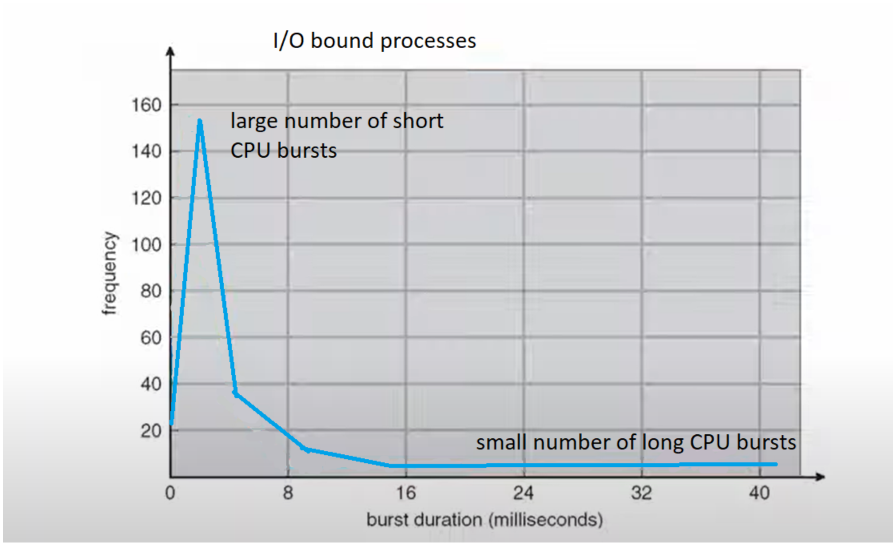
CmpE 322

January 29, 2022

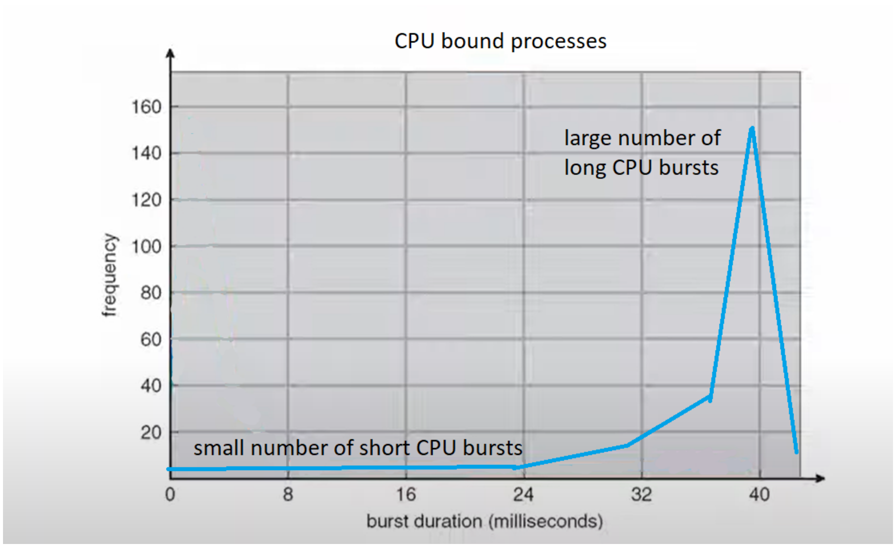
Contents

1 Discussion VI

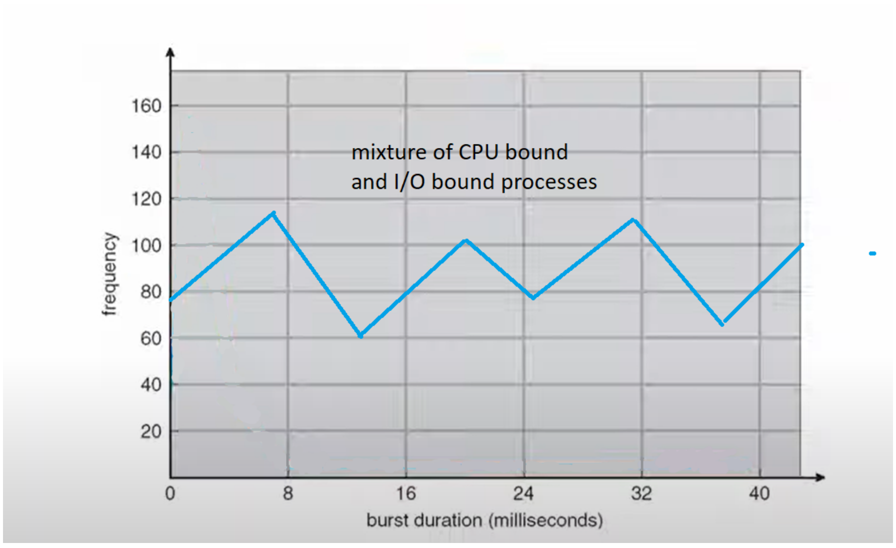
1) (a) All processes are interactive:



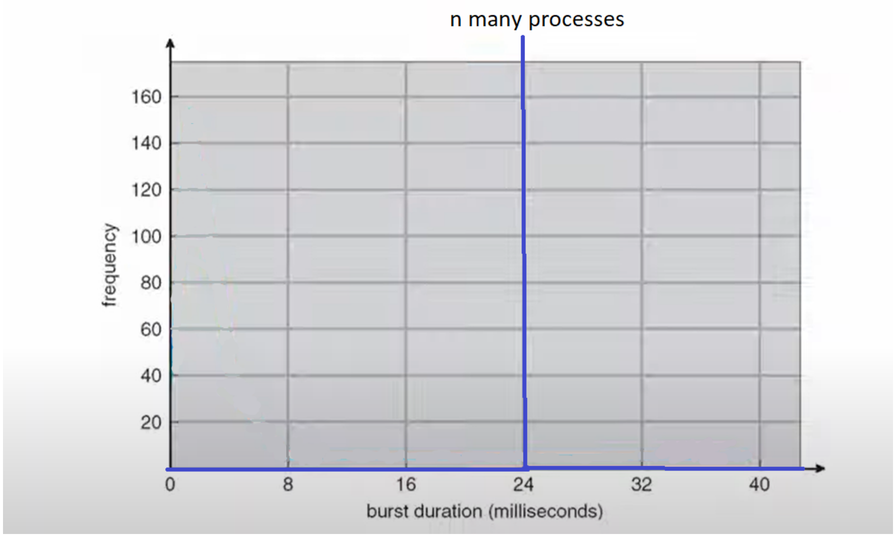
(b) All process are CPU-heavy with very few I/O operations:



(c) You have a mixture of the two types above:



(d) All processes have exactly the same burst length of 24 ms:



2) (a) Can we identify which job is the shortest?

Answer: For short-term scheduling, there is no way to know the length of the next CPU burst.

(b) If we can, how?

Answer: -

(c) If we cannot, what can we do about it for scheduling?

Answer: One approach to this problem is to try to approximate shortest-job-first scheduling. We may not know the length of the next CPU burst, but we may be able to predict its value. We expect that the next CPU burst will be similar in length to the previous ones. By computing an approximation of the length of the next CPU burst, we can pick the process with the shortest predicted CPU burst. The next CPU burst is generally predicted as an **exponential average** of the measured lengths of previous CPU bursts. Formula for the next CPU burst time estimation:

$$\tau_{n+1} = \alpha t_n + (1 - \alpha)\tau_n$$

3) (a) What is the difference between a preemptive vs. a nonpreemptive scheduler? State the pros and cons.

Answer: Under **nonpreemptive** scheduling, once the CPU has been allocated to a process, the process keeps the CPU until it releases the CPU either by terminating or by switching to the waiting state voluntarily, say for I/O operation. In **preemptive** scheduling, a process that is in the running state can be removed from the CPU by force. Switch from running to ready state when an interrupt occurs and switch from waiting to ready state when a process completes its I/O operation.

The executing process in preemptive scheduling is **interrupted** in the middle of execution whereas, the executing process in non-preemptive scheduling is not interrupted in the middle of execution.

Preemptive scheduling has the **overhead** of switching the process from running state to ready state. Whereas non-preemptive scheduling has no overhead of switching the process from running state to ready state.

In preemptive scheduling, if a process with high priority arrives in the ready queue, then the process in the running state can be interrupted. Non-preemptive scheduling doesn't provide such flexibility. Critical processes are allowed to access CPU as they arrive into the ready queue, no matter what process is executing currently.

CPU utilization is more efficient in preemptive scheduling ocmpared to non-preemptive scheduling.

Preemptive scheduling improves the average response time. On the other hand, non-preemptive scheduling offers low scheduling overhead. It tends to offer high throughput since we wait for a process to terminate, it is conceptually a very simple method and less computational resources are needed for scheduling.

(b) What is a race condition? Why can a preemptive scheduler cause a race condition? Is it good or bad? Do we want to promote race conditions or avoid them?

Answer: A situation where several processes access and manipulate the same data concurrently and the outcome of the execution depends on the particular order in which the access takes place, is called a **race condition**. (p.g. 207 - kernel race condition)

To guard against the race condition above, we need to ensure that only one process at a time can be manipulating the shared data.

In preemptive scheduling, one process can be preempted all of a sudden while it is updating the data, in order to let second process run. If the second process then tries to read the data, it may lead to a race condition.

4) (a) What is the difference between the scheduler and the dispatcher?

Answer: The dispatcher is the module that gives control of the CPU to the process selected by the short-term scheduler (scheduler). Switching context, switching to user mode, jumping to the proper location in the user program to restart the program are some of the responsibilities of the dispatcher.

Scheduler makes the decision, which process CPU should be allocated to, and the dispatcher applies that decision via context switching. It is the duty of the dispatcher to dispatch or transfer a process from one state to another.

Scheduler works independently, while the dispatcher has to be dependent on the scheduler.

If we want to switch from process P1 to P2, we should remember the values of the registers while P1 is being executed. When we return back to P1, we should continue with the previous register values. Dispatcher should save the values of the registers like stack pointer, program counter, instruction register etc. These values are stored in PCB and storing action is performed by the dispatcher.

(b) Two of your classmates, Ali and Ayşe, make interesting claims:

• Ali says he developed a scheduler that increases waiting time but reduces turnaround time.

• Ayşe says she developed a scheduler that increases waiting time but reduces response time.

Which one would you believe? Ali, Ayşe, both, or none?

Answer: I shouldn't believe Ali. Because reducing turnaround time and increasing waiting time means reducing the time spent in CPU and time spent for I/O operations. However we cannot modify the amount of time spent for such actions since they depend on the processes, not on the scheduling algorithm.

Ayşe may have developed such a scheduler. Furthermore, that scheduler can be using an algorithm similar to Round Robin Scheduling algorithm. With a small time quantum, it may give a low response time for each process but on the other hand it increases the waiting time of the processes, sum of the periods spent by a process waiting in the ready queue.

5) SJF - Gantt Chart

Answer: Shortest-Job-First : Non-preemptive scheduling

Time	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
PID	1	1	1	1	3	3	4	4	2	2	2				

Average waiting time: $(0+7+1+1)/4 = 2.25$ unit

6) SRTF - Gantt Chart

Answer: Shortest-Remaining-Time-First : Non-preemptive scheduling

Time	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
PID	1	1	1	1	3	3	4	4	2	2	2				

Average waiting time: $(0+7+1+1)/4 = 2.25$ unit

7) Priority Scheduling - Gantt Chart

Answer: Preemptive Priority scheduling

Time	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
PID	1	1	1	3	3	1	4	4	2	2	2				

Average waiting time: $(2+7+9+1)/4 = 4.75$ unit

8) Multilevel Feedback Queue

Answer: RR and FCFS:

Time	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
PID	1	1	1	2	2	2	3	3	3	4	4	2	1	1	1	3	

Average waiting time: $(9+2+9+2+3+0)/6 = 4.17$ unit

9) h

Answer:h

10) (a) Please give two example applications which can be handled by a soft real time system and explain the reason for that?

Answer: In soft real time systems, failure to meet a deadline is considered less "good", however it doesn't lead to a system failure. MP3 player is one of the examples of soft real time applications for which small amounts of delay don't cause huge problems. A video game is another soft real time application, in which FPS (frame per second) is so important for a better gaming experience. Tasks related to the game should be completed according to a schedule. However, if there are some delays, it does not necessarily cause the game to terminate or lead to any other error. DVD players, online transaction systems and web browsing are other examples of soft real time systems.

(b) Please give two example applications which should be handled by hard real time system and explain the reason for that?

Answer: Hard real time applications require certain compliance with the deadlines. Nuclear reactor control systems, anti-missile systems, air traffic control systems, autopilot systems can be given as examples of hard real time applications. In each of these systems, predefined deadlines shouldn't be missed, otherwise it may lead to huge problems, or even disasters.

(c) Can LINUX be used for hard real time applications?

Answer: No, Linux attempts to use a "fairness" policy in scheduling threads and processes to the CPU. This gives all applications in the system a chance to make progress in order to increase the interactivity. However, this results in unpredictable delays preventing an activity from completing on time. Therefore a conventional OS like Linux cannot be used for hard real time applications.

Additionally, there is a Linux distribution designed for hard real time applications namely RTLinux.

11) (a) A computer system is already running with a scheduling mechanism with a certain average turnaround time. If we double the number of processes to be scheduled what will happen to the turnaround time?

Answer: Since we don't know the execution time of new processes all cases are possible. If execution time of new processes are very small compared to others and if we execute them first then average turnaround time will decrease. If average execution time of new processes is close to average execution time of old processes, average turnaround time will increase since we have more processes present in the waiting state. Turnaround time is the time from submission of a process to the completion of it (includes waiting time etc.)

(b) What are the advantages and disadvantages of simulation vs. analytical models while evaluating scheduling algorithms?

Answer: One of the advantages of the analytic evaluation is that it produces a formula or a number to evaluate the performance of the algorithm for that workload. Analytical modeling is deterministic, it's simple and fast. It gives us exact numbers so that we can compare the scheduling algorithms with each other easily. However, it requires exact numbers for input, and its answers apply only to those cases. Also, analytic evaluation require us to provide the processes with their CPU bursts beforehand to analyze the complexities. But this may not always be the case.

To get a more accurate evaluation of scheduling algorithms, we can use simulations. Running simulations involves programming a model of the computer system. As the simulation executes, statistics that indicate algorithm performance are gathered and printed. In simulations, we can use probability distributions for CPU burst times, arrivals, departures, and so on. However, simulations can be expensive, often requiring hours of computer time. A more detailed simulation provides more accurate results, but it also takes more computer time. The design, coding, and debugging of the simulator can be a major task.

Even a simulation is of limited accuracy. The only completely accurate way to evaluate a scheduling algorithm is to code it up, put it in the operating system, and see how it works.