ECE482 — Introduction to Operating Systems

Homework 4

Manuel — UM-JI (Fall 2022)

Non-programming exercises:
- Write in a neat and legible handwriting
- Clearly explain the reasoning process
- Write in a complete style (subject, verb and object)

Progamming exercises:
- Write a single README file per homework
- Push to git and create a release with tag h4

ECE4821: submit together with ECE4820

ECE4820 Exercises

Ex. 1 — *Simple questions*

1. Consider a system in which threads are implemented entirely in user space, with the run-time system getting a clock interrupt once a second. Suppose that a clock interrupt occurs while some thread is executing in the run-time system. What problem might occur? Can you suggest a way to solve it?
2. Suppose that an operating system does not have anything like the `select` system call (`man select` for more details on the command) to see in advance if it is safe to read from a file, pipe, or device, but it does allow alarm clocks to be set that interrupt blocked system calls. Is it possible to implement a threads package in user space under these conditions? Discuss.

Ex. 2 — *Race condition in Bash*

Write a Bash script which generates a file composed of one integer per line. The script should read the last number in the file, add one to it, and append the result to the file.

1. Run the script in both background and foreground at the same time. How long does it take before observing a race condition?
2. Modify the script such as to prevent the race condition.

Ex. 3 — *Programming with semaphores*

The following C code creates two threads which increment a common global variable. When run it generates a random and inaccurate output. In order to solve this problem we want to use semaphores.

1. On Linux, find the file *semaphore.h*.
2. Read the documentation to understand how to use the functions described in the file *semaphore.h*.
3. Using semaphores adjust the program such as to always return the correct answer.

cthread.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#define N 1000000
int count = 0;
```

```c
void * thread_count(void *a) {
  int i, tmp;
  for(i = 0; i < N; i++) {
    tmp = count;
    tmp = tmp+1;
    count = tmp;
  }
}
int main(int argc, char * argv[]) {
  int i;
  pthread_t *t=malloc(2*sizeof(pthread_t));
  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);
}
```

ECE4821 Exercise

Ex. 4 — *Monitors*

During the lecture monitors were introduced (3.30). They use condition variables as well as two instructions, `wait` and `signal`. A different approach would be to have only one operation called `waituntil`, which would check the value of a boolean expression and only allow a process to run when it evaluates as `True`. What would be the drawback of such a solution?