timer Fijo
- iv. Round Robin con quantum = 4 y Timer Variable

(b) Para cada algoritmo calcule el TR y TE para cada job así como el TPR y el TPE.

Solución en la carpeta de Diagramas de Gantt.

(c) En base a los tiempos calculados compare los diferentes algoritmos.

Para este lote de procesos, el algoritmo con mejores promedios fue el SJF y el que tuvo peores promedios fue el Round Robin de Timer Variable con Quantum 4.

6. Se tiene el siguiente lote de procesos:

(a) Realice los diagramas de Gantt según los siguientes algoritmos de scheduling:

Job	Llegada	Unidades de CPU
1	0	4
2	2	6
3	3	4
4	6	5
5	8	2

- i. FCFS (First Come, First Served)
- ii. SJF (Shortest Job First)
- iii. Round Robin con quantum = 1 y Timer Variable
- iv. Round Robin con quantum = 6 y Timer Variable

(b) Para cada algoritmo calcule el TR y TE para cada job así como el TPR y el TPE.

Solución en la carpeta de Diagramas de Gantt.

(c) En base a los tiempos calculados compare los diferentes algoritmos.

El algoritmo con mejores promedios fue el SJF, mientras que el algoritmo con peor promedio fue el Round Robin de Timer Variable con Quantum 1.

(d) En el algoritmo Round Robin, que conclusión se puede sacar con respecto al valor del quantum.

Respecto al valor del Quantum podemos decir que mientras más chico sea, más cambios de contexto habrá, haciendo que los promedios se disparen, y cuanto más grande sea, se comporta de manera similar al FCFS disminuyendo los promedios.

(e) ¿Para el algoritmo Round Robin, en qué casos utilizaría un valor de quantum alto y que ventajas y desventajas obtendría?

Se podría utilizar un quantum alto para manejar lotes de procesos CPU Bound ya que estos en lo general ocupan por un largo tiempo los recursos de la CPU, disminuyendo las tasas de tiempo de retorno de todos estos procesos.

La desventaja de esto sería que, si en el lote de procesos nos encontramos procesos cortos o por ejemplo procesos I/O Bound que por lo general no hacen mucho consumo de la CPU, con un quantum alto podríamos generar que las tasas de tiempo de espera se disparen, haciendo que los procesos cortos tengan que esperar muchos ciclos de CPU para poder ejecutarse.

7. Una variante al algoritmo SJF es el algoritmo SJF apropiativo o SRTF (Shortest Remaining Time First):

(a) Realice el diagrama del Gantt para este algoritmo según el lote de trabajos del ejercicio 6.

Solución en la carpeta de Diagramas de Gantt.

(b) ¿Nota alguna ventaja frente a otros algoritmos?

Genera un buen promedio de tiempo de espera, por lo tanto, favorece ventajosamente a procesos I/O Bound ya que estos dependen del tiempo de espera, también los procesos cortos se verían beneficiados.

8. Suponga que se agregan las siguientes prioridades al lote de procesos del ejercicio 6, donde un menor número indica mayor prioridad:

Job Prioridad	
1	3
2	4
3	2
4	1
5	2

(a) Realice el diagrama de Gantt correspondiente al algoritmo de planificación por prioridades según las variantes:

i. No Apropiativa

ii. Apropiativa

(b) Calcule el TR y TE para cada job así como el TPR y el TPE.

Solución en la carpeta de Diagramas de Gantt.

(c) ¿Nota alguna ventaja frente a otros algoritmos? ¿Bajo qué circunstancias lo utilizaría y ante que situaciones considera que la implementación de prioridades podría no ser de mayor relevancia?

Las ventajas es que para tareas críticas del sistema operativo que necesiten ser ejecutadas inmediatamente, van a ser muy buenos, además de que brindan una mayor adaptabilidad en nuestro sistema si decidimos llevar una organización por prioridades. Los casos donde los utilizaría son justamente los mencionados en las ventajas.

9. Inanición (Starvation)

(a) ¿Qué significa?

Cuando un proceso sufre de Starvation/Inanición significa que está en una situación donde no puede avanzar o ejecutarse debido a la competencia por recursos del sistema entre diferentes procesos. Ocurre cuando un proceso no puede obtener los recursos necesarios para su ejecución porque otros están acaparándolos.

(b) ¿Cuál/es de los algoritmos vistos puede provocarla?

De los algoritmos vistos los que pueden generar inanición serían el SJF, el STRF y el de Prioridades. Los 2 primeros generándola para procesos largos y el tercero en procesos de baja prioridad.

(c) ¿Existe alguna técnica que evite la inanición para el/los algoritmos mencionados en b?

- Envejecimiento (Aging): consiste en aumentar la prioridad de un proceso a medida que pasa el tiempo sin ser seleccionado para su ejecución.
- Establecimiento de límites de espera: consiste en establecer límites de tiempo máximo para la espera de un proceso en la cola de listos, haciendo que, si llega a este límite, pase a ejecutarse.
- Prioridades dinámicas: podemos tener sistemas que ajusten las prioridades de los procesos dinámicamente según el comportamiento y las necesidades del proceso.

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 más).

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.

Para los siguientes algoritmos de Scheduling:

- FCFS
- Round Robin con quantum = 2 y timer variable.

Y suponiendo que la cola de listos de todos los dispositivos se administra mediante FCFS, realice los diagramas de Gantt según las siguientes situaciones:

(a) Suponga que al lote de procesos del ejercicio 6 se agregan las siguientes operaciones de entrada salida:

Job	I/O (rec,ins,dur)
1	(R1, 2, 1)
2	(R2, 3, 1) (R2, 5, 2)
4	(R3, 1, 2) (R3, 3, 1)

(b) Suponga que al lote de procesos del ejercicio 6 se agregan las siguientes operaciones de entrada salida:

Job	I/O (rec,ins,dur)
1	(R1, 2, 3) (R1, 3, 2)
2	(R2, 3, 2)
3	(R2, 2, 3)
4	(R1, 1, 2)

Solución en la carpeta de Diagramas de Gantt.

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

El algoritmo Round Robin para procesos que son CPU Bound podría presentar como desventajas que los tiempos de espera se vuelven bastante grandes, que se genera cierto desperdicio de tiempo de CPU si utilizamos Timer Variable y un proceso termina antes de terminar su Quantum, y también podemos incluir como desventaja que al usar este algoritmo en procesos CPU Bound se puede producir un Overhead de context switch que reduzca la eficiencia del sistema.

En cuanto a los procesos I/O Bound, este algoritmo es más adecuado para estos tipos de procesos, pero presenta ciertas desventajas a su vez como pueden ser problemas de inanición, ineficiencia en la utilización de la CPU y retrasos en la finalización de estos procesos ya que pueden ser interrumpidos por el context switch.

(b) SRTF (Shortest Remaining Time First)

Este algoritmo beneficia a los procesos I/O Bound, pero para los procesos CPU Bound creo que este algoritmo podría generar inanición de procesos con larga duración y también podría generar una sobrecarga de cambios de contexto si la mayoría de procesos que están en la cola de listos tienen un tiempo de ejecución similar.

12. Para equiparar la desventaja planteada en el ejercicio 11), se plantea la siguiente modificación al algoritmo:

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 elige un proceso de la cola auxiliar se le otorga el procesador por tantas unidades de tiempo como le falta 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.

(a) Analice el funcionamiento de este algoritmo mediante un ejemplo. Marque en cada instante en que cola se encuentran los procesos.

(b) Realice el ejercicio 10)a) nuevamente considerando este algoritmo, con un quantum de 2 unidades y Timer Variable.

Solución en la carpeta de Diagramas de Gantt.

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.

Sí, esto podría ocurrir si un proceso termina su ejecución en una ráfaga de CPU antes de que se termine su Quantum. Sin embargo, en condiciones normales y con un sistema operativo bien diseñado, esto no ocurrirá.

14. El algoritmo SJF (y SRTF) tiene como problema su implementación, dada la dificultad de conocer la duración de la próxima ráfaga de CPU. Es posible realizar una estimación de la próxima, utilizando la media de las ráfagas de CPU para cada proceso. Así, por ejemplo, podemos tener la siguiente formula:

$$S_{n+1} = \frac{1}{n}T_n + \frac{n-1}{n}S_n \quad (1)$$

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 fórmula 1, con un valor inicial estimado de S1=10.

La fórmula anterior 1 les 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 más recientes y menor a casos viejos (o viceversa). Se plantea la siguiente formula:

$$S_{n+1} = \alpha T_n + (1 - \alpha)S_n \quad (2)$$

Con $0 < \alpha < 1$.

(b) Analice para que valores de α se tienen en cuenta los casos más recientes.

(c) Para la situación planteada en a) calcule que valores se obtendrían si se utiliza la fórmula 2 con $\alpha = 0, 2$; $\alpha = 0, 5$ y $\alpha = 0, 8$.

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

15. Colas Multinivel

Hoy en día los algoritmos de planificación vistos se han ido combinando para formar algoritmos más eficientes. Así surge el algoritmo de Colas Multinivel, donde la cola de procesos listos es dividida en varias colas, teniendo cada una su propio algoritmo de planificación.

(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.

Para procesos Interactivos usaría Round Robin y para procesos Batch usaría FCFS.

(b) Para el caso de las dos colas vistas en a: ¿Qué algoritmo utilizaría para planificarlas?

Yo utilizaría un algoritmo por Prioridades con Aging.

16. Suponga que en un SO se utiliza un algoritmo de planificación de colas multinivel. El mismo cuenta con 3 colas de procesos listos, en las que los procesos se encolan en una u otra según su prioridad. Hay 3 prioridades (1, 2, 3), donde un menor número indica mayor prioridad.

Se utiliza el algoritmo de prioridades para la administración entre las colas. Se tiene el siguiente lote de procesos a ser procesados con sus respectivas operaciones de I/O:

Job	Llegada	CPU	I/O (rec,ins,dur)	Prioridad
1	0	9	(R1, 4, 2) (R2, 6, 3) (R1, 8, 3)	1
2	1	5	(R3, 3, 2) (R3, 4, 2)	2
3	2	5	(R1, 4, 1)	3
4	3	7	(R2, 1, 2) (R2, 5, 3)	2
5	5	5	(R1, 2, 3) (R3, 4, 3)	1

Suponiendo que las colas de cada dispositivo se administran a través de FCFS y que cada cola de procesos listos se administra por medio de un algoritmo RR con un quantum de 3 unidades y Timer Variable, realice un diagrama de Gantt:

(a) Asumiendo que NO hay apropiación entre los procesos.

(b) Asumiendo que hay apropiación entre los procesos.

Solución en la carpeta de Diagramas de Gantt.

17. En el esquema de Colas Multinivel, cuando se utiliza un algoritmo de prioridades para administrar las diferentes colas los procesos pueden sufrir starvation.

La técnica de envejecimiento se puede aplicar a este esquema, haciendo que un proceso cambie de una cola de menor prioridad a una de mayor prioridad, después de cierto periodo de tiempo que el mismo se encuentra esperando en su cola. Luego de llegar a una cola en la que el proceso llega a ser atendido, el mismo retorna a su cola original.

Por ejemplo: Un proceso con prioridad 3 está en cola su cola correspondiente. Luego de X unidades de tiempo, el proceso se mueve a la cola de prioridad 2. Si en esta cola es atendido, retorna a su cola original, en caso contrario luego de sucederse otras X unidades de tiempo el proceso se mueve a la cola de prioridad 1. Esta última acción se repite hasta que el proceso obtiene la CPU, situación que hace que el mismo vuelva a su cola original.

(a) Para los casos a y b del ejercicio 16 realice el diagrama de Gantt considerando además que se tiene un envejecimiento de 4 unidades.

Solución en la carpeta de Diagramas de Gantt.

18. La situación planteada en el ejercicio 17, donde un proceso puede cambiar de una cola a otra, se la conoce como Colas Multinivel con Realimentación.

Suponga que se quiere implementar un algoritmo de planificación que tenga en cuenta el tiempo de ejecución consumido por el proceso, penalizando a los que más tiempo de ejecución tienen. (Similar a la tarea del algoritmo SJF que tiene en cuenta el tiempo de ejecución que resta).

Utilizando los conceptos vistos de Colas Multinivel con Realimentación indique que colas implementaría, que algoritmo usaría para cada una de ellas, así como para la administración de las colas entre sí.

Tenga en cuenta que los procesos no deben sufrir inanición.

Manejaría por ejemplo 3 colas cada una con un nivel de prioridad, los procesos más largos tendrían menor prioridad y los más cortos tendrían mayor prioridad, cada cola individualmente la manejaría usando un RR con un Quantum chico que favorece a los procesos de menor duración y para solucionar el problema de inanición de los procesos más largos, para administrar las colas entre ellas usaría un algoritmo por prioridades con aging para que los procesos más largos vayan escalando en prioridad a medida que los procesos más chicos van terminando su ejecución.

19. Un caso real: "Unix Clásico" (SVR3 y BSD 4.3)

(a) Analizando la jerarquía descrita para las bandas de prioridades: ¿Qué tipo de actividad considera que tendrá más prioridad? ¿Por qué piensa que el scheduler prioriza estas actividades?

Yo creo que las actividades que tendrán más prioridad son las que requieran una interacción con el usuario o las que conlleven que se use poco la CPU ya que si el scheduler selecciona estas actividades podría minimizar el tiempo de espera para esos procesos y maximizar el poco uso que estos procesos puedan requerir de CPU, dejándole paso a procesos CPU Bound de los cuáles el usuario promedio no se entera de su existencia.

(b) Para el caso de los procesos de usuarios, y analizando las funciones antes descritas: ¿Qué tipo de procesos se encarga de penalizar? (o equivalentemente se favorecen). Justifique

Los procesos que se verían favorecidos serían los procesos I/O Bound ya que con estos el usuario interactúa más.

(c) La utilización de RR dentro de cada cola: ¿Verdaderamente favorece al sistema de Tiempo Compartido? Justifique.

Yo creo que el favorecer o no lo determina el hecho de cómo se configure el Round Robin, es decir, tener en cuenta el tamaño del Quantum ya que si es muy pequeño puede generar un overhead de context switch, mientras que si es muy grande puede disminuir la capacidad de respuesta del sistema. Como también lo determina el número de niveles en las colas multinivel y la configuración de los niveles de prioridad. Pero en general, si se configura bien, el usar Round Robin puede proporcionar equidad, prevención de inanición, respuestas rápidas del sistema y una adaptabilidad a diversas cargas de trabajo.

20. A cuáles de los siguientes tipos de trabajos:

(a) cortos acotados por CPU

(b) cortos acotados por E/S

(c) largos acotados por CPU

(d) largos acotados por E/S

benefician las siguientes estrategias de administración:

(a) prioridad determinada estáticamente con el método del más corto primero (SJF).

(b) prioridad dinámica inversamente proporcional al tiempo transcurrido desde la última operación de E/S.

a) Cortos acotados por CPU se ven beneficiados.

Cortos acotados por E/S se ven beneficiados.

Largos acotados por CPU se ven penalizados siempre que haya procesos más cortos antes que estos.

Largos acotados por E/S se ven penalizados por lo mismo que los anteriores más el tiempo de espera para usar la E/S.

b) Tomándolo como mientras menor sea el tiempo transcurrido desde la última operación de entrada y salida mayor va a ser la prioridad, creo que los únicos procesos que se beneficiarían serían los que usan la E/S ya que serían los únicos capaces de escalar en prioridad.

21. Explicar por qué si el quantum "q" en Round-Robin se incrementa sin límite, el método se aproxima a FIFO.

Esto ocurre porque va a llegar un punto donde el Quantum sea tan grande que todos los procesos terminen o salgan de la CPU sin ser expulsados por el algoritmo, eliminando toda la rotación de procesos que plantea el Round Robin convirtiendo la cola en una circular del estilo FIFO.

22. Los sistemas multiprocesador pueden clasificarse en:

Homogéneos: Los procesadores son iguales. Ningún procesador tiene ventaja física sobre el resto.

Heterogéneos: Cada procesador tiene su propia cola y algoritmo de planificación.

Otra clasificación posible puede ser:

Multiprocesadores débilmente acoplados: Cada procesador tiene su propia memoria principal y canales.

Procesadores especializados: Existe uno o más procesadores principales de propósito general y varios especializados controlados por el primero (ejemplo: procesadores de E/S, procesadores Java, procesadores Criptográficos, etc.).

Multiprocesador fuertemente acoplado: Consta de un conjunto de procesadores que comparten una memoria principal y se encuentran bajo el control de un Sistema Operativo.

(a) ¿Con cuál/es de estas clasificaciones asocia a las PCs de escritorio habituales?

Con los Homogéneos o los Multiprocesador fuertemente acoplados.

(b) ¿Qué significa que la asignación de procesos se realice de manera simétrica?

Esto significa que todos los procesadores o núcleos de la CPU tienen la misma prioridad y pueden ejecutar cualquier tarea. No existe una distinción jerárquica entre los procesadores en términos de capacidad de procesamiento o acceso a recursos del sistema.

(c) ¿Qué significa que se trabaje bajo un esquema Maestro/esclavo?

Esto significa que establece una jerarquía entre los procesadores para coordinar y distribuir las tareas, el Maestro tiene el control central y toma decisiones críticas, mientras que los esclavos ejecutan las tareas asignadas. Esto puede facilitar la coordinación y especialización de tareas, pero también requiere una planificación cuidadosa y buena sincronización para un funcionamiento eficiente y confiable en el sistema.

23. Asumiendo el caso de procesadores homogéneos:

(a) ¿Cuál sería el método de planificación más sencillo para asignar CPUs a los procesos?

(b) Cite ventajas y desventajas del método escogido

“Si se asume que la arquitectura del multiprocesador es uniforme, en el sentido de que ningún procesador tiene una ventaja física particular con respecto al acceso a memoria principal o a dispositivos de E/S, entonces el enfoque más simple de la planificación consiste en tratar cada proceso como un recurso

colectivo y asignar procesos a procesadores por demanda. Surge la cuestión de si la asignación debería ser estática o dinámica.

Si un proceso se vincula permanentemente a un procesador desde su activación hasta que concluye, entonces se mantiene una cola a corto plazo dedicada por cada procesador. Una ventaja de esta estrategia es que puede haber menos sobrecarga en la función de planificación, dado que la asignación a un procesador se realiza una vez y para siempre. Asimismo, el uso de procesadores dedicados permite una estrategia conocida como planificación de grupo o pandilla, como se verá más adelante.

Una desventaja de la asignación estática es que un procesador puede estar ocioso, con su cola vacía, mientras otro procesador tiene trabajo acumulado. Para evitar esta situación, puede utilizarse una cola común. Todos los procesos van a una cola global y son planificados sobre cualquier procesador disponible. Así, a lo largo de la vida de un proceso, puede ser ejecutado en diferentes procesadores en diferentes momentos.”

Sistemas Operativos 5ta edición, William Stallings.

24. Indique brevemente a que hacen referencia los siguientes conceptos:

(a) Huella de un proceso en un procesador

Es el estado que un proceso va dejando en la caché de un procesador, información como la cantidad de memoria RAM que ocupa, la cantidad de CPU que consume, los archivos y dispositivos que utiliza, entre otras cosas.

(b) Afinidad con un procesador

Es la preferencia de un proceso para ejecutarse en un procesador específico.

(c) ¿Por qué podría ser mejor en algunos casos que un proceso se ejecute en el mismo procesador?

El por qué se combina con el concepto de huella ya que, si un proceso tiene mayor afinidad con un procesador, cada vez que se ejecute irá dejando su huella allí por lo tanto es más probable que el procesador tenga almacenada información de ese proceso en su caché, reduciendo el tiempo necesario para la ejecución del proceso en cuestión.

(d) ¿Puede el usuario en Windows cambiar la afinidad de un proceso? ¿y en GNU/Linux?

Sí, en ambos sistemas operativos es posible que el usuario pueda cambiar la afinidad de un proceso.

(e) Investigue el concepto de balanceo de carga (load balancing).

Es una técnica utilizada para distribuir de manera equitativa la carga de trabajo entre múltiples recursos. Tiene como objetivo principal mejorar la eficiencia, disponibilidad y rendimiento del sistema al evitar la sobrecarga de un recurso particular y aprovechar al máximo los recursos disponibles.

(f) Compare los conceptos de afinidad y balanceo de carga y como uno afecta al otro.

Los dos conceptos son estrategias importantes en la administración de procesos y recursos en un sistema. La afinidad se ve más centrada en la asignación específica de procesos a recursos (eficiencia local), mientras que el balanceo de carga busca la distribución equitativa de la carga entre los recursos (optimización del uso global de recursos).

Conflicto potencial: La afinidad puede limitar el balanceo de carga, ya que restringe el movimiento de procesos a otros procesadores, incluso si están menos cargados.

Efecto positivo: La afinidad también puede mejorar la eficiencia del balanceo al reducir los costos de migración entre procesadores (por ejemplo, reutilizando caché al mantener un proceso en el mismo núcleo).

25. Si a la tabla del ejercicio 6 la modificamos de la siguiente manera: Y considerando que el scheduler de los Sistemas Operativos de la familia Windows utiliza un mecanismo denominado *preferred processor* (procesador preferido). El scheduler usa el procesador preferido a modo de afinidad cuando el proceso está en estado ready. De esta manera el scheduler asigna este procesador a la tarea si este está libre.

Job	Llegada	CPU	Afinidad
1	0	4	CPU0
2	2	6	CPU0
3	3	4	CPU1
4	6	5	CPU1
5	8	2	CPU0

(a) Ejecute el esquema anterior utilizando el algoritmo anterior.

(b) Ejecute el esquema anterior. Pero ahora si el procesador preferido no está libre es asignado a otro procesador. Luego el procesador preferido de cada job es el último en el cual ejecuto.

(c) Para cada uno de los casos calcule el tiempo promedio de retorno y el tiempo promedio de espera.

Solución en la carpeta de Diagramas de Gantt.

(d) ¿Cuál de las dos alternativas planteadas es más performante?

La opción B.