Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ciencias y Sistemas
Sistemas Operativos 1
201709051

Hoja de Trabajo – CPU Scheduling

1. Explique cuál es la diferencia entre Scheduling Permisivo y No Permisivo.
   Scheduling Permisivo: Permite que un proceso siga ejecutándose en la CPU aunque esté esperando una operación de entrada/salida (E/S). El proceso se retira solo cuando es necesario para dar paso a otro proceso.
   Scheduling No Permisivo: Retira un proceso de la CPU tan pronto como se bloquea esperando una operación de E/S. El proceso se suspende hasta que la operación de E/S se complete, liberando la CPU para otros procesos.
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 Scheduling que podría provocar un bloqueo indefinido es Round Robin. Esto ocurre cuando un proceso se bloquea esperando una operación de E/S y los siguientes procesos en la cola tienen ráfagas de CPU muy largas, impidiendo que el proceso bloqueado vuelva a ejecutarse, lo que lleva a un bloqueo indefinido.

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: Tiene más probabilidades de experimentar cambios de contexto voluntarios debido a períodos de espera durante operaciones de E/S.

CPU-bound: Tiene más probabilidades de experimentar cambios de contexto no voluntarios debido a su uso intensivo de la CPU y la posible competencia con procesos de mayor prioridad.

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

```
hbmg@hbmg-pc:~/Documentos/Sopes1/Magistral/SO1_Actividades_201709051/Actividad6$ ps -l
F S   UID     PID    PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  1000   11089   10392  0  80   0 -  4441 do_wai pts/2    00:00:00 bash
1 S  1000   11295    1125  0  80   0 -   694 hrtime pts/2    00:00:00 zombie
4 R  1000   11297   11089  0  80   0 -  3844 -      pts/2    00:00:00 ps
hbmg@hbmg-pc:~/Documentos/Sopes1/Magistral/SO1_Actividades_201709051/Actividad6$
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>

int main() {
    pid_t pid = fork();

    if (pid < 0) {
        // Error al crear el proceso hijo
        perror("Error al crear el proceso hijo");
        exit(EXIT_FAILURE);
    } else if (pid == 0) {
        // Código ejecutado por el proceso hijo
        printf("Proceso hijo creado. PID: %d\n", getpid());
        sleep(10); // Espera 10 segundos después de que el padre termine
        printf("Proceso hijo terminado.\n");
        exit(EXIT_SUCCESS);
    }

    // Código ejecutado por el proceso padre
    printf("Proceso padre creado. PID del hijo: %d\n", pid);
    printf("Proceso padre terminando...\n");
    exit(EXIT_SUCCESS);
}
```