Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ciencias y Sistemas
Sistemas Operativos 1
Carlos Javier Martínez Polanco 201709282

Hoja de Trabajo – CPU Scheduling

1. Explique cuál es la diferencia entre Scheduling Permisivo y No Permisivo.

   Scheduling (planificación) Permisivo y No Permisivo son dos enfoques diferentes en el diseño de sistemas operativos en tiempo real, especialmente en sistemas multitarea y multiprocesador. Estos enfoques determinan cómo se asignan y se gestionan los recursos del sistema, como la CPU, entre los procesos en ejecución.

   Scheduling Permisivo:

   En un sistema operativo con un enfoque de scheduling permisivo, el planificador del sistema permite que los procesos que han sido bloqueados por razones como espera de E/S (Entrada/Salida) o esperando un evento se mantengan en la cola de planificación y no se desalojen de inmediato. Es decir, el planificador permite que los procesos que no están listos para ejecutarse permanezcan en la cola de ejecución, incluso si la CPU está disponible. Este enfoque puede ayudar a minimizar el tiempo de conmutación (overhead) de contexto y puede ser beneficioso en ciertos casos donde los procesos bloqueados pueden desbloquearse rápidamente.

Scheduling No Permisivo:

Por otro lado, en un sistema operativo con un enfoque de scheduling no permisivo, el planificador del sistema tiende a desalojar inmediatamente los procesos que están bloqueados y no pueden ejecutarse, liberando así la CPU para otros procesos que pueden estar listos para ejecutarse. Esto significa que, en cuanto un proceso se bloquea por una operación de E/S u otro evento, el planificador busca inmediatamente otro proceso listo para ejecutarse. Este enfoque puede ayudar a maximizar la utilización de la CPU y reducir la latencia en la respuesta de los procesos que están listos para ejecutarse.

2. ¿Cuál de los siguientes algoritmos de Scheduling podría provocar un bloqueo indefinido? Explique su respuesta.
   a. First-come, first-served
   b. Shortest job first
   c. Round robin
   d. Priority

El algoritmo de prioridad asigna la CPU al proceso con la prioridad más alta. Siempre que haya un proceso en ejecución con la máxima prioridad, otros procesos de menor prioridad pueden quedar bloqueados indefinidamente si continúan llegando procesos de mayor prioridad.

Por ejemplo, considera un sistema donde un proceso A con alta prioridad está en ejecución y siempre hay nuevos procesos llegando con prioridades más altas que el proceso B que está esperando para ser ejecutado. En este escenario, el proceso B nunca recibirá la CPU porque los procesos de mayor prioridad siempre la reclamarán primero, lo que resultará en un bloqueo indefinido para el proceso B.

Por otro lado, en los algoritmos como el First-come, first-served, Shortest job first y Round robin, todos los procesos eventualmente tendrán la oportunidad de ejecutarse, aunque puedan experimentar latencias o retrasos dependiendo de la situación de planificación. Sin embargo, en el algoritmo de prioridad, los procesos con prioridad más baja pueden quedar bloqueados indefinidamente si siempre hay procesos de mayor prioridad disponibles para la ejecución.

3. De estos dos tipos de programas:
   a. I/O-bound (un programa que tiene más I/Os que uso de CPU)
   b. CPU-bound (un programa que tiene más uso de CPU que I/Os)

   ¿Cuál tiene más probabilidades de tener cambios de contexto voluntarios y cuál tiene más probabilidades de tener cambios de contexto no voluntarios? Explica tu respuesta.

   I/O-bound:

   Estos programas realizan una gran cantidad de operaciones de entrada/salida (I/O). Por ejemplo, programas que leen y escriben archivos, realizan operaciones de red, etc. Estas operaciones de E/S pueden llevar más tiempo en completarse en comparación con el tiempo que requieren para la CPU. Cuando un programa I/O-bound realiza una operación de E/S, normalmente se bloquea y cede voluntariamente la CPU, ya que no puede continuar hasta que se complete la operación de E/S. Por lo tanto, los cambios de contexto en programas I/O-bound son más propensos a ser voluntarios, ya que el programa activamente decide ceder el control de la CPU mientras espera que se complete la operación de E/S.

CPU-bound:

Estos programas realizan principalmente operaciones que consumen CPU intensivamente, como cálculos complejos, procesamiento de datos grandes, etc. Estos programas tienden a mantener la CPU ocupada durante largos períodos de tiempo sin necesidad de realizar operaciones de E/S significativas. Como resultado, cuando un programa CPU-bound necesita ceder la CPU, normalmente es debido a un evento externo, como la expiración de un temporizador, la llegada de una interrupción, o el planificador del sistema decidiendo asignar el tiempo de CPU a otro proceso. Estos cambios de contexto son no voluntarios, ya que el programa no decide activamente ceder la CPU, sino que se ve obligado a hacerlo por el sistema operativo.

4. Utilizando un sistema Linux, escriba un programa en C que cree un proceso hijo (fork) que finalmente se convierta en un proceso zombie. Este proceso zombie debe permanecer en el sistema durante al menos 10 segundos.

   Los estados del proceso se pueden obtener del comando: ps -l

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
int main() {
  pid_t pid = fork();
  if (pid < 0) {
    // Error al crear el proceso hijo
    perror("fork");
    exit(EXIT_FAILURE);
  } else if (pid == 0) {
    // Este es el proceso hijo
    printf("Proceso hijo creado con PID: %d\n", getpid());
    exit(EXIT_SUCCESS);
  } else {
    // Este es el proceso padre
    printf("Proceso padre creado con PID: %d\n", getpid());
    // Espera durante al menos 10 segundos
    sleep(10);
    // Comprobamos el estado del proceso hijo
    system("ps -l");
    wait(NULL);
  }
  return 0;
}
```