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

Hoja de Trabajo – CPU Scheduling

Christian Alessander Blanco González - 202000173

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

   El scheduling no apropiativo, también llamado permisivo, permite que un proceso en ejecución retenga la CPU hasta que decida voluntariamente cederla o terminar. Un proceso acaparando la CPU indefinidamente puede llevar a causar inanición a 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

   Priority podría causar un bloqueo sin límite de tiempo. Si uno o más procesos de alta prioridad no terminan nunca o liberar la CPU, los procesos de baja prioridad pueden quedar bloqueados indefinidamente sin tener oportunidad para ejecutarse.

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 a menudo tienen más cambios de contexto voluntarios debido a sus frecuentes operaciones de entrada/salida, mientras que los programas CPU-

bound suelen experimentar más cambios de contexto no voluntarios debido al uso intenso de la CPU y la necesidad del sistema operativo para distribuir equitativamente el tiempo entre procesos.

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

int main() {
  pid_t pid;

  pid = fork(); // Crear un 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("Soy el proceso hijo con PID: %d\n", getpid());
    exit(EXIT_SUCCESS); // El proceso hijo termina
  } else {
    // Código del proceso padre
    printf("Soy el proceso padre con PID: %d\n", getpid());
    sleep(10); // Esperar 10 segundos
```

```
        printf("Esperando al proceso hijo...\n");
        waitpid(pid, NULL, 0); // Esperar al proceso hijo (sin recoger el estado)
        printf("Proceso hijo terminado\n");
    }
    return 0;
}
```