# Homework #1

## CMPSC 170 (divyagrawal)

Winter 2019

Due Date: Friday January 25, 2019 (in HW Box by 5PM)

Q1. Which of the following instructions should be privileged? Explain your answer.

* 1. Set value of timer.
  2. Read the clock.
  3. Clear memory.
  4. Issue a trap instruction.
  5. Turn off interrupts.
  6. Modify entries in device-status table.
  7. Switch from user to kernel mode.
  8. Access I/O device.

Q2. Early computers developed the concept of a master control program (pre-cursor to the operating systems) that required that main memory be shared between the master control program and the user program. One of the issues that arises in such an environment is to protect the memory region of the master control program from user program. What is the minimal hardware/software mechanism that is needed to provide this protection?

Q3. In the early era of computing, a notion of reentrant user program was widely in use. The idea of reentrant user program was that user programs could be swapped-out of the memory to the disk and be swapped-in when the necessary resource became available. Clearly, during this re-entry process, user programs were not guaranteed to occupy the same physical memory address space that they occupied before. Develop a hardware/software architectural solution to address this problem? Extend the scheme so that when multiple user programs are resident in memory, they are protected from each other.

Q4. Explain how the interrupt mechanism works in Unix? Use a clear and concise diagram.

Q5a. Develop a program to instantiate two processes (a parent and a child) and write signal handlers to send signals between the two processes. The parent process is a in loop alternating with a sleep command and sending a signal to the child. When the parent receives a signal from its child it prints: “RECEIVED SIGNAL FROM CHILD PROCESS <PID>””. The child process does the something similar. This exercise is to make you familiar with the signaling mechanism in Unix.

Q5b. Explore ways in which Unix signaling could be used not only to send a signal from one process to another but also send data from one process to the other (e.g., the string that should be displayed by the process receiving the signal).

Q6. Consider the following simple programs involving the Unix fork() and other related system calls. For each case, run the program both synchronously and asynchronously (i.e., as cmd and cmd&) multiple times and provide a clear narrative to explain the behavior of the programs.

**simfork1.c**

main()

{

int i;

printf("simpfork: pid = %d\n", getpid());

i = fork();

printf("Did a fork. It returned %d. getpid = %d. getppid = %d\n",

i, getpid(), getppid());

}

**simfork2.c**

main()

{

int i;

i = fork();

if (i == 0) {

printf("Child. getpid() = %d, getppid() = %d\n", getpid(), getppid());

sleep(5);

printf("After sleeping. getpid() = %d, getppid() = %d\n",

getpid(), getppid());

} else {

printf("Parent exiting now\n");

}

}

**simfork3.c**

#include <stdio.h>

int K;

main()

{

int i;

int j;

j = 200;

K = 300;

printf("Before forking: j = %d, K = %d ", j, K);

i = fork();

if (i > 0) { /\* Delay the parent \*/

sleep(1);

printf("After forking, parent: j = %d, K = %d\n", j, K);

} else {

j++;

K++;

printf("After forking, child: j = %d, K = %d\n", j, K);

}

}

**simfork4.c**

#include <stdio.h>

#include <fcntl.h>

#include <sys/types.h>

#include <unistd.h>

main()

{

int i;

int seekp;

int fd;

char \*s1;

char s2[1000];

fd = open("tmpfile", O\_WRONLY | O\_TRUNC | O\_CREAT, 0666);

s1 = "Before forking\n";

write(fd, s1, strlen(s1));

i = fork();

if (i > 0) {

sleep(1); /\* Delay the parent by one second \*/

s1 = "Parent";

} else {

s1 = "Child";

}

seekp = lseek(fd, 0, SEEK\_CUR);

sprintf(s2, "%s: After forking: Seek pointer = %d\n", s1, seekp);

write(fd, s2, strlen(s2));

}