Introducción a los Sistemas Operativos

Práctica 4

1. Responda en forma sintética sobre los siguientes conceptos:
(a) Programa y Proceso.
Un proceso es un programa en ejecución e incluye el código y su estado en memoria.
Un programa es un conjunto de instrucciones estáticas almacenadas en disco.
(b) Defina Tiempo de retorno (TR) y Tiempo de espera (TE) para un Job.
Tiempo de retorno (TR):Tiempo total desde que un job ingresa hasta que finaliza.
Tiempo de espera (TE): Tiempo total que un job pasa en cola, sin ejecutarse.
(c) Defina Tiempo Promedio de Retorno (TPR) y Tiempo promedio de espera (TPE) para un lote de JOBS.
Defina Tiempo Promedio de Retorno **TPR:** Promedio del tiempo de retorno de todos los jobs.
Tiempo promedio de espera **TPE:** Promedio del tiempo de espera de todos los jobs.
(d) ¿Qué es el Quantum?
Es el intervalo de tiempo máximo asignado a un proceso en sistemas con planificación de tipo Round Robin. Al finalizar el quantum, el proceso pierde el CPU si no ha terminado.
(e) ¿Qué significa que un algoritmo de scheduling sea apropiativo o no apropiativo (Preemptive o Non-Preemptive)?
Apropiativo significa que se le puede interrumpir un proceso en ejecución para darle cpu a otro
No apropiativo significa que desde que el proceso comienza hasta que termina o se bloquea no sale de cpu.
(f) ¿Qué tareas realizan?:
i. Short Term Scheduler: Selecciona qué proceso se ejecutará a continuación y asigna el CPU.
ii. Long Term Scheduler: Controla la admisión de procesos al sistema, determinando qué procesos entran a la cola de listos.
iii. Medium Term Scheduler: Realiza la suspensión y reactivación de procesos, gestionando memoria y recursos.
(g) ¿Qué tareas realiza el Dispatcher?
El dispatcher se encarga de cambiar el contexto entre procesos, pasar el control al proceso seleccionado por el Short Term Scheduler y configurar registros y tiempos necesarios para su ejecución.

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):
i. top Muestra en tiempo real los procesos activos y su consumo de recursos del sistema, como CPU y memoria. Es útil para monitorear y gestionar el rendimiento de los procesos.

ii. htop muestra en tiempo real los procesos activos y su consumo de recursos del sistema, como CPU y memoria. Es útil para monitorear y gestionar el rendimiento de los procesos.
iii. ps Muestra información sobre los procesos en ejecución. Con opciones como `ps aux`, se obtienen todos los procesos del sistema, incluyendo información sobre el PID, usuario, uso de CPU, memoria y comandos asociados.
iv. pstree presenta los procesos en un formato de árbol jerárquico, lo cual es útil para visualizar la relación de dependencia entre procesos, ya que muestra los procesos hijos y sus procesos padres.
v. kill Envía una señal a un proceso para realizar una acción, como terminarlo. Por ejemplo, `kill -9 <PID>` termina un proceso de inmediato. El PID (Process ID) debe conocerse para usar este comando.
vi. pgrep busca y devuelve el PID de procesos que coinciden con un patrón. Útil para combinar con `kill` para terminar procesos específicos sin conocer su PID exacto (`pgrep <nombre del proceso>`).
vii. killall permite terminar todos los procesos que coinciden con un nombre específico. A diferencia de kill, no requiere el PID, sino el nombre del proceso (killall <nombre del proceso>).
viii. renice Cambia la prioridad de un proceso en ejecución, ajustando su nivel de "nice" (prioridad de CPU). Una prioridad más baja aumenta la prioridad del proceso. Ejemplo: `renice 10 <PID>`.
ix. xkill Cierra una aplicación de forma gráfica. Cuando se ejecuta, el cursor se transforma en una "X" y permite hacer clic en la ventana de un programa para finalizarlo. Es útil para aplicaciones con interfaz gráfica que no responden.
x. atop Monitorea y presenta el rendimiento del sistema y de los procesos, similar a `top`, pero con mayor detalle. Proporciona un registro histórico de métricas del sistema, como uso de CPU, memoria y disco, lo que facilita el análisis de rendimiento a lo largo del tiempo. Puede que necesites instalarlo (`sudo apt install atop` en Debian).

(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?.

En la primera iteración, el proceso original crea un hijo. En la segunda iteración, tanto el proceso original como su primer hijo crean un proceso adicional, lo que resulta en un total de 4 procesos (2 procesos adicionales en esta iteración). En la tercera iteración, cada uno de estos 4 procesos crea un nuevo proceso, generando un total de 8 procesos en esta última iteración.

Así, en total se crean 2^3=8 2^3 = 8 2^3=8 procesos, incluidos el proceso original y los procesos hijos.

- Cada uno de estos procesos ejecuta la línea `printf("Proceso\n");`, por lo que **aparecen 8 líneas con la palabra "Proceso"** en la salida.

ii. ¿El número de líneas es el número de procesos que han estado en ejecución?.
Sí, cada línea de "Proceso" representa una ejecución de la línea `printf`, que es llamada una vez en cada proceso que se creó.
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, representa una ejecución de la línea `printf`, que es llamada una vez en cada proceso que se creó.
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?.
Los 8 proceso 1
ii. ¿Todas las líneas tendrán el mismo valor o algunas líneas tendrán valores distintos?.
Tendrían el mismo valor cada línea de código
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.
Proceso 1
Si pongo el p dentro del bucle el número de p va a aumentar a medida que aumente el for.
Si pongo c en 4 van a ser mas iteraciones 2 elevado a la 4.
(d) Comunicación entre procesos:
i. Investigue la forma de comunicación entre procesos a través de pipes
Los **pipes** (o tuberías) permiten la comunicación unidireccional entre procesos relacionados, generalmente entre un proceso padre y sus hijos. A través de un pipe, un proceso puede enviar datos a otro escribiendo en un extremo del pipe, mientras el otro proceso lee los datos desde el otro extremo. Los pipes son una forma de comunicación **en memoria** y siguen un esquema **FIFO (First In, First Out)**.
ii. ¿Cómo se crea un pipe en C?.
En C, se crea un pipe usando la función `pipe()`, que pertenece a la biblioteca `unistd.h`. Su prototipo es:

int pipe(int pipefd[2]);

`pipefd[0]` es el descriptor de archivo para **leer** desde el pipe.

`pipefd[1]` es el descriptor de archivo para **escribir** en el pipe.

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

El parámetro necesario es un **array de dos enteros** (`pipefd[2]`), que almacenará los descriptores de archivo del pipe.

Este arreglo es fundamental para que los procesos puedan identificar en qué extremo del pipe leer o escribir datos. Al escribir en `pipefd[1]`, los datos se colocan en el pipe y se pueden leer desde `pipefd[0]`.
iv. ¿Qué tipo de comunicación es posible con pipes?

Los pipes permiten **comunicación unidireccional**. Esto significa que un extremo se utiliza únicamente para escribir y el otro para leer, lo cual permite transferir datos en una sola dirección. Para una comunicación bidireccional (envío y recepción de datos en ambos sentidos), se necesitan dos pipes (uno para cada dirección de comunicación).

Este tipo de comunicación es ideal para que procesos relacionados, como un proceso padre e hijos, puedan intercambiar información sin necesidad de estructuras de datos compartidas complejas.

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

El pid, el pc, el estado del proceso.

La estructura que guarda los datos del SO es la PCB.

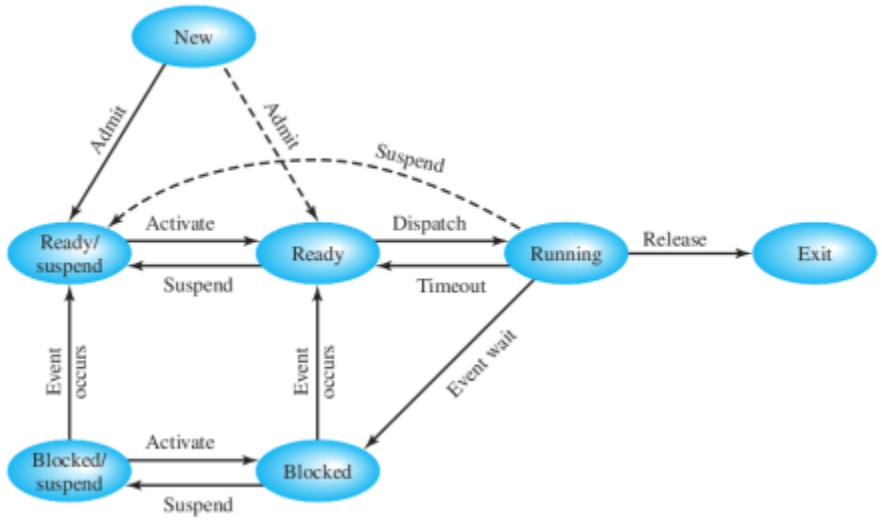
(f) ¿Qué significa que un proceso sea “CPU Bound” y “I/O Bound”?

Cpu bound es que el proceso pasa el mayor tiempo usando la cpu y i o bound es que un gran tiempo está esperando de operaciones de la entrada y la salida.

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

Son new, ready, waiting, running, exit, blocked, suspend.

(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?

De ready a running es el short term scheduler

De new a ready es el long term

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.

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

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

(d) Cite ventajas y desventajas de su uso.

FCFS (First Come First Served)

Los procesos se ejecutan en el orden en que llegan a la cola de procesos listos, sin interrupciones.
-Ejemplo: Dado los procesos P1, P2, y P3, con tiempos de llegada y duración de CPU:
  - P1: 5 ms
  - P2: 3 ms
  - P3: 2 ms
- Orden de ejecución: P1 se ejecuta primero (5 ms), luego P2 (3 ms) y finalmente P3 (2 ms).

  - Es más adecuado para sistemas de procesamiento en **lotes (batch)**, donde el tiempo de llegada es conocido y los tiempos de ejecución son largos.

  - Ventajas: Simplicidad y facilidad de implementación.
  - Desventajas: Puede generar problemas de **espera prolongada (starvation)** para procesos que llegan después de procesos largos, y tiene un bajo tiempo de respuesta.

SJF (Shortest Job First)

Los procesos con menor tiempo de ejecución se priorizan.
- **Ejemplo**: Con los mismos procesos que en el ejemplo anterior:
  - P1: 5 ms
  - P2: 3 ms
  - P3: 2 ms
SJF ejecuta primero P3 (2 ms), luego P2 (3 ms), y finalmente P1 (5 ms).

  - Requiere conocer o estimar el **tiempo de ejecución** de cada proceso.

  - Adecuado para sistemas donde los tiempos de ejecución son conocidos y se buscan tiempos promedio de espera bajos.

  - **Ventajas**: Minimiza el tiempo de espera promedio y mejora la eficiencia.
  - **Desventajas:** Puede generar espera prolongada (starvation) para procesos largos y no es adecuado para sistemas en tiempo real debido a la incertidumbre en la duración de algunos procesos.

Round Robin

Cada proceso se ejecuta durante un intervalo de tiempo fijo (quantum). Si no termina, se coloca al final de la cola.
- **Ejemplo**: Procesos P1, P2, P3, con quantum de 2 ms:
  - P1: 5 ms

  - P2: 3 ms
  - P3: 2 ms
- P1 se ejecuta 2 ms, luego P2 por 2 ms, luego P3 (2 ms, termina), vuelve a P1 (otros 2 ms), y luego a P2 (1 ms, termina).

- **Quantum**: El intervalo de tiempo que cada proceso tiene en la CPU. (parametro)

- Ideal para sistemas de tiempo compartido, ya que da la impresión de multitarea.

- **Ventajas:** Reduce el tiempo de espera y mejora la capacidad de respuesta, asegurando que ningún proceso espere indefinidamente.
 - **Desventajas:** El tiempo de espera puede ser mayor que en SJF, y si el quantum es demasiado alto o bajo, afecta la eficiencia.

Prioridades

Cada proceso tiene una prioridad. Los procesos con mayor prioridad se ejecutan primero.
- **Ejemplo**: Procesos P1, P2, y P3 con las prioridades:
  - P1: Prioridad 3
  - P2: Prioridad 2
  - P3: Prioridad 1 (mayor prioridad)
Primero se ejecuta P3, luego P2, y finalmente P1.

  - La **Prioridad** de cada proceso, que puede ser estática o dinámica.(parametro)

  - Adecuado para **sistemas en tiempo real** donde ciertos procesos necesitan ejecutarse antes que otros.

  - **Ventajas**: Flexibilidad para asignar prioridades según la importancia del proceso.
  - **Desventajas**: Puede provocar starvation en procesos de baja prioridad y necesita un mecanismo para manejar prioridades dinámicas o envejecimiento de procesos.

4. Para el algoritmo Round Robin, existen 2 variantes:
Timer Fijo
Timer Variable
(a) ¿Qué significan estas 2 variantes?
(b) Explique mediante un ejemplo sus diferencias.
(c) En cada variante ¿Dónde debería residir la información del Quantum?

En el algoritmo Round Robin, las variantes **Timer Fijo** y **Timer Variable** se refieren a cómo se gestiona el quantum, que es el tiempo asignado a cada proceso antes de que se interrumpa para dar paso a otro proceso.

(a)

Timer Fijo: En esta variante, todos los procesos tienen un **quantum constante**. Esto significa que el sistema operativo define un valor de quantum que será el mismo para todos los procesos, sin importar sus necesidades o características. Cada proceso recibe el mismo tiempo de CPU antes de ser interrumpido y reinsertado en la cola de espera si aún no ha terminado.

Timer Variable: En esta variante, el **quantum puede variar** para cada proceso. El sistema puede ajustar el quantum según las características del proceso o su comportamiento en el tiempo. Por ejemplo, procesos que necesitan menos tiempo de CPU pueden tener un quantum menor, mientras que procesos con más carga pueden tener un quantum mayor.

(b)

Dados los procesos P1,P2P1, P2P1,P2 y P3P3P3.

- **Timer Fijo**:
  - Quantum fijo de 4 unidades de tiempo para todos.
  - Se ejecuta P1P1P1 durante 4 unidades de tiempo y, si no ha terminado, se pasa a P2P2P2 por 4 unidades, luego a P3P3P3 y se repite.
  - Cada proceso tiene el mismo tiempo de ejecución en cada ciclo de la cola, independientemente de su estado o tipo.
- **Timer Variable**:
  - Quantum variable que depende del proceso.
  - Supongamos que P1P1P1 es intensivo en CPU y se le asignan 6 unidades, P2P2P2 es I/O intensivo y se le asignan 3 unidades, y P3P3P3 es interactivo y se le asignan 2 unidades.
  - En este caso, P1P1P1 se ejecutará más tiempo antes de ser interrumpido, mientras que P3P3P3, que necesita responder rápidamente a la interacción, tiene un quantum más corto.
  - Esto permite un ajuste más personalizado, mejorando la eficiencia y la experiencia del usuario en ciertos casos.

(c)

- **En Timer Fijo**: La información del quantum puede residir de forma centralizada en el sistema operativo, ya que es un valor único para todos los procesos. Este quantum se guarda en una variable de sistema o en una estructura de configuración accesible por el planificador.
- **En Timer Variable**: La información del quantum debe estar almacenada en una estructura individual para cada proceso. Cada proceso tendría un campo o atributo en su estructura de control de procesos (PCB) donde se almacena el valor de su quantum

específico. El planificador accede al quantum de cada proceso al momento de asignarle tiempo de CPU.

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.

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

Para mi el mejor de los algoritmos por si tiempo promedio es el Shortest job first porque tiene un promedio de tiempo de retorno de 22.8 y promedio de tiempo de espera de 13.4.

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.

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

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

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

(d) Qué conclusión queres que saque de ahí??? que se yo que en el de uno va a tardar un monton en ejecutarse??? Perdon ahi respondo

Un Quantum muy pequeño:

- El sistema parece ser más **justo**, ya que cada proceso recibe atención rápidamente.
- Sin embargo, hay un **incremento en el overhead** debido al cambio frecuente de contexto, lo que reduce la eficiencia.

Quantum muy grande:

- Los procesos tienen más tiempo para ejecutarse antes de ser interrumpidos, lo que **reduce el overhead** del cambio de contexto.
- Sin embargo, los procesos interactivos (que requieren respuestas rápidas) pueden experimentar **mayores tiempos de espera**, lo que afecta la experiencia del usuario.

El valor del quantum debe ser un **compromiso**:

- No tan pequeño que incremente el overhead.
- No tan grande que pierda la equidad y responsividad para los procesos interactivos.

(e)

Casos para usar un quantum alto

- **Sistemas con carga de procesos intensiva en CPU**: Si la mayoría de los procesos son cálculos intensivos que requieren largos tiempos de ejecución, un quantum alto permite aprovechar mejor el tiempo de CPU.
- **Sistemas batch o de procesamiento en lote**: En estos sistemas, no es tan crítico responder rápidamente, y la prioridad es completar las tareas lo antes posible.
- **Reducción del overhead**: En sistemas donde los cambios de contexto son costosos.

Ventajas de un quantum alto

1. **Menor overhead**: Reduce el número de cambios de contexto, lo que mejora el rendimiento global del sistema.
2. **Mayor tiempo efectivo de ejecución**: Los procesos tienen más tiempo para completar tareas sin interrupciones.
3. **Más eficiencia para procesos largos**: Procesos intensivos en CPU se benefician al no ser interrumpidos con frecuencia.

Desventajas de un quantum alto

*** interactivos= requieren respuestas rapida***

1. **Aumento del tiempo de respuesta para procesos interactivos**: Procesos que requieren respuestas rápidas (como aplicaciones de usuario o interacciones de red) pueden experimentar demoras, afectando la experiencia del usuario.
2. **Inequidad**: Procesos cortos o interactivos pueden quedar en desventaja frente a procesos largos.
3. **Baja responsividad**: En sistemas multitarea interactivos, los usuarios pueden percibir un sistema lento.

TR = TOTAL DE INSTANTES - CUANDO LLEGA (EN DONDE TERMINA)

TE = TOTAL DE INSTANTES - CUANDO LLEGA +1 - UNIDADES DE CPU

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.

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

La principal desventaja de SRTF es que requiere conocer (o estimar) el tiempo de ejecución restante de los procesos, lo cual puede no ser trivial en algunos sistemas. Además, puede generar **injusticia** hacia procesos largos que son interrumpidos repetidamente. Sin embargo, su capacidad para optimizar tiempo de retorno lo convierte en una opción ideal en ciertos escenarios.

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.

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

- El **algoritmo de planificación por prioridades** es útil cuando **hay una variedad de procesos** que requieren **atención diferenciada** o cuando ciertos procesos deben ejecutarse dentro de **plazos críticos**.
- **Ventajas**: Permite priorizar procesos importantes, aumenta la flexibilidad del sistema y puede garantizar el cumplimiento de los SLA.
- **Circunstancias de uso**: Sistemas en tiempo real, aplicaciones interactivas, y entornos donde la importancia de los procesos varía.
- **No relevante**: En sistemas simples o cuando todos los procesos son igualmente importantes, otros algoritmos como **FIFO** o **Round Robin** pueden ser más eficaces sin necesidad de manejar prioridades.

La implementación de prioridades es una herramienta poderosa, pero debe ser aplicada con criterio, dependiendo del **tipo de sistema** y las **necesidades de los procesos** que maneja.

9. Inanición (Starvation)

(a) ¿Qué significa?

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

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

(a)

La **inanición (starvation)** ocurre en un sistema de planificación cuando un proceso de baja prioridad nunca recibe tiempo de CPU porque otros procesos de mayor prioridad siempre lo preceden. Como resultado, el proceso queda en espera indefinida, sin posibilidad de ser ejecutado.

(b)

Planificación por prioridades (no apropiativa y apropiativa):

- Si se utiliza un sistema rígido donde los procesos con menor prioridad no tienen forma de ser atendidos mientras existan procesos de prioridad más alta, los de baja prioridad pueden experimentar inanición.

Shortest Job First (SJF) y Shortest Remaining Time First (SRTF):

- En sistemas donde llegan constantemente procesos cortos, los procesos más largos pueden quedar postergados indefinidamente, llevando a inanición.

Round Robin con Quantum inadecuado:

- Si el valor del quantum es demasiado pequeño y hay muchos procesos en cola, un proceso puede esperar tanto tiempo para volver a ejecutarse que efectivamente queda en inanición relativa (aunque no tan marcada como en otros algoritmos).

(c)

1. **Envejecimiento (Aging):**
    - Es una técnica que incrementa la prioridad de un proceso a medida que permanece en la cola de espera. De esta manera, los procesos que llevan mucho tiempo esperando eventualmente alcanzan una prioridad suficiente para ser atendidos.

    - Ejemplo: En planificación por prioridades, si un proceso con prioridad baja espera demasiado tiempo, su prioridad puede incrementarse gradualmente hasta ser lo suficientemente alta para ejecutarse.
2. **Cuotas de CPU para procesos largos:**
    - En algoritmos como **SJF o SRTF**, se pueden asignar cuotas mínimas de CPU a procesos largos para garantizar que eventualmente sean ejecutados.
3. **Quantum dinámico en Round Robin:**
    - Ajustar el quantum según la carga del sistema o el número de procesos en cola puede ayudar a evitar largas esperas para procesos específicos.
4. **Implementación híbrida:**
    - Mezclar características de diferentes algoritmos puede prevenir inanición. Por ejemplo, combinar **Round Robin** con **prioridades dinámicas** o **envejecimiento** puede garantizar justicia en la asignación de CPU.

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.

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)

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)

Sobreutilización del Quantum para procesos ligados a I/O:

- Los procesos ligados a I/O suelen requerir cortos tiempos de CPU para gestionar sus operaciones.

- Si un proceso de I/O usa menos CPU que el quantum asignado, la CPU se desperdicia al realizar un cambio de contexto innecesario.
- **Ejemplo:** Un proceso I/O que necesita 1 unidad de CPU pero tiene un quantum de 5 desperdicia 4 unidades.

Procesos ligados a CPU pueden experimentar latencia:

- Como todos los procesos tienen la misma prioridad en Round Robin, los procesos ligados a CPU pueden sufrir fragmentación, ya que su tiempo de ejecución se divide en múltiples ciclos.
- Esto provoca que los procesos intensivos en CPU tarden más en completarse.

Overhead de cambio de contexto:

- Si hay muchos procesos ligados a I/O, estos generan bloqueos frecuentes. Cada desbloqueo puede desencadenar cambios de contexto repetidos, aumentando el overhead y disminuyendo la eficiencia general.

(b)

Desventajas con procesos ligados a CPU e I/O:

1. **Procesos ligados a I/O pueden bloquear procesos largos:**
    - Procesos de I/O suelen tener tiempos de CPU cortos. Esto los hace prioritarios constantemente.
    - Un proceso largo de CPU podría experimentar inanición si hay muchos procesos cortos de I/O entrando al sistema.
2. **Complejidad en el manejo de I/O:**
    - Los procesos de I/O suelen alternar entre estados de ejecución y bloqueo. Esto implica cálculos frecuentes del tiempo restante al desbloquearse, lo que incrementa la **sobrecarga administrativa**.
    - En sistemas con muchas operaciones I/O, los cálculos continuos pueden afectar la eficiencia.
3. **Problemas con procesos intensivos en CPU:**
    - Si hay procesos largos ligados a CPU, serán interrumpidos frecuentemente por procesos de menor tiempo restante, fragmentando su ejecución y aumentando el tiempo total de finalización (**turnaround time**).

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 encuentran 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 faltó 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.

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 VRR con Timer Variable:

- El quantum **siempre llegará a 0** si el proceso sigue ejecutándose, ya que el decremento es constante mientras el proceso utiliza la CPU.
- En casos donde el proceso realice I/O o finalice antes, no es que el quantum no llegue a 0, sino que el proceso ya no necesita más CPU, lo que no invalida la premisa.

El diseño del VRR y el Timer Variable está pensado para evitar inconsistencias como un quantum "infinito".

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 fórmula:

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

Donde:

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

Si = valor estimado para el i-ésimo caso

M[M]

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 qué valores se obtendrían como estimación para las ráfagas de CPU del proceso si se utiliza la fórmula 1, con un valor inicial estimado de S 1=10.

La fórmula anterior 1 le da el mismo peso a todos los casos (siempre calcula la media).

Es posible reescribir la fórmula permitiendo darle un peso mayor a los casos más recientes y menor a casos viejos (o viceversa). Se plantea la siguiente fórmula:

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

Con 0 < α < 1.

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

(c) Para la situación planteada en a) calcule qué valores se obtendrían si se utiliza la fórmula 2 con α = 0, 2; α = 0, 5 y α = 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?

(a)

Rafaga	1	2	3	4	5	6	7
Tiempo	6	4	6	4	13	13	13
Duracion	10	6	5	5.3	4.975	6.58	7.65
Resultado	6	5	5.3	4.975	6.58	7.65	8.41

(b) em

(c)

Rafaga	α	1	2	3	4	5	6	7	8
Duracion	0.2	10	9.2	8.16	7.728	6.982	8.185	9.148	9.9184
	0.5	10	8	6	6	5	9	11	12
	0.8	10	6.8	4.53	5.71	4.34	11.27	12.65	12.93

Tiempo		6	4	6	4	13	13	13	

(d) Yo creo que el b el del 0.8

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 divida 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 utilizaria 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 qué cola elegir un proceso.

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

(a)

Procesos Interactivos:

- Estos procesos requieren **bajos tiempos de respuesta**, ya que suelen estar ligados a la interacción del usuario. Un algoritmo adecuado sería:
  - **Round Robin (RR):**
    - Garantiza un tiempo de respuesta aceptable distribuyendo la CPU de manera equitativa.
    - Usa un **quantum pequeño** para que los procesos interactivos puedan responder rápidamente a las solicitudes del usuario.

Procesos Batch:

- Estos procesos no requieren tiempos de respuesta inmediatos y suelen ser de larga duración, enfocados en la utilización eficiente de la CPU. Un algoritmo adecuado sería:
  - **Shortest Job First (SJF):**
    - Da prioridad a los procesos más cortos para minimizar el tiempo promedio de espera.
    - Si se desea mayor flexibilidad, se puede usar **SRTF (Shortest Remaining Time First)**, que es una versión apropiativa.

(b)

Uso de Prioridades con aging:

Se asigna **mayor prioridad inicial** a la cola de procesos interactivos, ya que requieren respuesta inmediata.

La cola de procesos batch tiene **menor prioridad**, pero a medida que los procesos en esta cola esperan más tiempo, su **prioridad aumenta gradualmente**.

Esto evita que los procesos batch se queden esperando indefinidamente mientras los interactivos se ejecutan continuamente.

- **Evita inanición:** Los procesos batch no quedan "olvidados" en la cola porque su prioridad aumenta con el tiempo.
- **Mantiene la eficiencia del sistema:** Los procesos interactivos siguen teniendo preferencia inmediata, pero los procesos batch también progresan si llevan mucho tiempo esperando.
- **Flexibilidad:** Se puede ajustar el ritmo de "envejecimiento" (aging) para balancear entre dar prioridad a los interactivos y permitir progreso para los batch.

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.

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.

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

- **Colas Definidas:**

  - **Cola 1 (alta prioridad):** Procesos que han consumido poco tiempo de CPU. Ideal para procesos interactivos que realizan tareas cortas.
  - **Cola 2 (media prioridad):** Procesos que han consumido un tiempo moderado de CPU.
  - **Cola 3 (baja prioridad):** Procesos que han consumido mucho tiempo de CPU. Se utiliza para procesos de tipo batch o de larga duración.

Algoritmo en Cada Cola:

- **Cola 1: Round Robin** con un quantum muy pequeño para garantizar respuesta rápida a los procesos interactivos.
- **Cola 2: Round Robin** con un quantum mayor al de la Cola 1, equilibrando equidad y eficiencia.
- **Cola 3: First Come, First Served (FCFS)**, ya que los procesos batch no requieren tanta interactividad.

Administración entre Colas:

- Los procesos se mueven entre colas en función de su **tiempo acumulado de ejecución**:
    - Si un proceso excede cierto umbral de tiempo de CPU en su cola actual, se mueve a una cola de menor prioridad.
    - Si un proceso en una cola de menor prioridad está inactivo o en espera, puede recibir una **realimentación ascendente** (por ejemplo, subir de cola si ha estado esperando demasiado tiempo). Este mecanismo evita la inanición.

Evitar la Inanición:

- Implementar una política de **aging**: los procesos que pasan demasiado tiempo en una cola de baja prioridad suben progresivamente en prioridad para evitar que queden bloqueados indefinidamente.

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

Estos sistemas estaban dirigidos principalmente a entornos interactivos de tiempo compartido. El algoritmo de planificación estaba diseñado para ofrecer buen tiempo de respuesta a usuarios interactivos y asegurar que los trabajos de menor prioridad (en segundo plano) no sufrieran inanición.

La planificación tradicional usaba el concepto de colas multinivel con realimentación, utilizando RR para cada uno de las colas y realizando el cambio de proceso cada un segundo (quantum). La prioridad de cada proceso se calcula en función de la clase de proceso y de su historial de ejecución. Para ello se aplican las siguientes funciones:

$$CPU_j(i) = \frac{CPU_j(i-1)}{2}$$

$$P_j(i) = Base_j + \frac{CPU_j(i)}{2} + nice_j$$

donde:

$CPU_j(i)$ = Media de la utilización de la CPU del proceso j en el intervalo i.

$P_j(i)$ = Prioridad del proceso j al principio del intervalo i (los valores inferiores indican prioridad más alta).

$Base_j$ = Prioridad base del proceso j.

$Nice_j$ = Factor de ajuste.

La prioridad del proceso se calcula cada segundo y se toma una nueva decisión de planificación. El propósito de la prioridad base es dividir los procesos en bandas fijas de prioridad. Los valores de CPU y nice están restringidos para impedir que un proceso salga de la banda que tiene asignada. Las bandas definidas, en orden decreciente de prioridad, son:

1. **Intercambio (Swap)**: Procesos relacionados con la gestión de la memoria.
2. **Control de Dispositivos de I/O por bloques**: Procesos que interactúan con dispositivos de almacenamiento (discos duros, por ejemplo).
3. **Gestión de Archivos**: Procesos que se encargan de la administración de los archivos y sistemas de archivos.
4. **Control de Dispositivos de I/O de caracteres**: Procesos que interactúan con dispositivos de entrada/salida de caracteres (teclados, terminales, etc.).
5. **Procesos de Usuarios**: Procesos generados por los usuarios para realizar tareas de cómputo general.

-

Veamos un ejemplo: Supongamos 3 procesos creados en el mismo instante y con prioridad base 60 y un valor nice de 0. El reloj interrumpe al sistema 60 veces por segundo e incrementa un contador para el proceso en ejecución.

Los sectores en celeste representan el proceso en ejecución.

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

Considero que tendria mas prioridad la actividad de intercambio /swap y el control de i/o por bloque por el **impacto en la eficiencia y estabilidad del sistema**: Si los procesos relacionados con **intercambio de memoria** o **I/O por bloques** no reciben suficiente prioridad, el sistema puede volverse lento o inestable, ya que la memoria no se gestionaría adecuadamente y las operaciones de lectura/escritura a disco podrían ser excesivamente lentas, afectando a otros procesos.

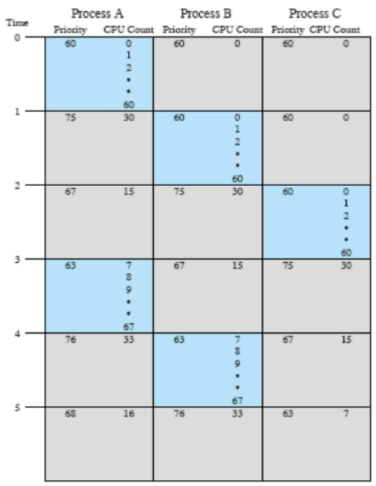
(b) Para el caso de los procesos de usuarios, y analizando las funciones antes descriptas: ¿Qué tipo de procesos se encarga de penalizar? (o equivalentemente se favorecen). Justifique

En este sistema, se penalizan los procesos que utilizan **excesivamente la CPU** o que se comportan de manera **ineficiente** (por ejemplo, procesos que no liberan la CPU cuando deberían). Esto se logra mediante el **historial de ejecución** y el ajuste de las prioridades con base en las funciones que se describieron.

Se favorecen los procesos que **se comportan de manera eficiente**, usando la CPU de manera justa, liberándola cuando no la necesitan y permitiendo que otros procesos también puedan ejecutarse de forma equitativa. Los **procesos interactivos**, que requieren tiempos de respuesta rápidos, también son favorecidos por el sistema.

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

Sí, la utilización de RR dentro de cada cola favorece a un sistema de tiempo compartido porque proporciona **equidad en la distribución de los recursos de CPU**. Garantiza que **todos los procesos tengan oportunidad de ejecutarse**, evita la **inanición** de procesos de baja prioridad y mejora el **tiempo de respuesta** para los procesos interactivos. Sin embargo, es importante ajustar correctamente el tamaño del quantum para equilibrar el rendimiento de procesos largos y cortos, y garantizar que el sistema funcione de manera eficiente sin generar una sobrecarga de cambio de contexto.



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) beneficia a todos los procesos cortos que haya porque siempre se va a ejecutar el que necesite menos tiempo de cpu

(b)Los largos acotados por cpu porque cuanto más tiempo dure la entrada y salida más va a durar y el corto acotado por entrada salñida ya que el algoprtmo va a hacer que dure mas.

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

Si se incrementa sin límite el RR se aproxima a un algoritmo fifo porque el sistema permite que el proceso se ejecute hasta que termine o hasta que no haya otros procesos listos para ejecutarse.

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:

Multiprocesador 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?

Multiprocesador débilmente acoplados y homogéneos

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

La **asignación simétrica de procesos** significa que **todos los procesadores del sistema** tienen **igual acceso y responsabilidades** en la ejecución de procesos, **sin jerarquías** o un procesador principal que tenga un control superior sobre los demás.

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

En un esquema **Maestro/Esclavo**, existe una **relación jerárquica entre los procesadores o unidades** del sistema:

- **El Maestro** es el **procesador principal** o el controlador central que **coordina las operaciones** y toma decisiones importantes sobre la asignación de tareas o recursos.
- **Los Esclavos** son los procesadores secundarios o unidades de trabajo que **siguen las órdenes del Maestro** y realizan tareas específicas bajo su control. Los esclavos no toman decisiones por sí mismos ni gestionan tareas por su cuenta, sino que dependen del Maestro para recibir instrucciones.

23. Asumiendo el caso de procesadores homogéneos:

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

RR

(b) Cite ventajas y desventajas del método escogido

Ventajas:

- Asegura una **distribución equitativa de la CPU** entre todos los procesos.
- Proporciona un **bueno tiempo de respuesta** para procesos interactivos.

Desventajas:

- Si el **quantum es muy pequeño**, se generan muchos cambios de contexto, lo que puede reducir la eficiencia del sistema.
- Si el **quantum es muy grande**, se acerca más a un sistema FIFO.

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

(a) Huella de un proceso en un procesador

Estado que el proceso va dejando en la cache de un procesador, La **huella de un proceso** en un procesador se refiere a la **cantidad de recursos** (memoria, registros, caché, etc.) que un proceso utiliza mientras se ejecuta.

(b) Afinidad con un procesador

La **afinidad de un proceso con un procesador** (también conocida como **"cpu affinity"** o **"processor affinity"**) es la asignación de un proceso a un procesador específico, o un

conjunto de procesadores, de manera que el sistema operativo favorezca la ejecución del proceso en ese procesador en lugar de moverlo entre varios procesadores. Esto puede mejorar el rendimiento al aprovechar mejor las cachés del procesador, ya que el procesador podría retener más datos relacionados con el proceso en su caché local.

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

Caché del procesador: Cuando un proceso se ejecuta varias veces en el mismo procesador, puede aprovechar los **datos almacenados en la caché local del procesador**, reduciendo la latencia y mejorando el rendimiento. Los datos y las instrucciones previamente utilizados pueden ser más rápidamente accesibles si no se necesita moverlos entre procesadores.

Eficiencia del contexto: Mantener un proceso en el mismo procesador evita la sobrecarga de tener que **guardar y restaurar el contexto** del proceso (es decir, los registros, el contador de programa, etc.) cada vez que se mueve entre procesadores. Esto puede ahorrar tiempo y recursos en cambios de contexto.

Afinidad de hardware: Algunos sistemas tienen características de hardware que pueden ser más eficaces cuando los procesos se ejecutan en un núcleo o procesador específico, lo que puede mejorar el rendimiento general del sistema.

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

Windows: Sí, los usuarios pueden cambiar la afinidad de un proceso en Windows. Esto se puede hacer desde el Administrador de tareas de Windows

GNU/Linux: También es posible cambiar la afinidad de un proceso en GNU/Linux, utilizando herramientas como **`taskset`**. El comando `taskset` permite asignar un proceso a uno o más núcleos específicos.

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

El **balanceo de carga** es una técnica utilizada en sistemas multiprocesador o distribuidos para distribuir equitativamente el trabajo entre múltiples unidades de procesamiento, como núcleos de CPU o servidores. El objetivo del balanceo de carga es mejorar la eficiencia y evitar la sobrecarga en una unidad específica, asegurando que todos los recursos disponibles se utilicen de manera óptima.

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

El balanceo de carga puede **contradecir** la afinidad porque para lograr una distribución equitativa de la carga, los procesos pueden ser movidos entre procesadores, lo que **desventaja la afinidad** al eliminar las optimizaciones relacionadas con la reutilización de la caché del procesador.

El equilibrio entre ambos es importante: se pueden usar técnicas como el balanceo de carga **dinámico adaptativo**, que también tenga en cuenta la afinidad de los procesos para optimizar tanto el rendimiento del sistema como el uso eficiente de los recursos.

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 esta en estado ready. De esta manera el sheduler 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 ejecutó.

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

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

La más performante es la que va cambiando el procesador ya que no quedan procesos esperando tanto tiempo.