

# Operating System Lab Project **4** Report

Reza Abdoli

Mohammad Reza Alavi

Farbod Azimmohseni

Commit Id: a4d93625e67de95edebdaff8c9bfe94c663b1f32

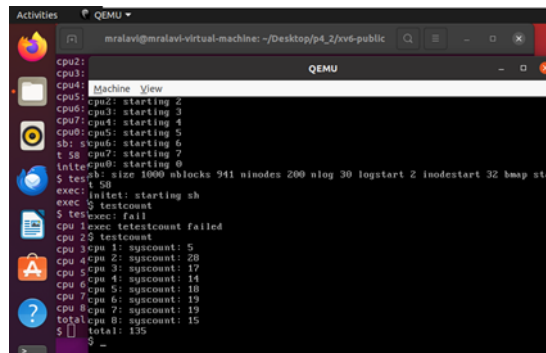
December 31, 2023

## Questions

1. If we do not disable interrupts, an Interrupt handler may try to access a variable but because some one else has acquired a lock, it will be blocked. But interrupts have high priority and need to run immediately. So we need to disable them. We will face deadlock if some processes wait for 2 different locks. Each acquiring one and waiting on the other.
2. `cli()` disables interrupts and `sti()` enables them. The `pushcli` function disables interrupts for that cpu. It has a `ncli` variables which keeps the number of times we have disabled interrupts and increases this value each time. The `popcli()` does the opposite thing. The difference is that `cli()` and `sti()` are assembly functions. These two are called inside `pushcli()` and `popcli()`. Also, we need other informations that `pushcli()` and `popcli()` keep track of.
3. Because process with active spinlock is always in ready queue it decreases cpu utilization and Causes low performance. We can use sleeplock instead.
4. This instruction atomically swaps the value loaded from the memory and value of another register. Atomic instructions like this are used in lock implementations to ensure synchronisation of them.
5. In xv6 if a sleeplock is available the process will succesfully acquire it. Otherwise it need to wait. Instead of busy waiting we will put it in the wait queue until lock is available again. A process might get waked up when another process releases the lock.
6. Process are either in ready queue or wait queue(there might be other queues based on type of the os). `yield` function on each tick of the clock will call `sched`. `sched()` does a context from the running process to the scheduler and make the scheduler run the next process.
7. **Semaphore** is a sleep lock in linux. When a task attempts to acquire a semaphore that is already held, the semaphore places the task onto a wait queue and puts the task to sleep.
8. Lock free methods rely on atomic instructions and hardware support to write a race free program. Challenge of lock-free programming might be lack of hardware support.

## Cache Coherency

1. We may need to set a control bit in order to check if value of a local cache is valid or not and synchronize data in each cache. This operation has memory overhead and will decrease performance.
2. Unlike semaphores and mutexes, **ticket lock** prevents starvation of processes with giving priority to the process by their arrive time.
3. Using a command `taskset -c` we can set a proces for a core.



```
cpu2: starting 2
cpu3: starting 3
cpu4: starting 4
cpu5: starting 5
sb: starting 6
t 58: starting 7
init:cpu0: starting 0
t 58: sb: size 1000 mblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap star
exec: 1 50
exec: init: starting sh
exec: $ testcount
$ testcount: fail
cpu 1 exec: testcount failed
cpu 2 $ testcount
cpu 3 cpu 1: syscount: 5
cpu 4 cpu 2: syscount: 28
cpu 5 cpu 3: syscount: 17
cpu 6 cpu 4: syscount: 14
cpu 7 cpu 5: syscount: 18
cpu 8 cpu 6: syscount: 19
cpu 9 cpu 7: syscount: 19
total:cpu 8: syscount: 15
$ total: 135
```

Figure 1: testcount user program

## Priority Lock

1. It may cause starvation of a process because a process with higher priority will always acquire the lock after releasing and reentering its critical section.
2. We may try to add a weight for number of times a process has acquired a lock and lower the priority of process dependently. This implementation avoids starvation of the process and also works with processes which have the same initial priority.
3. **Ticket lock** assigns a number for each process and each time a process releases the lock it will search for its next match. Because it increases the number of ticket value it has some kind of fairness and avoids starvation.

```
process with pid 8 released the lock
list of other people in list:
procces with pid: 9 has rank 1
procces with pid: 10 has rank 2
procces with pid: 11 has rank 3
-----process with pid 9 acquired the lock
done
process with pid 9 released the lock
list of other people in list:
procces with pid: 10 has rank 1
procces with pid: 11 has rank 2
-----process with pid 10 acquired the lock
done
process with pid 10 released the lock
list of other people in list:
procces with pid: 11 has rank 1
-----process with pid 11 acquired the lock
done
process with pid 11 released the lock
list of other people in list:
-----$
```

Figure 2: priority lock