

Operating Systems exercises – week 4 – Selected answers.

1) Give two advantages threads have over multiple processes? What major disadvantage do they have? Suggest one application that would benefit from the use of threads, and one that would not.

Answer:

Threads are very inexpensive to create and destroy, and they use very little re-sources while they exist. They do use CPU time for instance, but they don't have totally separate memory spaces. Switching time between threads is less than between processes. Unfortunately, threads must "trust" each other to not damage shared data. For instance, one thread could destroy data that all the other threads rely on, while the same could not happen between processes unless they used a system feature to allow them to share data.

Any program that may do more than one task at once could benefit from multitasking. However, a real speed up benefit is only obtained if kernel supported threads are used. A server process that handles many similar requests would benefit from multi-threading. "Single-minded" processes would not benefit from multiple threads; for instance, a program that displays the time of day.

2) What resources are used when a thread is created? How do they differ from those used when a process is created?

Answer:

A context must be created, including a register set storage location for storage during context switching, and a local stack to record the procedure call arguments, return values, and return addresses, and thread-local storage. A process creation results in memory being allocated for program instructions and data, as well as thread-like storage.

3) What are the differences between user-level threads and kernel-supported threads? Under what circumstances is one type "better" than the other?

Answer:

User-level threads have no kernel support, so they are very inexpensive to create, destroy, and switch among. However, if one blocks, the whole process blocks. Kernel-supported threads are more expensive because system calls are needed to create and destroy them and the kernel must schedule them. They are more powerful because they are independently scheduled and block individually.

If a process with multiple threads involves a lot of I/O then kernel supported threads would be better - it avoids all the threads being blocked when any one of the threads makes a system call for I/O. If a process with multiple threads has little I/O then user-level threads which are more efficient to switch between would be better.

4) Define the difference between preemptive and nonpreemptive scheduling. State why strict nonpreemptive scheduling is unlikely to be used in an environment where interactive computing was important.

Answer:

Preemptive scheduling allows a process to be interrupted in the midst of its execution,

taking the CPU away and allocating it to another process. Nonpreemptive scheduling ensures that a process relinquishes control of the CPU only when it finishes with its current CPU burst.

5) What is a CPU burst? An I/O burst?

Answer:

- CPU burst: length of time required by a process to use the CPU to execute its instructions before it makes a system call.
- I/O burst: a time interval when a process is waiting for I/O to complete.

6) Consider the following set of processes, with the length of the CPU-burst time given in milliseconds:

Process	Burst Time	Priority
P1	5	2
P2	1	1
P3	3	3
P4	1	4
P5	7	3

At time 0, the processes are queued in the order of arrival P1, P2, P3, P4, P5.

a) Draw four Gantt charts illustrating the execution of these processes using

i) FCFS

ii) SJF (nonpreemptive)

iii) nonpreemptive Priority (with a smaller priority number implying a higher priority)

iv) RR (quantum = 1) scheduling

b) What is the turnaround time of each process for each of the scheduling algorithms in part a?

c) What is the waiting time of each process for each of the scheduling algorithms in part a?

d) Which of the schedules in part a results in the minimal average waiting time (over all processes)?

No answer – question and answer to be included in portfolio that is to be handed in.

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

Answer:

It will favor the I/O-bound programs because of the relatively short CPU burst request by them; however, the CPU-bound programs will not starve because the I/O-bound programs will relinquish the CPU relatively often to do their I/O.

8) Explain how interleaving of the execution of processes can corrupt shared data values.

The following are common scheduling criteria:

- a) I/O boundedness of a process
- b) processor boundedness of a process
- c) whether the process is batch or interactive
- d) urgency of a fast response
- e) process priority
- f) frequency of a process being pre-empted by higher priority process
- g) priorities of processes waiting for resources held by other processes
- h) accumulated waiting time
- i) accumulated execution time
- j) estimated run-time to completion

For each of the following, indicate which of the preceding scheduling criteria is most appropriate

- i) In a real-time spacecraft monitoring system, the computer must respond immediately to signals received from the spacecraft
- ii) Although a process has been receiving occasional service, it is making only nominal progress.
- iii) How often does the process voluntarily give up the processor for I/O before its quantum has expired.
- iv) Is the user present and expecting fast interactive response times, or is the user absent.
- v) One goal of processor scheduling is to minimize average waiting times
- vi) Processes holding resources in demand by other processes should have higher priorities.
- vii) Nearly complete processes should have higher priorities.

No answer – question and answer to be included in portfolio that is to be handed in.

9) Determining the quantum is a complex and critical task. Assume that the average context-switching time between processes is s , and the average amount of time an I/O bound processes uses before generating an I/O request is t ($t \gg s$). Discuss the effect of each of the following quantum settings, q .

- a) q is slightly greater than zero
- b) $q = s$
- c) $s < q < t$
- d) $q = t$
- e) $q > t$
- f) q is an extremely large number

a) q is slightly greater than zero

Processes make effectively no progress, executing 1 instruction per quantum and CPU spends almost all its time context switching. Computer comes to a effective stop.

b) $q = s$

Almost as bad as a), CPU spends half of its time context switching (if we include scheduling/dispatching in with context switch) and processes only execute a relatively small number of instructions per quantum.

c) $s < q < t$

This provides steadily better improvement in efficiency as less time is spent in context switches, but still CPU burst of I/O bound jobs will require several time quanta and context switch overheads.

d) $q=t$

This provides a reasonable chance for I/O bound jobs to complete CPU burst before quantum expires.

e) $q > t$

Increasingly processes complete with time quantum allocated, including longer CPU bound jobs. There results a deterioration in responsiveness to interactive users as waiting time increase.

f) q is an extremely large number

Round robin with very long q are effectively equivalent to First Come First Served.

10) Discuss the effect of each of the following methods of assigning q .

- a) q fixed and identical for all users
- b) q fixed and unique to each process
- c) q variable and identical for all processes
- d) q variable and unique to each process

i) arrange the schemes above in order from lowest to highest runtime overhead

ii) arrange the schemes in order from least to most responsive to variations in individual processes and system load

iii) Relate your answers in i) and ii) to one another.

Answer:

i) d, b, c, a

ii) a, c, b, d

iii) Inverse relationship.

11) In single-user dedicated systems, it is generally obvious when a program goes into an infinite loop. But in multiuser systems running large numbers of processes. It cannot easily determined that an individual process is not progressing.

a) Can the operating system determine that a process in an infinite loop?

b) What reasonable safeguards might be built into operating system to prevent processes in infinite loops from running indefinitely.

Answer:

a) No. There is no general way in which an OS can determine whether a program is in an infinite loop. There is a general result in theoretical computer science (Halting problem) which effectively means that in general it is impossible to determine whether the result from a program is undecided i.e. the algorithm will never return a result.

b) Assuming a pre-emptive scheduling strategy - you could allocate a maximum amount of CPU time to a process. CPU time usage is normally recorded by an OS, so the only overhead would be that the Scheduler would have to check to see if the maximum CPU time was exceeded and terminate the process if it was.

12) The following are common scheduling objectives.

a) to be fair

b) to maximize throughput

c) to maximize the number of interactive users receiving acceptable response times

- d) to be predictable
- e) to minimize overhead
- f) to balance resource utilisation
- g) to achieve a balance between response and utilisation
- h) to avoid indefinite postponement
- i) to obey priorities
- j) to give preference to processes that hold key resources
- k) to give lower grade of service to high overhead processes
- l) to degrade gracefully under heavy loads

Which of the preceding objectives most directly applies to each of the following?

- i) If a user has been waiting for an excessive amount of time.
- ii) The user who runs a payroll job for a 100 employee company expects the job to take about the same amount of time each week.
- iii) The system should admit processes to create a mix that will keep most devices busy.
- iv) The system should favour important processes.
- v) An important process arrives but cannot proceed because an unimportant process is holding the resources the important process needs
- vi) During peak periods, the system should not collapse from the overhead it takes to manage a large number of processes
- vii) The system should favour I/O bound processes
- viii) Context switches should execute as quickly as possible

Answer:

- i) h
- ii) d
- iii) f
- iv) i
- v) j
- vi) l
- vii) c
- viii) e