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

Hoja de Trabajo – CPU Scheduling

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

El Scheduling Permisivo permite que los procesos se ejecuten en cualquier momento, sin restricciones temporales, promoviendo la equidad al otorgar a todos los procesos la misma oportunidad de ejecución, aunque esto puede llevar a ineficiencias al permitir que procesos de baja prioridad bloqueen a los de alta prioridad. Ejemplos comunes incluyen Round Robin y First Come First Served (FCFS). Por otro lado, el Scheduling No Permisivo impone restricciones temporales a la ejecución de los procesos, priorizando la eficiencia al otorgar más tiempo de ejecución a los procesos de alta prioridad. Aunque potencialmente más eficiente que el permisivo, puede carecer de equidad. Ejemplos destacados son Shortest Job First (SJF) y Priority Scheduling.

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

Si bien una buena parte de la responsabilidad de ese “bloqueo” depende del tipo de tarea que es programada, existe una mayor probabilidad de ocurrencia con el algoritmo FCFS , puesto asigna el CPU al primer proceso que llega a la cola de ready. Este proceso puede ejecutarse durante un tiempo indefinido, sin que ningún otro proceso pueda ejecutarse, incluso si tienen mayor prioridad o son más cortos.

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.

   Los programas I/O-bound, que hacen un uso intensivo de operaciones de entrada/salida, suelen experimentar cambios de contexto voluntarios, ya que frecuentemente ceden la CPU mientras esperan la finalización de estas operaciones. Por otro lado, los programas CPU-bound, que tienen mayor demanda de CPU y menos operaciones de E/S, tienden a experimentar cambios de contexto no voluntarios, ya que el sistema operativo interviene para garantizar una distribución justa de recursos, decidiendo cuándo cambiar al siguiente proceso en la cola de planificación.

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

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

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

   if (child_pid < 0) {
      perror("fork");
      exit(EXIT_FAILURE);
   } else if (child_pid == 0) {
      printf("Proceso hijo, PID: %d\n", getpid());
      exit(EXIT_SUCCESS);
   } else {
      printf("Proceso padre, PID: %d\n", getpid());
      sleep(10);
   }

   return 0;
}
```