**CG2271 Real Time Operating Systems**

**Lab 6 - Processes in Unix**

**Answer Book**

|  |  |
| --- | --- |
| Member 1 Name: Vu Ngoc Quang | Member 1 Matric No: A0099878W |
| Member 2 Name: Prachi Singhal | Member 2 Matric No: A0103527N |

Question 1 (3 marks)

From my research on google, a "pid" is a data type that represents Process IDs. Whenever a command is issued in UNIX, it creates a new process. A process is an instance of a running program. The operating system tracks processes through a five digit ID number known as the pid or process ID. Each process in the system has a unique pid, which is always a non-negative integer. Pids eventually repeat because all the possible numbers are used up and the next pid rolls or starts over. At any one time, no two processes with the same pid exist in the system because it is the pid that UNIX uses to track each process.

It is useful because it can be used to terminate a frozen or a misbehaving program. This command makes it possible to end a program that cannot otherwise be stopped except by rebooting (i.e., restarting) the system. It enhances the stability and robustness of unix-like operating system.

The parent's parent is bash, which is a default UNIX Shell for the operating system.

Question 2 (2 marks)

The values of k between parent and child are independent because the parent and child have individual branches, as in the parent executes in the “if” branch whereas the child executes in the “else” branch.

Question 3 (3 marks)

The output is mixed because the delay for the parent process and the child process is different. Therefore, the print command for the parent process is called faster than the print command for the child process.

The child continues to produce output after the end of lab6a because the child outputs to the buffer. When the parent process ends and returns to shell, the data inside the buffer are still waiting to be printed out. Thus, even when return to shell, the output is still being printed out.

Question 4 (3 marks)

The changes I made were:

**if**(pid=fork())

{

printf("Process ID of child: **%d\n**", pid);

**for**(i=0; i<10; i++)

{

printf("Parent: i=**%d** k=**%d\n**", i, k);

k++;

sleep(PARENT\_DELAY);

}

wait(&status); // line inserted

}

We added the wait(&status) line. Previously, the program quit to shell after the parent process ended its execution. After inserting the wait line, the program has to wait for all the child processes to end before quitting to shell.

Question 5 (2 marks)

This is what I see on the screen: Parent sent message: Hello child! and 128

My single statement description is: The program creates a children process that writes the number 128 and the string “Hello child!” into the pipe and then reads those data from the pipe and prints to the screen.

Question 6 (2 marks)

The sizeof function returns a value of the data type size\_t, which is an unsigned integer type used to represent the sizes of objects. Hence, sizeof(pnum) and sizeof(cnum) returns the size of the integer objects, whereas sizeof(buffer) returns the length of the string object.

Question 7 (3 marks)

My completed code is attached below:

#include <stdio.h>

#include <math.h>

#include <time.h>

#include <stdlib.h>

#include <unistd.h>

#define NUMELTS 16384

**int** prime(**int** n)

{

**int** ret=1, i;

**for**(i=2; i<=(**int**) sqrt(n) && ret; i++)

ret=n % i;

**return** ret;

}

**int** main()

{

**int** data[NUMELTS];

**int** fd[2];

**int** countByChild=0, countByParent=0;

**int** i=0;

**int** buffer;

pipe(fd);

srand(time(NULL));

**for**(i=0; i<NUMELTS; i++)

data[i]=(**int**) (((**double**) rand() / (**double**) RAND\_MAX) \* 10000);

**if**(fork()) {

**for**(i=0; i<8192; i++) {

**if** (prime(data[i])) {

countByParent++;

}

}

wait(&countByChild);

close(fd[1]);

read(fd[0], &buffer, **sizeof**(buffer));

printf("Number of prime found by parent: **%d\n**", countByParent);

printf("Number of prime found by children: **%d\n**", buffer);

printf("Total number of prime found: **%d\n**", countByParent + buffer);

} **else** {

**for**(i=8192; i<NUMELTS; i++) {

**if** (prime(data[i])) {

countByChild++;

}

}

//printf("Number of prime found by children: %d\n", countByChild);

close(fd[0]);

write(fd[1], &countByChild, **sizeof**(countByChild));

}

**return** 0;

}