VE482 — Introduction to Operating Systems Homework 4

Xu Shengyuan 518370910200

Ex. 1 — Simple questions

1.1

It can happen that at runtime the system is in the position of blocking or unblocking threads and is busy operating on the scheduling queue. This would be a very inopportune time for a clock interrupt handler to start checking these queues to determine if it is time for a thread switch, as they might be in an inconsistent state. One solution is to set a flag when entering the runtime system. The clock handler sees this, sets its own flag, and returns. When the runtime system is finished, it checks the clock flag, sees that a clock interrupt has occurred, and then runs the clock handler.

1.2

Yes, it's possible but inefficient. The thread that wants to execute the system call first sets an alarm timer and then executes the call. If the call blocks, the timer returns control to the thread package. Of course, most of the time the call does not block and the timer must be cleared. Therefore, each system call that might block must be executed as three system calls. If the timer turns off early, all sorts of problems can occur. This is not an attractive way to build threaded packages.

Ex. 2 — Monitors

`waituntil` costs much more resources. It will check the value every time a variable changes, while, on the contrary, the conbination of `wait` and `signal` will only awaken the process with the signal.

Ex. 3 — Race condition in Bash

3.1

`race.sh`:

```
#!/bin/bash

FILE=./ex3.out

if ! test -f "$FILE"; then
    echo 1 >> $FILE
fi
for i in `seq 1 100`
do
    LASTNUM=$(tail -n 1 $FILE)
    LASTNUM=$((LASTNUM + 1))
    echo $LASTNUM >> $FILE
done
```

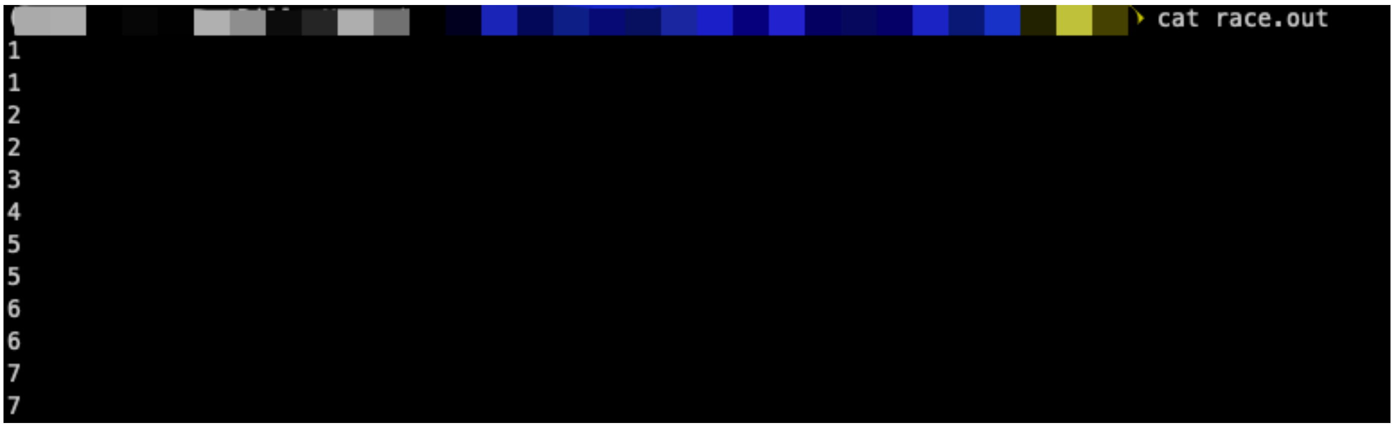
`race_driver.sh`:

```
#!/bin/bash
./ex3.sh &
./ex3.sh

sleep 3s

exit 0
```

The race condition almost happens immediately.



```
[redacted] › cat race.out
1
1
2
2
3
4
5
5
6
6
7
7
```

3.2

`no_race.sh`

```
#!/bin/bash

FILE=./no_race.out

if ! test -f "$FILE"; then
    echo 1 >> $FILE
fi
for i in `seq 1 100`
do
    flock -n -x 33
    if [ $? -eq 1 ];
    then
        exit;
    fi
    LASTNUM=$(tail -n 1 $FILE)
    LASTNUM=$((LASTNUM + 1))
    echo $LASTNUM >> $FILE
done
```

`no_race_driver.sh`

```
#!/bin/bash
./no_race.sh &
./no_race.sh

sleep 3s

exit 0
```

Ex. 4 — Programming with semaphores

modified `cthread.c`:

```
#include<stdio.h>
#include<stdlib.h>
#include<pthread.h>

#define N 1000000

int count = 0;
```

```c
sem_t sem; // +++

void * thread_count(void *a) {
  int i, tmp;
  for(i = 0; i < N; i++) {
    sem_wait(&sem); // lock
    tmp = count;
    tmp = tmp+1;
    count = tmp;
    sem_post(&sem); // unlock
  }
  return NULL;
}

int main(int argc, char * argv[]) {
  int i;
  pthread_t *t=malloc(2*sizeof(pthread_t));
  sem_init(&sem,0,1); // +++
  for(i=0;i<2;i++) {
    if(pthread_create(t+i, NULL, thread_count, NULL)) {
      fprintf(stderr,"ERROR creating thread %d\n", i);
      exit(1);
    }
  }
  for(i=0;i<2;i++) {
    if(pthread_join(*(t+i), NULL)) {
      fprintf(stderr,"ERROR joining thread\n");
      exit(1);
    }
  }
  if (count < 2 * N) printf("Count is %d, but should be %d\n", count, 2*N);
  else printf("Count is [%d]\n", count);
  pthread_exit(NULL);
  free(t);
  sem_destroy(&sem);// +++
  return 0;// +++
}
```