# 5.11

Of these two types of programs:

a. I/O-bound

b. CPU-bound

which is more likely to have voluntary context switches, and which is more likely to have nonvoluntary context switches? Explain your answer.

I/O-bound programs are more likely to have voluntary context switches, while CPU-bound programs are more likely to have nonvoluntary context switches.

When a program is I/O-bound, it means that it spends most of its time waiting for input/output operations to complete, such as reading or writing data from/to a disk or network. During this waiting period, the program can voluntarily give up its turn to execute to other processes that are ready to run, using a system call like "yield" or "sleep". This voluntary context switching allows the operating system to allocate CPU time to other processes and keep the system responsive.

On the other hand, when a program is CPU-bound, it means that it spends most of its time performing computational tasks that require heavy processing power. In this case, the program may not have the opportunity to voluntarily give up the CPU, and the operating system may have to interrupt its execution with a nonvoluntary context switch to give other processes a chance to run. This type of context switch can occur, for example, when the CPU time allocated to the process has expired, or when a higher-priority process becomes ready to run.

In summary, the type of program determines the likelihood of voluntary or nonvoluntary context switches, with I/O-bound programs more likely to have voluntary context switches, and CPU-bound programs more likely to have nonvoluntary context switches.

‘

# 5.12

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: The CPU utilization criterion aims to keep the CPU as busy as possible by ensuring that there is always a process running on it. However, this can conflict with the response time criterion, which aims to minimize the time that a process waits before it starts executing. If the scheduler selects CPU-bound processes with high CPU burst times, the response time for I/O-bound processes can increase, leading to poor system performance. On the other hand, if the scheduler selects I/O-bound processes with low CPU burst times, the CPU utilization may decrease, resulting in underutilization of the CPU.

b. Average turnaround time and maximum waiting time: The average turnaround time criterion measures the average time it takes for a process to complete, including both CPU and I/O time. The maximum waiting time criterion measures the longest time that a process has to wait before it can start executing. These two criteria can conflict because reducing the maximum waiting time may increase the average turnaround time, and vice versa. For example, a scheduler that prioritizes short jobs can reduce the maximum waiting time, but this may increase the average turnaround time if long jobs are waiting in the queue.

c. I/O device utilization and CPU utilization: The I/O device utilization criterion aims to keep the I/O devices as busy as possible by ensuring that there is always an I/O request in progress. However, this can conflict with the CPU utilization criterion, which aims to keep the CPU as busy as possible. If the scheduler selects I/O-bound processes, the I/O device utilization may increase, but the CPU utilization may decrease, leading to underutilization of the CPU. Conversely, if the scheduler selects CPU-bound processes, the CPU utilization may increase, but the I/O device utilization may decrease, leading to underutilization of the I/O devices.

# 5.13

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.

Lottery scheduling is a probabilistic scheduling algorithm that allocates CPU time to processes based on their ownership of lottery tickets. In BTV operating system, each process is assigned a certain number of lottery tickets proportional to its priority. Higher-priority processes are given more tickets, increasing their chances of winning the lottery and receiving CPU time.

To ensure that higher-priority threads receive more attention from the CPU than lower-priority threads, BTV can assign more lottery tickets to high-priority threads. This increases their probability of winning the lottery and receiving CPU time. For example, if a high-priority process is assigned 10 tickets and a low-priority process is assigned 2 tickets, the high-priority process has a higher chance of winning the lottery and receiving CPU time.

Additionally, BTV can also adjust the length of the CPU time slice given to the winning process based on its priority. For example, high-priority processes can be given longer time slices than low-priority processes, ensuring that they receive more CPU time overall. This can be achieved by adjusting the number of lottery tickets required to win a time slice. For example, if high-priority processes require fewer tickets to win a time slice, they will receive longer time slices and more CPU time.

Overall, by assigning more lottery tickets to higher-priority threads and adjusting the length of CPU time slices based on priority, BTV can ensure that higher-priority threads receive more attention from the CPU than lower-priority threads in a probabilistic manner.

# 5.14

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 two options for maintaining run queues on multicore systems are:

1. Each processing core has its own run queue:

Advantages:

- Can exploit parallelism: Each core can run a separate process simultaneously, improving overall system throughput.
- Decentralized scheduling: Each core can make its own scheduling decisions independently, potentially reducing overhead and latency.
- Reduces contention: Each core has its own run queue, reducing contention for shared resources like locks and synchronization primitives.

Disadvantages:

- Load balancing: It can be difficult to balance the load across multiple run queues, leading to some cores being overburdened while others are underutilized.
- Fragmentation: Processes may be spread across multiple run queues, leading to fragmentation and reduced cache locality.
- Complexity: The need to coordinate scheduling decisions across multiple run queues can add complexity to the system.

2: A single run queue is shared by all processing cores:

Advantages:

- Simplifies scheduling: All cores share the same run queue, simplifying scheduling decisions.
- Load balancing: Processes can be easily balanced across all cores by assigning processes from the run queue to different cores.
- Cache locality: Processes are more likely to be grouped together in memory, improving cache locality.

Disadvantages:

- Contentions: Multiple cores may contend for access to the shared run queue, leading to increased overhead and latency.
- Synchronization: Access to the shared run queue requires synchronization mechanisms like locks, which can add overhead and increase contention.
- Scalability: As the number of cores increases, the shared run queue may become a bottleneck, limiting scalability.

# 5.16

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 is 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.

The regressive round-robin scheduler favors CPU-bound processes over I/O-bound processes.

This is because CPU-bound processes typically use up their entire time quantum without blocking for I/O, which means that they will have 10 milliseconds added to their time quantum and their priority boosted. Over time, this will give CPU-bound processes longer time quantum and higher priority levels, which will increase their chances of getting allocated the CPU.

On the other hand, I/O-bound processes are more likely to block for I/O before using up their entire time quantum, which means that their time quantum will be reduced by 5 milliseconds each time they block, without any increase in priority level. This can cause their time quantum to be repeatedly reduced until it becomes very small, making it less likely that they will get allocated the CPU.

Overall, the regressive round-robin scheduler is biased towards processes that use the CPU heavily and are less likely to block for I/O. This means that CPU-bound processes will have an advantage over I/O-bound processes in terms of CPU allocation.