Sri Lanka Institute of Information Technology

IT2060 – Operating Systems and Systems Administration

Year 2, Semester 1- 2024

Assignment Report

IT23222786

<Budara.V.P.R.>

Table of Contend

<u>Question 1: Consider the following C program and answer the questions.</u>

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

    int main()
    {
     pid_t pid;

     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++) //LineB
     wait();

     return 0;
     }
```

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

There are 11 processes total run in this program (parent and child both).
<u>10 child processes</u> are created.

Justification
In this program have 10 iterations and the loop runs 10 times. every time when loop condition is true Fork () system call function will called. So every time that loop is start to run fork () creates child process and also when the fork () function is called it will return 0 to child process and child's process 's PID to the parent process.

If (pid == 0) form this condition program will check whether pid of the program is equal to 0 so only the child process is will execute this code because child process pid is equal to 0. Each child process executes all the statements inside the if condition. Hence, It excites the function_A and the return 0 so **all the child process will immediately terminates after executing return 0 therefore system only create 10 new process because there are 10 iteration so fork() call only 10 times.**

- **Only 10 new process will create in this program**

- If there no return 0 inside the else if condition

```
else if (pid == 0)
      {
function_A();

   }
```

In this above program there is a fork () system call within a loop. The loop has 10 iterates. So each call to fork () creates a new child process. As this is a for loop number of processes are double with each iteration.

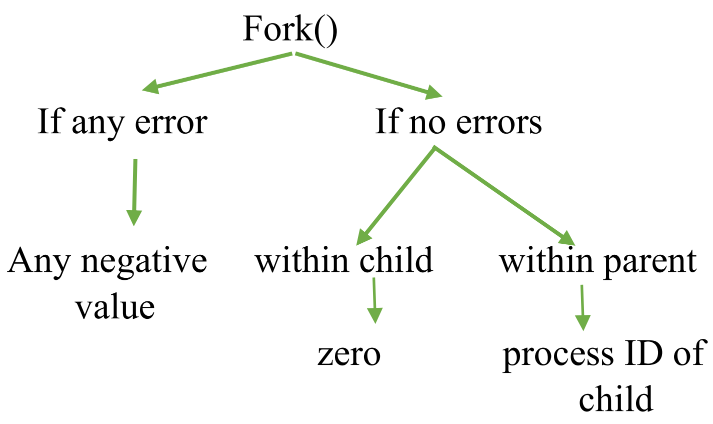
- At the fist parent process starts the loop as i = 0.
- In the first iteration, fork () is called and creating 1 child process.
  Now the total number of processes = 2
- Then in the 2nd iteration both parent and first child call fork () then creating two more processes
  Now the total number of processes = 4
- In the **10th iteration** there are **1024 processes** with the parent process
  1024 – 1 = 1023 (child process)

We use this calculation for calculate the number of processes

Total number of processes = 2^n
= 2^{10} (10 for number of iterations)
= 1024
Total number of child process = 1024 – 1 (removing the parent process)

(ii) **Which process, the parent or the child, executes function A()? Justify your answer?**

- **Child process will execute the Function A**



Always fork () returns child process ID in parent process and in the child process fork() return 0. In this program function_A is inside the condition of **if (pid == 0)** and it check whether pid equal to the zero. Therefor function A is only executing in the Child process. The parent process does not run this because pid of parent process equal to positive number.

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

```
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

}
```

- **The parent process prints the child's PID in Line A.**

The **fork ()** system call creates a new process. After **fork()** is called the parent process receives the child ‘s process ID and the child process receives 0 as the pid. In the return statement inside the **if (pid ==0)** block terminates all the child process after the execution of **function_A** because of that only the parent process will execute the **printf** statement   and the child process and does not reach the print statement.

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

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

The wait () system call function in Line B inside the for loop is use for ensures that the parent process waits for all its child processes to finish before terminating it.

<u>Question 2: Consider the following C program and answer the questions.</u>

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.

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

}
```

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

No of Valid iterations

$$\text{Toal no of Process} = 2^{\boxed{K}}$$

$$= \mathbf{2^5}$$

$$= \mathbf{\underline{\underline{32}}}$$

- At the fist parent process starts the loop as i = 0 .
- In the first iteration, fork () is called and creating 1 child process .

  Now the total number of processes = 2
- Then in the 2nd iteration both parent and first child call fork() then creating two more processes

  Now the total number of processes = 4
- likewise, the fork() function creates duplicate processes from all existing ones.

- At the 5th iteration **there are 32 process in the memory when executing the program** .

(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/wait.h>

void CPU()
 {
    printf("cpu function executed.\n");
 }

int main ()
{
   Pid_t pid;
   int k = 3;

   for(int i =0; i < k; i++)
   {
       pid=fork ();
       if (pid == 0)
       {
       //child process
       CPU();
       return 0;
       }
       else if (pid >0 )
       {
          Wait(NULL); // wait for child's termination
       }
        else
       {
         prinf("Fork failed.\n");
       }
    }
```

```
  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);
         }
}
```

1) **`int value = 40;`**

   Global integer variable call “value “ is declared and initialized to 40

2) **`int main()`**

   Begging of the main function.

3) **`pid_t pid;`**

   The pid_t is a predefined struct in C programming, used for storing process IDs for safety and readability. It's more suitable for storing variables meant for process IDs than int. variable pid of type pid_t is declared to store process IDs .

4) **`pid = fork();`**

   The fork() system call generates a child process. Fork() gives 0 to the child process and the child program ID to the parent process.

5) **`if (pid == 0)`**

The if-else control structure is being used. To filter out child processes, this program first checks if the pid variable value is zero.

6) **`value = value + 15;`**

in the child process, value = 40 becomes 55 by adding 15 **(value = 40 + 15,value = 55)**

7) **`else if (pid > 0)`**

check if the pid > 0 (pid is a positive value /current process is a parent process)

8) **`value = value - 15;`**

if current process is a parent process value = 40 becomes 25 by subtracting 15 **(value = 40 – 15, value = 25)**

9) **`printf("PARENT: value= %d \n", value);`** `//Line A`

prints the value of ‘value’ in the parent process (value = value – 15, value = 40 – 15 , value = 25)

`output - PARENT: value=  25`

10) **`wait (NULL);`**

The parent process wait till the child process to finish before terminating

<u>Question 4: Consider the following programs A and B and answer the questions given below.</u>

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

```
//program A
int main ()
{
   pid_t pid ;
   int i ;

   for (i = 0 ; i<4 ; i++)

         pid = fork ();

}
```

In this program there is a fork () function inside a for loop that iterates 4 times. Every time when the fork () is called, a new process is created (child process).

- First iteration – After the fork () is called then creating a child process
  Now total process = 2
- Second iteration – Both parent and child process call 'fork()'2 more child process are created
  Now total process = 4

- Third iteration – parent process and all other 3 child process are creating 4 more child process
  Now total process = 8

- Fourth iteration - parent process and all other 7 child process are creating 8 more child process
  Now total process = 16
  (1 parent process + 15 children processes)

- ❖ **After the every iteration, total number of process are coming double after the fork () self-call .**

- o There are four iterations in program A

No of
Valid iterations

Toal no of Process $= 2^{\boxed{K}}$

$= \mathbf{2^4}$

$= \mathbf{\underline{\underline{16}}}$

There are 16 run in this program and there are 15 new processes are created

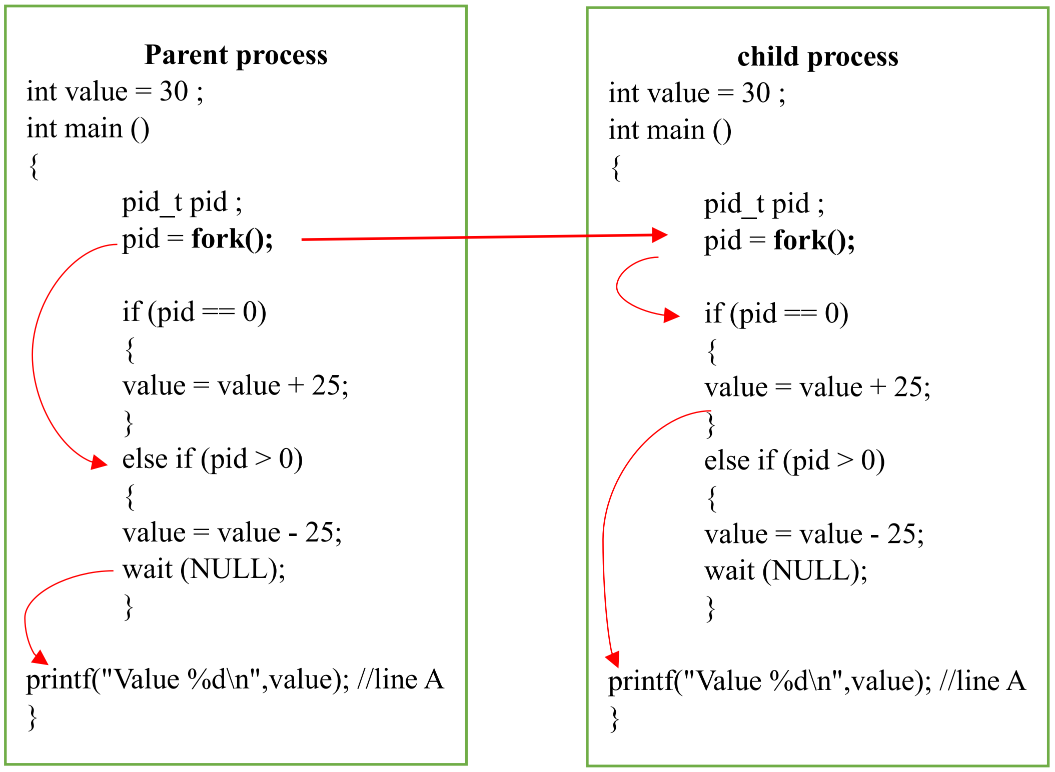
$\mathbf{2^4 - 1}$

$\mathbf{\underline{\underline{15}}}$

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

```
//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
}
```



```
Parent process
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
}
```

```
child process
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
}
```

Fork () return Child's PID
parent process

Child process PID

1st Step – goes to
If else block

2nd Step

Value -=25

3rd Step

Wait (NULL)
Wait until child process execute first.

4th Step

Printf()

5th Step output

Value 5

1st Step – goes to
if block

2nd Step

Value +=25

3rd Step

Printf()

4th Step output

Value 55

- Begging of the program B value variable equal to 30 (initialized) then the fork () creates a new process (child process).
- In the Child process
  If the pid == 0 condition is executed because the value of the child process's pid variable is equal to zero. Therefore, value (30) will **added 25** and the answer will be assigned to the value **(value += 25)**. Then reaches the Line A and print the value variable. **`(Value (30) + 25 = 55)`**
- In the Parent process
  The parent process will be executing the else if condition because parent process's pid is greater than zero. Therefore, value (30) will deducted 25 and the answer will be assigned to the value (value -= 25). In next statement the wait (NULL); call makes the parent process wait for the child process to terminate before proceeding. Then reaches the Line A and print the value variable. **(Value (30) - 25 = 5)**
- The printf command will be performed by both the parent and child processes since it is not inside the if or else code segments. As a result, it will publish the values assigned by both the parent and the child processes.
- The Output should be

```
Value 55
Value 5
```

References

- Lecture Slides of Lecture 03 and worksheet
- https://www.geeksforgeeks.org/fork-system-call
- https://www.gnu.org/software/libc/manual/html_node/Creating-a-Process.html