OPERATING SYSTEM

Lecture 7

Deadlock and Starvation

Dr Victoria Mukami

INTRODUCTION

This lecture focuses on deadlock and starvation. We will look at the conditions of deadlock, deadlock approaches deadlock prevention, avoidance, detection, and recovery. Finally, we shall review starvation as an operating system concept.

Learning objectives

By the end of this topic, you should be able to:

1. Show an understanding of the conditions that lead to deadlock.
2. Outline the different approaches to dealing with deadlocks
3. Understand starvation as an operating system concept

OVERVIEW

We have so far reviewed the operating system's role during memory allocation. We looked at various memory allocation schemes that have been used since the early memory allocation scheme till the current operating systems. We additionally have reviewed the way jobs are allocated within the processor. We now work on the deadlock concept

DEADLOCK

A deadlock can be defined as a situation where no agreement can be reached due to a specific disagreement. In computers, deadlock is a situation where there is a permanent block by a set of processes that are competing for resources [2]. A deadlock occurs when a particular blocked process requests a resource that is currently held by another blocked process.

A good example of a deadlock would be when during rush hour and there is an intersection with a roundabout that is a four-way. The cars at the four sides of the roundabout all enter the roundabout and are going to different sides. The cars behind the ones at the front follow the same route and soon there is gridlock. The roundabout is said to be deadlocked. This happens because the first car cannot move since car two is on the way. Car two cannot move because car three is on the way. Car three cannot move because car four is on the way and car four cannot move because it is blocked by car one.

This is like a deadlock where process one asks for a resource held by process two, while process two requires a resource held by process one, thereby leading to a deadlock.

CONDITIONS FOR DEADLOCKS

Several conditions lead to deadlocks. These include mutual exclusion, hold and wait, no pre-emption and circular wait. Let us now review each of these conditions.

Mutual Exclusion

This happens when a resource is held exclusively. This means that only one process can use a particular resource at a particular time. This is also known as the first condition of deadlock [1].

Hold and Wait

This happens when a process holds allocated resources as it waits for other resources. Good example is tickets for a particular concert are on sale. The last two remaining tickets are taken by one person as they wait for the money. The other person has the money but cannot access the tickets. This is the second condition for deadlock [1].

No pre-emption

This is a situation where a resource can only be released voluntarily by the process holding it. A resource cannot be forcibly taken away from a process. This is the third condition for deadlock [1].

Circular Wait

This is the fourth condition of deadlock [1] and it occurs when there is a chain made up of two or more processes and each process requires a particular resource by the next and so on. Think of the traffic gridlock example.

The first three conditions that are mutual exclusion, hold and wait and no pre-emption are conditions that could cause a deadlock. When the circular wait is introduced within those first three conditions then a deadlock exists.

STRATEGIES TO DEAL WITH DEADLOCKS

Four approaches exist that are aimed at resolving or preventing deadlocks. The approaches include prevention, avoidance, detection, and recovery.

Deadlock Prevention

Before a deadlock occurs, the operating system needs to ensure that it can prevent one from occurring. We have so far looked at four conditions that lead to deadlock. The operating system needs to make sure that each of these conditions are taken care of.

Mutual Exclusion: With the mutual exclusion condition, each process gets its resource exclusively. In operating systems, some resources must be held exclusively such as memory, the CPU, and other dedicated devices. While resources such as files may allow simultaneous access, they only allow one write access at a time [1]. It is therefore not possible for mutual exclusion to be disallowed as some resources require to be held exclusively.

Hold and Wait: In the hold and wait if you remember, a process will hold a resource and wait until it can use it. This can be prevented by ensuring that all the resources required by a process are issued in one go [2]. This of course has its disadvantages as a resource may wait for a long time waiting for resources or the resources allocated to a process may remain unused for a long period [2]. It may be hard to prevent the hold and wait especially in interactive systems.

No Pre-emption: To prevent this a process could be denied any additional requests until it releases its original resources and then can later request the same resources together with the additional resource [2]. An additional preventative measure would be for the operating system to pre-empt when a resource may require additional resources and provide this at the same time. Of course, this only works if both processes do not have the same priority levels.

Circular Wait: This can be prevented by the operating system ensuring that the formation of a circle does not exist [1]. Use of linear ordering [2] and numbering systems [1] could be used to ensure circles do not form. This of course is inefficient as some processes may be denied resources.

Deadlock Avoidance

As seen in the previous section, it is very hard for an operating system to prevent or remove one of the four conditions of deadlock. It may be easier for the operating system to know in advance the sequence of events for every active process [1]. Banker's algorithm is an algorithm that explains how deadlock avoidance can be achieved. The banker’s algorithm is based on specific principles. It looks at a bank with a fixed amount of capital. The principles that guide this algorithm state [1]

1. No customer can be given a loan that is higher than the bank's capital.
2. Customers are initially assigned a maximum loan limit as they open their accounts.
3. Customers cannot be able to borrow more than their maximum loan limit.
4. All the loans when summed up should not be more than the bank’s capital.

Let us look at this statement practically. Let us assume the bank has 100,000 in capital regardless of the currency. If three customers come in and they have a credit of 40,000 each, only two can be given a loan of their maximum amount without breaking rule 4. Before reviewing how this algorithm works with avoidance, let us look at two states of a computer system. A safe state exists when resources are allocated without it resulting in a deadlock. An unsafe state is a state that could result in a deadlock when resources are allocated.

Within a computer, the same principle could be applied. The operating system will only allocate resources so long as it remains in a safe state. While the banker’s algorithm can ensure that deadlocks are handled, it is not practical due to several reasons [1].

- Jobs must predict the maximum resources needed
- The number of resources must remain constant
- The number of jobs must remain fixed
- Resources are not well utilized since the algorithm will keep vital resources unavailable to guard against unsafe states
- Jobs are kept waiting for resources therefore scheduling suffers due to poor utilization [1].

This algorithm would not work especially for interactive systems which are the most used.

Deadlock Detection

Prevention strategies try to solve the problem of deadlock by limiting access to resources or by imposing restrictions on processes. While prevention strategies try to solve the problem of deadlock by limiting access to resources and by imposing restrictions on processes deadlock detection does not limit resource access or restrict actions [2]. An operating system can have it so that a check is made frequently as each resource is requested or non-frequently when the operating system detects an incoming deadlock. The first method ensures early detection, and it is simple for the system to perform. A downside to this would be the overhead resources required to run the check frequently. The algorithm, on the other hand, would try to detect all sequences that need to be completed and could cause a deadlock. It marks processes based on whether they may cause a deadlock. In this way, the system can use one of the strategies to prevent the deadlock.

Deadlock Recovery

The last phase is recovery after a deadlock occurs. This involves ensuring fixing the deadlock as fast as possible and returning the system to normal [1]. The most common feature in deadlock recovery algorithms is choosing one process that will act as the victim and will be terminated [1]. Four methods can be used to recover from a deadlock.

1. All active jobs within the system are terminated regardless of whether they are part of the deadlock or not. Each job, therefore, needs to be restarted from the beginning.
2. All the processes that are deadlocked are aborted. When this happens, it means that the deadlocked processes must be resubmitted by the user [1].
3. The third method involves an elimination method. This happens by deleting one process at a time within the deadlock and checking whether the deadlock is resolved. Once the deadlock is resolved, the remaining jobs continue with processing while the halted restart again at the beginning.
4. The final method involves placing a record of a job so that if the job needs to be interrupted then it would not need to start from the beginning. This method is best suited for jobs with a lengthy run time [1].

STARVATION

The best way to understand starvation is by reviewing the philosopher's problem. Basically, and in summary, the philosopher looks at an eating arrangement. Five philosophers are living in a house where a table is set. Each philosopher is served spaghetti and will need two forks to eat their spaghetti. They are all seated at a round table.

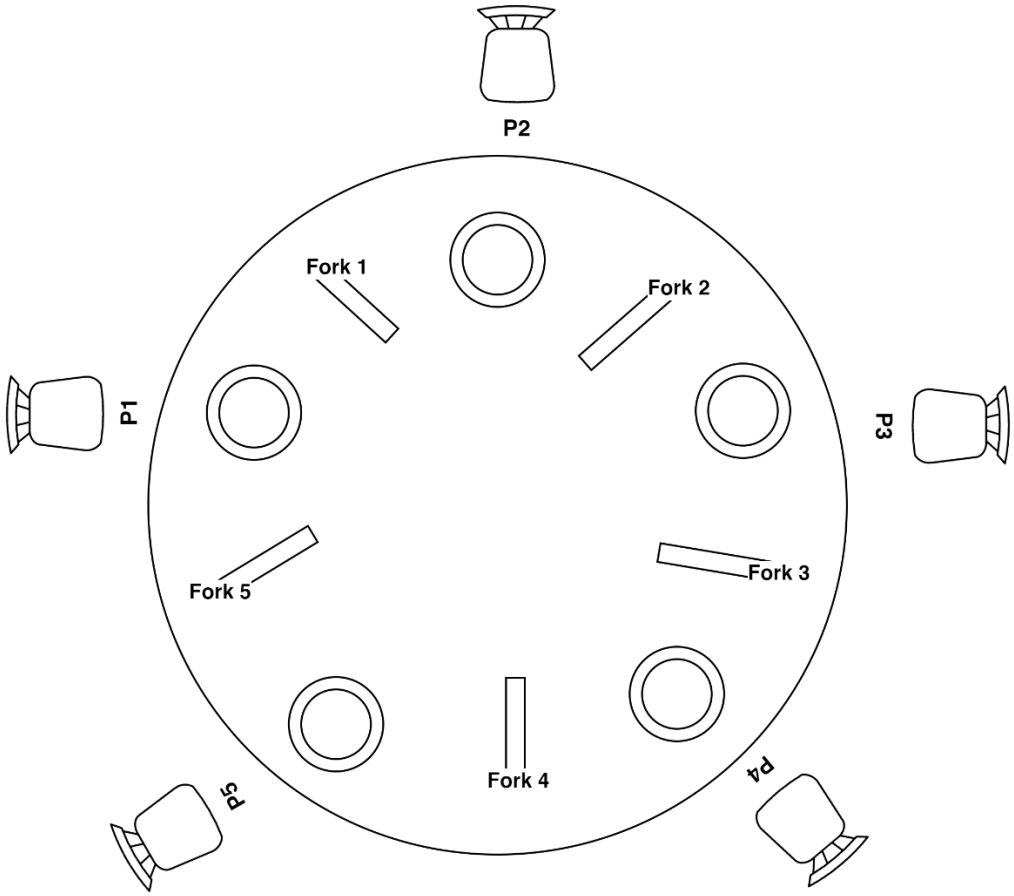

Figure 1: Philosophers dining table

Figure 1 shows the dining table described. As seen, there are 5 philosophers, 5 spaghetti plates and five forks. Each philosopher needs two forks to eat. Philosopher 1 (P1) for instance, would pick Fork 1 and Fork 5 and use them to eat their food. While doing this, Philosopher 2 and Philosopher 5 cannot eat as they do not have a second fork with which to eat. Assuming Philosopher 1 finishes eating and puts down the fork then 2 and 5 can take up. But if philosopher 3 decides to eat, then 2 cannot be able to eat. While philosopher 2 is waiting for food, philosopher 1 decides to eat again. Philosopher 2 ends up starving as they cannot get a fork until both are available.

This is a direct metaphor for the computing world where the forks are like resources while the competing processes are the philosophers [1]. If the operating system does not watch for the starving processes, then the processes would stay within the system forever without ever getting processed.

An algorithm is used to detect starving processes and how these processes can get the resources needed to end their starvation. This algorithm can block incoming processes until the starving processes are processed. As usual, a balance needs to be done to ensure that the system doesn't run high overheads or leave starving jobs indefinitely.

SUMMARY

This lecture was a focus on deadlock and starvation. We have reviewed the conditions of deadlock and the various deadlock approaches. These included deadlock prevention, avoidance, detection, and recovery. Finally, we reviewed the philosopher's problem concerning starvation as an operating system concept.

DISCUSSION TOPIC

We have started learning about concurrency and its related terms: deadlocks and starvation. Based on the operating system that you are running, find out when deadlocks happen or better yet an example of a deadlock and measures that the operating system has taken to solve the said deadlock.

REFERENCES

[1] McHoes, A., & Flynn, I., Understanding Operating Systems. Boston: Cengage Learning, 2018

[2] Stallings, W., Operating Systems: Internals and Design Principles. Harlow: Pearson Education Limited, 2018.