PROCESOS EN LOS SISTEMAS OPERATIVOS

MARIA ANGELICA BUITRAGO MENDEZ

DAVID ENRIQUE BALLESTEROS

\*JOSE FERNANDO GALINDO SUAREZ

FUNDACION UNIVERSITARIA UNIPANAMERICANA

INGENIERIA DE TELECOMUNICACIONES

SISTEMAS OPERATIVOS

MAYO DE 2019

\*Tutor

INTRODUCCION

Planificación de procesos en Sistemas Operativos. Conjunto de políticas y mecanismos incorporados al sistema operativo, a través de un módulo denominado planificador, que debe decidir cuál de los procesos en condiciones de ser ejecutado conviene ser despachado primero y qué orden de ejecución debe seguirse. Esto debe realizarse sin perder de vista su principal objetivo que consiste en el máximo aprovechamiento del sistema, lo que implica proveer un buen servicio a los procesos existentes en un momento dado.

MARCO TEORICO

**PROCESOS**

Un proceso es un programa en ejecución. Existen 3 estados en los que puede encontrarse un proceso, estos son: "Listo", "Bloqueado" y "En ejecución". Para el control de estos internamente son almacenados en una lista, cada uno de los nodos guarda información de un proceso. En esa información se almacena, entre otros aspectos, el estado en que se encuentra el proceso, el tiempo que el proceso ha usado el CPU, e información de E/S (entrada/salida). Los sistemas operativos cuentan con un componente llamado planificador, que se encarga de decidir cuál de los procesos hará uso del procesador. La toma de esta decisión, así como el tiempo de ejecución del proceso, estará dada por un algoritmo, denominado Algoritmo de Planificación.

**OBJETIVOS DE LA PLANIFICACIÓN DE PROCESOS**

La Planificación de procesos tiene como principales objetivos la equidad, la eficacia, el tiempo de respuesta, el tiempo de regreso y el rendimiento.

* *Equidad:* Todos los procesos deben ser atendidos.
* *Eficacia:* El procesador debe estar ocupado el 100% del tiempo.
* *Tiempo de respuesta:* El tiempo empleado en dar respuesta a las solicitudes del usuario debe ser el menor posible.
* *Tiempo de regreso*: Reducir al mínimo el tiempo de espera de los resultados esperados por los usuarios por lotes.
* *Rendimiento:* Maximizar el número de tareas que se procesan por cada hora.

**ALGORITMOS DE PLANIFICACIÓN**

* *Primero en llegar primero en ser servido*

Conocido como FCFS (First Come 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.

* *Prioridad al más corto*

Su nombre es SJF (Shortest Job First). El proceso que se encuentra en ejecución cambiará de estado voluntariamente, o sea, no tendrá un tiempo de ejecución determinado para el proceso. A cada proceso se le asigna el tiempo que usará cuando vuelva a estar en ejecución, y se irá ejecutando el que tenga un menor tiempo asignado. Si se da el caso de que dos procesos tengan igual valor en ese aspecto emplea el algoritmo FCFS.

* *Round Robin*

A cada proceso se le asigna un tiempo determinado para su ejecución, 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. Este es un algoritmo basado en FCFS que trata la cola de procesos que se encuentran en estado de listos como una cola circular.

* *Planificación por prioridad*

En este tipo de planificación a cada proceso se le asigna una prioridad siguiendo un criterio determinado, y de acuerdo con esa prioridad será el orden en que se atienda cada proceso.

* *Planificación garantizada*

Para realizar esta planificación el sistema tiene en cuenta el número de usuarios que deben ser atendidos. Para un número "n" de usuarios se asignará a cada uno un tiempo de ejecución igual a 1/n.

* *Planificación de Colas Múltiples*

El nombre se deriva de MQS (Multilevel Queue Schedulling). En este algoritmo la cola de procesos que se encuentran en estado de listos es dividida en un número determinado de colas más pequeñas. Los procesos son clasificados mediante un criterio para determinar en qué cola será colocado cada uno cuando quede en estado de listo. Cada cola puede manejar un algoritmo de planificación diferente a las demás.

**TIEMPOS**

En la Planificación de procesos se tiene en cuenta diferentes tiempos que pueden ser calculados, como son el "Tiempo de espera medio", el "Tiempo de retorno del proceso" y el "Tiempo de retorno medio".

* Tiempo de espera medio

Es el promedio de tiempos en que los procesos están en estado de listos. En algoritmos FCFS este tiempo suele ser bastante largo. En algoritmos SJF para los procesos largos este tiempo suele ser muy grande, pues se estarán ejecutando constantemente los procesos más cortos y los más largos se encontrarán constantemente en espera, por lo que pueden entrar en inanición. En Planificación por prioridad los procesos de prioridad baja podrían no ejecutarse nunca. Para dar solución a este problema el envejecimiento de un programa eleva su prioridad.

* Tiempo de retorno del proceso

Es el tiempo que transcurre desde la creación de un proceso hasta que termina la ejecución del programa que le dio lugar.

* Tiempo de retorno medio

Es la suma de los tiempos de retorno de cada uno de los procesos dividida entre la cantidad de procesos.

DESARROLLO DE LA ACTIVIDAD

*Programar un algoritmo de planificación tipo round robin que permita identificar como el procesador o los procesadores atienden muchos procesos simultáneamente sin alterar la vista como los percibe el usuario final.*

*El algoritmo debe permitir:*

*1. Simular la cantidad de procesos que entran al procesador con la misma prioridad al procesador en forma FIFO (primeros en llegar primeros en ser atendidos)*

*2. Simular por cada proceso que llega al procesador un tiempo (QUANTUM) el cual deberá ser cronometrado según tiempo límite de expiración de este y se debe mostrar la interrupción cuando agote su tiempo y deberá dar paso a un nuevo proceso.*

*3. Determinar los procesos de acuerdo con la prioridad asignada para su ejecución en memoria (CE1. Determina los procesos de acuerdo a la prioridad asignada para su ejecución en memoria).*

*NOTA: Realizar el algoritmo de Round Robin en un lenguaje de programación que permita realizar lo expuesto en “DEMO: PROCESOS EN LOS SISTEMAS OPERATIVO CON ROUND ROBIN”, en la sección de recursos académicos dispuestos en la plataforma*

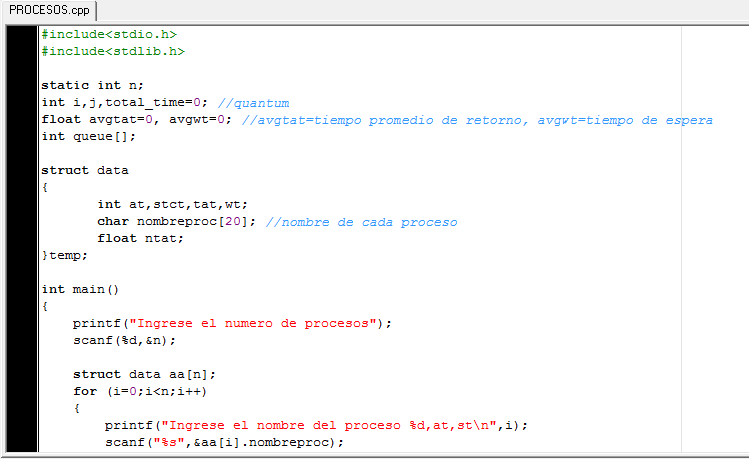
Cuando hablamos de informática y ordenadores, el algoritmo Round Robin es usado para ayudar a crear procesos o trabajos programados para asegurar que cada proceso requerido para hacer un trabajo tiene bastante tiempo para hacerlo. En los ordenadores, la CPU puede hacer divisiones de tiempo para configurar un tiempo determinado para cada proceso a usar en cada ciclo. Usando Round Robin asigna una porción de tiempo para cada proceso que está en funcionamiento.

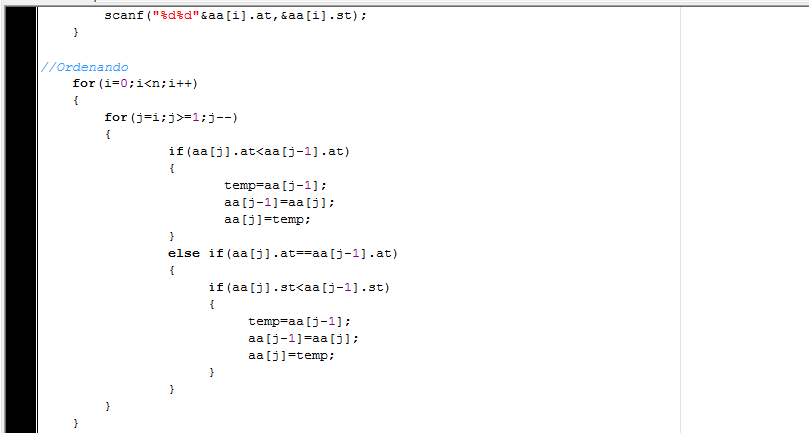
Por ejemplo, en un ordenador el usuario empieza tres aplicaciones, que pueden ser la aplicación de correo, el navegador web y un procesador de texto. Estas aplicaciones son cargadas en el sistema de memoria, y cada uno de ellos puede funcionar de forma transparente para el usuario. Round Robin se encarga de compartir los recursos entre las tres aplicaciones y sus procesos (y muchas otras que están funcionando de forma invisible para el usuario). Esta forma de manejar los procesos funciona bien porque cada aplicación consigue una cierta cantidad de tiempo por cada proceso cíclico. Un proceso cíclico es la cantidad de tiempo que le lleva a la CPU gestionar cada proceso el mismo tiempo.

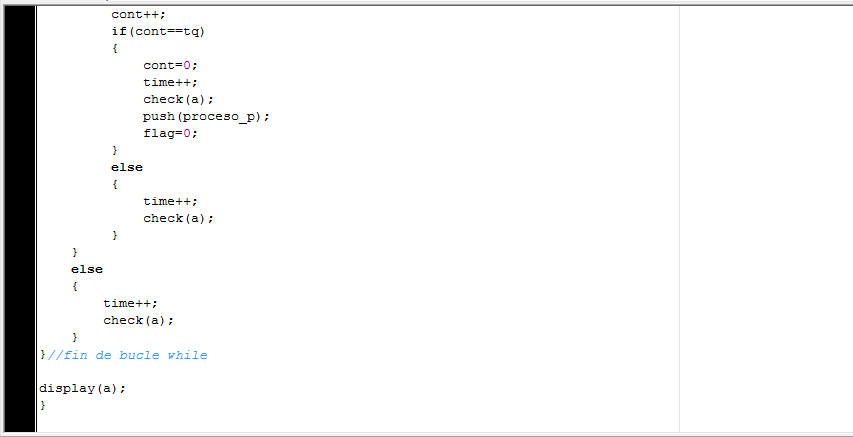
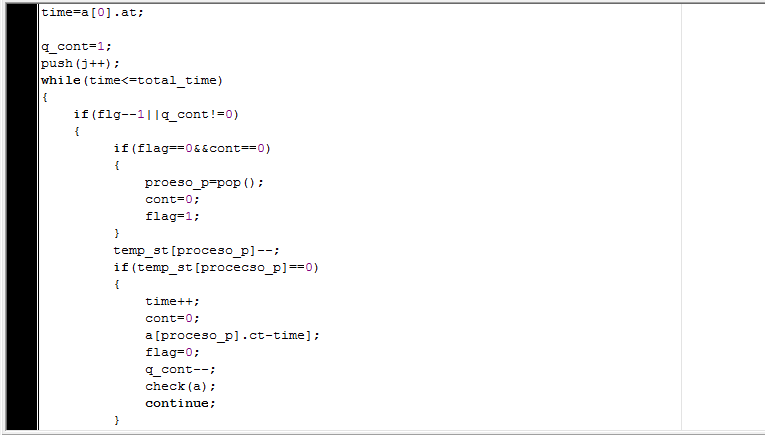
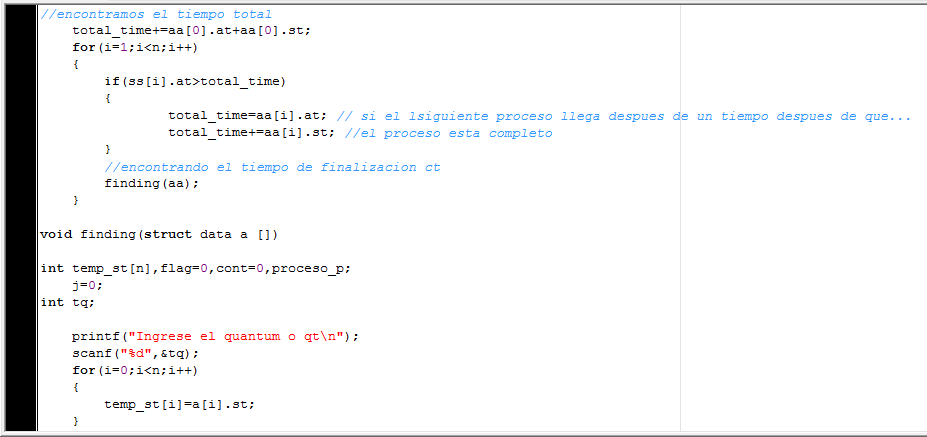
A continuación mostraremos un algoritmo en el lenguaje C, usaremos el siguiente ejemplo:

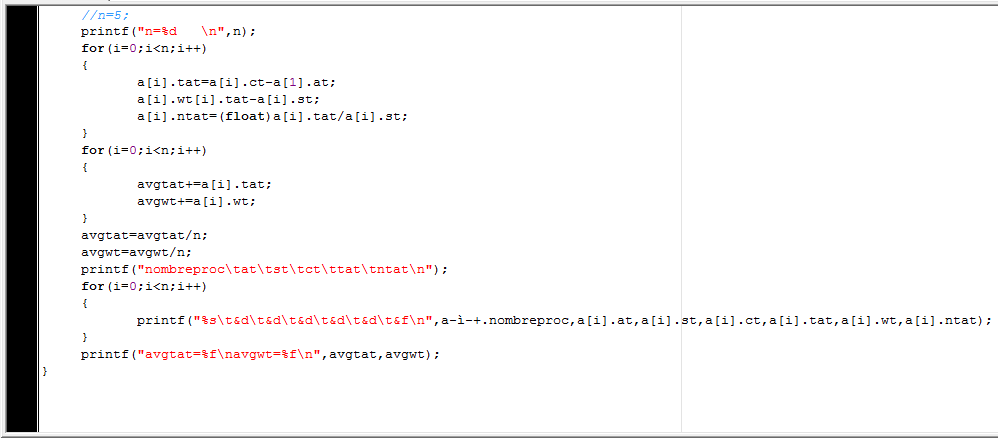
|  |  |  |
| --- | --- | --- |
| **PROCESO** | **TIEMPO DE LLEGADA (AT)** | **TIEMPO DE EJECUCION (BT)** |
| a | 0 | 4 |
| b | 1 | 5 |
| c | 2 | 2 |
| d | 3 | 1 |
| e | 4 | 6 |
| f | 6 | 3 |

**Desarrollo de Algoritmo en C++**

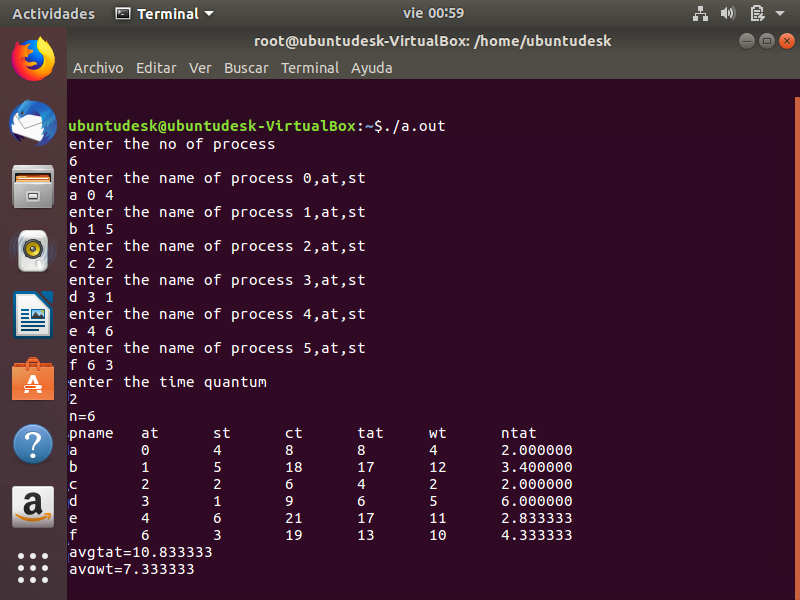
****

****

****

****

El resultado al compilarlo en la terminal de Ubuntu es el siguiente:



LA BARBERIA

En ciencias de la computación, el problema del barbero durmiente es un problema de sincronización. El problema consiste en una barbería en la que trabaja un barbero que tiene un único sillón de barbero y varias sillas para esperar. Cuando no hay clientes, el barbero se sienta en una silla y se duerme. Cuando llega un nuevo cliente, éste o bien despierta al barbero o —si el barbero está afeitando a otro cliente— se sienta en una silla (o se va si todas las sillas están ocupadas por clientes esperando). El problema consiste en realizar la actividad del barbero sin que ocurran condiciones de carrera. La solución implica el uso de semáforos y objetos de exclusión mutua para proteger la sección crítica.

Un semáforo es una variable protegida (o tipo abstracto de datos) que constituye el método clásico para restringir o permitir el acceso a recursos compartidos (por ejemplo, un recurso de almacenamiento) en un entorno de multiprocesamiento. Fueron inventados por Edsger Dijkstra y se usaron por primera vez en el sistema operativo THEOS.

En electrónica y en programación concurrente, se conoce como condición de carrera al error que se produce en programas o circuitos lógicos que no se han construido adecuadamente para su ejecución simultánea con otros procesos.

* El próximo pseudo-código garantiza la sincronización entre el barbero y el cliente, pero puede llevar a inanición del cliente. *wait()* y *signal()* son funciones provistas por el [semáforo](https://es.wikipedia.org/wiki/Sem%C3%A1foro_(programaci%C3%B3n)).
* Se necesita:

Semáforo barberoListo = 0 // (Mutex, sólo 1 o 0)

Semáforo sillasAccesibles = 1 // (Mutex) Cuando sea 1, el número de sillas libres puede aumentar o disminuir

Semáforo clientes = 0 // Número de clientes en la sala de espera

int sillasLibres = N // N es el número total de sillas

* Función barbero (Proceso/hilo-thread):

while(true) // Ciclo infinito

{

wait(clientes) // Espera la señal de un hilo cliente para despertar.

wait(sillasAccesibles) // (Ya está despierto) Espera señal para poder modificar sillasLibres.

sillasLibres += 1 // Aumenta en uno el número de sillas libres.

signal(barberoListo) // El barbero está listo para cortar y manda señal al hilo cliente.

signal(sillasAccesibles) // Manda señal para desbloquear el acceso a sillasLibres

// Aquí el barbero corta el pelo de un cliente (zona de código no crítico).

}

* Función cliente (Proceso/hilo-thread):

wait(sillasAccesibles) // Espera la señal para poder acceder a sillasLibres.

if (sillasLibres > 0) // Si hay alguna silla libre, se sienta en una.

{

sillasLibres -= 1 // Decrementando el valor de sillasLibres en 1.

signal(clientes) // Manda señal al barbero de que hay un cliente disponible.

signal(sillasAccesibles) // Manda señal para desbloquear el acceso a sillasLibres.

wait(barberoListo) // El cliente espera a que el barbero esté listo para atenderlo.

// Se le corta el pelo al cliente.

}

else // Si no hay sillas libres.

{

signal(sillasAccesibles) // Manda señal para desbloquear el acceso a sillasLibres.

// El cliente se va de la barbería y no manda la señal de cliente disponible.

}

***ANALISIS DEL PROBLEMA***

* Barbería con un solo Barbero
* Una silla de Barbero
* n sillas para los clientes

***CONDICIONES DEL PROBLEMA***

* Diferentes procesos (clientes) quieren acceder a la silla del barbero
* Diferentes clientes quieren entrar a la Barberia

***SOLUCION***

* Un mutex que se encarga de no dejar que un cliente tome la silla del babero, si esta se encuentra ocupada
* La otra variable cuenta el numero de clientes que esperan al barbero. Si hay mas de (n) no deja entrar a otro cliente a la Barberia, hasta que se desocupe una silla.

***DIAGRAMA DE FLUJO – EL BARBERO***



***DIAGRAMA DE FLUJO – LA BARBERIA***



BIBLIOGRAFIA

1. <https://www.ecured.cu/Planificaci%C3%B3n_de_procesos_en_un_sistema_operativo>
2. <https://es.wikipedia.org/wiki/Problema_del_barbero_durmiente>
3. <https://es.slideshare.net/18segundos/barbero-dormilon?next_slideshow=1>
4. <https://w3.ual.es/~acorral/DSO/Tema_2.pdf>
5. <http://exa.unne.edu.ar/depar/areas/informatica/SistemasOperativos/TFLauraGonzalez.pdf>
6. <http://ls.fi.upm.es/pc/Anteriores/Examenes/1993sep/node4.html>