**Practica 4 ISO**

***1***

***Programa y Proceso.***

Proceso: Programa en ejecución. Los conceptos de tarea, job y proceso hacen referencia a lo

Mismo. Según su historial de ejecución: CPU Bound (ligados a la CPU) e I/O Bound (ligados a entrada/salida). Es dinámico, tiene program counter y su ciclo de vida comprende desde que se lo ejecuta hasta que termina. Se almacena en la ram.

Programa: Es estático, no tiene program cunter y existe desde que se edita hasta que se borra, se almacena en el hdd/ssd.

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

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

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

***Defina Tiempo Promedio de Retorno (TPR) y Tiempo promedio de espera (TPE)***

***para un lote de JOBS.***

TPR = Tiempo Total de Retorno de Todos los Trabajos / Número de Trabajos

TPE = Tiempo Total de Espera de Todos los Trabajos / Número de Trabajos

***¿Qué es el Quantum?***

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

***¿Qué significa que un algoritmo de scheduling sea apropiativo o no apropiativo (Preemptive***

***o Non-Preemptive)?***

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

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

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

***¿Qué tareas realizan?:***

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

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

Short term scheduler: determina que proceso pasara a ejecutarse

***¿Qué tareas realiza el Dispatcher?***

El Dispatcher es una parte fundamental de un sistema operativo y realiza varias tareas esenciales para la gestión eficiente de los procesos y recursos del sistema.

* Selecciona qué proceso se ejecutará a continuación.
* Realiza cambios de contexto para cambiar entre procesos.
* Asigna recursos necesarios a los procesos.
* Gestiona interrupciones y eventos.
* Puede supervisar tareas en segundo plano.

**2 procesos**

top:

***Top*** proporciona una vista en tiempo real de los procesos que se están ejecutando. Muestra información detallada sobre el uso de CPU, memoria y otros recursos por parte de los procesos.

***htop:*** es una alternativa avanzada a top. Proporciona una vista más interactiva y colorida de los procesos en ejecución y su uso de recursos. Permite navegar por la lista de procesos y realizar acciones como matar procesos directamente desde la interfaz.

***ps:*** muestra una lista de procesos en ejecución en el sistema. Puede proporcionar información detallada sobre los procesos, como su ID, estado, uso de CPU y más.

***Pstree:*** muestra la jerarquía de procesos en forma de árbol, lo que facilita la visualización de las relaciones padre-hijo entre los procesos. Ayuda a comprender cómo los procesos están relacionados entre sí en el sistema.

***kill:*** es un comando que se utiliza para enviar señales a procesos en ejecución. La señal más comúnmente utilizada es SIGTERM (15), que indica al proceso que se cierre de manera ordenada. También se puede utilizar SIGKILL (9) para forzar la terminación de un proceso.

***pgrep:*** busca procesos por su nombre o características y devuelve los IDs de proceso que coinciden con el patrón.

***pkill:*** envia señales a procesos que coinciden con un patrón especificado.

***killall:*** envia señales a todos los procesos que coincidan con un nombre de proceso específico. Se debe usar con precaución, ya que puede detener múltiples procesos de una vez.

***renice:*** cambia la prioridad de un proceso en ejecución. Puede aumentar o disminuir la prioridad de CPU de un proceso para darle más o menos recursos de CPU.

***xkill:*** es un comando específico de sistemas con entorno gráfico X Window System. Permite seleccionar una ventana en la interfaz gráfica y forzar la terminación del proceso que la controla.

***atop:*** es una herramienta de monitoreo de recursos similar a top, pero con características adicionales. Proporciona una visión más detallada de la actividad del sistema, incluyendo información sobre CPU, memoria, disco y red, y permite registrar datos para análisis posterior.

***Comunicación entre procesos:***

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

La comunicación entre procesos a través de pipes es una forma de IPC (Inter-Process Communication). Los pipes son especialmente útiles cuando se trata de procesos emparentados, es decir, procesos que se derivan de un proceso padre común. Existen dos tipos de pipes: los pipes anónimos (pipe) y los pipes con nombre (fifo). Los pipes anónimos son utilizados para la comunicación entre procesos relacionados y comparten un espacio de memoria, mientras que los pipes con nombre permiten la comunicación entre procesos no relacionados.

***ii. ¿Cómo se crea un pipe en C?.***

En C, se puede crear un pipe utilizando la función pipe(). Esta función toma un arreglo de dos descriptores de archivo como argumento. El primer descriptor se usa para lectura (conexión de lectura), y el segundo se usa para escritura (conexión de escritura). “int pipe(int filedes[2]);”.

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

***utiliza.***

La función pipe() toma un arreglo de dos descriptores de archivo como parámetro. Este arreglo se utiliza para almacenar los descriptores de archivo de la conexión de lectura y escritura del pipe. La razón de utilizar un arreglo es que permite que tanto el proceso padre como el proceso hijo accedan a estos descriptores después de la llamada a pipe(). Los descriptores de archivo son utilizados para la comunicación bidireccional entre los procesos a través del pipe.

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

Los pipes son principalmente utilizados para la comunicación de datos en flujo unidireccional. Los datos escritos en un extremo del pipe son leídos desde el otro extremo. Esto permite la transferencia de información de un proceso emisor a un proceso receptor.

***¿Cuál es la información mínima que el SO debe tener sobre un proceso? ¿En qué estructura***

***de datos asociada almacena dicha información?***

Esta información se almacena en una estructura de datos llamada "PCB" (Process Control Block) o "Bloque de Control de Proceso". Cada proceso tiene su propio PCB que contiene estos datos.

* ID de proceso (PID): Un identificador único para cada proceso en el sistema.
* Estado del proceso: Indica si el proceso está en ejecución, listo para ejecutarse, en espera, o terminado.
* Prioridad del proceso: Información sobre la prioridad de ejecución del proceso.
* Contexto de procesador: Información sobre los registros del procesador y otros datos necesarios para reanudar la ejecución del proceso.
* Información sobre los recursos asignados al proceso, como descriptores de archivo abiertos, memoria asignada, etc.

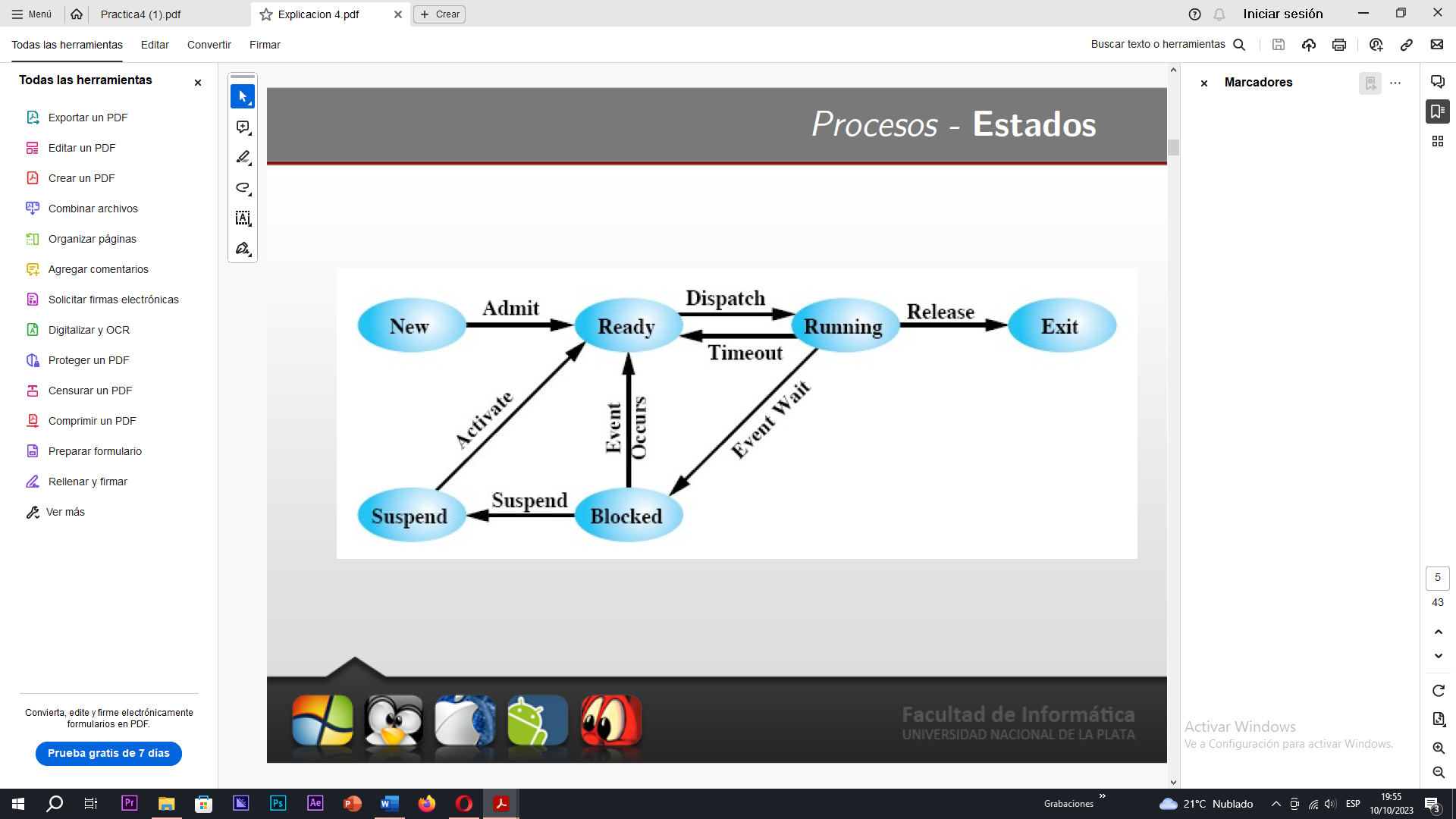
***(f) ¿Qué significa que un proceso sea “CPU Bound” y “I/O Bound”?***

"CPU Bound" significa que un proceso consume principalmente tiempo de CPU y está ocupado realizando cálculos intensivos en la CPU. Un proceso "I/O Bound" está más enfocado en operaciones de entrada/salida y pasa la mayor parte de su tiempo esperando que se completen operaciones de E/S, como lectura/escritura en disco o red.

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

* Nuevo (New): representa el estado de un proceso recién creado.
* Listo (Ready): El proceso está listo para ejecutarse, pero el scheduler aún no lo ha seleccionado para la ejecución.
* En ejecución (Running): El proceso está siendo ejecutado por la CPU en ese momento.
* En espera (Blocked): El proceso está esperando que ocurra algún evento, como una operación de E/S, para continuar su ejecución.
* Listo (Ready): Similar al estado "Listo", pero el proceso ha sido suspendido y no se considera para la ejecución.
* Suspendido (Suspended): Similar al estado "En espera", pero el proceso ha sido suspendido y no se considera para la ejecución.
* Salir (Exit): representa el estado de un proceso que ha completado su ejecución y está a punto de ser eliminado.

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



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

El scheduler encargado de las transiciones entre los estados de un proceso es conocido como "scheduler de planificación" o "scheduler de procesos". El tipo específico de scheduler utilizado puede variar según el sistema operativo, pero su función principal es decidir qué proceso se ejecutará a continuación y gestionar la asignación de recursos de CPU a los procesos en función de su prioridad y otros factores. Los tipos comunes de schedulers incluyen planificación de lotes, planificación en tiempo real, planificación por prioridad, entre otros.

***FCFS (Fisrt Coome First Served)***

Este algoritmo emplea una cola de procesos, asignando un lugar a cada proceso por el orden de llegada. Cuando el proceso llega es puesto en su lugar en la cola después del que llegó antes que él y se pone en estado de listo. Cuando un proceso comienza a ejecutarse no se interrumpe su ejecución hasta que termina de hacerlo.

Ejemplo: Supongamos que tres procesos llegan en el siguiente orden: P1, P2 y P3. FCFS los ejecutará en el mismo orden: P1, luego P2 y finalmente P3.

Ventaja del algoritmo FIFO:

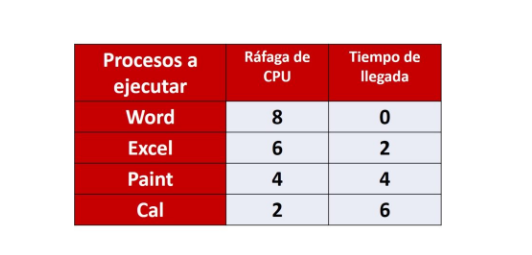
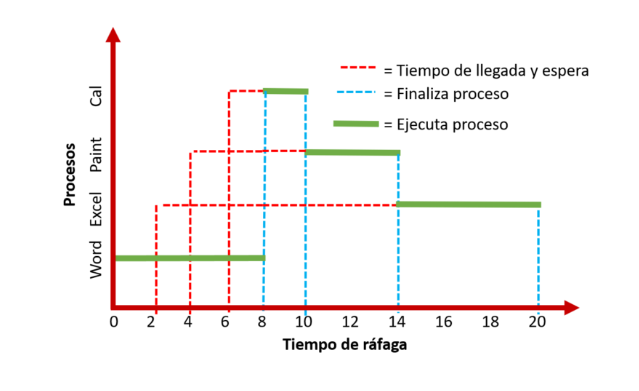
* No hay inversión de prioridades.
* Implementación simple.

Desventajas del algoritmo FIFO:

* Ineficiente en la minimización del tiempo de espera.
* No es adecuado para cargas de trabajo variadas.
* Puede resultar en problemas de inanición.

***SJF (Shortest Job First)***

El algoritmo de planificación de procesos primero coloca al que tiene el trabajo más corto, supone que los tiempos de ejecución se conocen de antemano. Tiene política nonpreemptive. Procesos cortos se colocan delante de procesos largos. Los procesos largos pueden sufrir starvation (inanición). Cuando hay varios trabajos de igual importancia a ser iniciados en la cola de entrada, el planificador selecciona el trabajo más corto primero SJF(Shortest Job First) y en caso de empate prioriza con el algoritmo primero en entrar primero en salir (FIFO, First In First Out).



Ejemplo: Si tenemos tres procesos P1 (5 ms), P2 (2 ms) y P3 (8 ms), SJF ejecutará primero P2 (el más corto), luego P1 y finalmente P3.

Ventajas:

* Minimiza el tiempo de espera promedio.
* Maximiza la utilización de la CPU.
* Eficiente para cargas de trabajo predecibles.

Desventajas:

* Injusto para procesos largos.
* Requiere estimaciones precisas del tiempo de CPU.
* Sensible a la llegada de nuevos procesos.
* Complejidad en sistemas multitarea.

***Round Robin***

A cada proceso se le asigna un tiempo determinado para su ejecución (**Quantum (Q):** medida que determina cuanto tiempo podrá usar el procesador cada proceso), el mismo tiempo para todos. En caso de que un proceso no pueda ser ejecutado completamente en ese tiempo se continuará su ejecución después de que todos los procesos restantes sean ejecutados durante el tiempo establecido Cuando un proceso es expulsado de la CPU es colocado al final de la Ready Queue y se selecciona otro (FIFO circular). Este es un algoritmo basado en FCFS que trata la cola de procesos que se encuentran en estado de listos como una cola circular.

Ejemplo: Si tenemos tres procesos P1, P2 y P3 y se les asigna un quantum de 10 ms cada uno, se ejecutarán en turnos de 10 ms. Por ejemplo, P1 ejecuta 10 ms, luego P2 ejecuta 10 ms, y así sucesivamente.

Ventajas de Round Robin:

* Justo para todos los procesos.
* Adecuado para cargas de trabajo variadas.
* Implementación sencilla.
* Baja latencia de respuesta.

Desventajas de Round Robin:

* Tiempo de espera promedio más largo para procesos largos.
* Ineficiente para procesos cortos.
* No optimiza la utilización de la CPU.
* Dependencia del tamaño del quantum.

Timer Varaible

El “contador” se inicializa en Q (contador := Q) cada vez que un proceso es asignado a la CPU.

* Es el más utilizado
* Utilizado por el simulador

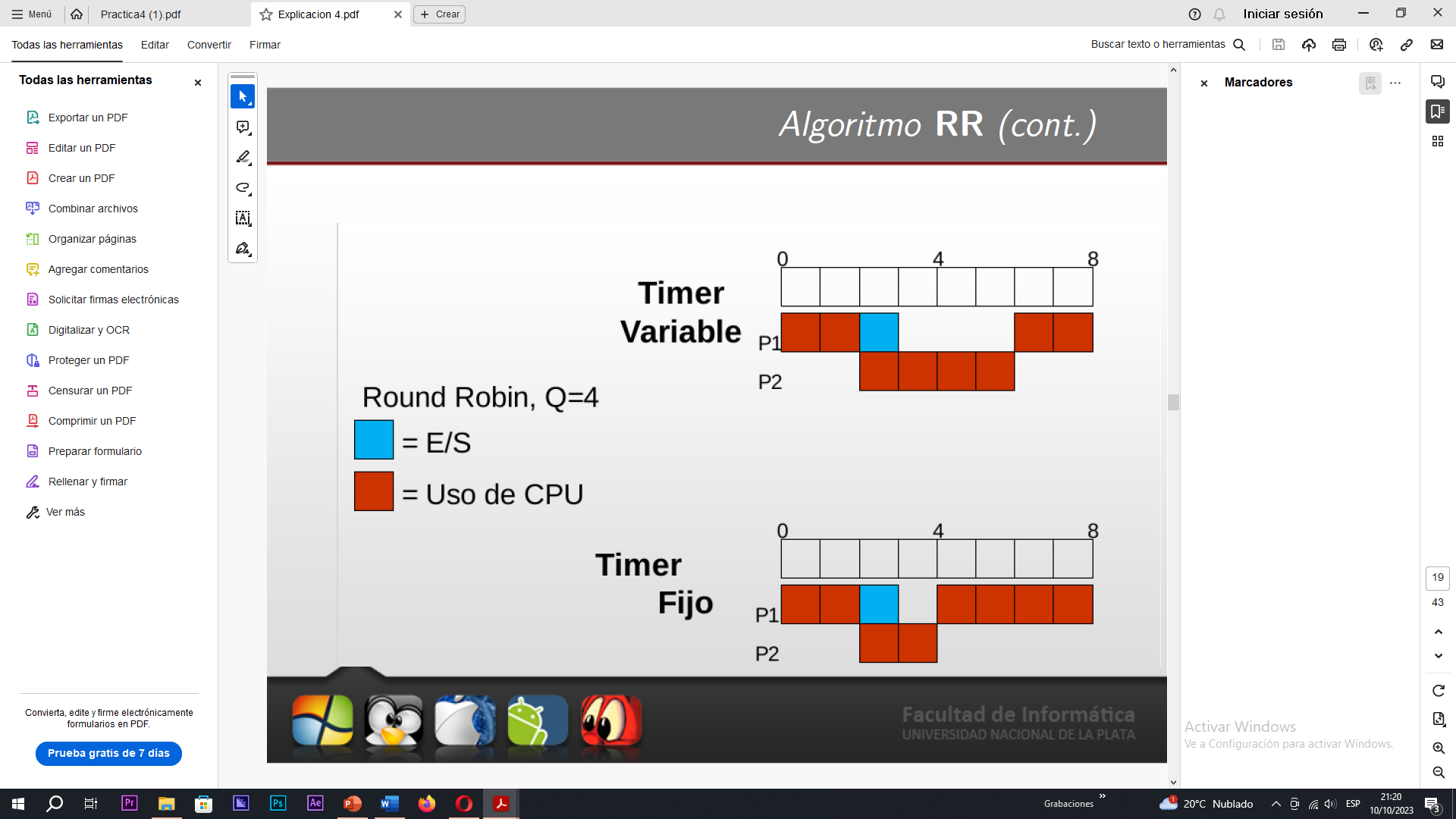
La información del quantum puede residir en la configuración del sistema operativo, pero también puede incluirse en la información de control del proceso. Los algoritmos de planificación pueden adaptar el quantum en función de criterios específicos y cambiarlo dinámicamente durante la ejecución en respuesta a las características de los procesos y las condiciones del sistema.

Timer Fijo

El “contador” se inicializa en Q cuando su valor es cero.

* if (contador == 0) contador = Q;
* Se puede ver como un valor de Q compartido entre los procesos.

La información del quantum debe residir en la configuración del sistema operativo. El valor del quantum es constante y se establece previamente en el sistema operativo antes de ejecutar los procesos.



***Prioridades***

En este tipo de planificación a cada proceso se le asigna una prioridad siguiendo un criterio determinado (menor valor -> mayor prioridad), y de acuerdo con esa prioridad será el orden en que se atienda cada proceso. Se selecciona el proceso de mayor prioridad de los que se encuentran en la Ready Queue, hay una por cada nivel de prioridad

* Procesos de baja prioridad pueden sufrir starvation (inanicion). Solución: permitir a un proceso cambiar su prioridad durantesu ciclo de vida: Aging o Penalty
* Puede ser un algoritmo preemptive o no.

Ejemplo: Si tenemos tres procesos P1 (prioridad alta), P2 (prioridad media) y P3 (prioridad baja), la CPU se asignará primero a P1, luego a P2 y, finalmente, a P3.

Ventaja de la planificación por prioridades:

* Permite asignar recursos según la importancia.

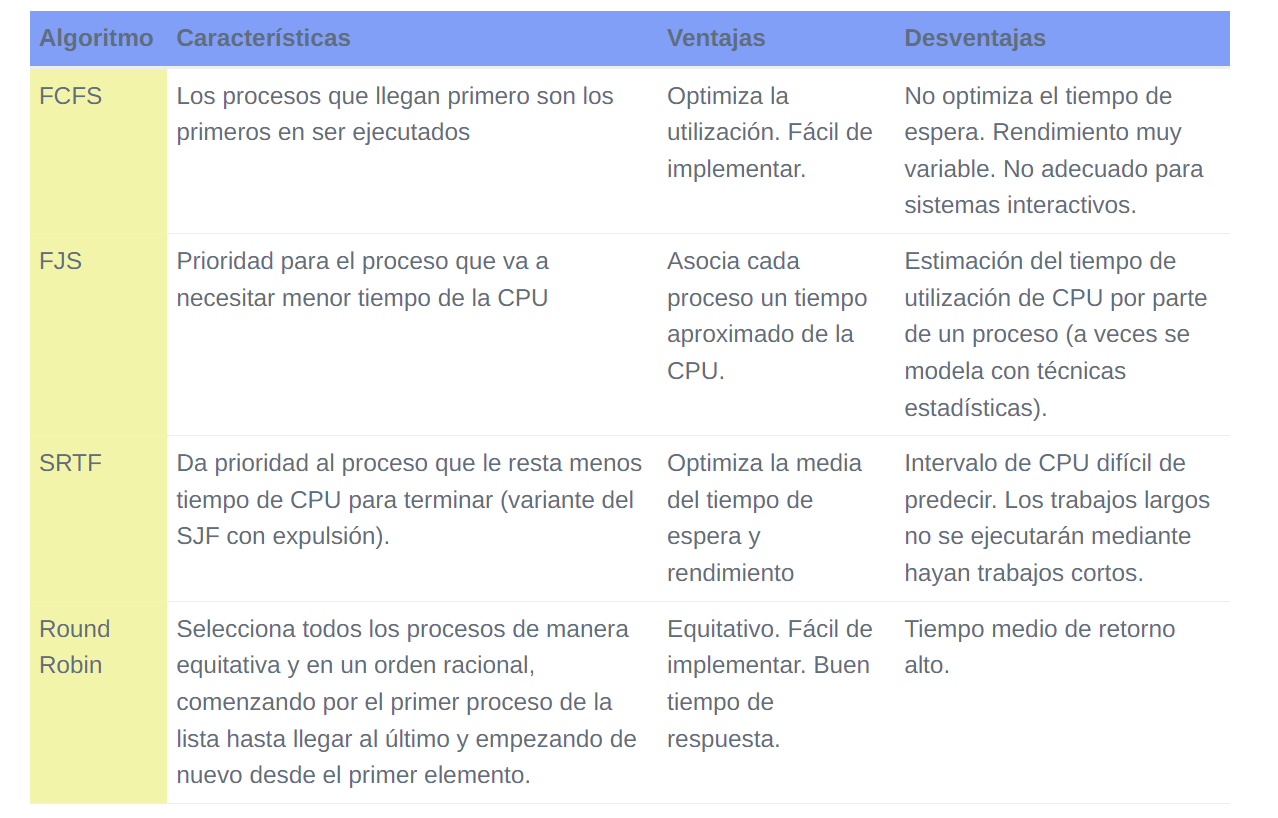
Desventajas de la planificación por prioridades:

* Puede llevar a la inanición de procesos de baja prioridad.
* Requiere una gestión precisa de prioridades.
* Puede resultar en ineficiencia de recursos.
* Se vuelve compleja en sistemas grandes.

***Algoritmos más adecuados según los tipos de procesos y/o SO***

En sistemas donde la mayoría de los procesos son de E/S, puede ser beneficioso utilizar un algoritmo de planificación que permita una conmutación rápida de procesos, como Round Robin, para garantizar una respuesta rápida a las solicitudes de E/S.

Por otro lado, si se trata de un sistema en el que la computación intensiva en CPU es predominante y los procesos de E/S son menos frecuentes, un algoritmo como SJF o la planificación por prioridades podría ser más adecuado para optimizar el rendimiento general del sistema.



***Parámetros requeridos:***

* FCFS no requiere parámetros específicos; funciona en el orden de llegada.
* SJF requiere conocer o estimar la duración de cada proceso.
* Round Robin requiere definir el tamaño del quantum, que es el tiempo asignado a cada proceso.
* La planificación por prioridades necesita que se asigne una prioridad a cada proceso.

***Inanición (Starvation)***

La inanición en este contexto implica que un proceso o entidad no puede acceder a los recursos o la atención necesarios para avanzar, lo que resulta en una degradación o detención de su funcionamiento.

***¿Cuál/es de los algoritmos vistos puede provocarla?***

Algoritmos que pueden provocar inanición: Varios algoritmos y situaciones pueden causar la inanición en sistemas informáticos. Algunos ejemplos incluyen:

* Algoritmo de Planificación de Procesos: En sistemas operativos, ciertos algoritmos de planificación de procesos pueden dar prioridad a procesos específicos, lo que podría hacer que otros procesos tengan dificultades para obtener tiempo de CPU. Esto puede resultar en la inanición de procesos de menor prioridad.
* Algoritmo de Planificación de Colas: En sistemas de colas (como en la gestión de trabajos en una impresora), si no se implementa un mecanismo de justicia para garantizar que todos los trabajos tengan la oportunidad de avanzar, algunos trabajos pueden quedar atrapados en una cola y no ser atendidos.

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

* Envejecimiento (Aging): El envejecimiento es una técnica que aumenta gradualmente la prioridad de un proceso a medida que pasa más tiempo en estado de espera. Esto significa que, incluso si un proceso comienza con una baja prioridad, con el tiempo su prioridad aumenta si no se le asigna tiempo de CPU. Como resultado, los procesos que han estado esperando durante mucho tiempo eventualmente obtendrán una prioridad más alta y tendrán una oportunidad de competir con procesos de mayor prioridad por los recursos.
* Penalización por Tiempo de Espera (Penalty): Otra solución es aplicar una penalización o puntuación negativa a los procesos que han estado esperando durante un tiempo prolongado. Esto significa que, a medida que un proceso espera, su puntuación disminuye, lo que disminuye su prioridad relativa. Cuando la puntuación de un proceso alcanza un cierto umbral, se le otorga una prioridad más alta y tiene más probabilidades de ser seleccionado para la ejecución.

***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 algoritmo de Round Robin Variable (VRR) con Timer Variable, es posible que el quantum de un proceso nunca llegue a 0. Esto se debe a la naturaleza variable del quantum en este esquema y a cómo se maneja el contador del quantum.

En un algoritmo de Round Robin convencional, cada proceso se ejecuta durante un tiempo fijo conocido como el quantum. Una vez que el quantum se agota, se produce un cambio de contexto y otro proceso obtiene la CPU. Sin embargo, en el VRR con Timer Variable, el quantum se representa mediante un contador que se decrementa en 1 unidad en cada interrupción de reloj. El valor de inicio de este contador puede ser variable, y el sistema operativo tiene la flexibilidad de asignar diferentes valores de inicio de contador a diferentes procesos, dependiendo de varios factores.

Si el valor de inicio del contador para un proceso es lo suficientemente grande y las interrupciones de reloj son relativamente poco frecuentes, es posible que el proceso nunca llegue a agotar su quantum. Esto significa que el proceso puede continuar ejecutándose sin interrupción de cambio de contexto durante un tiempo prolongado.

La ventaja de este enfoque es que los procesos que requieren más tiempo de CPU pueden tener un quantum inicial más grande, lo que reduce la sobrecarga de cambio de contexto en comparación con un Round Robin convencional. Sin embargo, también puede llevar a problemas de equidad si un proceso con un contador de quantum grande monopoliza la CPU durante mucho tiempo, lo que podría afectar negativamente a otros procesos en la cola de planificación.

En resumen, en un algoritmo de VRR con Timer Variable, es posible que el quantum de un proceso nunca llegue a 0 debido a la flexibilidad en la asignación de valores iniciales de contador y la naturaleza variable de la ejecución de los procesos.

***Colas Multinivel***

***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?***

Para administrar colas multinivel que contienen dos tipos de procesos, Interactivos y Batch, se pueden utilizar los siguientes algoritmos de planificación para cada una de las colas:

* Cola de Procesos Interactivos: Un algoritmo apropiado para procesos interactivos es el Round Robin (RR). Este algoritmo proporciona una respuesta rápida a las solicitudes de entrada/salida y asegura que ningún proceso monopolice la CPU durante mucho tiempo. Los procesos interactivos suelen requerir una respuesta rápida a eventos del usuario, por lo que RR es adecuado.
* Cola de Procesos Batch: Para procesos batch, que generalmente son procesos de larga duración que no requieren una respuesta inmediata a eventos de entrada/salida, un algoritmo adecuado es Priority Scheduling. Puede asignarse una prioridad baja a los procesos batch para que no compitan de manera desigual con los procesos interactivos en términos de tiempo de CPU. Esto permite que los procesos interactivos se ejecuten con prioridad y los procesos batch se ejecuten cuando la CPU esté menos ocupada.

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

Para planificar las dos colas vistas en, es necesario determinar cuándo y cómo se eligen procesos de cada cola. Para hacer esto, se puede usar un algoritmo de planificación multinivel, donde se establece un algoritmo para determinar cuál de las dos colas debe tener prioridad en cada ciclo de planificación. Un enfoque común es utilizar una política de asignación de prioridades dinámicas basada en la carga del sistema o la prioridad de los procesos. Aquí hay una sugerencia de cómo podría funcionar:

* Se pueden asignar prioridades más altas a las colas de procesos interactivos en condiciones normales. Esto asegura que los procesos interactivos tengan prioridad.
* Si la cola de procesos batch se llena o si la carga del sistema es alta (muchos procesos en cola), se podría cambiar temporalmente la prioridad para dar a la cola de procesos batch una oportunidad de ejecución.
* Un algoritmo como Feedback Queue Scheduling, que se basa en la cantidad de tiempo que un proceso ha estado en cola, podría ser útil. Por ejemplo, se puede mover un proceso de la cola de procesos interactivos a la cola de procesos batch si pasa demasiado tiempo en la cola de procesos interactivos sin ser atendido.

En resumen, el enfoque de planificación de colas multinivel permite adaptar la asignación de recursos a la naturaleza de los procesos y las condiciones del sistema, asegurando una buena respuesta a los procesos interactivos y permitiendo la ejecución de procesos batch cuando la carga del sistema lo permita. El algoritmo exacto a utilizar dependerá de los requisitos y características específicas del sistema operativo y las aplicaciones que se ejecuten.

***18***

Para implementar un algoritmo de planificación que tome en cuenta el tiempo de ejecución consumido por los procesos y penalice a los que más tiempo de ejecución tienen en un sistema de Colas Multinivel con Realimentación, puedes utilizar el siguiente enfoque:

1. Colas Multinivel: Divide la cola en varios niveles, cada uno representando un rango de tiempo de ejecución. Por ejemplo, podrías tener tres colas, donde cada una se encargue de procesos con diferentes niveles de tiempo de ejecución:
   * Cola de nivel alto (Procesos con poco tiempo de ejecución).
   * Cola de nivel medio (Procesos con tiempo de ejecución moderado).
   * Cola de nivel bajo (Procesos con tiempo de ejecución alto).
2. Algoritmo para cada cola:
   * Cola de nivel alto: Utiliza un algoritmo de planificación de tipo Round Robin para garantizar que los procesos en esta cola obtengan una cantidad justa de tiempo de CPU sin importar su tiempo de ejecución.
   * Cola de nivel medio: Implementa un algoritmo de planificación de tipo SJF (Shortest Job First). Este algoritmo permite que los procesos con el tiempo de ejecución más corto se ejecuten primero.
   * Cola de nivel bajo: Utiliza un algoritmo de planificación basado en prioridad, donde la prioridad se calcula en función del tiempo de ejecución restante. Cuanto mayor sea el tiempo de ejecución restante, menor será la prioridad. Esto penaliza a los procesos con un alto tiempo de ejecución.
3. Administración entre colas: Para evitar la inanición, debes implementar una estrategia de realimentación. Cuando un proceso consume su cuota de tiempo en una cola específica, se mueve a la cola de nivel inmediatamente superior. Esto permite que los procesos con menor tiempo de ejecución tengan prioridad para ejecutarse en las colas superiores, mientras que los procesos con más tiempo de ejecución se mueven hacia colas inferiores. Además, si un proceso en una cola de nivel superior no se completa, debería descender de nuevo a su cola original después de un tiempo específico para evitar la inanición.

Este enfoque garantiza que los procesos con un tiempo de ejecución más corto se ejecuten antes que los procesos con un tiempo de ejecución más largo, pero aún permite que los procesos más largos tengan acceso a la CPU para evitar la inanición. La realimentación entre las colas es esencial para mantener un equilibrio entre la equidad y la eficiencia.

***19***

a) En el sistema Unix Clásico con colas multinivel y realimentación, las actividades de usuarios interactivos tendrían más prioridad. Esto se debe a la jerarquía de bandas de prioridad que se describe en el sistema, donde las actividades de intercambio y el control de dispositivos de I/O por bloques tienen prioridades más bajas que los procesos de usuario. El scheduler prioriza estas actividades para garantizar un tiempo de respuesta rápido a los usuarios interactivos, lo que significa que el sistema se enfoca en brindar una experiencia interactiva fluida y sensible.

b) En el caso de los procesos de usuarios, las funciones de prioridad se calculan utilizando la fórmula Pj(i) = Basej + CPUj(i) / 2 + nicej. Dado que el valor nice (factor de ajuste) se establece en 0, el factor más importante para determinar la prioridad es la media de utilización de la CPU del proceso. Esto significa que los procesos que han consumido menos tiempo de CPU tendrán prioridades más altas, y se favorecerán en términos de acceso a la CPU. Los procesos que no han consumido tanto tiempo de CPU se favorecen en detrimento de los procesos que han consumido más tiempo, lo que ayuda a evitar que los procesos largos de fondo sufran inanición y que los procesos de usuario obtengan una respuesta rápida.

c) La utilización de Round Robin (RR) dentro de cada cola es beneficiosa en un sistema de tiempo compartido como el Unix Clásico. El RR garantiza que todos los procesos en una cola tengan acceso equitativo a la CPU durante un intervalo de tiempo fijo (quantum), lo que es importante para mantener una experiencia interactiva justa y equitativa para los usuarios. Esto significa que, independientemente del tiempo de ejecución de un proceso, todos los procesos en una cola tendrán la oportunidad de ejecutarse en orden, lo que ayuda a prevenir la inanición y garantiza que los procesos interactivos no sean bloqueados por tareas largas en segundo plano. En resumen, RR favorece el sistema de tiempo compartido al proporcionar un equilibrio entre la equidad y la eficiencia en la ejecución de procesos.

***Explicar porqué si el quantum "q.en Round-Robin se incrementa sin límite, el método se***

***aproxima a FIFO.***

Esto se debe a la forma en que funciona el algoritmo Round-Robin y cómo la duración del quantum afecta su comportamiento. En el algoritmo Round-Robin, cada proceso se ejecuta durante un quantum de tiempo fijo y luego se coloca nuevamente al final de la cola de procesos listos. Esto garantiza que todos los procesos tengan la oportunidad de ejecutarse de manera equitativa y que ningún proceso monopolice la CPU durante mucho tiempo.

Si el quantum se incrementa sin límite, significa que cada proceso tiene la oportunidad de ejecutarse durante todo el tiempo que desee o necesite antes de ceder la CPU al siguiente proceso en la cola. En esencia, esto significa que ya no hay un límite de tiempo fijo para cada proceso, y los procesos se ejecutarán hasta su finalización antes de pasar al siguiente proceso.

En esta configuración, el algoritmo se comporta de manera muy similar a la política FIFO, ya que los procesos se ejecutan en el orden en que llegaron a la cola de procesos listos. El primer proceso en llegar es el primero en ejecutarse y se ejecuta hasta que se completa antes de pasar al siguiente proceso. Esta es la característica principal de la política FIFO: los procesos se ejecutan en el orden en que llegaron, sin importar sus necesidades de tiempo de CPU.

En resumen, cuando el quantum en el algoritmo Round-Robin se incrementa sin límite, el algoritmo se convierte en una aproximación a la política FIFO, ya que los procesos se ejecutan en el orden en que llegaron y se les permite completar su ejecución antes de pasar al siguiente proceso, lo que elimina la característica de tiempo fijo del quantum de Round-Robin.

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 S.O .

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

Las PCs de escritorio habituales generalmente se asocian con la clasificación de "Homogéneos" y "Multiprocesador fuertemente acoplado".

* Homogéneos: En las PCs de escritorio habituales, los procesadores suelen ser similares en arquitectura y rendimiento, y no hay ninguna ventaja física significativa de uno sobre el otro. Todos los núcleos de CPU en una PC de escritorio típica funcionan de manera similar.
* Multiprocesador fuertemente acoplado: En una PC de escritorio típica, los procesadores (núcleos de CPU) comparten una memoria principal y están bajo el control del sistema operativo. Esto se debe a que las PCs de escritorio están diseñadas para un uso general y no están especializadas para tareas específicas. Los núcleos de CPU en una PC de escritorio pueden ejecutar una variedad de aplicaciones y tareas, y todos comparten un acceso a la memoria principal, lo que permite una mayor coordinación y comunicación entre los procesadores.

***¿Qué significa que la asignación de procesos se realice de manera simétrica?***

Asignación de procesos de manera simétrica: Significa que en un sistema multiprocesador, todos los procesadores tienen la misma capacidad y recursos, y se utilizan para ejecutar tareas de manera equitativa. La asignación de procesos simétrica garantiza que no haya una sobrecarga desigual en ningún procesador y que todos los procesadores tengan una carga de trabajo similar.

***¿Qué significa que se trabaje bajo un esquema Maestro/esclavo?***

Trabajar bajo un esquema Maestro/Esclavo: Esto se refiere a un modelo de procesamiento en el que un procesador principal (el maestro) coordina y controla las actividades de uno o más procesadores secundarios (los esclavos). El procesador maestro asigna tareas y distribuye cargas de trabajo a los procesadores esclavos y coordina su funcionamiento. Este enfoque es común en sistemas con múltiples núcleos, donde un núcleo o procesador principal administra y supervisa las tareas de otros núcleos o procesadores, asegurando una ejecución eficiente y coordinada.

***Asumiendo el caso de procesadores homogéneos***

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

Si estamos tratando con procesadores homogéneos en un sistema multiprocesador, el método de planificación más sencillo para asignar CPUs a los procesos sería el método de "Round Robin". Este método es ampliamente utilizado y fácil de implementar en sistemas multiprocesador. El método Round Robin asigna un quantum de tiempo fijo a cada proceso en un ciclo circular, asegurando una distribución justa del tiempo de CPU entre todos los procesos.

***Cite ventajas y desventajas del método escogido***

Ventajas del método Round Robin:

* Simplicidad: El algoritmo Round Robin es fácil de entender y de implementar, lo que reduce la complejidad de la planificación de procesos en sistemas multiprocesador.
* Equidad: Garantiza que todos los procesos tengan acceso equitativo a la CPU, lo que es importante para evitar la inanición y asegurar que ningún proceso monopolice la CPU.
* Tiempo de respuesta predecible: Debido al uso de un quantum de tiempo fijo, los usuarios pueden prever cuánto tiempo deberá esperar su proceso para ejecutarse nuevamente.

Desventajas del método Round Robin:

* Ineficiencia en procesos de larga duración: Si algunos procesos son mucho más largos que otros, pueden generar sobrecarga de cambio de contexto debido a la alternancia frecuente entre los procesos en la cola.
* No se adapta automáticamente a las prioridades: El Round Robin no tiene en cuenta las prioridades de los procesos, lo que puede ser problemático en situaciones donde algunos procesos son más críticos que otros.
* Desperdicio de tiempo de CPU: Si un proceso completa su trabajo antes de que expire su quantum, el tiempo restante del quantum se desperdicia, ya que el proceso debe esperar su turno nuevamente en lugar de permitir que otro proceso lo utilice.

Si bien el método Round Robin es sencillo y justo, puede no ser la elección óptima en todos los casos, especialmente cuando se requiere una planificación más sofisticada para priorizar ciertos tipos de procesos o evitar problemas de ineficiencia. En tales casos, se pueden considerar otros algoritmos de planificación más avanzados.

***Huella de un proceso en un procesador***

La huella de un proceso se refiere a la cantidad de recursos de CPU y memoria que un proceso utiliza en un procesador. Una huella más grande indica que el proceso consume más recursos de CPU y memoria.

***Afinidad con un procesador***

La afinidad con un procesador se refiere a la capacidad de un sistema operativo para asignar un proceso específico a un núcleo o procesador en un sistema multiprocesador. Establecer la afinidad significa que el proceso se ejecutará en un núcleo particular y no se migrará entre núcleos.

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

Al ejecutar un proceso en el mismo procesador (manteniendo la afinidad), se pueden obtener beneficios de rendimiento, ya que se evita el costo de cambio de contexto y se reduce la latencia de acceso a la memoria local, lo que puede ser especialmente importante en aplicaciones de alto rendimiento o en sistemas en tiempo real***.***

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

En Windows: Los usuarios pueden cambiar la afinidad de un proceso utilizando el Administrador de tareas. Seleccionan el proceso, hacen clic con el botón derecho, van a "Ir a detalles" y luego pueden establecer la afinidad en la pestaña "Detalles".

En GNU/Linux: Los usuarios pueden utilizar comandos como taskset para establecer la afinidad de un proceso en un núcleo específico.

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

El balanceo de carga se refiere a la distribución equitativa de la carga de trabajo entre múltiples recursos de procesamiento (como núcleos de CPU o servidores) para maximizar la utilización de los recursos y evitar la congestión o el desequilibrio de carga.

***Compare los conceptos de afinidad y balanceo de carga y como uno afecta al otro.***

La afinidad y el balanceo de carga pueden entrar en conflicto en algunos casos, ya que la asignación de afinidad puede obstaculizar la capacidad del sistema para equilibrar la carga de manera efectiva. Sin embargo, en sistemas bien diseñados, se pueden encontrar soluciones que permitan una combinación adecuada de afinidad y balanceo de carga para optimizar el rendimiento general del sistema.