DHA SUFFA UNIVERSITY
Department of Computer Science
CS-2004L
Operating Systems
Fall 2020

LAB 03 – Process Creation By using fork() and wait() system call

OBJECTIVE(S)

Learn about creating process using fork() and wait() system call

Process

- A computer program which is in execution is called a process.
- It is a dynamic entity.
- It needs resources, CPU or I/O allocated when created.

ps Command:

ps command is used to show what processes are currently running on your system.

For example, you might see the following output when you issue the command:

```
File  Edit  View  Search  Terminal  Help
ansha@ansha-VirtualBox:~$ ps
  PID TTY          TIME CMD
 2281 pts/1    00:00:00 bash
11498 pts/1    00:00:00 ps
ansha@ansha-VirtualBox:~$
```

Process ID:

A Linux or Unix process is running an instance of a program. For Example, Firefox is running a process if you are browsing the internet. Each time you start Firefox browser, the system is automatically assigning a unique process identification number (PID). A PID is automatically assigned to each process when it is created. To find out the PID of processes, the following commands can be used.

System Calls to get process IDs:

- getpid(): returns the process ID of the calling process.
- getppid(): returns the process ID of the parent of the calling process.

DHA SUFFA UNIVERSITY
Department of Computer Science
CS-2004L
Operating Systems
Fall 2020

Example 1:

```
#include <iostream>
#include <unistd.h>
int main()
{
        std::cout<<"I am process "<<getpid() << std::endl;
        return 0;
}
```

Output:

```
ansha@ansha-VirtualBox: ~/Desktop/Process
File  Edit  View  Search  Terminal  Help
ansha@ansha-VirtualBox:~/Desktop/Process$ g++ -o task1 Process_id.cpp
ansha@ansha-VirtualBox:~/Desktop/Process$ ./task1
I am process 2845
ansha@ansha-VirtualBox:~/Desktop/Process$
```

Process Creation

In Unix, the process is created by first duplicating the parent process. This is called forking, after the fork system call. In C++, you invoke the fork system call with the fork() function.

Fork()

It is a system call that creates a new process under the Unix Operating system. It takes no arguments. The purpose of fork() is to create a new process, which becomes the child process of the caller. After a new child process is created, both processes will execute the next instruction following the fork() system call. Therefore, we have to distinguish the parent from the child. This can be done by testing the returned value of fork().

Example 2: using fork() system call

```
#include <iostream>
#include<unistd.h> /*For Fork*/
int main()
{
        std::cout<<"Before Forking"<<std::endl;
        fork();
        std::cout<<"After Forking"<<std::endl;
        return 0;
}
```

Output:

```
File  Edit  View  Search  Terminal  Help
ansha@ansha-VirtualBox:~/Desktop/Process$ g++ -o fork Fork.cpp
ansha@ansha-VirtualBox:~/Desktop/Process$ ./fork
Before Forking
After Forking
ansha@ansha-VirtualBox:~/Desktop/Process$ After Forking
```

Example 3: using getpid() to get PID

```
#include <iostream>
#include<unistd.h> /*For Fork*/
int main()
{
        std::cout<<"0 Process Number = "<< getpid() <<std::endl;
        fork();
        std::cout<<"1 Process Number = "<< getpid() <<std::endl;
        return 0;
}
```

Output:

```
File  Edit  View  Search  Terminal  Help
ansha@ansha-VirtualBox:~/Desktop/Process$ g++ -o fork Fork.cpp
ansha@ansha-VirtualBox:~/Desktop/Process$ ./fork
0 Process Number = 2981
1 Process Number = 2981
ansha@ansha-VirtualBox:~/Desktop/Process$ 1 Process Number = 2982
```

Example 04 : Getting the complexity of fork()

```
#include <iostream>
#include<unistd.h> /*For Fork*/
int main()
{
        std::cout<<"0 Process Number = "<<getpid() <<std::endl;
        fork();
        std::cout<<"1 Process Number = "<<getpid() <<std::endl;
        fork();
        std::cout<<"2 Process Number = "<<getpid() <<std::endl;
        return 0;

}
```

Output:

```
ansha@ansha-VirtualBox:~/Desktop/Process$ g++ -o fork Fork.cpp
ansha@ansha-VirtualBox:~/Desktop/Process$ ./fork
0 Process Number = 3756
1 Process Number = 3756
2 Process Number = 3756
ansha@ansha-VirtualBox:~/Desktop/Process$ 2 Process Number = 3758
1 Process Number = 3757
2 Process Number = 3757
2 Process Number = 3759
```

Some abstract examples of fork() usages:
Your shell uses a fork to run the programs you invoke from the command line.

- Web servers like apache use fork to create multiple server processes, each of which handles requests in its own address space. If one dies or leaks memory, others are unaffected, so it functions as a mechanism for fault tolerance.
- Google Chrome uses a fork to handle each page within a separate process. This will prevent client-side code on one page from bringing your whole browser Down.
- Fork is used to spawn processes in some parallel programs (like those written using MPI). Note this is different from using thread, which don't have their own address space and exist *within* a process. Scripting languages use a fork indirectly to start child processes. For example, every time you use a command like subprocess. Popen in Python, you fork a child process and read its output. This enables programs to work together.
  (Courtesy: tgamblin – StackOverflow Community)

Output of the fork() system call:

- = 0 : – Child process successfully created
- > 0 :– i.e. the process ID of the child process to the parent process.
- < 0 :– Creation of child process was unsuccessful.

The returned process ID is of type pid_t. Normally, the process ID is an integer. Therefore, after the system calls the fork(), a simple test can tell which process is the child. Note that Unix will make an exact copy of the parent's address space and give it to the child. Therefore, the parent and child processes have separate address spaces.
If the call to fork() is executed successfully, Unix will:

- Make two identical copies of address spaces, one for the parent and the other for the child.
- Both processes will start their execution at the next statement following the fork() call.

Example:

```
pid = fork();
// Both child and parent will now start execution from here.
if(pid < 0) {
    //child was not created successfully
    return 1;
}
else if(pid == 0) {
    // This is the child process
    // Child process code goes here
}
else {
    // Parent process code goes here
}
```

wait( )

The parent process may then issue a wait system call, which suspends the execution of the parent process while the child executes. When the child process terminates, it returns an exit status to the operating system, which is then returned to the waiting parent process.

Example 05 :

```
#include <iostream>
#include<unistd.h> /*For Fork*/
#include<sys/types.h> /*pid_t*/
#include<sys/wait.h> /*for wait*/
#define COUNT 5
```

```
int main()
{
        pid_t pid;
        int i;
        pid = fork();
        for(i=1; i<=COUNT; i++)
        {
                std::cout<<"This is from pid : " << pid << " and i is :"
                << i << std::endl;
                wait(0);
        }
        return 0;
}
```

Output:

```
ansha@ansha-VirtualBox:~/Desktop/Process$ g++ -o fork Fork.cpp
ansha@ansha-VirtualBox:~/Desktop/Process$ ./fork
This is from pid : 4565 and i is :1
This is from pid : 0 and i is :1
This is from pid : 0 and i is :2
This is from pid : 0 and i is :3
This is from pid : 0 and i is :4
This is from pid : 0 and i is :5
This is from pid : 4565 and i is :2
This is from pid : 4565 and i is :3
This is from pid : 4565 and i is :4
This is from pid : 4565 and i is :5
ansha@ansha-VirtualBox:~/Desktop/Process$
```

Example 06 : Working of wait() system call

```
#include <iostream>
#include<unistd.h> /*For Fork*/
#include<sys/types.h> /*pid_t*/
#include<sys/wait.h> /*for wait*/

int main()
{
        std::cout<<"Current process Id : "<<getpid()<<std::endl;

        pid_t childProcessId = fork();

        // If fork call Code after
        if(childProcessId < 0)
        {
                std::cout<<"Failed to Create a new Process"<<std::endl;
        }
        else if (childProcessId == 0)
        {
                // This code will be executed in Child Process Only
                std::cout<<"Child Process Id : "<< getpid()
                <<std::endl << "Its parent ID : "<< getppid() <<std::endl;
        }
        else if (childProcessId > 0)
        {
                // This code will be executed in Parent Process Only
                std::cout<<"Parent Process Id : "<< getpid()<<std::endl
                <<"Its child Process Id :"<< childProcessId << std::endl;
        }
         wait(NULL);
         return 0;
}
```

- **wait(NULL)** parent process will only wait until the child process is completed.

Output:

```
ansha@ansha-VirtualBox:~/Desktop/Process$ g++ -o fork Fork.cpp
ansha@ansha-VirtualBox:~/Desktop/Process$ ./fork
Current process Id : 4527
Parent Process Id : 4527
Its child Process Id :4528
Child Process Id : 4528
Its parent ID : 4527
ansha@ansha-VirtualBox:~/Desktop/Process$
```

Example 07 :

```
#include <iostream>
#include<unistd.h> /*for Fork*/
#include<sys/types.h> /*for pid_t*/
#include<sys/wait.h> /*for wait*/
#define COUNT 5
int main()
{
        int i, pid;
        pid = getpid();
        std::cout<<"Before fork , pid is " << pid <<std::endl;
        pid = fork();
        std::cout<<"After fork , pid is " << pid <<std::endl;
        if(pid==0)
        {
                std::cout << "Child Process starts "<<std::endl;
                for(int i=0; i<3; ++i)
                {
                        std::cout<< "i is " <<i <<std::endl;
                }
                std::cout << "Child Process end " <<std::endl;
        }
        else
        {
                std::cout << "Wait starts " <<std::endl;
                std::cout << "Process ID: " << getpid() << std::endl;
                pid = wait(0);
                std::cout<< "* pid :" <<pid << std::endl;
                std::cout <<"* After wait " <<std::endl;
```

```
                std::cout <<"* Parent ID " << getppid() << std::endl;
        }
        return 0;

}
```

Output:

```
ansha@ansha-VirtualBox:~/Desktop/Process$ g++ -o fork Fork.cpp
ansha@ansha-VirtualBox:~/Desktop/Process$ ./fork
Before fork , pid is 5028
After fork , pid is 5029
Wait starts
Process ID: 5028
After fork , pid is 0
Child Process starts
i is 0
i is 1
i is 2
Child Process end
* pid :5029
* After wait
* Parent ID 3741
ansha@ansha-VirtualBox:~/Desktop/Process$
```

Lab Task:

- Submit all examples given in manual.
- Write a program to search the key element in the parent process and print the key to be searched in the child process.

DHA SUFFA UNIVERSITY
Department of Computer Science
CS-2004L
Operating Systems
Fall 2020

ASSIGNMENT # 03

1) Write a program to sort the numbers in the parent process and print the unsorted numbers in the child process. For example :

Input : 5, 2, 3, 1, 4
Output :

Parent process
sorted numbers are
1, 2, 3, 4, 5

Child process
numbers to sort are
5, 2, 3, 1, 4

2) Execute the following codes and identify the differences.

(a)

```
#include <iostream>
#include <unistd.h> /* _exit, fork */
#include <sys/types.h> /* pid_t */
#include <sys/wait.h> /* for wait */
#include <stdlib.h> /* exit */

using namespace std;
int main()
{
        if (fork() == 0 )
        {
                cout << "1" ;
        }
        else
        {
                wait(0);
                cout << "0";
                exit(0);
        }
        cout << "0";
        return 0;
}
```

(b)

```
#include <iostream>
#include <unistd.h> /* _exit, fork */
#include <sys/types.h> /* pid_t */
#include <sys/wait.h> /* for wait */
#include <stdlib.h> /* exit */

using namespace std;
int main()
{
        if (fork() == 0 )
        {
                cout << "1" ;
        }
        else
        {
                //wait(0);
                cout << "0";
                exit(0);
        }
        cout << "0";
        return 0;
}
```

SUBMISSION GUIDELINES

- Make a folder named with your Roll No e.g. **cs181xxx**
- Place the screenshots of code and output of all questions in word file and write differences in the same file for question 2, the file must be labeled with Roll No and Lab No. e.g. **'cs181xxx_Lab01'.**
- Place the word file and .cpp files in the folder.
- Submit the folder at LMS