OBL3-OS

March 15, 2019

This is a mandatory assignment. Use resources from the course to answer the following questions. Take care to follow the numbering structure of the assignment in your submission. Some questions may require a little bit of web searching. Some questions require you to have access to a Linux machine, for example running natively or virtually on your own PC, or by connecting to gremlin.stud.iie.ntnu.no over SSH (Secure Shell). Working in groups is permitted, but submissions must be individual.

1 Synchronisation

1. The principle of process isolation in an operating system means that processes must not have access to the address spaces of other processes or the kernel. However, processes also need to communicate.

1. Give an example of such communication.

**A client needs to communicate with the database thread by polling from it to receive data do display to the user. If not, the client remains offline.**

1. How does this communication work?

**Threads synchronize between each other using semaphores, spinlocks, and mutual exclutions. Semaphores are variables that control access to shared resources. They can be used for locks. Spinlocks are locks where the processor waits in a loop for the lock to become free. A mutex is a requirement that threads never enter their own critical section simultaneously.**

1. What problems can result from inter-process communication?

**Deadlocks and process starvation.**

2. What is a critical region? Can a process be interrupted while in a critical region? Explain.

**Piece of code that only one thread can execute at once. A process can be interrupted while in a critical region if it is not locked and the other process capitalizes on a resource - either if a deadlock or starvation occurs. Interruption is either by another process in starvation or deadlocks, or by the OS in a deadlock where it temporary deprives a resource to solve it. Being critical means that the address space is shared and can affect the outcome of another potential process.**

3. Explain the difference between busy waiting (polling) versus blocking (wait/signal) in the context of a process trying to get access to a critical section.

**Polling is continuously requesting data. A blocked process is blocked by the OS, and will be notified when its required data is available to use.**

4. What is a race condition? Give a real-world example.

**Output of a concurrent program depends on the order of operations between threads.**

**eksempel prosess-tabell:**

**En prosess-tabell har begrenset plass. Hver prosess må «forke» for å fortsette, men ingen får «forket» nok til å fortsette pga. begrenset plass i tabellen. vranglås!**

**løsning: la en og en prosess «forke» istedenfor alle parallelt slik at de får fullført en og en, og frigjort plass til neste prosess. Første tilfellet tar kortere tid, men bruker mer (for mye) minne. Andre tilfellet tar lenger tid, men bruker mindre minne. Derfor har rekkefølgen noe å si.**

5. What is a spin-lock, and why and where is it used?

**A spin lock is a lock where the processor waits in a loop for the lock to become free. Assumes lock will be given back soon. It is used for expected short periods of execution time.**

6. List the issues involved with thread synchronization in multi-core architectures. Two lock algorithms are MCS and RCU (read-copy-update). Describe the problems they attempt to address. What hardware mechanism lies at the heart of each?

***Synchronization problems in multi-core processors:***

**Regular test-and-set instructions become very expensive, due to the fact that they are atomic and prevent other processors from doing a test-and-set on the same memory at the same time to ensure mutual exclusion.**

**This old method also does not guarantee FIFO ordering among processors competing for the lock, creating a long wait time for processors in unlucky scenarios.**

**Lastly, each core has its own cache, leading to a large amount of pinging between caches to synchronize data. This pinging is time-expensive and resource heavy.**

***MCS:***

**MCS aims to solve these problems by giving the MCS lock a pointer to a CPU spinlock structure, which then also contains a pointer to the following CPU in line, which is 0 if the line is empty. Each CPU has its own spinlock structure, and spins locally to the CPU, thus reducing cache pinging and minimalizing the need to spin on the main lock itself and pinging values back and forth.**

***RCU:***

**RCU seeks to solve these problems by allowing reads to occur concurrently with updates, through creating multiple versions of shared data that exist until all pre-existing read critical sections are finished; at which time they are then promptly deleted. This allows minimal spinning on locks, considering that readers do not need to acquire the locks at all.**

***Hardware mechanism in the heart of each:***

**Both algorithms rely on atomic read-modify-update instructions, but MCS assumes a generally busy lock, while RCU assumes a lock acquired by mostly readers.**

2 Deadlocks

1. What is the difference between resource starvation and a deadlock?

**Resource starvation is occupying a resource for long, then eventually giving it away (when it’s finished) to the waiting process. A deadlock does not resolve itself as each process is dependent on the other.**

1. What are the four necessary conditions for a deadlock? Which of these are inherent properties of an operating system?

* **gjensidig utelukkelse: ressurs er ledig eller allokert til prosess. andre får ikke tilgang. mange er av denne typen**
* **hold på ressurs og vent: prosess kan holde på ressurs samtidig som den spør etter nye**
* **no pre-emption: ressurser kan bare frigis av prosessen selv. kun prosessen selv som kan frigi ressursen**
* **sirkulær venting: som i veikrysset**

3. How does an operating system detect a deadlock state? What information does it have available to make this assessment?

* Looks for cycles of resources and processes between each other.
* Uses descriptors to see what points to what.
* From there, it removes resources from a process, then gives it back as the deadlock resolves.
* Alternatively executes a rollback with a previous checkpoint for processes that often lock up.

3 Scheduling

1. Uniprocessor scheduling

1. When is first-in-first-out (FIFO) scheduling optimal in terms of average response time? Why?

**Because there is no overhead. Meaning if a given number of tasks need to be completed and order does not matter, FIFO will complete first.**

1. Describe how Multilevel feedback queues (MFQ) combines first-in-first-out, shortest job first, and

round robin scheduling in an attempt at a fair and efficient scheduler. What (if any) are its

shortcomings?

**MFQ uses multiple FIFO-queues. It implements SJF by putting new processes at the top level queue, but downgrades it if it takes too long. This way, shorter jobs are effectively higher priority. Round robin is used at the base level of the scheduling.**

2. Multi-core scheduling

1. Similar to thread synchronisation, a uniprocessor scheduler running on a multi-core system can be very inefficient. Explain why (there are three main reasons). Use **MFQ as an example.**

**MFQ, while viable on a uniprocessor, can cause issues when multiple cores are introduced. This is due to;**

**- Contention for scheduler spinlock arises.**

**- Cache slowdown because ready list data-structure pings between caches, due to it having to be moved between CPUs.**

**- Limited cache reuse; threads data from previous run is probably in an older cache from a previous core, therefore not quickly obtainable from a new core when it requires a memory read.**

1. Explain the concept of work-stealing.

**Work stealing is a scheduling strategy for multithreaded computer programs. I works by splitting work into threads. In other words, threads with nothing to do steals work from others. This results in more time-efficient execution.**