Sri Lanka Institute of Information Technology

IT2060 – Operating Systems and Systems Administration
Year 2, Semester 1 - 2024

Assignment Report
IT 23144408

Page of Content

Question 01	3
Question 02	5
Question 03	7
Question 04	9
Reference	12

Question 1:

Consider the following C program and answer the questions.

```c
#include <stdio.h>
#include <unistd.h>

for (int i = 0; i < 10; i++)
{
    if ((pid = fork()) < 0)
        // error
    else if (pid == 0)
    {
        function_A();
        return 0;
    }
    printf("process ID: %d \n", pid); // Line A
}

for (int i = 0; i < 10; i++) // Line B
    wait();
```

(i) How many new processes are created in the program? Justify your answer?

10 processes

- The fork() function is called 10 times, creating 10 new processes.
- Each fork() call generates a child process.
- The child processes execute function_A() and then terminate immediately due to the return 0 command.
- The child processes do not create any further child processes.
- If we consider another the scenario where the return 0 command is absent from the code, 1023 new processes would be created.

(ii) Which process, the parent or the child, executes function_A()? justify your answer?

Child Process

Always child process equel to 0 and parent id always get child process id it must be a positive number. In the question function_A() have under the if condition be a “else if (pid == 0) ” . In this condition check if the pid equal to the zero. Therefor function A is include under the child process.

(iii) Whose PID, the parent or the child, is printed in Line A? Justify your answer?

Child’s pid

The pid of the child process is printed in Line A. In the parent process, ’fork ()’ returns the pid of the child. This pid is printed by the printf statement. The child process does not reach this printf statement because it terminates immediately after calling ‘function_A()’.

(iv) What is the purpose of the for loop with the wait() in Line B?

There was a wait() function inside the line B for loop. This function ensure that the parent process waits for all of its child processes to complete before it terminates. The wait() system call makes the parent process block until one of its child processes finishes.

Question 2:

Consider the following C program and answer the questions.

```
int main ()
{
        for(i =0; i < K; i++) {
                pid=fork ();
        }
}
```

Assume that the variables i and pid, and constant K have been properly defined, and initialized. There are no syntax errors in the above code.

(i) For K=5, How many processes are in the memory when the program is executed?

2^5 = 32//

•Initial process (parent) starts the loop
•In the first iteration, fork() is called, creating 1 child process. Now there are 2 processes.
•In the second iteration, both the parent and the first child call fork(), creating 2 more processes. Now there are 4 processes.
•This pattern continues for 5 iterations.
•After the 5th iteration: 2^5=32 processes.
•the question asks for how many **processes** are created the correct answer is **32** processes.

(ii) Modify the above program so that only the parent process creates 3 child processes, and each newly created process calls a function CPU( ). In addition, make the parent process wait for each child's termination.

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

#define k 3

void cpu(){
    int pid = getpid();
    printf("Child Process %d\n", pid);
}

int main(){
    int i;
    pid_t pid;

    for(i = 0; i < k; i++){

        pid = fork();

        if(pid == 0){
            cpu();
            return 0;
        }
        else if(pid > 0){
             wait(NULL);
        }
        else{
            printf("fork failed.\n");
            return 1;
        }
    }
    return 0;
}
```

Question 3:

Consider the following program. Explain the meanings of every line in the code and mention the output in Line A?

```
int value = 40;
int main()
{
        pid_t pid;
        pid = fork();
        if (pid == 0) {
                value = value + 15;
        }
        else if (pid > 0)
        {
                value = value - 15;
                printf("PARENT: value= %d \n", value); //Line A
                wait (NULL);
        }
}
```

Answer:

int value = 40;

This line declares a global integer variable value and initializes it with 40. This variable is shared across both the parent and child processes, but each process gets its own copy after the fork().

Inside the main method,

pid_t pid;
pid = fork();

Here, **pid_t** is a data type used for process IDs. The **pid** variable will hold the process ID returned by the **fork()** function.

fork() is a system call used to create a new process. The newly created process is called the **child** process, and the process that called fork() is the **parent** process.

- If fork() returns **0**, it means the code is running in the child process.
- If fork() returns a **positive value**, it means the code is running in the parent process, and this is the PID of the child process.
- If fork() returns **-1**, it indicates an error occurred, and the fork failed

```
if (pid == 0) {
        value = value + 15;
}
```

This block of code is executed only in the child process because **pid == 0**. The child process modifies its own copy of value by adding 15, so value becomes 55 in the child process.

```
else if (pid > 0)  {
        value = value - 15;
        printf("PARENT: value= %d \n", value); //Line A
        wait (NULL);
}
```

This block is executed only in the parent process because pid > 0.
The parent process modifies its own copy of value by subtracking 15, so value becomes 25 in the parent process.
The printf() function outputs the current value of value, which is 25, followed by a newline. This output occurs in the parent process.
wait(NULL); makes the parent process wait until the child process has finished executing. This ensures that the parent does not exit before the child completes.

Output in line A :

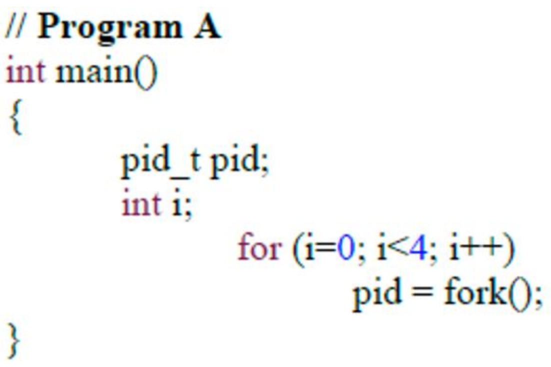
```
PARENT: value=25
```

Question 4:

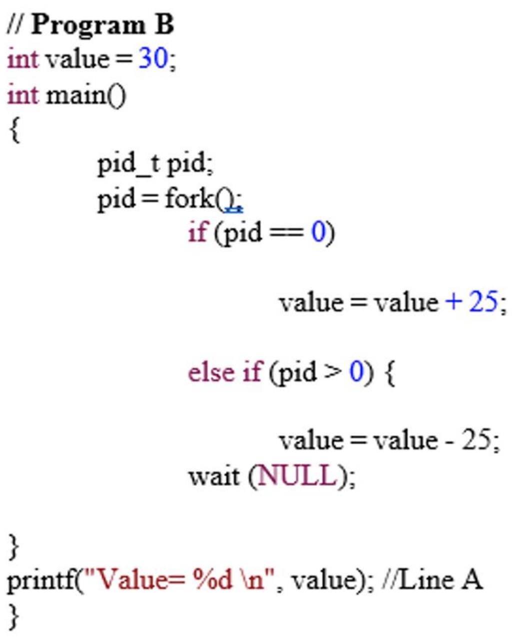
Consider the following programs A and B and answer the questions given below.

(i) How many times will the fork () function be called in Program A? (i.e., how many processes are created?) Justify your answer.

(ii) What is the output of Line A in Program B? Justify your answer.



```
// Program A
int main()
{
        pid_t pid;
        int i;
                for (i=0; i<4; i++)
                        pid = fork();
}
```



```
// Program B
int value = 30;
int main()
{
        pid_t pid;
        pid = fork();
                if (pid == 0)

                        value = value + 25;

                else if (pid > 0) {

                        value = value - 25;
                wait (NULL);

}
printf("Value= %d \n", value); //Line A
}
```

i) a. **fork() is called 15 times**
   b. **Total Process created : 16**

The fork() function creates a new process. Each time fork() is called, it creates a new child process that is a duplicate of the parent process.

The loop runs 4 times (i = 0 to i = 3).

Initially, there is one process (the parent).
In each iteration of the loop, each existing process will call fork(), doubling the number of processes.

After 1st iteration: 1 parent + 1 child = 2 processes
After 2nd iteration: 2 processes * 2 = 4 processes
After 3rd iteration: 4 processes * 2 = 8 processes
After 4th iteration: 8 processes * 2 = 16 processes

The total number of fork() calls is the sum of the processes created in each iteration: 1 (initial) + 1 + 2 + 4 + 8 = 15 new processes created.
Therefore, fork() is called 15 times.

A total of 16 processes will be created (15 children + 1 initial parent).

ii) **Value= 55**
Value= 5

In this program has fork() system call. This use to create new process. newly created process is called the child process, and the process that called fork() is the parent process.

If fork() returns **0**, it means the code is running in the child process.
If fork() returns a **positive value**, it means the code is running in the parent process, and this is the PID of the child process.
If fork() returns **-1**, it indicates an error occurred, and the fork failed

```
if (pid == 0) {
        value = value + 15;
}
```

This block of code is executed by the child process because **pid == 0**.
The child process increments value by 25, so value becomes 55 in the child process.

```
else if (pid > 0) {
        value = value - 25;
        wait (NULL);
}
```

This block is executed by the parent process because pid > 0.

The parent process decrements value by 25, so value becomes 5 in the parent process. (The child process and the parent process have separate memory spaces. Changing value in one process does not affect the value in the other process.)

The wait(NULL); call makes the parent process wait for the child process to terminate before proceeding

Therefor Line A print twice, because parent and child process are executing this line separately.
Firstly print child process printf function and after that print parent process printf function, because parent process have wait(NULL) function.

Reference

- https://www.geeksforgeeks.org/fork-system-call
- fork() and exec() System Calls (youtube.com)