Tecnológico de Costa Rica

Escuela de Ingeniería en Computación
Principios de Sistemas Operativos

05 de setiembre de 2017

Samantha Arburola León
2013101697

Operating System Concepts

Chapter 6 - Practice Exercises

6.1 A CPU-scheduling algorithm determines an order for the execution of its scheduled processes. Given n processes to be scheduled on one processor, how many different schedules are possible? Give a formula in terms of n.

$n!$ **i.e. factorial n**

6.2 Explain the difference between preemptive and nonpreemptive scheduling

The first one allows a process to be interrupted on its execution, taking the CPU away and assigning it to other process.

Nonpreemptive scheduling ensures a process leave the CPU only when it finishes with its current CPU burst.

6.3 Suppose that the following processes arrive for execution at the times indicated. Each process will run for the amount of time listed. In answering the questions, use nonpreemptive scheduling, and base all decisions on the information you have at the time the decision must be made

Process	Arrival Time	Burst Time
P_1	0.0	8
P_2	0.4	4
P_3	1.0	1

a. What is the average turnaround time for these processes with the FCFS scheduling algorithm?

   10.53

b. What is the average turnaround time for these processes with the SJF scheduling algorithm?

   9.53

c. The SJF algorithm is supposed to improve performance, but notice that we chose to run process P_1 at time 0 because we did not know that two shorter processes would arrive

soon. Compute what the average turnaround time will be if the CPU is left idle for the first 1 unit and then SJF scheduling is used. Remember that processes P_1 and P_2 are waiting during this idle time, so their waiting time may increase. This algorithm could be called future-knowledge scheduling

6.86

Turnaround time = finishing time - arrival time and FCFS = FCFS - arrival time

6.4 What advantage is there in having different time-quantum sizes at different levels of a multilevel queueing system?

Processes that need more frequent servicing can be in a queue with a small time quantum

Processes with no need for frequent servicing can be in a queue with a larger quantum thus making more efficient use of the computer.

6.5 Many CPU-scheduling algorithms are parameterized. For example, the RR algorithm requires a parameter to indicate the time slice. Multilevel feedback queues require parameters to define the number of queues, the scheduling algorithm for each queue, the criteria used to move processes between queues, and so on. These algorithms are thus really sets of algorithms (for example, the set of RR algorithms for all time slices, and so on). One set of algorithms may include another (for example, the FCFS algorithm is the RR algorithm with an infinite time quantum). What (if any) relation holds between the following pairs of algorithm sets?

a. Priority and SJF

Shortest job has the highest priority

b. Multilevel feedback queues and FCFS

Lowest level of MLFQ is FCFS

c. Priority and FCFS

FCFS set the highest priority to the job having been in existence

d. RR and SJF

None

6.6 Suppose that a scheduling algorithm (at the level of short-term CPU scheduling) favors those processes that have used the least processor time in the recent past. Why will this algorithm favor I/O-bound programs and yet not permanently starve CPU-bound programs?

- I/O-bound programs would not require much CPU usage, having short CPU bursts.
- CPU-bound programs require large CPU bursts and do not have to worry about starvation because I/O-bound programs finish running quickly allowing CPU-bound programs to use the CPU often.

6.7 Distinguish between PCS and SCS scheduling

Process contention scope scheduling is performed local to each process.
System contention scope scheduling is performed on the operating system with kernel threads.
* On systems that use many-to-one or many-to-many, PCS and SCS scheduling is different.

6.8 Assume that an operating system maps user-level threads to the kernel using the many-to-many model and that the mapping is done through the use of LWPs. Furthermore, the system allows program developers to create real-time threads. Is it necessary to bind a real-time thread to an LWP?

El tiempo es crucial para las aplicaciones en tiempo real.
Si un hilo está marcado como tiempo real pero no está enlazado a un LWP, el hilo puede tener que esperar para ser conectado a un LWP antes de ejecutarse.

Hay que tener en cuenta si se ejecuta un subproceso en tiempo real y luego procede a bloquear

Mientras se bloquea el subproceso en tiempo real, el LWP al que se ha conectado se ha asignado a otro subproceso. Cuando el hilo en tiempo real ha sido programado para ejecutarse de nuevo, primero debe esperar para ser conectado a un LWP

Al vincular un LWP a un hilo en tiempo real, se está asegurando que el hilo pueda ejecutarse con un retardo mínimo una vez que esté programado.

6.9 The traditional UNIX scheduler enforces an inverse relationship between priority numbers and priorities: the higher the number, the lower the priority. The scheduler recalculates process priorities once per second using the following function: Priority = (recent CPU usage / 2) + base where base = 60 and recent CPU usage refers to a value indicating how often a process has used the CPU since priorities were last recalculated. Assume that recent CPU usage is 40 for process P1, 18 for process P2, and 10 for process P3. What will be the new priorities for these three processes when priorities are recalculated? Based on this information, does the traditional UNIX scheduler raise or lower the relative priority of a CPU-bound process?

Priority = (recent CPU usage / 2) + base where base = 60
P1 = (40/2) + 60 = 80
P2 = (18/2) + 60 = 69
P3 = (10/2) + 60 = 65

Disminuye la prioridad relativa de los procesos vinculados a la CPU.