**COMPSCI 377: Operating Systems**

Homework 2

Due: Monday, Feb 6, 2016 at 4pm

1. List the sequence of state transitions for the following program.

main(){

int i=1;

while(i<5){

i = i+1; //increment i

}

cout << i; // print i

while(i>0){

i = i-1; //decrement i

}

cout << i; // print i

}

new -> ready -> running -> waiting -> running -> waiting -> running -> terminated

1. Run the program with the following flags: ./process-run.py -l 5:100,5:100 What should the CPU utilization be (e.g., the percent of time the CPU is in use?) Why do you know this? Use the -c and -p flags to see if you were right. What about ./process-run.py -l 5:80,5:100?

The CPU Utilization should be 100%, because both processes have 100% of their instructions as CPU instructons.   
  
For 5:80,5:100, since two processes are being run, another process will be run in the same time that the first process is waiting on IO, so the utilization should be 100% CPU. It appears one IO instruction takes up 4 instructions worth in CPU time, as well.

1. Write pseudo code for a parent process to create a child process, and for the parent & child to each print their process ids and quit.

Pid\_t fork(void)

Main(){

int pid = fork();

if (pid == 0)

cout << getpid();

else

cout << getpid();

}

1. Run the fork/exec code shown in Fig 5.1(p1.c), Fig 5.2(p2.c), Fig 5.3(p3.c). In each case, compile the code and run it. Observe what happens. Briefly (in 1-2 sentences) explain what each example program does and what output it produced.

P1.c prints the pid of the child and parent process that is created in one fork.

P2.c is the same as p1.c with the additional synchronization that the parent waits for the child to terminate before printing the parent pid.

P3.c is the same as P2.c, with the additional program invocation for (‘wc p3.c’) on the child process, which replaces the child process with wc and does not print the last line of the printf in the child process.

1. Examine the following file: /usr/include/asm/unistd\_32.h (Make sure you use the class VM) This file shows the numbers assigned to each linux system call, which is used to construct the sys call table. What is system call #4? How many system calls do you see in this file?

System call #4 is write. There are 377 system calls in the file.

1. Write pseudo-code to show how a program can take an input n, where n >= 1, and fork a child, which forks a grand-child and so on until n descendants are created. Each descendent should print “I am descendent # <i>” and quit. Each process should wait until its immediate child has terminated before it can print the above message and quit.

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/wait.h>

int main(int argc, char \*argv[]){

char \*end;

int n = strtol(argv[1], &end, 10);

if ( n < 1 )

return 1;

for(int i = 0; i < n; ++i){

if(fork() == 0){

printf("I am descendent %d\n", i + 1);

return 0;

}

else{

wait(NULL);

printf("I am descendent %d\n", i);

}

}

}