Assignment 2

Saad Saiyed sas162@uregina.ca
200415258

1. Define short-term scheduler and long-term scheduler, and explain the main differences between them. (6 marks)

To make multiple processes available for a user, schedulers are appointed to run them simultaneously. There are two types of schedulers, (i) Long-term and (ii) Short-term Scheduler. The main difference is that long-term scheduler takes the process from the job pool and appoints them to ready queue and short-term scheduler takes processes from the ready queue and provide CPU. (i) Long-term Scheduler: It keeps the left out process in routine. To avoid mass execution, some (less prioritized) processes are sent to the job pool where they wait for the long-term scheduler to take them to ready queue. These processes sometimes stay there for quite a while and for this reason some operating system avoids using the long-term scheduler. (ii) Short-term Scheduler: It is used to control the execution of processes which are inside the ready queue by providing CPU to each individual process. It is a bad example of multi-programming as it has very short and simultaneous usage.

2. Explain the concept of a context switch. (6 marks)

The concept of switching through one process to another and also to make sure the following process does not conflict is referred to as context switch. This process is followed by CPU. To understand context switching we need to understand how multiprocessing (multitasking) is achieved. Each process has three states in which it can proceed: runnable, running and blocked. The pre and post methods are: created and terminated. The process is created and sent to the ready queue. Now scheduler takes one process and keeps it into running if another existing process was already using CPU has its time doing. If this is not the case, the process is ready to be kept in runnable or blocked according to the instructions present in the process. This transfer of process is executed by the context switch. If the process is done executing or terminated for some reason, the process exits through 'terminated' path.

3. Explain the terms at most once and exactly once, and indicate how these terms relate to remote procedure calls. (6 marks)

The term at most once implies that the call from a client is produced and given to server only once and that is ensured using the repeated check on timestamps of every message. Similarly, the term exactly once ensures that the call was received by the server once and only once. Which confirms that the server has executed the message and responded to it.

4. Identify and briefly explain each of the four major categories of benefits of multithreaded programming. (6 marks)

The four major categories of benefits of multithreading are as follows: (1) Responsiveness: a program containing multithreaded programming can perform even if part of it is

blocked or busy performing lengthy operations or calculation. As a result, it makes design responsive at a certain level. This is useful when programming user interfaces in a program.

(2) Resource Sharing: it is possible for processes to share resources using some additional methods like; message passing. This can be preloaded by the programmer. Processes containing multiple threads of activity in same address space, shares resources assigned to that process.

(3) Economy: creating processes takes a lot of memory and time in compared to threads. Threads shares resources provided by the process as well as its location. So creating multiple threads is far more time-efficient than creating a new process.

(4) Scalability: multithreading is more effective multiprocessor CPUs, as threads can run on multiple processers even on which process is not present at the same time. Which is not the case with processes containing a single thread. Each of these processes can run on single processer even if CPU has multiple processors. So increasing the number of threads makes a program run in parallel with multiple processes (threads inside process).

5. Briefly describe the benefits and challenges for multithreaded programming that are presented by multicore systems. (8 marks)

As we discussed in earlier questions, there are four major categories of benefits of multi-threaded programming presented by the multicore system; responsiveness, resource sharing, economy, scalability. Even if some part of the program is blocked, multithread programming allows the program to run. This enhances responsiveness in terms of user interactions. All the process containing multiple threads has the ability to share memory within the same address space. This is obtained using threads, as they share memory and resources allocated to its process. This makes it cheap in terms of space and time to use threads instead of the new process. And so making threads in a process is far more efficient than creating new processes.

The multithreaded program allows the program to run faster but this makes it harder on programmers and software designers to create such manageable programs. First of all, challenges start from dividing all the tasks which should work in parallel. All parallel task must be ensured to use and perform a similar amount of work. All of these tasks need to access data associated with the program, which makes all the data to be accessed by all the threads running on different cores. To achieve this goal we need to make sure all data is secured and not unnecessary threads receive confidential information. Further, if we all threads to get access to data, we need to store the memory location of these data and there we introduce data dependences.

6. Define coarse-grained multithreading and fine-grained multithreading, and explain their differences. (6 marks)

In coarse-grained multithreading each instruction in each thread is executed until there is a stall or cycle gap in a running thread. In each stall, a clock cycle is lost to change the thread.

In a process in which multithreading is programmed where threads are executed in a round-robin fashion, that multithreaded programming is referred to as fine-grained multi-threading. In this procedure, each instruction of individual thread is executed simultaneously.

Differences:

• The procedure of multithreading in which switch between threads is done only when there is a stall in the instruction of currently running thread which wastes one clock cycle is known as coarse-grained multithreading. The procedure of multithreading in which switch between threads is done without any worries about cache miss caused by instructions enrolled in threads is known as fine-grained multithreading.

• Coarse-grained multithreading is less efficient than fine-grained multithreading, as switching between threads cost one clock cycle per switching.

• Coarse-grained takes more usage of CPU than fine-grained as it takes fewer threads for coarse-grained to keep CPU busy.

7. Explain process starvation and how ageing can be used to prevent it. (6 marks)

The term 'aging' refers to the increase of priority gradually according to the time span. Process starvation occurs when a process is stuck in a ready queue waiting for its turn to execute. The main reason when this takes is when a low priority process gets stuck with high priority processes. The scheduler avoids low priority process and continues executing different instructions of multiple high priority processes. So at this time 'aging' occurs to the process with the low priority and eventually the scheduler has to choose this process as its priority is increased. But the aging takes place only when a certain process is stuck of some period of time.

8. How does the dispatcher determine the order of thread execution in Windows? (6 marks)

The execution order is decided by the dispatcher using 32 level priority scheme. The priority is determined in two classes; variable class and real-time class. The threads between priority level from 1-15 are categorized as a variable class. And threads from 15 and above till 31 are prioritized as a real-time class. The queue system is followed by the dispatcher for every instance of scheduling priority and sorts them from high priority to low priority. After that, it searches through the queue starting from highest until it finds a ready to run state thread. If that is not the case, the dispatcher will run a sleeping thread.

9. Define critical section, and explain two general approaches for handling critical sections in operating systems. (8 marks)

When one or more processes get the same code segment to execute that segment is known as the critical section. These sections share different parameters and resources which are necessary to maintain consistency.

There are two general approaches for handling critical sections in the operating system: preemptive kernels and non-preemptive kernels. When a process is running in a kernel mode and if the kernel is a preemptive kernel, it allows the process to preempted or prevented. The opposite of that happens in the non-preemptive kernel. It does not allow a process running a kernel-mode to be preempted. In the non-preemptive kernel, there is only one process running at a time and so it is free from the race conditions. That is not the case with the preemptive kernel. As multiple processes can be running in the kernel mode at the same time, the sharing of data needs to be maintained manually as free from race conditions.

10. Describe the dining-philosophers problem, and explain how it relates to operating systems. (6 marks)

The dining-philosophers problem was with synchronization of the large class of concurrency control problems (the problem of allocating the memory and resources to several processes in a way that prevents starvation and deadlock).

11. Define the two-phase locking protocol. (6 marks)

There are two types of a lock system which includes, exclusive and shared lock system. These locks are implemented in a very simple lock-based protocol. As they are very simple they do not provide serialized workflow every time. To achieve serializability, two-phase locking plays an important role.

The two-phase locking method is used when in a transaction we need serializability which is possible if we use locking and unlocking in two phases. This process goes under;

Growing Phase: this is used if we want to make new lock but we cannot release any existing locks. Shrinking Phase: opposite to growing phase you can release existing locks but cannot create new ones.

12. Describe how an adaptive mutex functions. (6 marks)

A Solaris operating system uses an adaptive mutex to prevent the data sharing which runs as a standard spinlock. If the sharable is data is being shared and also that thread which has the lock is locked and also if different CPU is running the thread then the thread will continue spinning until the lock is released and allow data to be accessible. If the thread which has a lock is in the ready queue and not executing, it sleeps and waits for the available lock to be free. As a result in single-core processing system thread in the waiting stage always waits until it gets any available locks.

13. Describe a scenario in which the use of a reader-writer lock is more appropriate than using another synchronization tool, such as a semaphore. (6 marks)

In a synchronous system like semaphore, sharable data is used by only one process at a time. But if some process is just want to read the data, it is highly distinguishable. So if a process just want to read or reader-write than a reader-write lock is used to let the data be shared amongst all the other readers. If there are many reading processes reader-write lock is very efficient.

14. What is the difference between deadlock prevention and deadlock avoidance? (6 marks)

Deadlock prevention makes sure that one of the many conditions of deadlock must not be satisfied which makes a deadlock not to be used. Deadlock avoidance is a precaution method which ensures that the process comes with the instructions about the information of all the resources it will use until the end of the execution.

15. Describe a wait-for graph, and explain how it detects deadlock. (6 marks)

If an only single instance of all resources is created then we can define a deadlock detection algorithm that is similar to the resources-allocation graph this is called wait-for-graph. We can easily get this graph if we alter some of the components of resources-allocation graph such as collapsing edges and removing the resource nodes. In order to detect the deadlock, the program needs to invoke an algorithm which searches through the graph for a cycle and also has to maintain the wait-for graph.

16. Describe how a safe state ensures that deadlock will be avoided. (6 marks)

Safe state is a state in which the deadlock is not possible. This is achieved by constantly looking for a sequence of processes to execute and terminate itself. Still if after all the process are terminated there is a threat that system will eventually exit safe state and at that time there is a possibility of a deadlock happening. In order to prevent that, a safe state must be always running. Which is possible if we assign necessary resources as long as the system is in the safe state.