

Introducción a los Sistemas Operativos

Práctica 4

# Objetivo

El objetivo de esta práctica es que el alumno comprenda los aspectos base acerca de la planifi- cación de procesos en un Sistema Operativo (tipos de planificadores, algoritmos y sus variantes, etc.). Además, para la autocorrección de los ejercicios, es deseable la utilización del simulador que se encuentra en cátedras virtuales1.

1. Responda en forma sintética sobre los siguientes conceptos:
   1. Programa y Proceso.
   2. Defina Tiempo de retorno (**TR**) y Tiempo de espera (**TE**) para un Job.
   3. Defina Tiempo Promedio de Retorno (**TPR**) y Tiempo promedio de espera (**TPE**) para un lote de JOBS.
   4. ¿Qué es el Quantum?
   5. ¿Qué significa que un algoritmo de scheduling sea apropiativo o no apropiativo (Pre- emptive o Non-Preemptive)?
   6. ¿Qué tareas realizan?:
      1. *Short Term Scheduler*
      2. *Long Term Scheduler*
      3. *Medium Term Scheduler*
   7. ¿Qué tareas realiza el *Dispatcher* ?
2. Programa

Es estático

No tiene program cunter

Existe desde que se edita hasta que se borra

Proceso/Tarea/Job

Es dinámico

Tiene program counter

Su ciclo de vida comprende desde que se lo ejecuta hasta que termina

Según su historial de ejecución, los podemos clasificar en:

 CPU Bound (ligados a la CPU)

 I/O Bound (ligados a entrada/salida

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

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

1. Tiempo Promedio de Retorno (**TPR**) para un lote de JOBS: Es la suma de los tiempos de retorno de los N procesos, dividido N

Tiempo promedio de espera (**TPE**) para un lote de JOBS: Es la suma de los tiempos de espera de los N procesos, dividido N

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

 Pequeño: overhead de context switch

 Grande: todo lo que entra terminaría, seria ejecución secuencial

1. 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

(overhead: Al cambiar de un proceso a otro el sistema operativo (mediante el dispatcher) genera lo que se conoce como overhead, que es tiempo desperdiciado por el procesador para realizar un cambio de contexto (context switch)

1.  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

1. Un dispatcher en un sistema operativo es un componente que administra la asignación de tiempo de CPU a diferentes procesos en un entorno multitarea. Su función principal es programar la ejecución de procesos de manera eficiente, permitiendo que múltiples programas se ejecuten de manera aparentemente simultánea, compartiendo los recursos de la CPU de forma equitativa o priorizada.
2. Procesos:
   1. Investigue y detalle para que sirve cada uno de los siguientes comandos.
      1. *Top:* Muestra información en tiempo real sobre el uso del sistema y listado de procesos en ejecución.
      2. *Htop:* Versión avanzada de "top" con una interfaz más amigable y funciones interactivas
      3. *Ps:* Lista los procesos en ejecución en el sistema con información básica.
      4. *Pstree:* Muestra la jerarquía de procesos en forma de árbol.
      5. *Kill:* Envía señales para controlar la terminación de procesos.
      6. *Pgrep,pkill,* killall Herramientas para buscar y gestionar procesos por nombre o atributos.
      7. *killall: Mata todos los procesos*
      8. *renice:* Cambia la prioridad de procesos en ejecución.
      9. *Xkill:* Cierra aplicaciones gráficas seleccionadas en entornos gráficos
      10. *Atop:* Herramienta de monitoreo avanzada que muestra información detallada sobre recursos del sistema.
   2. Observe detenidamente el siguiente código. Intente entender lo que hace sin necesidad de ejecutarlo.

#i n c l u d e <s t d i o . h>

#i n c l u d e <sys / types . h> #i n c l u d e <unistd . h>

i n t main ( void ) {

i n t c ; pid\_t pid ;

p r i n t f ( " Comienzo . : \ n " ) ;

f o r ( c = 0 ; c < 3 ; c++ )

{

pid = f o r k ( ) ;

}

p r i n t f ( " Proceso \n " ) ; return 0 ;

}

* + 1. *¿Cuántas líneas con la palabra “Proceso” aparecen al final de la ejecución de este programa?.*
    2. *¿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.

En la ejecución de este programa, aparecerán 4 líneas con la palabra "Proceso" al final. Esto se debe a que la línea "Proceso" se imprime una vez en la función principal (proceso padre) y tres veces en el bucle **for** (procesos hijos), ya que se ejecuta **printf("Proceso\n");** dentro del bucle.

No, el número de líneas impresas no es igual al número de procesos que han estado en ejecución. El número de líneas impresas es mayor porque cada vez que se crea un proceso hijo en el bucle **for**, se imprime una línea adicional. En este caso, hay 1 proceso principal (padre) y 3 procesos hijos creados en el bucle **for**, lo que da un total de 4 líneas impresas con "Proceso".

* 1. 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.

#i n c l u d e <s t d i o . h>

#i n c l u d e <sys / types . h> #i n c l u d e <unistd . h>

i n t main ( void ) {

i n t c ,p=0; pid\_t pid ;

f o r ( c = 0 ; c < 3 ; c++ )

{

pid = f o r k ( ) ;

} p++;

p r i n t f ( " Proceso %d\n " , p ) ; return 0 ;

}

* + 1. *¿Qué valores se muestran por consola?.*
    2. *¿Todas las líneas tendrán el mismo valor o algunas líneas tendrán valores distin- tos?.*
    3. *¿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.

El valor de la variable **p** se muestra al final del programa, y se mostrará el valor que tenga después de las tres iteraciones del bucle **for** y de los incrementos realizados en cada proceso hijo.

Todas las líneas tendrán el mismo valor, ya que el código muestra el valor de **p** al final del programa, después de que todos los procesos hayan completado su ejecución. Sin embargo, cada proceso hijo realiza un incremento de **p** antes de mostrarlo, por lo que si fuera a mostrar el valor intermedio, podrías ver diferentes valores temporalmente en cada proceso hijo.

El valor que se muestra en la consola será el valor final de **p**, que será igual a 1 más el número de procesos hijos creados en el bucle **for**. Dado que se crean tres procesos hijos, **p** tendrá un valor de 4 al final del programa.

* 1. Comunicación entre procesos:
     1. *Investigue la forma de comunicación entre procesos a través de pipes.*
     2. *¿Cómo se crea un pipe en C?.*
     3. *¿Qué parametro es necesario para la creación de un pipe?. Explique para que se utiliza.*
     4. *¿Qué tipo de comunicación es posible con pipes?*

i.Los pipes proporcionan una comunicación unidireccional, lo que significa que se pueden usar para enviar datos desde un proceso emisor (escritura) a un proceso receptor (lectura) de manera secuencial.

ii. En C, puedes crear un pipe utilizando la función **pipe()**. La declaración de esta función se encuentra en la biblioteca **<unistd.h>**. Aquí tienes un ejemplo simple de cómo crear un pipe:

#include <unistd.h>

int main() {

int mypipe[2];

if (pipe(mypipe) == -1) {

perror("pipe");

return 1;

}

// Ahora mypipe[0] es el extremo de lectura y mypipe[1] es el extremo de escritura del pipe.

// Puedes usar estos descriptores de archivo para comunicarte entre procesos.

return 0;

}

iii.No se requiere ningún parámetro específico para crear un pipe. La función **pipe()** simplemente toma un array de dos enteros como argumento, y crea el pipe utilizando esos descriptores de archivo. E

iv. Los pipes permiten la comunicación unidireccional y secuencial entre procesos. El proceso emisor escribe datos en un extremo del pipe, y el proceso receptor los lee desde el otro extremo. Los datos se transmiten en un flujo continuo y se consumen a medida que se leen. Los pipes son adecuados para situaciones en las que un proceso necesita enviar datos a otro proceso de manera ordenada y secuencial, pero no son adecuados para comunicación bidireccional o para compartir grandes cantidades de datos.

* 1. ¿Cuál es la información mínima que el SO debe tener sobre un proceso?¿En que es- tructura de datos asociada almacena dicha información?

Tabla

Descripción generada automáticamente

La información mínima que un sistema operativo debe tener sobre un proceso incluye su identificador único (PID), estado actual, el P.C., la ubicación de la imagen del programa en memoria y registros del procesador. Toda esta información se almacena en una estructura de datos llamada Tabla de Control de Procesos (PCB). El PCB es esencial para gestionar procesos y se actualiza a medida que el proceso cambia de estado o experimenta eventos.

La PCB es lo primero que se crea cuando se realiza un fork y lo ´ultimo que se desaloja cuando termina.

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

Según su historial de ejecución, los podemos clasificar en:

 CPU Bound (ligados a un uso intensivo de la CPU) Ej: Procesamiento de matrices

 I/O Bound (ligados a un uso intensivo de entrada/salida Ej: Word

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

1. **Nuevo (New):** El proceso se crea inicialmente en este estado. Representa un proceso que ha sido creado pero aún no se le ha asignado recursos o CPU.
2. **Listo (Ready):** El proceso está listo para ejecutarse, pero aún no se le ha asignado tiempo de CPU.
3. **Ejecución (Running):** El proceso se está ejecutando actualmente en la CPU.
4. **Bloqueado (Blocked):** El proceso está temporalmente detenido debido a una operación de entrada/salida u otro evento que requiere espera. El proceso no utiliza la CPU mientras está bloqueado.
5. **Suspendido (Suspend):** El proceso se encuentra en un estado de suspensión, lo que significa que se ha detenido temporalmente, generalmente debido a una decisión del sistema operativo. Un proceso suspendido no se ejecuta ni utiliza recursos de CPU.
6. **Finalizado (Exit):** El proceso ha terminado su ejecución y se encuentra en el proceso de ser eliminado o limpiado por el sistema operativo.
   1. Explique mediante un diagrama las posibles transiciones entre los estados.

Diagrama

Descripción generada automáticamente

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

1.  Long term scheduler: New, Exit

 Medium term scheduler: Ready, Blocked, Suspend

 Short term scheduler: Running

1. Para los siguientes algoritmos de scheduling:

 *FCFS* (Fisrt Coome First Served)

 *SJF* (Shortest Job First)

 *Round Robin * *Prioridades*

* 1. Explique su funcionamiento mediante un ejemplo.
  2. ¿Alguno de ellos requiere algún parámetro para su funcionamiento?
  3. Cual es el mas adecuado según los tipos de procesos y/o SO.
  4. Cite ventajas y desventajas de su uso.

A,b,c,d) Supongamos que tenemos tres procesos con t. en ms: A (5 ms), B (3 ms) y C (8 ms)

(-) **FCFS (First-Come, First-Served)**:

* Funcionamiento: Los procesos se ejecutan en el orden en que llegan a la cola de listos. El primero en llegar es el primero en ser atendido.
* Ejemplo: Si llegan en ese orden, FCFS ejecutaría los procesos en el orden A, B, C.
* Requisitos: No requiere parámetros adicionales
* Ventajas-Desventajas, según tipo: No favorece a ningún tipo de procesos, pero en principio podíamos decir que los CPU Bound terminan al comenzar su primer ráfaga, mientras que los I/O Bound no
*  Es un algoritmo preemptive

(-) **SJF (Shortest Job First)**:

* Funcionamiento: Elige el proceso más corto en términos de tiempo de ejecución. Minimiza el tiempo de espera promedio.
* Ejemplo, SJF ejecutaría primero B (3 ms), luego A (5 ms), y finalmente C (8 ms).
* Requisitos: No requiere parámetros adicionales (el proceso mas corto se calcula en base a ejecución previa)
* Ventajas-Desventajas, según tipo: Procesos cortos se colocan delante de procesos largos, Los procesos largos pueden sufrir starvation (inanición)
*  Es ser un algoritmo no preemptive

(-) **Round Robin**:

* Funcionamiento: Asigna un quantum de tiempo de CPU a cada proceso en orden cíclico. Si un proceso no termina su ejecución en el quantum, se mueve al final de la cola de listos.
* Ejemplo: Supongamos que el quantum es 4 ms y los mismos procesos. El orden de ejecución sería A (4 ms), B (4 ms), C (4 ms), A (1 ms), C (4 ms), B (3 ms), y así sucesivamente.
* Requisitos: Requiere un quantum
* Ventajas-Desventajas, según tipo: Si el quantum es muy alto, se produce una ventaja de los Cpu bound, si es muy baja, se pierde mucho tiempo al cambiar de procesos, si está en equilibrio es uno de los mejores si se utiliza un sistema que posea Io y Cpu bound
*  Es un algoritmo preemptive

(-) **Prioridades**:

* Funcionamiento: Asigna prioridades a los procesos, donde los procesos de mayor prioridad se ejecutan antes que los de menor prioridad.
* Ejemplo: Si A tiene prioridad alta, B tiene prioridad media y C tiene prioridad baja, entonces A se ejecutará antes que B y C.
* Requisitos: No requiere parámetros adicionales (Requiere que se sepan los valores de prioridad del proceso)
* Ventajas-Desventajas, según tipo: Procesos de baja prioridad pueden sufrir starvation (inanición) Solución: permitir a un proceso cambiar su prioridad durante su ciclo de vida → Aging o Penalty
*  Puede ser un algoritmo preemptive o no

Resumen Ventajas-Desventajas:

* FCFS: Ventaja - Fácil de entender. Desventaja - Puede llevar a tiempos de espera largos para procesos largos.
* SJF: Ventaja - Minimiza el tiempo de espera promedio. Desventaja - Requiere conocimiento previo de los tiempos de ejecución.
* Round Robin: Ventaja - Equidad en el uso de CPU. Desventaja - Puede tener una sobrecarga significativa de cambio de contexto.
* Prioridades: Ventaja - Permite la asignación de recursos según la importancia de los procesos. Desventaja - Posibilidad de inanición de procesos de baja prioridad.

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

# Timer Fijo

 **Timer Variable**

* 1. ¿Qué significan estas 2 variantes?
  2. Explique mediante un ejemplo sus diferencias.
  3. En cada variante ¿Dónde debería residir la información del Quantum?

1. Timer variable:

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

Gráfico, Gráfico en cascada

Descripción generada automáticamente  Es el mas utilizado y es utilizado por el simulador

Timer fijo:

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

 if (contador == 0) contador = Q;

 Se puede ver como Q compartido entre los procesos

1. Supongamos 2 procesos A y B, CPUBound con T.R -> Infinito, pero A tubo una interrupción de e/s en el 3° uso de CPU, si el quantum es 4….

-En caso de que sea con timer Variable, A va a ejecutar 2 veces, tiene su interrupción por lo que termina su tiempo y B ahora ejecuta su quantum completo de 4, luego se le retorna el control a A

-En caso de que sea con timer fijo, B va a ejecutar 2 veces, tiene su interrupción por lo que ahora se ejecuta B, 2 veces hasta que se agota el quantum con el que comenzó A, por lo tanto termina su tiempo y ahora vuelve a ejecutarse A

1.  El “contador” puede ser:  Global (Quantum fijo-¿variable?)

 Local → PCB (Quantum variable)

1. 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 |

* 1. Realice los diagramas de Gantt según los siguientes algoritmos de scheduling:
     1. FCFS (First Come, First Served)
     2. SJF (Shortest Job First)
     3. Round Robin con quantum = 4 y Timer Fijo
     4. Round Robin con quantum = 4 y Timer Variable
  2. Para cada algoritmo calcule el TR y TE para cada job así como el TPR y el TPE.
  3. En base a los tiempos calculados compare los diferentes algoritmos.

TODO HECHO EN EXCEL (VER)

1. Se tiene el siguiente lote de procesos:
   1. 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 |

* + 1. FCFS (First Come, First Served)
    2. SJF (Shortest Job First)
    3. Round Robin con quantum = 1 y Timer Variable
    4. Round Robin con quantum = 6 y Timer Variable
  1. Para cada algoritmo calcule el TR y TE para cada job así como el TPR y el TPE.
  2. En base a los tiempos calculados compare los diferentes algoritmos.
  3. En el algoritmo Round Robin, que conclusión se puede sacar con respecto al valor del quantum.
  4. ¿Para el algoritmo Round Robin, en que casos utilizaría un valor de quantum alto y que ventajas y desventajas obtendría?

1. Una variante al algoritmo SJF es el algoritmo SJF apropiativo o SRTF (Shortest Remaining Time First):
   1. Realice el diagrama del Gantt para este algoritmo según el lote de trabajos del ej. 6.
   2. ¿Nota alguna ventaja frente a otros algoritmos?
2. 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 |

* 1. Realice el diagrama de Gantt correspondiente al algoritmo de planificación por priori- dades según las variantes:
     1. No Apropiativa
     2. Apropiativa
  2. Calcule el TR y TE para cada job así como el TPR y el TPE.
  3. ¿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?

1. Inanición (*Starvation*)
   1. ¿Qué significa?
   2. ¿Cuál/es de los algoritmos vistos puede provocarla?
   3. ¿Existe alguna técnica que evite la inanición para el/los algoritmos mencionados en [b](#_bookmark4)?
2. Significa que un proceso queda mucho tiempo sin ser atendido
3. STF -SRTF -PRIORIDADES
4. Si…En prioridades cambiarle aumentarle la prioridad a un proceso que lleva mucho tiempo (Aging o Penalty)

Además, se puede planificar colas múltiples, para que se ejecute un proceso con prioridad baja cada N tiempo.

O agregar un tiempo de espera máximo, que una vez vencido ejecuta ese proceso

1. 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 dis- positivo 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:

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

* 1. Suponga que al lote de procesos del ejercicio [6](#_bookmark3) 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) |

* 1. Suponga que al lote de procesos del ejercicio [6](#_bookmark3) se agregan las siguientes operaciones de entrada salida:

1. 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:
   1. Round Robin; Desventajas= Trashing y poco tiempo para Io bound si tiene mucha E/s
   2. SRTF (Shortest Remaining Time First) = Starbation y CPU bound jodidisimos
2. Para equiparar la desventaja planteada en el ejercicio [11](#_bookmark5)), se plantea la siguiente modifica- ción al algoritmo:

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

* 1. Analice el funcionamiento de este algoritmo mediante un ejemplo. Marque en cada instante en que cola se encuentran los procesos. /No
  2. Realice el ejercicio 10)a) nuevamente considerando este algoritmo, con un quantum de 2 unidades y Timer Variable.

1. 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.

Si, bajo ciertas condiciones de carga y ejecución, es posible que el quantum de un proceso I-O bound con muchas ráfagas seguidas y largas esperas de E/S en un algoritmo VRR con Timer Variable nunca llegue a cero debido a la prioridad constante de otros procesos I-O bound que vuelven de interrupción y se ejecutan previamente y que el mismo proceso hace muchas ráfagas cortas de un quantum muy pequeño .

1. 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.

Interfaz de usuario gráfica, Aplicación, Word

Descripción generada automáticamente

S1 no es calculado

1. Suponga un proceso cuyas ráfagas de CPU reales tienen como duración: 6, 13, 4, 6, 4, 13, 13 Calcule que valores se obtendrían como estimación para las ráfagas de CPU del proceso si se utiliza la formula [1](#_bookmark6), con un valor inicial estimado de S1=10.

La formula anterior [1](#_bookmark6) le da el mismo peso a todos los casos (siempre calcula la media). Es posible reescribir la formula permitiendo darle un peso mayor a los casos mas recientes y menor a casos viejos (o viceversa). Se plantea la siguiente formula:

*Sn*+1 = *αTn* + (1 − *α*)*Sn* (2)

Con 0 ⟨ *α* ⟨ 1.

1. Analice para que valores de *α* se tienen en cuenta los casos mas recientes.
2. Para la situación planteada en a) calcule que valores se obtendrían si se utiliza la formula [2](#_bookmark8) con *α* = 0*,* 2; *α* = 0*,* 5 y *α* = 0*,* 8.
3. Para todas las estimaciones realizadas en [a](#_bookmark7) y [c](#_bookmark9) ¿Cuál es la que mas se asemeja a las ráfagas de CPU reales del proceso?, Se asemeja + la c de 0,8
   1. n= 7 ; Tn = 13 ; Sn = 10

Imagen que contiene objeto, reloj

Descripción generada automáticamente

Usando S2=queda 10,4 Unidades de C.P.U (11)

* 1. \*Si *α = 0* , no se tiene en consideración los casos mas viejos (Los resultados + viejo desaparece)

\* Si *α = 1* , se tiene en consideración los casos mas viejos

* 1. *Sn*+1 = *αTn* + (1 − *α*)*Sn*  ; n= 7 ; Tn = 13 ; Sn = 10

*α* = 0*,* 2; S2=10,6 (11) unidades de cpu

*α* = 0*,* 5 S2=11,5 (12) unidades de cpu

*α* = 0*,* 8. S2=12,4 (13) unidades de cpu

(Aclaracion, la formula es una sumatoria, para calcular S7 hay que repetir todo, ver Calculos Papel)

1. Colas Multinivel

Hoy en día los algoritmos de planificación vistos se han ido combinando para formar al- goritmos 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.

* 1. Suponga que se tienen dos tipos de procesos: *Interactivos* y *Batch*. Cada uno de estos procesos se coloca en una cola según su tipo. ¿Qué algoritmo de los vistos utilizaría para administrar cada una de estas colas?.

A su vez, se utiliza un algoritmo para administrar cada cola que se crea. Así, por ejemplo, el algoritmo podría determinar mediante prioridades sobre que cola elegir un proceso.

* 1. Para el caso de las dos colas vistas en [a](#_bookmark10): ¿Qué algoritmo utilizaría para planificarlas?

1. Adminsitraria la cola Batch con FCFS o RR q Grande

Adminsitraria la cola Batch con SRT o RR q Chico

1. Para planificar las colas, utilizaría un algoritmo de Prioridades con aging no apropiativo donde todos entren al RR q-chico, y si lo utilizan todo varias veces pase a FCFS (Beneficia a I-O Bound)
2. 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:

1. Asumiendo que NO hay apropiación entre los procesos.
2. Asumiendo que hay apropiación entre los procesos.
3. 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 esta 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 de 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.

1. Para los casos [a](#_bookmark12) y [b](#_bookmark13) del ejercicio [16](#_bookmark11) realice el diagrama de Gantt considerando además que se tiene un envejecimiento de 4 unidades.
2. La situación planteada en el ejercicio [17](#_bookmark14), 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 adminis- tración de las colas entre sí.

Tenga en cuenta que los procesos no deben sufrir inanición.

Cola nivel 1 R.R Quantum Chico

Cola nivel 2 R.R. Quantum Mediano

…

Cola nivel n FCFS

Método de pasaje entre colas: Si el proceso termina el quantum varias veces, baja de nivel

Administrador: Prioridades con envejecimiento

1. Un caso real: “Unix Clasico “ (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:

Imagen que contiene Diagrama

Descripción generada automáticamente*Ci* − 1)

donde:

2 (3)

*CPUj* (*i*) + *nie* (4)

2 *j*

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

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

*Basej* = Prioridad base del proceso *j*. *Nicej* = Factor de ajuste.

La prioridad del proceso se calcula cada segundo y se toma una nueva decisión de planifica- ción. 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:

 Intercambio

 Control de Dispositivos de I/O por bloques  Gestión de archivos

 Control de Dispositivos de I/O de caracteres  Procesos de usuarios

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.

1. Analizando la jerarquía descripta para las bandas de prioridades: ¿Que tipo de acti- vidad considera que tendrá más prioridad? ¿Por qué piensa que el scheduler prioriza estas actividades?
2. 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

1. La utilización de RR dentro de cada cola: ¿Verdaderamente favorece al sistema de Tiempo Compartido? Justifique.
   1. Calendario

      Descripción generada automáticamenteSegún la jerarquía descrita para las bandas de prioridades, las actividades relacionadas con el "Intercambio" tendrían más prioridad. Esto se debe a que se asigna la prioridad más alta a esta banda, seguida por el "Control de Dispositivos de I/O por bloques", "Gestión de archivos", "Control de Dispositivos de I/O de caracteres" y finalmente los "Procesos de usuarios". La razón detrás de esta priorización es la importancia de las operaciones de intercambio en sistemas operativos, ya que afectan directamente al rendimiento general del sistema.

(Dibujo descripto en enunciado)

1. En el contexto del ajuste de prioridades descrito, un valor negativo de "nicej" aumenta la prioridad del proceso. Un proceso con un valor negativo de "nicej" se considera "agradable" o "amigable" en términos de prioridad. Por lo tanto, favorece a los procesos que tienen una carga de E/S (Entrada/Salida) más intensiva en lugar de aquellos que son intensivos en CPU.

En general, los procesos I/O Bound (vinculados a E/S) tienden a beneficiarse más, ya que los mismos poseen un valor pequeño de ciclos de cpu, lo que tira su promedio para abajo y en consecuencia aumenta su prioridad, lo que implica que tienen más posibilidades de obtener recursos de CPU. Por otro lado, los procesos CPU Bound (vinculados a la CPU) pueden tener valores grandes de promedio de uso de ciclos, lo que reduce su prioridad y les asigna menos recursos de CPU en comparación con los procesos I/O Bound.

1. La utilización de Round Robin (RR) dentro de cada cola tiene el propósito de proporcionar un tiempo de respuesta rápido a los procesos interactivos. En un entorno de tiempo compartido, donde varios usuarios pueden estar interactuando con el sistema simultáneamente, el uso de RR permite que cada proceso tenga la oportunidad de ejecutarse en ráfagas cortas antes de pasar al siguiente proceso en la cola. Esto contribuye a una experiencia de usuario más receptiva.
2. A cuáles de los siguientes tipos de trabajos:
3. cortos acotados por CPU
4. cortos acotados por E/S
5. largos acotados por CPU
6. largos acotados por E/S

benefician las siguientes estrategias de administración:

1. prioridad determinada estáticamente con el método del más corto primero (SJF).
2. prioridad dinámica inversamente proporcional al tiempo transcurrido desde la última operación de E/S
3. Explicar porqué si el quantum "q.en Round-Robin se incrementa sin límite, el método se aproxima a FIFO.

**A) Cortos acotados por CPU:**

* **SJF (Shortest Job First):** Beneficia a los trabajos cortos al priorizar los más breves.
* **Prioridad inversamente proporcional al tiempo desde la última E/S:** No beneficia directamente a los trabajos cortos acotados por CPU.

b) **Cortos acotados por E/S:**

* **SJF:** Puede beneficiar al minimizar el tiempo de espera global.
* **Prioridad inversamente proporcional al tiempo desde la última E/S:** Beneficia directamente a los trabajos cortos acotados por E/S al asignarles mayor prioridad.

c) **Largos acotados por CPU:**

* **SJF:** No beneficia a los trabajos largos acotados por CPU.
* **Prioridad inversamente proporcional al tiempo desde la última E/S:** No los beneficia (Al contrario)

d) **Largos acotados por E/S:**

* **SJF:** No beneficia directamente
* **Prioridad inversamente proporcional al tiempo desde la última E/S:** Los perjudica, ya que si pasa mucho tiempo de E/S el sistema le va a asignar una menor prioridad

21) En Round-Robin (RR) con un quantum infinitamente grande, se aproxima a FIFO porque cada proceso tiene una oportunidad de ejecución completa antes de pasar al siguiente, eliminando la rotación interrumpida y asemejándose a FIFO

1. 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 memo- ria principal y canales.

 **Procesadores especializados**: Existe uno o más procesadores principales de propó- sito 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

1. ¿Con cuál/es de estas clasificaciones asocia a las PCs de escritorio habituales?
2. ¿Qué significa que la asignación de procesos se realice de manera simétrica?
3. ¿Qué significa que se trabaje bajo un esquema Maestro/esclavo?
4. **Las PCs de escritorio habituales suelen asociarse con la clasificación de multiprocesador homogéneo fuertemente acoplados. En una PC típica, los procesadores son del mismo tipo y no hay una jerarquía de privilegios entre ellos, y todos comparten la misma memoria principal. Esto contribuye a un alto grado de acoplamiento entre los procesadores, ya que comparten directamente la memoria y se gestionan de manera coordinada por el sistema operativo.**
5. La asignación de procesos de manera simétrica significa que todos los procesadores en un sistema multiprocesador tienen la misma capacidad para ejecutar cualquier tarea. No hay distinción de roles o privilegios entre los procesadores, y todos son capaces de llevar a cabo cualquier tipo de trabajo. Esta simetría facilita la distribución equitativa de la carga de trabajo entre los procesadores.
6. En un esquema Maestro/Esclavo, un procesador principal (maestro) coordina y distribuye tareas a varios procesadores secundarios (esclavos) que ejecutan las operaciones asignadas. El maestro controla la asignación de trabajo y la gestión global del sistema. Este esquema se utiliza a menudo en sistemas heterogéneos o en aplicaciones específicas donde algunos procesadores tienen funciones especializadas. Por ejemplo, en un entorno de procesamiento paralelo, el maestro puede dividir la tarea en subproblemas y asignarlos a los esclavos para su procesamiento.
7. Asumiendo el caso de procesadores homogéneos:
8. ¿Cuál sería el método de planificación más sencillo para asignar CPUs a los procesos?
9. Cite ventajas y desventajas del método escogido
10. El método de planificación más sencillo en procesadores homogéneos es el **FIFO (First-In-First-Out)** o "Primero en llegar, primero en ser servido". En este método, los procesos se ejecutan en el orden en que llegan a la cola de listos.**:**
11. **Ventajas:** **Simplicidad:** El método FIFO es fácil de entender e implementar. No requiere algoritmos complejos ni cálculos.
12. **Desventajas:** **Tiempo de espera promedio más largo:** Puede llevar a un mayor tiempo de espera, ya que los proceso son ejecutados según llegada, sin tener en cuenta su duración, provocando inanición si hay mucho CPU bound largo.
13. Indique brevemente a que hacen referencia los siguientes conceptos:
14. Huella de un proceso en un procesador
15. Afinidad con un procesador
16. ¿Por qué podría ser mejor en algunos casos que un proceso se ejecute en el mismo procesador?
17. ¿Puede el usuario en *Windows* cambiar la afinidad de un proceso? ¿y en *GNU/Linux*?
18. Investigue el concepto de balanceo de carga (load balancing).
19. Compare los conceptos de afinidad y balanceo de carga y como uno afecta al otro.

**a) Huella de un proceso en un procesador:** La "huella de un proceso en un procesador" se refiere a la cantidad de recursos y memoria que un proceso utiliza mientras se ejecuta en el procesador. La huella incluye la cantidad de CPU, memoria RAM y otros recursos que el proceso consume durante su ejecución.

**b) Afinidad con un procesador:** La "afinidad con un procesador" se refiere a la preferencia o restricción de un proceso para ejecutarse en un procesador específico. Un proceso con afinidad a un procesador determinado tiende a ejecutarse preferentemente en ese procesador.

**c) ¿Por qué podría ser mejor en algunos casos que un proceso se ejecute en el mismo procesador?** Ejecutar un proceso en el mismo procesador puede ser beneficioso debido a la memoria caché local y a la minimización de la necesidad de transferir datos entre cachés. Esto puede reducir los tiempos de acceso a memoria y mejorar el rendimiento, ya que los datos pueden permanecer en la caché local, evitando así la latencia asociada con la transferencia de datos entre cachés o memorias principales.

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

* **Windows:** Sí, en Windows, los usuarios pueden cambiar la afinidad de un proceso. Pueden asignar o cambiar los núcleos del procesador en los que un proceso específico puede ejecutarse a través del Administrador de tareas o mediante el uso de herramientas específicas.
* **GNU/Linux:** Sí, en GNU/Linux, los usuarios también pueden cambiar la afinidad de un proceso utilizando comandos como **taskset** en la línea de comandos o herramientas gráficas proporcionadas por los entornos de escritorio.

**e) Investigación del concepto de balanceo de carga (load balancing):** El "balanceo de carga" se refiere a la distribución equitativa de la carga de trabajo entre varios recursos, como procesadores o servidores, con el objetivo de maximizar la eficiencia, minimizar el tiempo de respuesta y evitar la sobrecarga de recursos individuales.

**f) Comparación de los conceptos de afinidad y balanceo de carga:**

* **Afinidad:** Se refiere a la preferencia de un proceso por ejecutarse en un procesador específico.
* **Balanceo de carga:** Busca distribuir la carga de trabajo de manera equitativa entre múltiples recursos para optimizar el rendimiento global del sistema.

**Relación entre afinidad y balanceo de carga:**

* La afinidad puede influir en el balanceo de carga al determinar en qué procesador se ejecutan los procesos. Si varios procesos tienen afinidad con un solo procesador, puede haber una carga desequilibrada.
* El balanceo de carga puede influir en la afinidad al intentar distribuir la carga equitativamente, lo que podría afectar la preferencia de ejecución de un proceso en un procesador específico.

1. Si a la tabla del ejercicio [6](#_bookmark3) la modificamos de la siguiente manera: Y considerando que 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 |

1. Ejecute el esquema anterior utilizando el algoritmo anterior.
2. Ejecute el esquema anterior. Pero ahora si el procesador preferido no está libre es asignado a otro procesador. Luego el procesador preferido de cada job es el último en el cual ejecuto.
3. Para cada uno de los casos calcule el tiempo promedio de retorno y el tiempo promedio de espera.

¿Cuál de las dos alternativas planteadas es mas performante? Resuelto en Excel

Importante :\* Cuando se tiene el algoritmo 2\* chance, se le saca a cada proceso el bit de referenciado hasta llegar a una pagina que lo tenga en 0 (Es la que sacas), y

\*Los algoritmos, si termina un proceso, no les importa, SE SIGUE CON LA POLITICA TAL CUAL

(Correcciones hechas sobre fecha parcial)