Operating Systems Problem session 3

Dibran Dokter 1047390 & Marnix Lukasse 1047400

September 27, 2019

6.10

To do this the process that tries to acquire the lock while the resource is not available will be blocked and placed in a queue. The process that holds the resource needs to signal that the resource is released and take the first process from the queue and continue it. To save the process to the queue it should save its PCB.

6.14

a) Number of processes can be read by multiple processes at the same time. This can cause a race condition.
Number of processes can be incremented and decremented at the same time because there is nothing keeping it from doing so. This would cause a race condition.

b) To prevent these race conditions the functions acquire and release should be placed around the reading and increasing and decreasing of the number of processes.

c) Yes.

6.17

To implement wait() and signal() in processors using test_and_set we can use the following system which is equal.

We define a list of booleans. This is our lock for the different resources.

When wait is called we call the test_and_set method on the requested lock. If the lock is set to true, the resource is in use and we put the process in a waiting list. Then when the process using the resource sets the lock to false we resume the waiting process. This process sets the lock back to true when it enters the critical section.

When signal is called we release the resource by leaving our critical section and setting the lock to false. This allows the first waiting process in the list to be scheduled.

6.22

When we have a fairness policy the processes are seen as equal and the readers have to wait until the writer is done when a writer enters the queue. This causes a lower throughput because the readers could be reading at the same time during this time but instead they have to wait for the writer to finish.

To solve the reader/writer problem without causing starvation we copy the data and allow the writer to change that data while the readers are still reading the old data. When the writer is done writing we can redirect the new readers to the written data and when the readers are done reading the old data we remove the old data.

Another way to solve this problem is to implement an algorithm that has some characteristics similar to round-robin. When a reader is reading and a writer wants to write the writer waits for an opportunity (until no readers are reading). Normally this results in starvation. Now we could implement a time bound that the writer should wait. When this bound has elapsed we preempt the readers and write the data. Similarly we do this for the readers.

7.8

One process can never request more than the number of possible resources. So one process can be started at all times

Condition a. Makes sure that a single process cannot try to claim more resources than the system has. If this was possible this would most definitely cause a deadlock.

Condition b. The problem could arise that all processes are just missing one resource and all resources are taken. In this case the number of claimed resources would be maxNeeds - n (namely 1 less for every process.) To prevent this situation from occurring it follows that the number of available resources should always be bigger than maxNeeds + n because in this case the case where they all need one more resource cannot exist.

7.9

The request for a chopstick can be granted when the chopstick is not the last one, or if after taking the chopstick at least one philosopher has two chopsticks. P

7.10

The request for a chopstick can be granted when there are at least 3 chopsticks left.
When there are 2 chopsticks left we can take one if after taking it there is at least one philosopher with 2 chopsticks.
When there is only one chopstick left we can take one if after taking it there is at least one philosopher with 3 chopsticks.

7.12

a) The state is unsafe, there is no order in which the processes can finish. The bottleneck here is resource D, both P_0 and P_4 require 3 more or resource D. This while there is only 1 of resource D is available, and only 1 of resource D is held by another process. Therefore we will never be able to run P_0 or P_4

b) The state is safe, the process can complete in following order for example: $\{P_1, P_2, P_0, P_3, P_4\}$