CO327 – Operating Systems

Assignment – 3

E/14/017

1. Why is it important for the scheduler to distinguish I/O-bound programs from CPU-bound programs?

   I/O-bound programs only doing a small amount of computation before performing IO. Such programs typically do not use up their entire CPU quantum. CPU-bound programs use their entire quantum without performing any blocking IO operations. We could make better use of the computer's resources by giving higher priority to I/O-bound programs and allow them to execute ahead of the CPU-bound programs.

2. Explain the difference between preemptive and non-preemptive scheduling.
   Non-Preemptive Scheduling
   Tasks within a Non-Preemptive system will run until completed.
   The Scheduler then checks all tasks' states and schedules the next highest priority task with a Ready state.
   With Non-Preemptive Scheduling, once a task has its assignment to the CPU, it cannot be taken away, even if short tasks have to wait for longer tasks to complete.
   The scheduling management across all tasks is "fair" and response times are predictable as high priority tasks cannot bump waiting tasks further down the queue.
   The Scheduler ensures each task gets its' share of the CPU, avoiding any delay with any task.
   The 'amount of time' allocated to the CPU may not necessarily be equal, as it depends on how long the task takes to complete.
   Preemptive Scheduling
   This scheduling model allows tasks to be interrupted – in contrast to Non-Preemptive Scheduling that has a "run-to-completion" approach.
   The interrupts, which could be initiated from external calls, invokes the Scheduler to pause a running task to manage another higher priority task – so the control of the CPU can be preempted.
   The highest priority task in a Ready state is executed, allowing rapid response to real-time events.
   Some of the cons with Preemptive Scheduling involve the increase of overheads on resources when using interrupts and issues can occur with two tasks sharing data, as one may be interrupted while updating shared data structures, and could negatively affect data integrity.
   On the other hand, it is practical to be able to pause a task to manage another one that could be critical.

3. Discuss how the following pairs of scheduling criteria conflict in certain settings.

(a) CPU utilization and response time

(b) Average turnaround time and maximum waiting time

(c) I/O device utilization and CPU utilization

a) CPU utilization and response time: CPU utilization is increased if the overheads associated with context switching is minimized. The context switching overheads could be lowered by performing context switches infrequently. This could however result in increasing the response time for processes.
b) Average turnaround time and maximum waiting time: Average turnaround time is minimized by executing the shortest tasks first. Such a scheduling policy could however starve long-running tasks and thereby increase their waiting time.
c) I/O device utilization and CPU utilization: CPU utilization is maximized by running long-running CPU-bound tasks without performing context switches. I/O device utilization is maximized by scheduling I/O-bound jobs as soon as they become ready to run, thereby incurring the overheads of context switches.

4.One technique for implementing lottery scheduling works by assigning processes lottery tickets, which are used for allocating CPU time. Whenever a scheduling decision has to be made, a lottery ticket is chosen at random, and the process holding that ticket gets the CPU. The BTV operating system implements lottery scheduling by holding a lottery 50 times each second, with each lottery winner getting 20 milliseconds of CPU time (20 milliseconds × 50 = 1 second). Describe how the BTV scheduler can ensure that higher-priority threads receive more attention from the CPU than lower priority threads.

By assigning more lottery tickets to higher-priority processes.

5.A variation of the round-robin scheduler is the regressive round-robin scheduler. This scheduler assigns each process a time quantum and a priority. The initial value of a time quantum is 50 milliseconds. However, every time a process has been allocated the CPU and uses its entire time quantum (does not block for I/O), 10 milliseconds are added to its time quantum, and its priority level is boosted. (The time quantum for a process can be increased to a maximum of 100 milliseconds.) When a process blocks before using its entire time quantum, its time quantum is reduced by 5 milliseconds, but its priority remains the same. What type of process (CPU-bound or I/O-bound) does the regressive round-robin scheduler favor? Explain.

This scheduler would favor CPU-bound processes as they are rewarded with a longer time quantum as well as priority boost whenever they consume an entire time quantum. This scheduler does not penalize I/O-bound processes as they are likely to block for I/O before consuming their entire time quantum, but their priority Remains the same.

6. 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 non-preemptive scheduling, and base all decisions on the information you have at the time the decision must be made.

Process	Arrival Time	Burst Time
P1	0.0	8
P2	0.4	4
P3	1.0	1

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

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

(c) The SJF algorithm is supposed to improve performance, but notice that we chose to run process P1 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 P1 and P2 are waiting during this idle time, so their waiting time may increase. This algorithm could be called future knowledge Scheduling.

a. 10.53

b. 9.53

c. 6.86

Remember that turnaround time is finishing time minus arrival time, so you have to subtract the arrival times to compute the turnaround times. FCFS is 11 if you forget to subtract arrival time.

7. What advantage is there in having different time-quantum sizes at different levels of a multilevel queuing system?

Processes that need more frequent servicing, for instance, interactive processes such as editors, 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, requiring fewer context switches to complete the processing, and thus making more efficient use of the computer.

8.Most scheduling algorithms maintain a run queue, which lists processes eligible to run on a processor. On multicore systems, there are two general options: (1) each processing core has its own run queue, or (2) a single run queue is shared by all processing cores. What are the advantages and disadvantages of each of these approaches?

The primary advantage of each processing core having its own run queue is that there is no contention over a single run queue when the scheduler is running concurrently on 2 or more processors. When a scheduling decision must be made for a processing core, the scheduler only need to look no further than its private run queue. A disadvantage of a single run queue is that it must be protected with locks to prevent a race condition and a processing core may be available to run a thread, yet it must first acquire the lock to retrieve the thread from the single queue. However, load balancing would likely not be an issue with a single run queue, whereas when each processing core has its own run queue, there must be some sort of load balancing between the different run queues.

9.Consider a preemptive priority scheduling algorithm based on dynamically changing priorities. Larger priority numbers imply higher priority. When a process is waiting for the CPU (in the ready queue, but not running), its priority changes at a rate α. When it is running, its priority changes at a rate β. All processes are given a priority of 0 when they enter the ready queue. The parameters α and β can be set to give many different scheduling algorithms.
(a) What is the algorithm that results from $\beta > \alpha > 0$?
(b) What is the algorithm that results from $\alpha < \beta < 0$?
First case is FCFS; second case is LIFO (last in, first out).

10. Explain why interrupt and dispatch latency times must be bounded in a hard-real-time system.

following tasks: save the currently executing instruction, determine the type of interrupt, save the current process state, and then invoke the appropriate interrupt service routine. Dispatch latency is the cost associated with stopping one process and starting another. Both interrupt and dispatch latency need to be minimized in order to ensure that real-time tasks receive immediate attention. Furthermore, sometimes interrupts are disabled when kernel data structures are being modified, so the interrupt does not get serviced immediately. For hard real-time systems, the time-period for which interrupts are disabled must be bounded in order to guarantee the desired quality of service.

11. Write a short note on the current status and the historical evolution of the Linux Scheduler.

The first ever Linux process scheduler that came with kernel version 0.01 in 1991 used minimal design and, obviously, was not aimed at massive multiprocessor systems. There existed only one process queue and the code iterated all the way through it when choosing a new process to run. Luckily, in the old days, maximum number of tasks was greatly restricted (macro NR_TASKS in the first version of the kernel was set to 32).