BSc (Hons) in Information Technology

Operating Systems and System Administration

Year 02 Semester 01

Sri Lanka Institute of Information Technology

IT2060 – Operating Systems and Systems Administration

Year 2, Semester 1- 2024

Assignment Report

Student ID - IT23193222

Table of content

Content **Pages**

Question 1: Consider the following C program and answer the questions.

```
#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++) //Line B
 wait();

   return 0;
 }
```

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

- **10 new processes** are created in the program.

The fork() function is called 10 times due to the for loop. Each time fork() is called, a new child process is created. The parent process and each newly created child process continue to execute the loop, resulting in a total of 2^10 or 1024 processes theoretically. However, due to the return 0; statement inside the child process block (when pid == 0), each child process returns and stops further execution of the loop after calling function_A(). Therefore, each iteration of the loop generates only 1 new process, leading to **10 new processes** created in total (one per iteration).

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

- **The Child process** executes function_A().

After 'fork()' is called, it return 0 to the child process and the process ID of the child to the parent process. The condition 'else if (pid = 0)' check if the current process is the child process. If true 'function_A()' is executed by the child process. The parent process does not execute 'functionA()' and instead continues to the next iteration of the loop.

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

- **The child's PID** is printed in Line A.

fork() return the child's process ID (PID) to the parent process. The statement 'printf("process ID: %d\n" , pid);' prints the value stored in 'pid'. Since 'pid' holds the child's PID in the process, the parent process print the child's PID. The child process never reaches Line because it return from the 'main()' function immediately after executing function_A().

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

- The purpose of the for loop with wait() in Line B is to **ensure that the parent process wait for all 10 child processes to terminate before it itself exits.**

Wait() is a system call that causes the parent process to wait for the child process to terminate. The loop iterates 10 times corresponding to the 10 child processes created in previous loop. This ensure that the parent process does not terminate until all of its child processes have finished executing.

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.

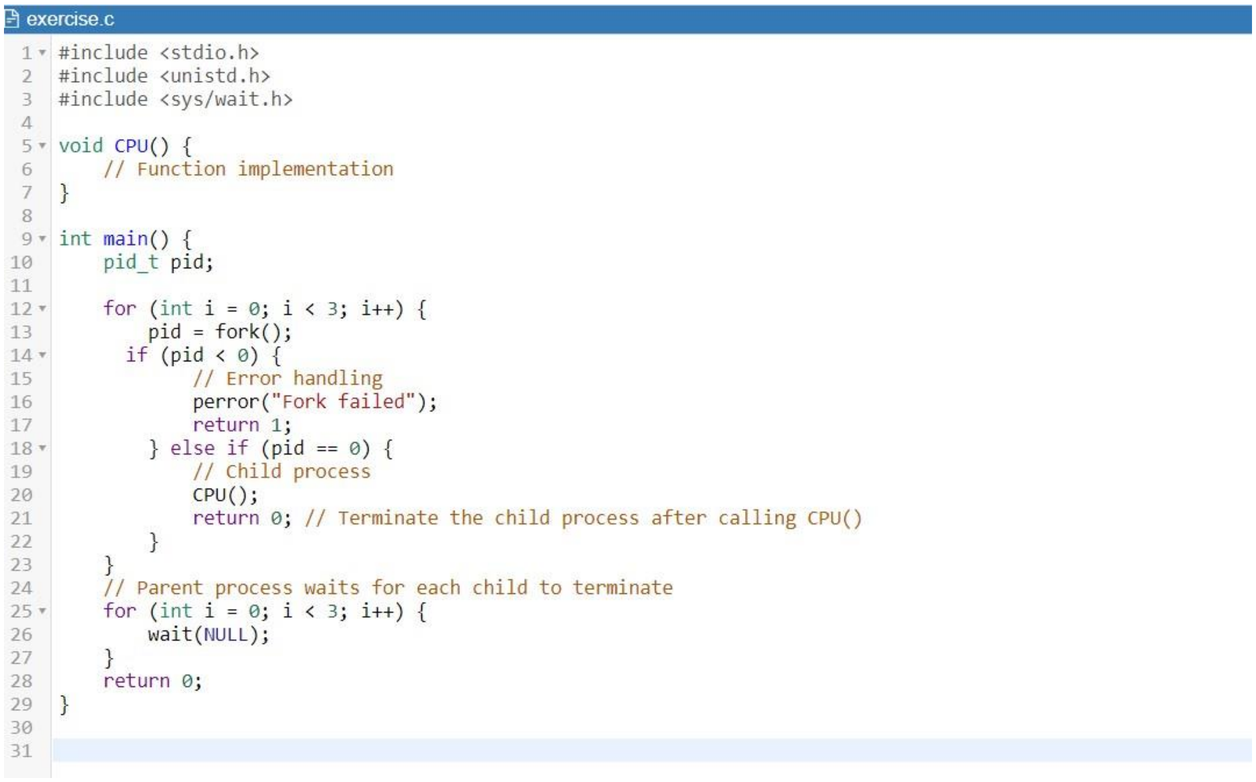
(i) For K=5, How many processes are in the memory when the program is executed?

The number of processes double with each iteration of the loop. For K = 5,

Total number of processes = 2^5 = 32

- **32 processes are in memory**

(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.



exercise.c

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

void CPU() {
    // Function implementation
}

int main() {
    pid_t pid;

    for (int i = 0; i < 3; i++) {
        pid = fork();
      if (pid < 0) {
            // Error handling
            perror("Fork failed");
            return 1;
        } else if (pid == 0) {
            // Child process
            CPU();
            return 0; // Terminate the child process after calling CPU()
        }
    }
    // Parent process waits for each child to terminate
    for (int i = 0; i < 3; i++) {
        wait(NULL);
    }
    return 0;
}

```

The parent creates exactly 3 child processes and each child calls the cpu() function. The parent waits for each child to finish before creating the next one.

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);
         }
     }
```

- Int value = 40; → A global integer variable value is declared and initialized to 40. The variable is shared by both the parent and child processes after the fork() system call.
- Int main() { → The main() function begins marking the entry point of the program.
- Pid_t pid; → A variable pid of type pid_t is declared. This variable will store the process ID returned by the fork() system call.
- Pid = fork(); → The fork() system call is invoked to create a new child process.
  - Int the child process, fork() return 0.
  - In the parent process fork() return the PID of the child process (a positive integer).
  - If fork() fails, it return '-1' .

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

  This block is executed only by the child process because pid == 0 in the child process.
The child process updates the value by adding 15, so the new value in the child process becomes ‘ 40 + 15 = 55’.

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

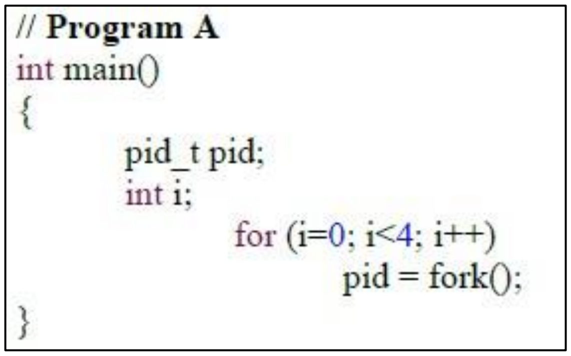
This block is executed only by parent process because pid > 0 . The parent process updates the value by subtracting 15, so the new ‘value’ in the parent process become 40 – 15 =25 . The printf statement at Line A output the current value of ‘value’ in the parent process to the console. Wait(Null); makes the parent process wait for the child process to finish executing.

- **Output at Line A**

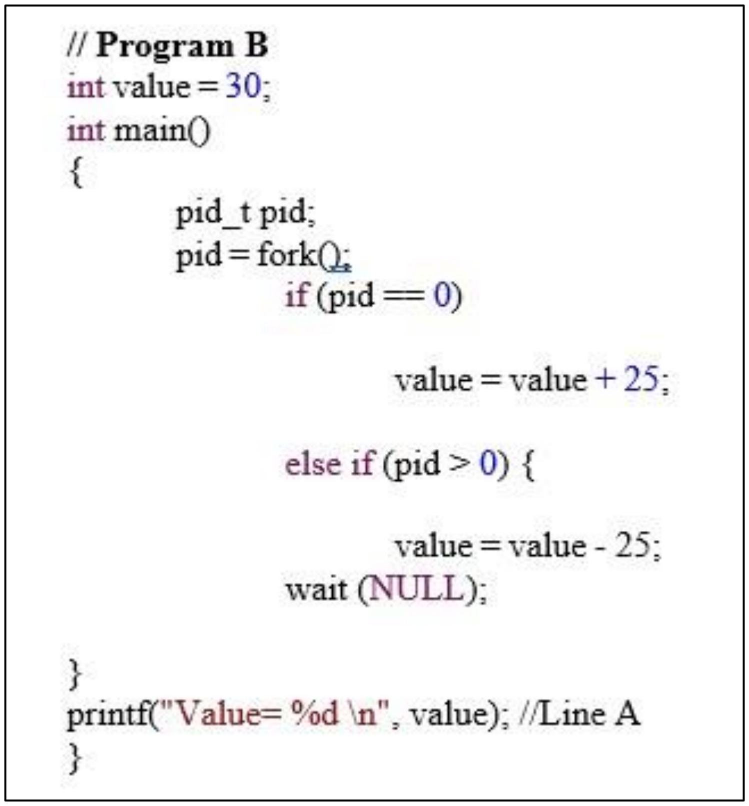
Parent process: When the printf statement at the Line A is executed by the parent process it output the current value of ‘value’ in the parent process, which is 25.

- Output → PARENT: value = 25

Question 4: Consider the following programs A and B and answer the questions given below.



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

16 processes are created.

The fork() function is called within a for loop that iterates 4 times. Each fork() call doubles the number of process.
The number of processes created can be calculate as follows:
Initial process (before any fork()) : 1 process.
After 1st fork() : 2 process.
After 2nd fork() : 4 process…..

So, the total number of process created after the loop is 2^4 = 16

- Since the original process also counts, and the fork happens after entering the loop, the total number of fork calls is 2^4 - 1 = 15.

However, **the fork function is called 15 times**, creating a total of 16 processes (including the original one).

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

When fork() is called it creates a child process. The pid in the child process is 0 while in the parent process it is a positive value.

In the child process: value = value + 25; result in value = 55.

In the parent process value = value – 25; result in value = 5.

Output :-

In the child process: value = 55
In the parent process: value = 5

References

Stack overflow - https://stackoverflow.com/questions/75522179/how-many-processes-are-created-in-this-c-programme

Chegg - https://www.chegg.com/homework-help/questions-and-answers/many-processes-created-program-shown-including-parent-process-include-include-int-main-int-q56658720

Quora - https://www.quora.com/What-is-the-number-of-child-process-created-by-fork-fork-fork

GeeksforGeeks - https://www.geeksforgeeks.org/fork-practice-questions/