Practica 4 PN

1)

a)

Programa

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

Proceso

- Es dinámico
- Tiene program counter
- Su ciclo de vida comprende desde que se lo ejecuta hasta que

termina

b)

Retorno: tiempo que transcurre entre que el proceso llega al

sistema hasta que completa su ejecución

Espera: tiempo que el proceso se encuentra en el sistema

esperando, es decir el tiempo que pasa sin ejecutarse (TR -

Tcpu)

1. c) Tiempo promedio retorno: La suma de los tiempos de retorno de los procesos dividido la cantidad de procesos

Tiempo promedio de espera: Suma de tiempo de espera de todos los procesos dividido la cantidad de procesos

Dependiendo del algoritmo empleado dicho tiempo variará

d)

Es una medida que determina cuanto tiempo podría usar el procesador cada proceso

1. e) Nonpreemptive: una vez que un proceso esta 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

f)

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

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 pasará a ejecutarse

g)

Dispatcher: hace cambio de contexto, cambio de modo de

ejecución...”despacha” el proceso elegido por el Short Term (es decir, salta

a la instrucción a ejecutar)

2)

a)

top: Despliega los procesos de Linux corriendo actualmente con ciertos detalles: usuario que lo está ejecutando, PID, uso de CPU, etc.

c)

ps: Despliega la información de procesos activos

d)

pstree: Despliega los procesos del sistema en forma de diagrama de árbol

1. e) kill: Le envía una señal a un proceso (Señal por defecto es la de matar al proceso
2. f) killall: Le envía una señal a todos los procesos que estén siendo ejecutados por un programa
3. g) renice: Altera la prioridad de procesos en ejecución
4. I) Aparece 4 veces la palabra proceso en el primer programa
5. II) Si

3)

A)

FCFS: Cuando hay que elegir un proceso para ejecutar, se selecciona el mas viejo

No favorece a ningún tipo de procesos, pero en principio prodríamos decir que los CPU Bound terminan al comenzar su primer ráfaga, mientras que los I/O Bound no.

Algoritmo FIFO (cont.)

#Ejemplo 1

TAREA ''1'' INICIO=0

[CPU,9]

TAREA ''2'' INICIO=1

[CPU,5]

TAREA ''3'' INICIO=2

[CPU,3]

TAREA ''4'' INICIO=3

[CPU,7]

Se ejecutaría: primero el 1, luego el 2, el 3 y el 4. Cada uno ejecutándose por completo ya que no es un algoritmo que favorezca a un proceso por sobre otro

SJF:

Política no apropiativa que selecciona el proceso con la ráfaga más corta, este se calcula basado en la ejeución previa. Los procesos se colocan delante de los largos, los cuales pueden sufrir inanición

Algoritmo SJF(cont.)

#Ejemplo 1

TAREA ''1'' INICIO=0

[CPU,9]

TAREA ''2'' INICIO=1

[CPU,5]

TAREA ''3'' INICIO=2

[CPU,3]

TAREA ''4'' INICIO=3

[CPU,7]

Se ejecutaría: Primero el 1, luego el 3, luego el 5 y por último el 4. Al no ser apropiativo y llegar primero el 1, se debe terminar de ejecutar para ejecutar el más corto de la cola de listos.

B)

Política nonpreemptive que selecciona el proceso con la ráfaga más corto

- Calculo basado en la ejecución previa
- Procesos cortos se colocan delante de procesos largos
- Los procesos largos pueden sufrir starvation (inanición)

#Ejemplo 1

TAREA ''1'' PRIORIDAD=3 INICIO=0

[CPU,9]

TAREA ''2'' PRIORIDAD=2 INICIO=1

[CPU,5]

TAREA ''3'' PRIORIDAD=1 INICIO=2

[CPU,3]

TAREA ''4'' PRIORIDAD=2 INICIO=3

[CPU,7]

Siguiendo este ejemplo pero con SJF: Inicia el proceso 1, que es el primero en llegar, al terminar su ejecución se ejecuta el proceso 3, luego el 2 y por último el 4

Round Robin:

Politica basada en un reloj

- Quantum (Q): medida que determina cuanto tiempo

podría usar el procesador cada proceso:

- Pequeñoo: overhead de context switch
- Grande: ¿pensar?

- Cuando un proceso es expulsado de la CPU es colocado al

final de la Ready Queue y se selecciona otro (FIFO circular)

Existe un “contador” que indica las unidades de CPU en las

que el proceso se ejecuto. Cuando el mismo llega a 0 el

proceso es expulsado

- El “contador” puede ser:
- Global
- Local → PCB
- Existen dos variantes con respecto al valor inicial del

“contador” cuando un proceso es asignado a la CPU:

- Timer Variable
- Timer Fijo

Algoritmo de uso de prioridades:

Cada proceso tiene un valor que representa su prioridad → menor valor, mayor prioridad

- Se selecciona el proceso de mayor prioridad de los que se

encuentran en la Ready Queue

- Existe una Ready Queue por cada nivel de prioridad
- Procesos de baja prioridad pueden sufrir starvation (inanici´on)
- Solución: permitir a un proceso cambiar su prioridad durante

su ciclo de vida → Aging o Penalty

- Puede ser un algoritmo preemptive o no

1. c) Los más adecuados son aquellos que son apropiativos y no generan inanición

4)

a)

RR timer variable:

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

RR 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

1. C) El variable sería Global a todos los procesos

En el fijo sería local al proceso en su PCB

9)

A)Inanición: Se refiere a una situación en la que un proceso nunca llega acceder a los recursos que necesita para continuar su ejecución. Esto ocurre cuando uno o más procesos reciben continuamente prioridad sobre otros, impidiendo que los de baja prioridad accedan a recursos compartidos

B)

Algoritmos que pueden provocar inanición:

- SJF
- Algoritmos de uso de prioridades
- SRTF

11)

Round robin:

- Fragmentación excesiva del tiempo de CPU:
    - Al aplicar Q fijo para todos los procesos, aquellos ligados a E/S, típicamente necesitan menos CPU, pueden ser programados innecesariamente, desperdiciando tiempo de CPU
    - Esto puede aumentar el overhead de cambio de contexto, especialmente si el quantum es muy pequeño
- Desempeño desigual:

    - Los procesos ligados a CPU necesitan grandes cantidades de tiempo de ejecución continuo. Con quantum pequeño, estos se verán frecuentemente interrumpidos, incrementando su tiempo total de finalización
    - Los procesos ligados a E/s podrían experimentar tiempos de espera adicionales en la cola mientras esperan su turno
- Procesos en espera luego de E/S:
    - Aquellos procesos que “Llamen” a E/S, que ya estaban ejecutandose en CPU, luego de hacer la E/s deberán esperar a que todos los procesos que estén por delante se ejecuten, y quizá no llegue a terminar su tarea hasta ese momento.

Shortest Remaining Time First:

- Favorece los procesos cortos:
    - Los proceso ligados a E/S, que tienden a ser cortos, pueden ser programados frecuentemente. Esto puede resultar inanición para los procesos largos, retrasando su finalización
- Alto overhead por interrupciones frecuentes:
    - Como el algoritmo es apropiativo, verifica constantemente si un nuevo proceso con menor tiempo restante ha llegado

13)

Sí, es posible que el quantum de un proceso nunca llegue a 0 en un sistema con VRR y timer variable, debido a interrupciones por operaciones de E/s, proridades más altas, o redistribuciones del quantum.

15)

Interactivo:

Son necesarios algoritmos apropiativos para evitar que un proceso acapare la CPU

Ejem:

RR

Prioridades

Colas multinivel

SRTF

Batch:

Se utilizan algoritmos no apropiativos, ya que no requiere interactividad con el usuario.

Ejem:

FCFS

SJF

19)

(a) **¿Qué tipo de actividad considera que tendrá más prioridad? ¿Por qué piensa que el scheduler prioriza estas actividades?**

La actividad con mayor prioridad es **Intercambio**. Esto se debe a que es la banda de mayor prioridad en el sistema, según el esquema de planificación jerárquica que se describe. En un sistema operativo, la **gestión de la memoria y el intercambio de procesos entre la memoria principal y el almacenamiento secundario (swap)** son esenciales para garantizar que los recursos sean utilizados de manera eficiente, especialmente cuando la memoria está bajo presión. Al dar prioridad a las actividades de intercambio, el sistema asegura que los procesos más críticos, como aquellos que requieren acceso rápido a la memoria o que están ejecutándose en primer plano, no se vean interrumpidos por procesos menos importantes.

(b) **¿Qué tipo de procesos se encarga de penalizar? (o equivalen a los que se favorecen). Justifique.**

Los procesos **de usuarios** son los que se favorecen o penalizan según su **historial de utilización de la CPU** y el valor de su **factor nice**. Los procesos con un historial de uso intensivo de la CPU serán penalizados, ya que su **prioridad se ajusta a la baja** según la ecuación de prioridad Pj(i), lo que significa que su prioridad disminuirá y estarán en una cola con menor prioridad, siendo interrumpidos más frecuentemente. En cambio, aquellos procesos que han tenido una menor utilización de la CPU o que tienen un valor nice mayor (que indica una preferencia por menor utilización de recursos) se verán favorecidos, pues sus **prioridades aumentan**, permitiendo que continúen ejecutándose con mayor facilidad y sin ser interrumpidos frecuentemente.

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

Sí, la utilización del algoritmo **Round Robin (RR)** dentro de cada cola favorece al sistema de **Tiempo Compartido**. El algoritmo RR es muy adecuado para entornos interactivos, ya que permite una asignación justa y predecible del tiempo de CPU a cada proceso, lo que resulta en un **buen tiempo de respuesta** para los usuarios interactivos. Dado que el sistema tiene múltiples colas de prioridad y RR se aplica dentro de cada cola, cada proceso obtiene una **cuota equitativa de tiempo de CPU** en función de su prioridad, lo que minimiza el riesgo de que procesos de bajo nivel de prioridad sufran inanición. Además, el **quantum** de un segundo es adecuado para asegurar que ningún proceso, incluso los de baja prioridad, monopolice la CPU, permitiendo que los procesos de primer plano (interactivos) puedan recibir su tiempo de ejecución de forma más equitativa.

20)

Prioridad determinada estáticamente con el método del más corto primero (SJF):

- Cortos acotados por CPU

Favorece a los trabajos

prioridad dinámica inversamente proporcional al tiempo transcurrido desde la última
operación de E/S:

- cortos acotados por E/S
- largos acotados por E/S

21)

Cuando el quantum en el algoritmo Round Robin (RR) se agranda, cada proceso tiene más tiempo para ejecutarse antes de ser interrumpido. Si el quantum es lo suficientemente grande, el algoritmo RR se comporta de manera similar a FIFO (First In, First Out), ya que, en muchos casos, el proceso en ejecución terminará antes de que se agote su quantum, lo que significa que no será interrumpido antes de finalizar. En este caso, el sistema no realiza un cambio

de contexto frecuente, y cada proceso obtiene un tiempo de ejecución continuo, lo que emula el comportamiento de FIFO, donde los procesos se ejecutan en el orden en que llegan sin interrupciones.

22)

A)
La mayoría de las
PCs de escritorio actuales tienen **procesadores fuertemente acoplados**, ya que generalmente **comparten memoria principal** (RAM) y están bajo el control de un único **Sistema Operativo**. Además, los procesadores en una PC de escritorio suelen ser **homogéneos**, ya que generalmente se utiliza un solo tipo de procesador para la gestión de todas las tareas.

b)

En un sistema simétrico, no hay procesadores dedicados o especializados para tareas específicas. Todos los procesadores son
tratados de manera igualitaria, y se puede asignar cualquier proceso a cualquier CPU disponible.

c)

El
esquema Maestro/Esclavo es un modelo en el que **un procesador principal (maestro)** controla y coordina la ejecución de uno o más **procesadores subordinados (esclavos)**. En este modelo, el procesador maestro generalmente se encarga de la toma de decisiones y la gestión de los recursos, mientras que los esclavos realizan tareas específicas bajo la dirección del maestro.

23)

En un sistema de procesadores homogéneos,
Round Robin es uno de los métodos más sencillos porque permite una asignación equitativa de los procesos entre las CPUs sin necesidad de priorizar

o gestionar diferentes tipos de procesos.

Ventajas del método Round Robin (RR):

1. **Equidad**: Cada proceso recibe la misma cantidad de tiempo de CPU (quantum), lo que garantiza que todos los procesos tengan una oportunidad justa de ejecutarse.
2. **Simplicidad**: RR es fácil de implementar, ya que no requiere complejas estructuras de datos o cálculos de prioridades.
3. **Sin inanición**: Dado que cada proceso se ejecuta por un tiempo limitado, no hay procesos que se queden esperando indefinidamente (inanición), lo que es especialmente importante en sistemas multitarea.

Desventajas del método Round Robin (RR):

1. **Rendimiento limitado**: Si el **quantum** es muy grande, RR se asemeja a un enfoque de **FIFO**, lo que podría hacer que los procesos largos o con alta carga de CPU no se gestionen eficientemente.
2. **Desperdicio de tiempo**: Si el quantum es demasiado pequeño, el sistema puede hacer muchos cambios de contexto, lo que **reduce la eficiencia** debido al **gasto de recursos** en la conmutación de procesos.
3. **No tiene en cuenta la prioridad**: RR no considera la prioridad de los procesos, lo que puede hacer que procesos más importantes o de mayor prioridad no se gestionen de manera óptima si no se ajusta el quantum.