# 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 Synchronization**

**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.**

An example of communication between threads is inter process communication.

1. **How does this communication work?**

Inter process communication (IPC) is a mechanism which allows processes to communicate with each other and synchronize their actions. Processes can communicate with each other by shared memory or message passing. Both methods of communication can be implemented by the operating system.

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

With inter-process communication we can have problems like starvation, deadlocks, data inconsistency and shared buffer problems.

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

A critical region is a piece of code that only one thread can execute at once. We use this to protect the part of the program where the shared resource is accessed. Since only one thread can execute at a time, this thread will run till its finished, and will not be interrupted by other threads.

**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.**

With polling a process will continuously ask for access to the critical section while with blocking a process is suspended by the operating system and will be automatically notified when the critical section becomes available.

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

A race condition occurs when the operating system attempts to preform two or more operations at the same time where the output of a concurrent program depends on the order of operations between threads.

**Example:** you have $1,000 in the bank. You pay your rent which is $2,000 and received a payment of $10,000 from work. However, due to a race condition, you are short of $1,000 to pay your rent.

This happened because Bank staff A takes the current value of $1,000 and adds $10,000 to it, while bank staff B takes the current value of $1,000 and subtracts $2,000 from it. Bank staff A updates the value to $11,000. Bank staff B updates the value to -$1,000.

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

A spinlock is a lock where the processor waits in a loop for the lock to become free. We use spin-locks when threads are likely to be blocked for short periods. They are used to avoid overhead from operating system process rescheduling or context switching.

**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?**

With thread synchronization in multi-core architectures we can have the issue of a lock becoming a bottleneck. This happens because a large number of threads all try to acquire the lock at the same time. This wastes CPU cycles since threads become blocked and the operating system spends more and more time switching between threads. Because of this the problem of deadlocks of threads occur. To help combat this we use lock algorithms like MCS and RCU.

# **2 Deadlocks**

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

A deadlock is when no process proceeds because they cannot get ahold of the resources they need to continue. Generally, this happens when a process holds resources which another process needs to continue, while the resources that the original process needs is acquired by someone else. This causes a deadlock where none of the threads can continue. Starvation is when a process with low priority gets blocked in favor for a process with a higher priority, so it never gets the chance to run on the CPU.

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

**1. Bounded resources:** There are a finite number of threads that can simultaneously use a resource.

**2. No preemption:** Once a thread acquires a resource, its ownership cannot be revoked until the thread acts to release it.

**3. Wait while holding:** A thread holds one resource while waiting for another. This condition is sometimes called multiple independent requests because it occurs when a thread first acquires one resource and then tries to acquire another.

**4. Circular waiting:** There is a set of waiting threads such that each thread is waiting for a resource held by another.

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

The operating system analyzes a graph over what resources are allocated to the different threads, and what resources are needed to execute the different threads. For this the operating system needs information about resource allocation. It can also use a variation of Dijkstra’s Banker’s Algorithm to detect deadlocks.

# **3 Scheduling**

**1. Uniprocessor scheduling**

**(a) When is first-in-first-out (FIFO) scheduling optimal in terms of average response**

**time? Why?**

FIFO is optimal when you are working with small tasks. If a big task were to arrive before other smaller tasks, the system will seem inefficient since it will finish up the larger task before moving on to the smaller once (because of the FIFO principle). Which will make the response time seem longer than the optimal response time.

**(b) 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 round robin queues with different priority levels and time quantum, where tasks at a higher priority level preempt lower priority tasks. Tasks at the same level are scheduled in Round Robin fashion. Short tasks are favored over longer ones (SJF), and a new task enters as top priority (FIFO). Combining these MFQ is supposed to achieve responsiveness, low overhead, starvation-freedom, background task and fairness. MFQ unfortunately requires some means of selecting values for all the parameters to define the best scheduler, it is also the most complex scheme.

**2. Multi-core scheduling**

1. **Similar to thread synchronization, 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.**

If we were to use MFQ on a multicore system, we would have contention for scheduler spinlock, cache slowdown due to ready list data structure pinging from one CPU to another and limited cache reuse (thread’s data from last time it

ran is often still in its old cache).

**(b) Explain the concept of work-stealing.**

Work-stealing is a scheduling strategy where threads that have finished their own tasks can steal pending tasks from other threads. So instead of a core having no work it gets assigned a task from another processer´s overloaded queue.