OPERATING SYSTEMS

Lab-4: Fork Programs

Name – Vinay Santosh Menon
Registration Number - 20BAI1103

TASK:

To understand how fork() command works, and implementing it in C, C++ code.

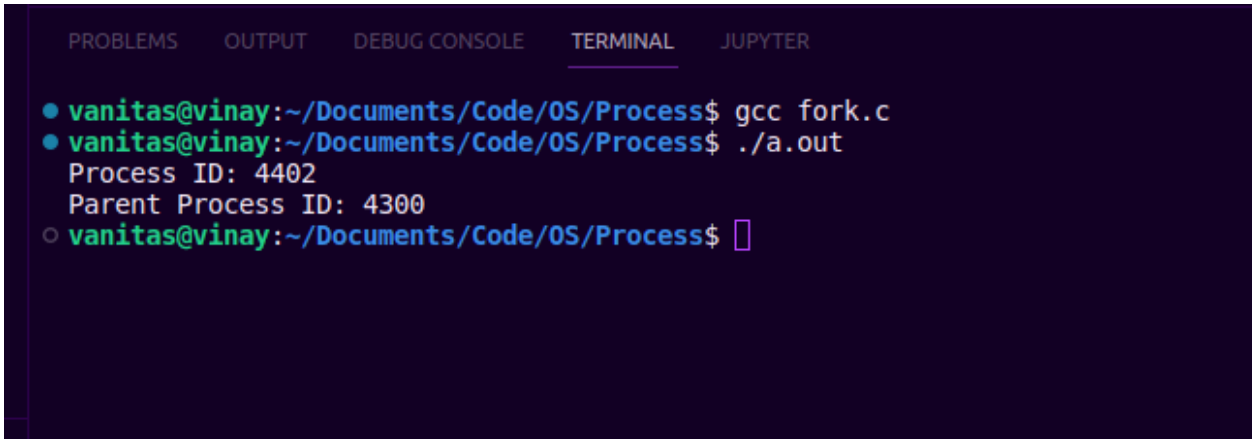
WHAT IS FORK():

Fork system call is used for creating a new process,
which is called child process, which runs concurrently
with the process that makes the fork() call (parent process).
Fork takes no arguments inside the brackets , process id of the child process is created, both parent and child starts execution in the next instruction.

Program-1:

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

int main()
{
   int p_id1,p_pid2;

   p_id1=getpid();
   p_pid2=getppid();

   printf("Process ID: %d\n",p_id1);
   printf("Parent Process ID: %d\n",p_pid2);

   return 0;
}
```

Output:



getppid() and getpid() in Linux. Both getppid() and getpid() are inbuilt functions defined in unistd.h library. getppid() : returns the process ID of the parent of the calling process. ... Otherwise, this function returns a value of 1 which is the process id for init process.

Program-2:

```c
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
   fork();
   printf("Hello world!\n");
   return 0;
}
```

```
vanitas@vinay:~/Documents/Code/OS/Process$ gcc fork1.c
vanitas@vinay:~/Documents/Code/OS/Process$ ./a.out
Hello world!
Hello world!
```

Program-3:

```c
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
   fork();
   fork();
```

```
    fork();
    printf("hello\n");
    return 0;
}
```

Output:

```
vanitas@vinay:~/Documents/Code/OS/Process$ gcc fork2.c
vanitas@vinay:~/Documents/Code/OS/Process$ ./a.out
hello
hello
hello
hello
hello
hello
hello
hello
```

Program-4:

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
        int i;
        printf("hello before fork \n");
        printf("i : %d\n",i);

        i=fork();
        printf("\n");
```

```
        if(i==0)
        {

                printf("Child has started\n\n");
                printf("child printing first time \n");

                printf("getpid : %d getppid : %d
\n",getpid(),getppid());
                sleep(5);
                printf("\nchild printing second time \n");
                printf("getpid : %d getppid : %d
\n",getpid(),getppid());
        }
        else
        {
                printf("parent has started\n");
                printf("getpid : %d  getppid : %d
\n",getpid(),getppid());
                printf("\n");

        }

        printf("Hi after fork i : %d\n",i);

        return 0;
}
```

Output:

```
vanitas@vinay:~/Documents/Code/OS/Process$ gcc fork3.c
vanitas@vinay:~/Documents/Code/OS/Process$ ./a.out
hello before fork
i : 0

parent has started
getpid : 7940  getppid : 4300

Hi after fork i : 7941

Child has started

child printing first time
getpid : 7941 getppid : 1767
vanitas@vinay:~/Documents/Code/OS/Process$
child printing second time
getpid : 7941 getppid : 1767
Hi after fork i : 0
```

Child status information: Status information about the child reported by wait is more than just the exit status of the child, it also includes.

Program-5:

```c
#include<stdio.h>
#include<sys/wait.h>
#include<unistd.h>
 int main()
{
   if (fork()== 0)
       printf("HC: hello from child\n");
   else
   {
       printf("HP: hello from parent\n");
       wait(NULL);
       printf("CT: child has terminated\n");
   }
    printf("Bye\n");
   return 0;
}
```

Output:

```
vanitas@vinay:~/Documents/Code/OS/Process$ gcc fork4.c
vanitas@vinay:~/Documents/Code/OS/Process$ ./a.out
HP: hello from parent
HC: hello from child
Bye
CT: child has terminated
Bye
```

Program-6:

```c
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
void main()
{
   int val;
   val = fork();
   printf("%d", val);
}
```

Output:

```
vanitas@vinay:~/Documents/Code/OS/Process$ gcc fork5.c
vanitas@vinay:~/Documents/Code/OS/Process$ ./a.out
82430vanitas@vinay:~/Documents/Code/OS/Process$
```

Fork system call is used for creating a new process, which is called child process, which runs concurrently with the process that makes the fork() call (parent process). After a new child process is created, both processes will execute the next instruction following the fork() system call. A child process uses the same pc(program counter), same CPU registers, same open files which use in the parent process. Negative Value: creation of a child process was unsuccessful. Zero: Returned to the newly created child process. Positive value: Returned to parent or caller. The value contains process ID of newly created child process.

Program-7:

```cpp
#include <iostream>
#include <sys/wait.h>
#include <unistd.h>

using namespace std;
int main()
{

   pid_t id1 = fork();
   pid_t id2 = fork();
   if (id1 > 0 && id2 > 0) {
       wait(NULL);
       wait(NULL);
       cout << "Parent Terminated" << endl;
   }
   else if (id1 == 0 && id2 > 0) {
       sleep(2);
       wait(NULL);
       cout << "1st child Terminated" << endl;
   }
   else if (id1 > 0 && id2 == 0) {

       sleep(1);
       cout << "2nd Child Terminated" << endl;
   }
   else {
       cout << "Grand Child Terminated" << endl;
   }
```

```
    return 0;
}
```

Output:

```
vanitas@vinay:~/Documents/Code/OS/Process$ g++ fork.cpp
vanitas@vinay:~/Documents/Code/OS/Process$ ./a.out
Grand Child Terminated
2nd Child Terminated
1st child Terminated
Parent Terminated
```

Program-8:

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
void forkexample()
{
   if (fork() == 0)
       printf("Hello from Child!\n");

   else
       printf("Hello from Parent!\n");
}
int main()
{
   forkexample();
   return 0;
}
```

Output:

```
82430vanitas@vinay:~/Documents/Code/OS/Process$ gcc fork6.c
vanitas@vinay:~/Documents/Code/OS/Process$ ./a.out
Hello from Parent!
Hello from Child!
vanitas@vinay:~/Documents/Code/OS/Process$
```

In the above code, a child process is created. fork() returns 0 in the child process and positive integer in the parent process. Here, two outputs are possible because the parent process and child process are running concurrently. So we don’t know whether the OS will first give control to the parent process or the child process. Important: Parent process and child process are running the same program, but it does not mean they are identical. OS allocate different data and states for these two processes, and the control flow of these processes can be different.