UM–SJTU Joint Institute

Introduction to Operating Systems

(VE482)

Homework 4

Name: Yu Xiao ID: 518021910696
Date: Oct 29, 2021

Ex.1 – *Simple questions*

1. It is possible that when the clock interrupt occurs, the run-time system is at the point of blocking or unblocking a thread. This may lead to some unexpected behaviours.

   One solution to this problem could be setting a flag when the run-time system is entered, so that the clock handler can see this and set its own flag. When the run-time system finishes its works, it can check the flag of the clock handler and see if there is a clock interrupt. If clock interrupt happened, it can call the clock handler now.

2. It is possible to implement a thread package in user space in such a case, but the efficiency of such a thread package will not be good enough. Since there is no system call like `select`, we may set an alarm clock for each thread that want to execute a system call. If the call is blocked, the control would be returned back to the thread package.

Ex.2 – *Monitors*

The execution of `waituntil` will have to check the value of the bool variable repeatedly, which will waste more resources than using `wait` and `signal`.

Ex.3 – *Race condition in Bash*

1. The bash script is implemented as shown below.

```bash
#!/bin/bash
FILE=./ex3.txt
if [ ! -f $FILE ]; then
    echo 0 > $FILE
fi
for i in {1..20}
do
    value=$(tail -n 1 $FILE)
    value=$((value + 1))
    echo $value >> $FILE
done
```

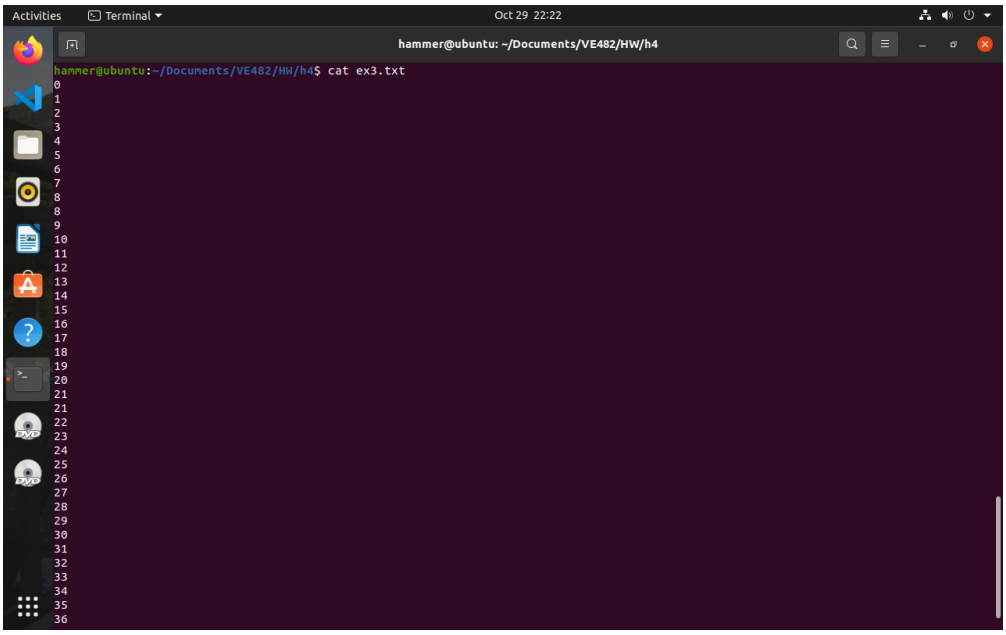
   Run the script with following command

```
./ex3_1.sh & ./ex3_1.sh
```

   In my own case, the race condition starts at the ninth line since there are two 8.

2. Modified version is shown below.

```bash
#!/bin/bash
FILE=./ex3.txt
if [ ! -f $FILE ]; then
    echo 0 > $FILE
fi
for i in {1..20}
do
    (
    flock -n -x 9
    value=$(tail -n 1 $FILE)
    value=$((value + 1))
    echo $value >> $FILE
    ) 9>>$FILE
done
```



Ex.4 – *Programming with semaphores*

For this exercise, please refer to `./h4/cthread.c`