Nombre: Steven Facundo Mejía Xolop
Carnet: 202104160
Curso: Sistemas Operativos
Sección: “A”

Hoja de Trabajo – CPU Scheduling

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

- **Scheduling Permisivo:**

  El scheduling permisivo se refiere a un enfoque de planificación de procesos que permite que los procesos utilicen la CPU durante el tiempo que lo deseen, sin imponer restricciones de tiempo. Bajo este enfoque, un proceso puede retener la CPU incluso si hay otros procesos en espera. Esto puede llevar a situaciones en las que un proceso monopolice la CPU, causando retrasos en la ejecución de otros procesos y una respuesta lenta del sistema en general.

- **Scheduling No Permisivo:**

  El scheduling no permisivo, por otro lado, implica la imposición de límites de tiempo en la asignación de CPU a los procesos. Bajo este enfoque, se establecen límites de tiempo para que cada proceso pueda utilizar la CPU, lo que garantiza que ningún proceso monopolice la CPU indefinidamente. Si un proceso excede su límite de tiempo asignado, se suspende y se da paso a otro proceso en espera. Esto ayuda a mantener una distribución equitativa del tiempo de CPU entre todos los procesos en ejecución y garantiza una respuesta más predecible del sistema.

2. ¿Cuál de los siguientes algoritmos de Scheduling podría provocar un bloqueo indefinido? Explique su respuesta.

- **Respuesta: a. First-come, first-served (FCFS)**

  El algoritmo FCFS es un algoritmo de planificación de procesos que asigna la CPU al primer proceso que llega y no la libera hasta que haya terminado su ejecución. Esto significa que si un proceso largo y pesado llega primero, todos los procesos que lleguen después tendrán que esperar hasta que el proceso inicial haya completado su ejecución, lo que puede causar un bloqueo indefinido para los procesos posteriores en la cola de espera.

3. De estos dos tipos de programas:

¿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 (programa que tiene más I/Os que uso de CPU)**

  Los programas I/O-bound tienen más probabilidades de tener cambios de contexto voluntarios. Esto se debe a que estos programas pasan la mayor parte de su tiempo esperando la finalización de operaciones de entrada/salida (I/O), durante las cuales el proceso puede optar por liberar voluntariamente la CPU, permitiendo que otros procesos la utilicen mientras tanto. Los cambios de contexto voluntarios son comunes en programas I/O-bound porque el proceso puede optar por liberar la CPU durante períodos de inactividad.

- **CPU-bound (programa que tiene más uso de CPU que I/Os):**

  Los programas CPU-bound tienen más probabilidades de experimentar cambios de contexto no voluntarios. Estos programas consumen una gran cantidad de tiempo de CPU y, a menudo, no tienen períodos de inactividad en los que puedan liberar voluntariamente la CPU. Por lo tanto, el sistema operativo puede verse obligado a realizar cambios de contexto no voluntarios para garantizar que otros procesos reciban una oportunidad justa de utilizar la CPU. Esto puede ocurrir cuando el planificador de procesos decide suspender el proceso actual y pasar a otro proceso en espera.

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 <sys/types.h>
#include <unistd.h>
#include <sys/wait.h>

int main() {
    pid_t pid = fork(); // Crear proceso hijo

    if (pid < 0) {
        fprintf(stderr, "Error al crear el proceso hijo\n");
        exit(EXIT_FAILURE);
    } else if (pid == 0) {
        // Código del proceso hijo
        printf("Proceso hijo ejecutándose...\n");
        exit(EXIT_SUCCESS); // Salir del proceso hijo
    } else {
        // Código del proceso padre
        sleep(10); // Esperar 10 segundos para permitir que el proceso hijo se convierta en zombie
        // No se llama a wait() aquí para demostrar que el proceso hijo se convierte en zombie
    }

    return 0;
}
```