Assignment 2

Vaibhav Sharma sharma3v@uregina.ca
200365101

1. Solutions

1.1 Solution 1

Long term scheduler or LTS is responsible for creating new processes and bringing them into the system. LTS main work is to create and move the jobs from "New" to "Ready" state. Whereas, the short term scheduler or STS is responsible for processing one of the jobs from "Ready" state to "Running" state. There are a lot of processes available in "Ready" state at a time and STS takes care of selecting one job and moving it to the "Running" state.

The number of processes present in the main memory at any given point of time is called the degree of multi-programming. The long term scheduler controls the degree of multi-programming and also selects a good combination of CPU bound and input/output bound processes. If LTS ends up bringing a lot of CPU bound jobs, it will increase the CPU utilization and drop the throughput.

1.2 Solution 2

Context switching is when the process in running state is saved and swapped by another process. For example, if the current running process is p1 and p2 needs to run then the processor will save the context of p1 and load the context of p2 into running state. Each and every time when the process is moving from one state to another, the context of the process will change. The minimum number of processes required in context switching is two processes with an exception of "Round robin" which only uses one process. Context switching time is the time it takes before the running process switches context with the next process.It is also considered as overhead for the system. The context switching does take some time, so if the context of the process is more than context switching time will also increase which is undesirable.

1.3 Solution 3

These term are used to define how client interacts with the server in an event of crash or restart. After a server crash/restart during RPC, client tries to initiate a conversation. RPC uses different models/semantics to manage outages. At most one is when the client sends a message to the server and receives a reply because then client knows server got at least one message and then client sends another to be sure. whereas, exactly one is a best case scenario where client sends a request and multiple requests are filtered for duplicates

to execute only one request.The problem with exactly one request is the server might never receive it due to crash/outage while client is sending a request.

1.4 Solution 4

Responsiveness - Multithreading allows programs to continue running even when other programs are using the processors and blocking the resource. It splits the work and reduces the time by which the program gets executed as they can run parallely.

Scalability - It is very easy to increase the work via either vertical scalability or horizontal scalability. multiple threads are executed parallely and one thread can be divided among different threads to execute separately.

Cost effective - Context switching is very easy in multithreading processing.

Using resources efficiently - Multithreading used message passing and Shared memory which allows it to create multiple threads under space address space.

1.5 Solution 5

Coarse-grained multithreading - a switch only happens when the thread in execution causes a stall, thus wasting a clock cycle.

Advantages : No need for very fast thread-switching; Doesn't slow down thread, since switches only when thread encounters a costly stall

Disadvantages : Since CPU issues instructions from 1 thread, when a stall occurs, the pipeline must be emptied or frozen; New thread must fill pipeline before instructions can complete

Fine-grained multithreading - switching among threads happens at each instruction, independently from the fact that the thread instruction has caused a cache miss.

Advantages : Switches between threads on each instruction; Multiples threads interleaved

Disadvantages : Other threads executed when one thread stall; But slows down execution of individual threads

1.6 Solution 6

Starvation occurs when the CPU is busy taking care of all the high priority tasks and all the low priority tasks never get their turn. Aging is used to prevent this as time is used to increase priority of a low priority task which is waiting from a long time. This makes sure that all the tasks are executed in a time manner else low priority tasks never get executed.

1.7 Solution 7

In windows, Dispatch uses a 32 level priority order where tasks are leveled from 1-32 according to their priority. Priorities are divided into two classes. variable class is from level 1-15 and then real factor class is from 16-31. Dispatch creates a queue of all the tasks in order of their priority and executes them one by one when there is a free resource available to complete the task.

1.8 Solution 8

When a code is sharing variables and resources cocurruntely, Operating system needs to make sure they are not being accessed by anything else while being used. The protected section/code/variable is called the critical section. Two different approaches to handle critical sections are:

Preemptive Kernel: It allows a process to be preempted while running in kernel mode

Non-Preemptive Kernel: It does not allow a process to be preempted. it must be finished before existing kernel mode.

1.9 Solution 9

The scenario involves five philosophers sitting at a round table with a bowl of food and five chopsticks. Each chopstick sits between two adjacent philosophers. The philosophers are allowed to think and eat. Since two chopsticks are required for each philosopher to eat, and only five chopsticks exist at the table, no two adjacent philosophers may be eating at the same time. A scheduling problem arises as to who gets to eat at what time. This problem is similar to the problem of scheduling processes that require a limited number of resources.

It relates to the operating system because the resources are shared among different tasks. In an operating system, these philosophers translate into different threads and they can run into problems like deadlock.

1.10 Solution 10

Two-phase locking protocol uses two phases to lock and unlock a transaction.

First phase - New locks are put on but locks cannot be unlocked. This is also known are expanding phase.

Second phase - locks can be unlocked but cannot request new locks. This is also known as shrinking phase.

1.11 Solution 11

If the shared data being accessed is already locked and the thread holding that lock is running on another CPU, the thread spins while waiting for the lock to be released, and the data to become available. If the thread holding the lock is not in the run state, the waiting thread sleeps until the lock becomes available. On a single processor system, spin locks are not used and the waiting thread always sleeps until the lock becomes available

1.12 Solution 12

Reader-writed lock is more efficient when there are several readers because it allows a process that is only reading shared data to corcurrently access shared data with other readers. Reader-Writer locks outweights the other locks when state of process is known such as read only or read/write.

1.13 Solution 13

Prevention: • The system does not require additional a priority information regarding the overall potential use of each resource for each process.

• In order for the system to prevent the deadlock condition it does not need to know all the details of all resources in existence, available and requested.

• Resource allocation strategy for deadlock prevention is conservative, it under commits the resources.

• All resources are requested at once.

Avoidance:

• The goal for deadlock avoidance is to the system must not enter an unsafe state.

• Deadlock avoidance is often impossible to implement.

• Needs to be manipulated until at least one safe path is found.

• There is no preemption.

1.14 Solution 14

If all resources have only a single instance, then we can define a deadlock-detection algorithm that uses a variant of the resource- allocation graph, called a wait-for graph. We obtain this graph from the resource-allocation graph by removing the resource nodes and collapsing the appropriate edges. To detect deadlocks, the system needs to maintain the wait-for graph and periodically invoke an algorithm that searches for a cycle in the graph.

1.15 Solution 15

A safe state ensures that there is a sequence of processes to finish their program execution. Deadlock is not possible while the system is in a safe state. However, if a system goes from a safe state to an unsafe state, deadlock is possible. One technique for avoiding deadlock is to ensure that the system always stays in a safe state. This can be done by only assigning a resource as long as it maintains the system in a safe state.