OBL3-OS

August 12, 2020 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.

(a) Give an example of such communication.

An example of communication between processes would be in the case of shared memory. An example of shared memory could be when a new process is forked from a parent process.

(b) How does this communication work?

To use the same memory as another process, each process sets the same entry in its page table. If the shared memory is larger than the page size, multiple page table entries have to be the same.

(c) What problems can result from inter-process communication?

There could be a race condition, as in the "Too much Milk" example, where one process checks if the operation is done after the other process has checked, then both try to perform the operation, and one or both fails. Other examples could be starvation or deadlocks

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

A critical region is a region of a programs memory containing code that accesses a shared state and should not be interrupted while in progress. When the critical section is being executed a program should have a lock on the objects its operating on to ensure that no other process or thread tries to change the shared state at the same time.

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 would check if it can access the critical section once every set time interval. This takes a lot of unnessecary processing, so a better option could be using blocking. With blocking a process waits for a signal from the previous user, before accessing the critical section.

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

A well-known example would be Therac-25. Therac-25 relied on software checks to control patient doses of radiation. Unfortunately there was a race condition which allowed the dosage to be changed to a dangerous ammount after the check for a safe dosage had been completed. This could have been avoided if the proess checking the dosage had a lock on that memory section until the procedure was complete.

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

For multiprocesser systems, simply disabling interrupts for safe synchronization is insufficient. Because of this we use locks on objects, for example by setting the memory as read-only for all other parts of the system. A spinlock uses a test_and_set instruction to check in a short loop if a lock is available or not. This is efficient if locks are used for short periods of time, but if locks are held for long intervals at a time, the continuous polling can create a bottleneck.

6. List the issues involved with thread synchronisation 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?

If an object is particularly popular, the lock for that object can be a point of contention as many processes want to access it. These locks can become bottlenecks, so the MCS and RCU alghoritms try to address this.

MCS uses a spinlock that instead of being optimized for usually free locks, is optimized for usually busy locks. It mainly does this by decreasing the frequency of polling based on the contention for a lock. The hardware implemented compare_and_swap instruction to create a queue for a lock, with signals to the next in line when a lock is freed.

RCS is ideal for resources that are read often, but rarely written to. It has space for multiple readers and one writer. The writer has to publish its changes in an atomic operation, and the readers will either see the old or new version. Because of this multiple versions of the object must be stored. RCU uses the hardware implemented test_and_set instruction.

2 Deadlocks

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

Locks have to be aquired and released. This can lead to waiting for another processes lock before continuing. If this happens in a cyclic pattern, a deadlock is created. An example could be if thread 1 aquires lock 1, and then tries to aquire lock 2 before releasing anything, and thread 2 aquires lock 2 and then attempts lock 1 before releasing. They get stuck in a deadlock with thread 2 trying to aquire lock 1 and thread 1 trying to aquire lock 2.

Resource starvation simply means that a process is stuck for an infinite ammount of time. There are other kinds of starvation than deadlock, but deadlock always implies resource starvation.

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

The four necessary conditions are bounded (finite) resources, no preemption, waiting (for another resource) while holding a resource and circular waiting. Bounded resources is one of the inherent resources of an operating system, and is one of the reasons the operating system acts as an illusionist.

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

Deadlock detection can be done radically or conservatively. In the radical case, any program waiting in a way that could suggest a deadlock will be treated and fixed as a deadlock. With a more conservative policy, the operating system will make sure theres a deadlock before treating it as one.

The operating system uses a Resource Allocation Graph. This describes the free and allocated resources and which processes are waiting for which resources. If a deadlock exists, the system waits a set ammount of time to be sure and then kills one process to resolve the deadlock.

3 Scheduling

1. Uniprocessor scheduling

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

FIFO scheduling means that the object that has been in the cache the lonest is removed first to make space for new entries. This is the best option if all tasks are the same length, and only requires the CPU. FIFO is also very poor if there are slightly more tasks than rotating than fit in the cache.

(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 achieves responsiveness using SJF, low overhead with FIFO and starvation-freedom with RR. It also defers system maintenance tasks to not interfere with user tasks, and tries to be fair approximating each tasks fair share of the processor with max-min. If we were searching for an issue, it doesnt allow for more important tasks to be prioritized over every other tasks if necessary.

2. Multi-core scheduling

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

MFQ is created for uniprocessor scheduling. If it was used on a multicore system it could lead to issues like spinlock contention, cache slowdown and limited cache reuse. Spinlock contention is caused bu multiple processors trying to access the lock. Cache slowdown would with MFQ be caused by multiple processors trying to access a shared data scheduling structure. This takes time, because accessing the cache on another processor on cache misses takes much longer than keeping the data structure in one processors cache on uniprocessor systems. Limited cache reuse becomes a problem if a thread runs across multiple processors and the next instruction for that thread is still in the threads last used cache, causing a miss in the current processors cache.

(e) Explain the concept of work-stealing.

To prevent limited cache reuse, a thread remains on a single processor with affinity scheduling. This causes some processors to be idle while the others are working. Work-stealing occurs when an idle processor does the work of working processors to spead up their work.