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.**

R/ En el contexto de la planificación de CPU, el enfoque permisivo permite que un proceso se ejecute sin interrupciones hasta que voluntariamente libere la CPU, lo que puede conducir a tiempos de respuesta lentos para otros procesos. En contraste, el enfoque no permisivo fuerza a los procesos a ceder el control de la CPU después de un período de tiempo predefinido, incluso si el proceso aún tiene trabajo pendiente, lo que garantiza una distribución más equitativa del tiempo de CPU entre los procesos y evita que un proceso monopolice la CPU durante largos períodos.

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**

   R/ El algoritmo de planificación que podría provocar un bloqueo indefinido es:
   b. Shortest job first (SJF)
   El algoritmo SJF prioriza la ejecución de procesos más cortos primero. Si hay un proceso muy largo (llamado proceso "invasor"), que llega antes que otros procesos más cortos, esos procesos más cortos tendrán que esperar indefinidamente hasta que se complete el proceso largo. Esto puede provocar un bloqueo indefinido para los procesos más cortos, ya que nunca tendrán la oportunidad de ejecutarse hasta que el proceso largo haya terminado, lo que puede no ocurrir si el proceso largo continúa generando trabajo. En situaciones extremas, este comportamiento podría conducir a un bloqueo del sistema.

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.

   R/ Un programa I/O-bound tiene más probabilidades de experimentar cambios de contexto voluntarios, ya que tiende a realizar muchas operaciones de entrada/salida (I/O), durante las cuales puede liberar la CPU voluntariamente mientras espera la finalización de la operación I/O. Por otro lado, un programa CPU-bound es más propenso a experimentar cambios de contexto no voluntarios, ya que requiere un uso continuo de la CPU y no libera la CPU voluntariamente, lo que puede llevar al sistema operativo a programar cambios de contexto para permitir que otros procesos tengan la oportunidad de ejecutarse.

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>
#include <sys/wait.h>

int main() {
    pid_t child_pid;

    // Crear proceso hijo
    child_pid = fork();

    if (child_pid < 0) {
        // Error al crear el proceso hijo
        perror("fork");
        exit(EXIT_FAILURE);
    } else if (child_pid == 0) {
        // Proceso hijo
        printf("Proceso hijo en ejecución.\n");
        exit(EXIT_SUCCESS);
    } else {
        // Proceso padre
        // Esperar un tiempo para que el proceso hijo se convierta en zombie
        sleep(10);

        // Comprobar el estado del proceso hijo
        printf("Estado del proceso hijo después de 10 segundos:\n");
        system("ps -l");

        // Esperar a que el proceso hijo termine (debería ser un zombie)
        wait(NULL);

        printf("Proceso padre terminando.\n");
    }

    return 0;
}f, seed, [])
}
```

```
Proceso hijo en ejecución.
Estado del proceso hijo después de 10 segundos:
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  1000  1234  1233  0  80   0 -  1247 wait   pts/0    00:00:00 programa
0 Z  1000  1235  1234  0  80   0 -     0 exit   pts/0    00:00:00 [programa] <defunct>
Proceso padre terminando.// Esperar a que el proceso hijo termine (debería ser un zombie)
        wait(NULL);

        printf("Proceso padre terminando.\n");
    }

    return 0;
}f, seed, [])
}
```