VE482 - Homework 4

Introduction to Operating Systems

Kexuan Huang 518370910126

October 24, 2021

Ex.1 Simple Questions

1. What problem might occur? Can you suggest a way to solve it?

- When the runtime system is about to unblocking or blocking a thread, changing the scheduling queues, which is in a highly inconsistent state, a clock interrupt may result in some unexpected behaviours that may cause an error.
- The solution could be setting a flag when runtime of the system is entered, where the clock handler can see this and can set the flag, then return to its original position. After the runtime system has finished its working like aforementioned state transition, it will inspect the clock flag and check whether a clock interrupt has occured, and then run the clock handler.

2. Is it possible to implement a threads package in user space under these conditions? Discuss.

Yes, it's still possible to implement a threads package in user space under these conditions. `select` system call can tell in advance whether it's safe for I/O operations, but if it's unavailable in certain OS, the alarm clock interrupts the thread which is blocking the system. Similarly, all other threads can be interrupts for a thread to do I/O. However, the efficiency might be reduced in this scenario.

Ex.2 Monitors

For only `waituntil` operation, the boolean refering to `waituntil` has to be checked by the system constantly whenever any varible changes, which is a waste of computing resources. For `wait` and `signal`, the system only awakes the process by `signal`, as a result, less resources are consumed.

Ex.3 Race Condition in Bash

Please refer to `ex3/` for more details.

Write a Bash script which generates a file composed of one integer per line.

```
#!/bin/bash
FILE=./race.out
if ! [ -s $FILE ]; then
    echo 0 >> $FILE
fi
for i in {1..100} do
    number=$(tail -n 1 $FILE)
    ((number++))
    echo $number >> $FILE
done
```

Run the script in both background and foreground at the same time.

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

1. How long does it take before observing a race condition?

The race condition occurs in line 12 and 13, they are both 11.

2. Modify the script such as to prevent the race condition.

```
#!/bin/bash
FILE=./race_free.out
if ! [ -s $FILE ]; then
    echo 0 >> $FILE
fi
for i in {1..100} do
    (
        flock -n 200
        number=$(tail -n 1 $FILE)
        ((number++))
        echo $number >> $FILE
    ) 200>> $FILE
done
```

Ex.4 Programming with Semaphores

Please see `ex4/` for more details.

```c
#include <pthread.h>
#include <semaphore.h>
#include <stdio.h>
#include <stdlib.h>
#define N 1000000
int count = 0;
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
  }
}
int main(int argc, char *argv[]) {
  int i;
  pthread_t *t = malloc(2 * sizeof(pthread_t));
  if (sem_init(&sem, 0, 1) != 0) {
    fprintf(stderr, "ERROR initializing semaphore\n");
    exit(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;
}
```