Operating Systems

Assignment 2

Department of Mathematics and Computer Science,

Lehman College, the City University of New York, Spring 2017

Due by March 27 (Monday), 2017 *(submit answers as hard copy)*

[Q.1] Answer the following questions

(1) Explain the difference between DMA and programmed I/O as I/O handling mechanisms.
(2) Why do we need run time stack? Explain why user stack and kernel stack are needed for executing an application in user address space.
(3) Explain why we need memory hierarchy in modern computer systems.
(4) Discuss the pros and cons of microkernel architecture.
(5) Explain the role of message type in message based communication in a single system environment.
(6) Explain why contemporary operating systems prefer implementing 1:1 threading model to *m:n* threading model.
(7) What is the role of matchmaker in RPC environment? Why do you need marshalling/un-marshalling in RPC/RMI environment?
(8) It is sometimes difficult to achieve a layered approach if two components of the operating system are dependent on each other. Identify a scenario in which it is unclear how to layer two system components that require tight coupling of their functionalities.

[Q.2] Answer the following questions.

(1) Draw a diagram that shows the migration of a process during its life cycle.
(2) Explain two situations when a process moves from running state to wait state.
(3) Explain two situations when a process moves from running state to ready state.
(4) Explain two situations when a process moves from wait state to ready state.

[Q.3] Answer the following questions.

(a) Process contexts include memory regions such as heap, run time stack, code(text), and data. Explain which memory region stores the variables and parameters in the following program (i.e., variable1, variable2, variable3, x, y, z).

```
/* include statements */

int variable1 =10;

int main(void)
{
  int variable2, *variable3;
  variable3 = malloc (10);
  variable2 = 20;
  xyz(variable1, variable2);
  return 0;
}
```

```
void xyz(x, y)
int x;
int y;
{
        int z;
        z = x + y;
        printf("value of z = %d\n", z); //line A
}
```

(b) Show runtime stacks of the main function and xyz function when a statement in line A is executed.

[Q.4] Answer questions based on the following program

```
/* include statements ….*/

int main()
{
  int PID, pid, status;
  char buf[3];

  pid = getpid();
  printf("\n The current process ID is %d\n", pid);
  PID = fork();
  if (PID==0) {
    printf("\n The child process prints this message from here........\n");
    printf("Enter a value:\n");
     scanf("%s", buf);

    printf("The child got the input buf...%s\n ", buf);
    printf("\nPID value in the child process = %d\n", PID);
    printf("…….%d\n", getpid());
    if (PID = fork() == 0) {
      printf("printout ...%d\n", getppid());
      exit(0);
    }
    printf("\n The child process id is %d\n", pid);
    printf("\n The child process now terminates");
  } else
    {
      printf("\n The parent process prints out this message from here........\n");
      pid = wait(&status);
      printf("\n The parent process waits until the child process terminates...", pid);
      printf("\nPID value in the parent process = %d\n", PID);
      pid = getpid();
      printf("\n The parent process id is %d\n", pid);
      printf("\n The parent process now terminates");
34      }
35   }
```

Let us assume the process id (PID) of the program above is 1500. The PIDs of the processes subsequently created are starting from the number 1501 (incremented by 1 whenever a new process is created). Note getppid() system calls returns the parent PID of the calling process.

(1) Identify the lines of code that will be executed by the child process created by the parent process.
(2) Explain the task done by the code segment (lines 19, 20, 21, and 22).
(3) After an input is requested by the process (see the statement 14), does the process hold the CPU continuously? Argue accordingly.
(4) What do we mean by orphaned process? Modify the above program to make the child process orphaned. Which process in UNIX becomes a parent process of an orphaned process?
(5) Show the output printed by statements 8, 18, 20, 32
(6) The parent process wants to launch firefox browser immediately after the child process terminates. Modify the program to do that. Let us assume the browser code firefox is located in the directory /usr/bin. (Note: remember execlp("/bin/ls", "ls", NULL) system call).

[Q.5] Answer the following questions based on the following program

```
/* include statements */

int value = 10;
int main()
{
  int pid = fork();
  if (pid == 0) {
    value += 20;
    printf("CHILD: value = %d\n", value); //line A
   }
  else if (pid > 0){
    wait(NULL);
    printf("PARENT: value = %d\n", value); //line B
   }
}
```

(1) What are the values printed out by the statements in line A and line B, respectively?
(2) If the printed values are different, explain why they are different.

[Q.6] Answer the following questions based on the following program.

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <stdlib.h>
```

```
int main()
{
   pid_t  pid;
   pid = fork();
   if (pid < 0) {
     printf("Some message...\n");
     exit(-1);
   }
   else if (pid == 0) {
     execlp("/bin/ls", "ls", NULL);
   }
   else {
    printf ("Another message\n");
    exit(0);
  }
}
```

(a) Let us assume the program is saved in a file named `q1.c`. Show the command to compile `q1.c` and generate `q1` as an executable file in Linux. Use GNU c compiler.

(b) If you remove `#include <unistd.h>`, `#include <sys/types.h>`, and `#include <stdlib.h>` statements from `q1.c`. There will be compilation errors and/or warnings. Identify which statement(s) may cause the compilation problem.

(c) How many children processes are created? Why do we need to have `if (pid < 0)` clause in the program? Is the `if(pid<0)` clause executed by a child process? Argue accordingly.

(d) Explain why there is possibility of having an orphaned child process in `q1.c`. Modify the program (add one statement) in such a way that there will not be any orphaned child process in `q1.c`.

(e) We want to make the parent and child processes to cooperate through the variable `var1` as shown below. Suggest a way to share the variable `var1`, and explain steps required to do that (in pseudo-code)

```
.......
int var1 = 10;
int main()
{
  int pid = fork();
  if (pid == 0) {
    value += 20;
   }
  else if (pid > 0){
    wait(NULL);
   }
}
```

[Q.7] Answer the following questions

(a) Explain the difference between built-in commands and external commands in a system like Linux. What is a Linux shell command to check a command is built-in?

(b) Explain when we use `strace` command in Linux. Also explain when you want to use `ipcrm` command.
(c) Explain two important data that the client must have to receive a message from a server in message-based communication in a single system environment.
(d) Explain how two file streams enable communication between parent and child processes through a pipe.
(e) Explain two streams used by a Java client socket to send and receive messages to and from the server socket.
(f) Explain why task group id is introduced in NPTL.

[Q.8] Answer the following questions in the context of process synchronization.

(a) What is busy waiting? Under what situation(s) a lock (i.e., a simple semaphore) with busy waiting (e.g., spinlock) would be useful?
(b) Explain the structure of semaphore that does not have busy waiting.
(c) Explain three requirements to be a solution to the critical section problem.
(d) An airplane ticket reservation system requires a database table lookup procedure and the table has information about available seats and prices. The table (TBL) is a resource shared by the lookup procedure and update procedure. Note, while any look up procedure instance gets information from the TBL, update procedure instance cannot access TBL and vice versa. Based on semaphores, develop a pseudo code of the lookup procedure (simply refer to the table as TBL in your pseudo-code) that allows multiple instances of lookup procedure can access TBL concurrently without creating any synchronization problems.

[Q.9] What will be printed out by the statements X and Y of the program shown below?

```
/* include statements */

int value = 0;

void *runner(void *param); /* the thread */

int main(int argc, char *argv[])
{
int pid;
pthread_t tid;
pthread_attr_t attr; /* set of attributes for the thread */

pid = fork();
if (pid==0) {
/* get the default attributes */
pthread_attr_init(&attr);
/* create the thread */
pthread_create(&tid,&attr,runner,NULL);
/* now wait for the thread to exit */
pthread_join(tid,NULL);
printf("value = %d\n", value); //statement X
}
else if (pid > 0) {
    wait(NULL);
        printf("value = %d\n", value); //statement Y
}

/**
 * The thread will begin control in this function
 */
void *runner(void *param)
{
        value = 5;
        pthread_exit(0);
}
```