

---

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.

```
#!/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
```

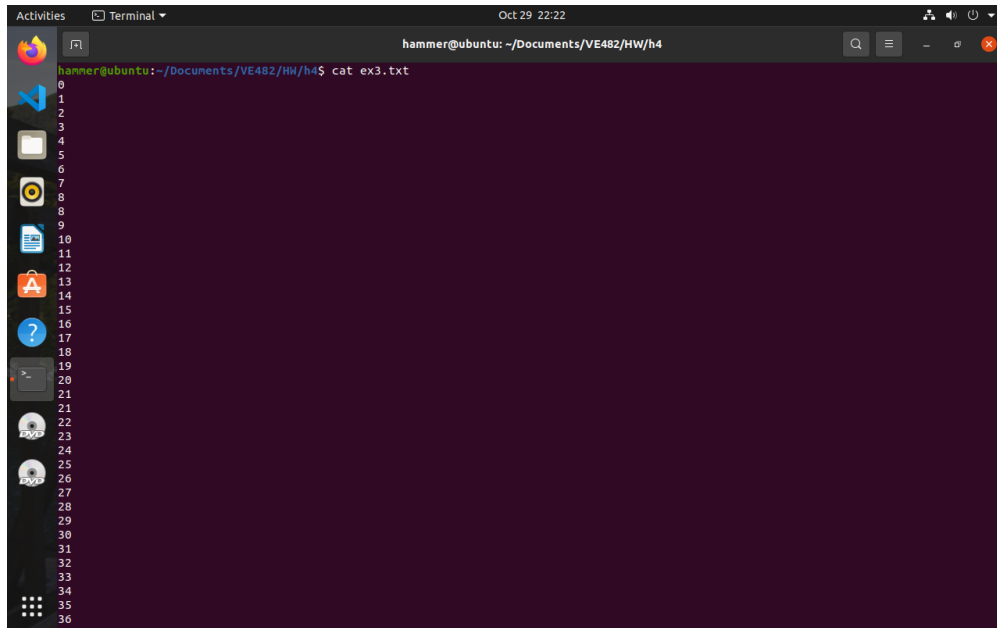
Run the script with following command

```
./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.

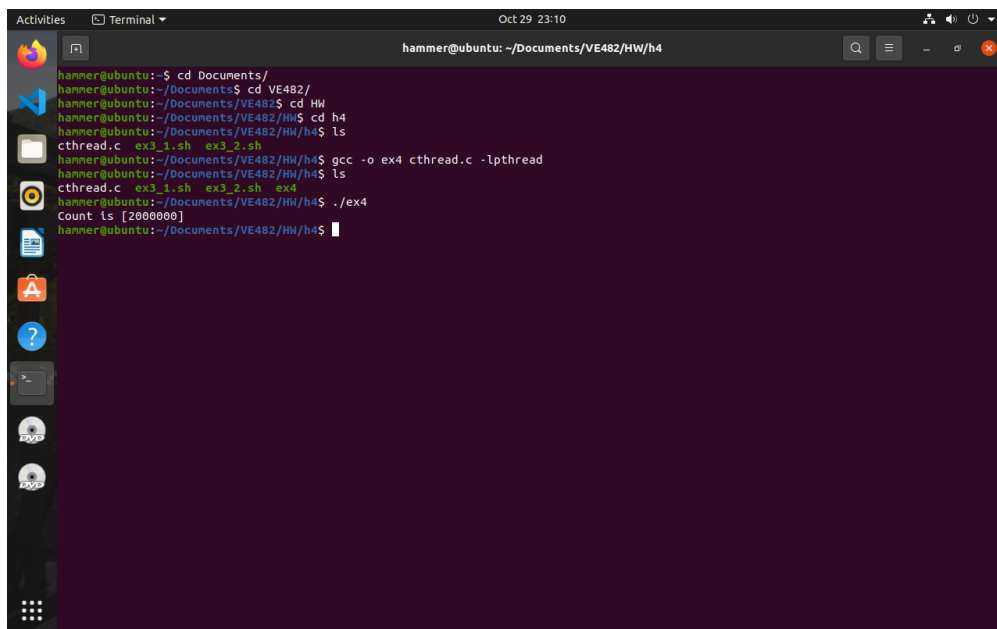
```
#!/bin/bash
FILE=./ex3.txt
if [ ! -f $FILE ]; then
    echo 0 > $FILE
fi
for i in {1..20}
do
    (
        flock -n -x 9
        value=$(tail -n 1 $FILE)
        value=$((value + 1))
        echo $value >> $FILE
    ) 9>>$FILE
done
```

A terminal window titled 'hammer@ubuntu: ~/Documents/VE482/HW/h4' with a timestamp of 'Oct 29 22:22'. The terminal shows the command 'cat ex3.txt' being executed. The output is a list of numbers from 0 to 36, with some numbers repeated: 0, 1, 2, 3, 4, 5, 6, 7, 8, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36. The terminal has a dark purple background and a sidebar on the left with various application icons.

```
hammer@ubuntu:~/Documents/VE482/HW/h4$ cat ex3.txt
0
1
2
3
4
5
6
7
8
8
9
10
11
12
13
14
15
16
17
18
19
20
21
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
```

#### Ex.4 – Programming with semaphores

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

A terminal window titled 'hammer@ubuntu: ~/Documents/VE482/HW/h4' with a timestamp of 'Oct 29 23:10'. The terminal shows a series of commands to compile and run a program. The commands are: 'cd Documents/', 'cd VE482/', 'cd HW', 'cd h4', 'ls' (showing 'cthread.c', 'ex3\_1.sh', 'ex3\_2.sh'), 'gcc -o ex4 cthread.c -lpthread', 'ls' (showing 'cthread.c', 'ex3\_1.sh', 'ex3\_2.sh', 'ex4'), and './ex4'. The output of the last command is 'Count is [20000000]'. The terminal has a dark purple background and a sidebar on the left with various application icons.

```
hammer@ubuntu:~$ cd Documents/
hammer@ubuntu:~/Documents$ cd VE482/
hammer@ubuntu:~/Documents/VE482$ cd HW
hammer@ubuntu:~/Documents/VE482/HW$ cd h4
hammer@ubuntu:~/Documents/VE482/HW/h4$ ls
cthread.c  ex3_1.sh  ex3_2.sh
hammer@ubuntu:~/Documents/VE482/HW/h4$ gcc -o ex4 cthread.c -lpthread
hammer@ubuntu:~/Documents/VE482/HW/h4$ ls
cthread.c  ex3_1.sh  ex3_2.sh  ex4
hammer@ubuntu:~/Documents/VE482/HW/h4$ ./ex4
Count is [20000000]
hammer@ubuntu:~/Documents/VE482/HW/h4$
```