Synchronization

Text

Description automatically generated

One example of this is when a shell starts other processes and kills them by using signals, for example UNIX kill command (SIGKILL).

With the use of signals (the one above). Other ways could be sockets, pipes, shared memory segments, UNIX sockets, memory mapped files, message queues, file locking (advisory locks), semaphores etc.

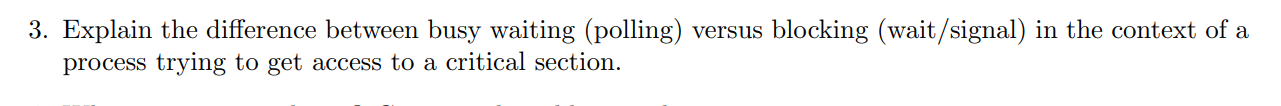


Race conditions could prove problematic if not handled appropriately. Mutexes (mutually exclusive flag), semaphores, or condvars (conditional variables) can be used to avoid this problem if race conditions arise within the same process. If the communication is between, let’s say an application and a server, the OS/kernel would need to take care of it. Message passing could then be used between the two to send messages that could then be used to avoid further complications or problems.

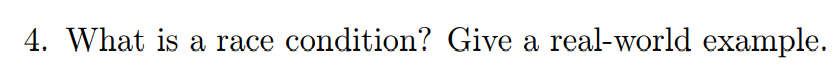


A critical section/region is a part of some code that operates on shared state (if two threads are operating on values of x for example). It is also important that the execution of a critical section should be atomic (as a single irreducible unit) to avoid race conditions, heisenbugs etc.

Therefore, a process cannot be interrupted while in a critical region, because of placed locks while executing atomic instructions.



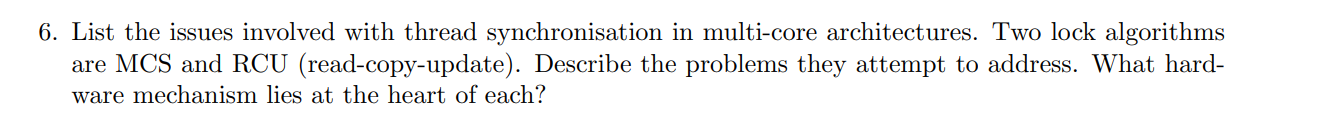
Polling is repeatedly checking for the state of interest’s availability. This results in the unnecessary use of the CPU. Blocking with wait/signal is a better and more efficient way of cooperation. With the use of signals/wait a process can get notified to start with signal() or wait in a queue for their turn with wait().



A race condition is when two processes/threads are working on the same shared data and the result of the execution is dependent on which order the instructions were executed. A real-world example of this is the “too much milk problem” where two threads compete in reading and updating the milk in the fridge.



A spin lock is a form of locking where a thread waits for its turn to access shared state by continually checking if its allowed (in a loop). This means that the operating system won’t need to preempt the thread. Because of this reason, spinlocks are used when gaining access to a lock is fast enough to not cause unnecessary delays or issues.



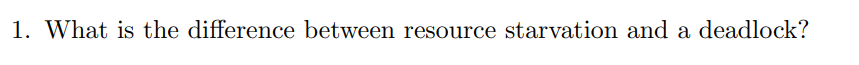
Making use of multiple cores could result in a higher lock contention. If there are multiple cores, then these cores must be used for efficiency. Since operating on shared state in the critical section on different cores is possible this could result in higher lock contention rate. This means that the whole process could slow down because of more back and forth communications for checking the availability of a lock.

RCU (read-copy update) is a mechanism used to avoid the use of lock primitives (thus waiting time can be greatly reduced). Updates by making copy and optimized for reading without lock. (Optimized for data read more often than written).

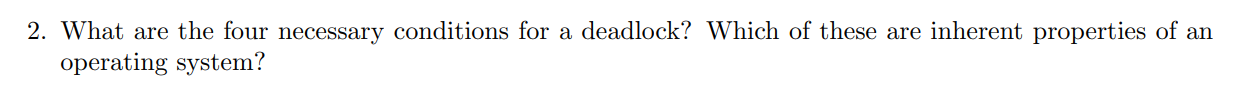
MCS spinlock is optimized for reducing processor bus traffic when trying to get a lock. Works by ordering lock-takers to reduce the amount of bus traffic.

Atomic compare and swap, atomic swap, atomic fetch and add, atomic fetch and subtract and mnemonics are useful macros that can be used to achieve this. This is because atomic read-modify-write instructions are supported on most processor architectures

# TASK 2



Resource starvation is when a process or a thread waits indefinitely trying to make progress, deadlock on the other hand is a form of starvation with the difference being that the starved processes are waiting for one another to progress (can be thought of as a waiting cycle ).

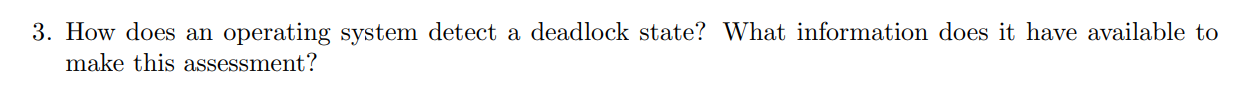


-bounded resources: finite numbers of threads that can simultaneously use a resource.

-no preemption: once a thread acquires a resource, the ownership of the resource cannot be revoked until the thread acts to release it. This means the kernel cannot take this privilege away.

-wait while holding: a thread holds one resource while waiting for another, this is called multiple independent requests because it occurs when a thread first acquires one resource and then attempts to acquire another resource.

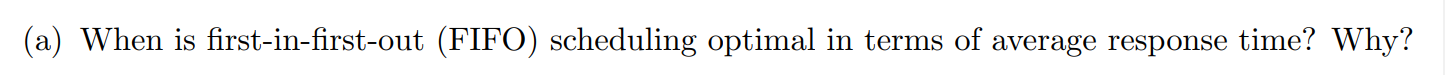
-circular waiting: set of threads waiting for a resource held by another.



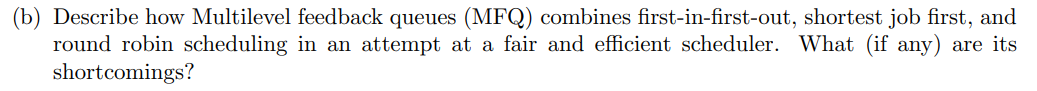
The OS has knowledge of which processes are requesting which resources, which processes are holding which resources and the thread state of each thread. This information can be used by the operating system to see if there is circular waiting, the other three criteria are already met by the design of most operating systems (for convenience).

# 3. Scheduling

1. uniprocessor scheduling



It is optimal when the tasks waiting to be executed only need the processor and if the variation in waiting time for each task is minimal. This is because if the task that came in front has a long waiting time and the tasks behind it, shorter waiting time then the average response time becomes much higher. Only thing that is left after serving the first task is context switching.

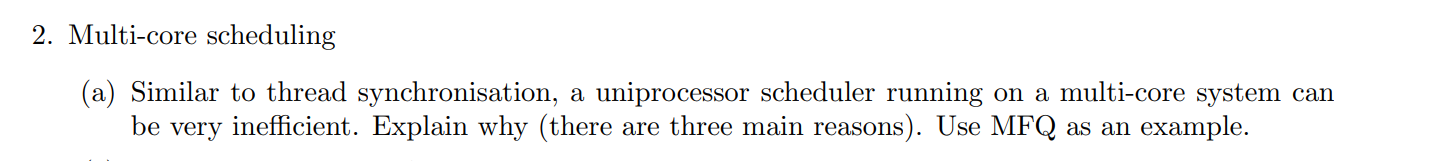


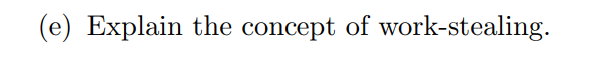
SJF (shortest job first): as the name suggests is prioritizing based on waiting time. Shortest job gets executed first.

Round robin: tasks take turns based on timer interrupts.

MFQ (Multi level feedback queue): Modern operating system use MFQ, by manipulating the assignments of tasks to priority queues, an MFQ scheduler can achieve responsiveness, low overhead, and fairness. It does this by combining FIFO, SFJ and round robin. A new task is added at the top of the priority (first in first out). If a task has a shorter time quantum than the CPU’s time quantum, then it will finish first (SJF). The idea of time quantum and priority queues comes from round robin.

Drawbacks for multiprocessor architectures.

 Drawbacks for multiprocessor architectures. Because it will increase **contention for the MFQ lock** as the number of processors increase. Another factor we must consider is **cache coherence overhead.** Since the cache where the MFQ is stored could be on another processor, retrieving remote data could make it twice as slow or worse, and since this happens while holding a lock this could worsen the bottleneck even more. This is also exacerbated by **limited cache reuse** (thread data displacement from the cache).



Work stealing is a way of dealing with idle cores, this way the workload can be balanced between the cores to improve efficiency.