Operating Systems

Opdracht 4

6.14

a) A process could call allocate_process() and another process could call exit() simultaniously, two things can happen then, the process counter is set one larger than the actual process count or one lower than the actual process count.
b) The locking and releasing needs to be done before and after each operation on number_of_processes. This means that locking needs to be done before the line:

   If (number_of_processes == MAX_PROCESSES)

   and

   --number_of_processes

   Releasing needs to be done before the lines:

   return -1;

   return new_pid;

   and after the line:

   --number_of_processes;

c) Yes

6.17

```
wait(boolean target) {
        while (!test_and_set(target) {
                add this process to list S;
                block();
        }
}

signal(target) {
        target = false;
        remove a process P from list S;
        wakeup(process P from list S);
}
```

6.22

Throughput is increased when we favor readers over writers. This way many readers can read the data in parallel while the writers have to wait because they can only perform their operation while they are the only writer writing to some data. This can cause starvation because when multiple readers keep reading data, the writers won't get a chance at writing. A possible solution could bet avoided by making a first in first out queue of readers and writers. When a writer finishes it's job, it wakes the first process in the queue to perform its operations. This way the process that has waited the longest time is executed first. When a reader wants to read some data and another reader is

already reading it can immidiately read data too, but if a writer is already waiting for the readers to finish, the reader is added to the queue to be executed after the writer. This would guarantee fairness and stop starvation from happening.

7.9

We can think of the chopsticks as if they are the resources and the philosophers as the processes. We can now say that if a philosopher makes a request for the chopsticks, we calculate:
Result = available chopsticks – 2

If result is bigger than or equal to zero, we can grant the philosopher acces to the chopsticks and give her two, then we need to update the available chopsticks before the next philosopher makes a request.

If the result is lower than zero, the philosopher has to wait for chopsticks to become free again.

7.10

Just as the anwser before, we can still check if the resulting amount of chopsticks is bigger than or equal to zero before we let the philosopher take the chopsticks. To do this we alter the check a little bit:

Result = available chopsticks – 3

If result is bigger than or equal to zero, we can grant the philosopher acces to the chopsticks and give her two, then we need to update the available chopsticks before the next philosopher makes a request.

If the result is lower than zero, the philosopher has to wait for chopsticks to become free again.