E/15/366

S.THINESH

CO327 – Operating Systems

Assignment – 3

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

I/O-bound programs have the property of performing only a small amount of computation before performing I/O. Such programs typically do not use up their entire CPU quantum. CPU-bound programs, on the other hand, use their entire quantum without performing any blocking I/O operations. Consequently, one 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. The IO bound programs would be starved. Imagine you treat them equally. Say each process gets 10 tics of time. You sleep the IO bound program every time it has a request say it has one once every 3 tics and needs 7 tics to have its request fulfilled but then the CPU bound program gets a full 10 tics. The IO bound program ends up getting 1/3of the processor that the cpu bound program does.

2.Explain the difference between preemptive and non-preemptive scheduling.

- The basic difference between preemptive and non-preemptive scheduling is that in preemptive scheduling the CPU is allocated to the processes for the limited time. While in Non-preemptive scheduling, the CPU is allocated to the process till it terminates or switches to waiting state.
- 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 ready state to running state, vise-verse, and maintaining the ready queue. On the other hands, 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 frequently arrives in the ready queue then the process with low priority have to wait for a long, and it may have to starve. On the other hands, in the non-preemptive scheduling, if CPU is allocated to the process with larger burst time then the processes with small burst time may have to starve.
- Preemptive scheduling is quite flexible because the critical processes are allowed to access CPU as they arrive into the ready queue, no matter what process is executing currently. Non-preemptive scheduling is rigid as even if a critical process enters the ready queue the process running CPU is not disturbed.
- The Preemptive Scheduling is cost associative as it has to maintain the integrity of shared data which is not the case with Non-preemptive Scheduling.

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?
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 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.

6.86

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

Processes which need more frequent servicing, such as interactive processes, can be in a queue with a small *q*. Processes that are computationally intensive can be in a queue with a larger quantum, requiring fewer context switches to complete the processing, making more efficient use of the CPU.

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?

With each processing core having its own run queue, there is not conflict over a single run queue when the scheduler is running simultaneously more than one processor. The disadvantage is that there needs to be some sort of load balancing between the different run queues. The advantage with single run queue is that load balancing would not be an issue, however, the disadvantage is thatit must be protected with locks to prevent a race condition.

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 $\beta > \alpha > 0$?

- FCFS

(b) What is the algorithm that results from $\alpha < \beta < 0$?

- LIFO (last in, first out).

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

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. v0.01 is process scheduler is just 20 lines of code and is very simple. For reference, the newest Linux kernel consists of ten thousand of lines. 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).
by the year 2000, operating systems designers considered scheduling to be a solved problem… (the) year 2004 brought an end to Dennard scaling, ushered in the multicore era and made energy efficiency a top concern in the design of computer systems. These events once again made schedulers interesting, but at the same time increasingly more complicated and often broken.
The scheduler is an operating system module that selects the next jobs to be admitted into the system and the next process to run. Operating systems may feature up to three distinct scheduler types: a long-term scheduler (also known as an admission scheduler or high-level scheduler), a mid-term or medium-term scheduler, and a short-term scheduler. The names suggest the relative frequency with which their functions are performed.