

Operating System

Lab Manual



Topic: Process System Call

Session: Fall 2024

School of Systems and Technology

UMT Lahore Pakistan

LAB 7: Process System Call

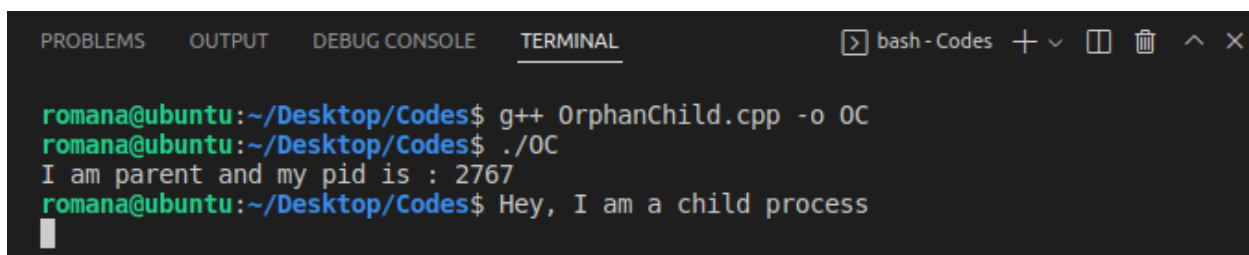
What is an orphan child?

a process which is executing (is alive) but its parent process has terminated (dead) is called an orphan process. Similarly, the orphan process in linux is adopted by a new process, which is mostly init process (pid=1). This is called re-parenting.

Reparenting is done by the kernel, when the kernel detects an orphan process in os, and assigns a new parent process.

New parent process asks the kernel for cleaning of the PCB of the orphan process and the new parent waits till the child completes its execution.

```
#include <iostream>
#include <sys/types.h>
#include <unistd.h>
using namespace std;
int main()
{
    int pid = fork();
    if(pid>0)
        cout << "I am parent and my pid is : "<< getpid() << endl;
    else
    {
        sleep(10);
        cout << "Hey, I am a child process\n";
        exit(0);
    }
    return 0;
}
```



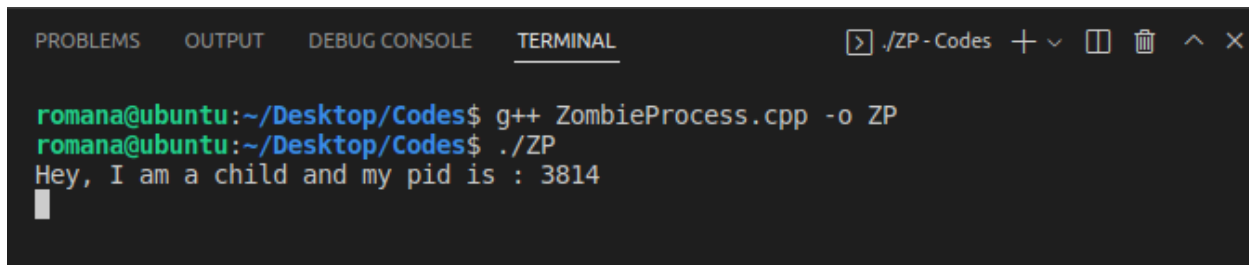
```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL bash - Codes + v [ ] [ ] ^ x
romana@ubuntu:~/Desktop/Codes$ g++ OrphanChild.cpp -o OC
romana@ubuntu:~/Desktop/Codes$ ./OC
I am parent and my pid is : 2767
romana@ubuntu:~/Desktop/Codes$ Hey, I am a child process
```

What is a zombie process?

Zombie state: When a process is created in UNIX using fork() system call, the address space of the Parent process is replicated. If the parent process calls wait() system call, then the execution of the parent is suspended until the child is terminated. At the termination of the child, signal is generated which is delivered to the parent by the kernel. Parent, on receipt of signal reads the status of the child from the process table. Even though the child is terminated, there is an entry in the process table corresponding to the child where the status is stored. When the parent collects the status, this entry is deleted. Thus, all the traces of the child process are removed from the system.

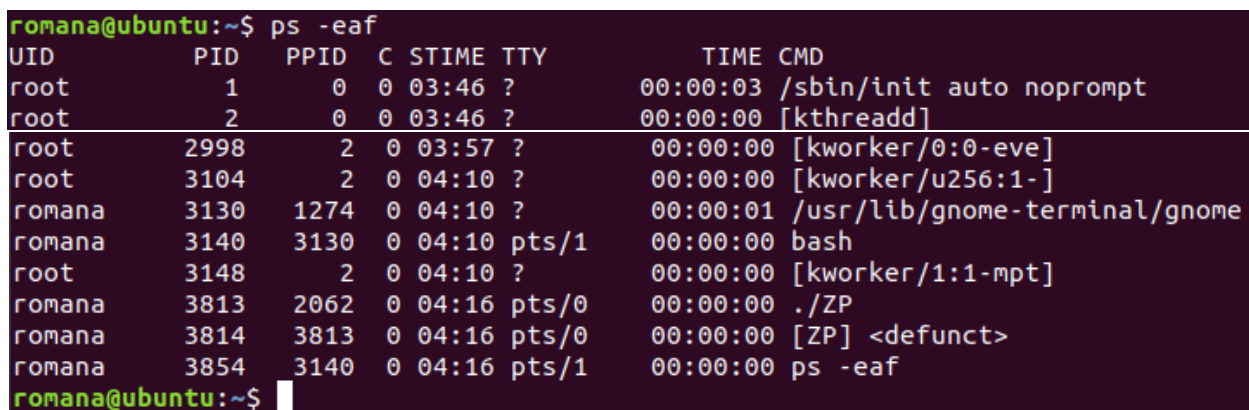
If the parent decides not to wait for the child's termination and executes its subsequent task, then at the termination of the child, the exit status is not read. Hence, there remains an entry in the process table even after the termination of the child. This state of the child process is known as the Zombie state.

```
#include <iostream>
#include <sys/types.h>
#include <unistd.h>
using namespace std;
int main()
{
    int pid = fork();
    if(pid>0)
    {
        sleep(60);
        cout << "I am parent and my pid is : "<<getpid()<< endl;
    }
    else
    {
        cout << "Hey, I am a child and my pid is : "<<getpid()<< endl;
        exit(0);
    }
    return 0;
}
```



A terminal window titled ".ZP - Codes" showing the compilation and execution of the program. The user 'romana' is at the prompt 'romana@ubuntu:~/Desktop/Codes\$'. They compile the program with 'g++ ZombieProcess.cpp -o ZP' and then run it with './ZP'. The output is 'Hey, I am a child and my pid is : 3814'.

```
romana@ubuntu:~/Desktop/Codes$ g++ ZombieProcess.cpp -o ZP
romana@ubuntu:~/Desktop/Codes$ ./ZP
Hey, I am a child and my pid is : 3814
```



A terminal window showing the output of the 'ps -eaf' command. The output is a table of processes. The process with PID 3814 is shown as '[ZP] <defunct>', indicating it is a zombie process. The process with PID 3854 is the current 'ps' command.

UID	PID	PPID	C	STIME	TTY	TIME	CMD
root	1	0	0	03:46	?	00:00:03	/sbin/init auto noprompt
root	2	0	0	03:46	?	00:00:00	[kthreadd]
root	2998	2	0	03:57	?	00:00:00	[kworker/0:0-eve]
root	3104	2	0	04:10	?	00:00:00	[kworker/u256:1-]
romana	3130	1274	0	04:10	?	00:00:01	/usr/lib/gnome-terminal/gnome
romana	3140	3130	0	04:10	pts/1	00:00:00	bash
root	3148	2	0	04:10	?	00:00:00	[kworker/1:1-mpt]
romana	3813	2062	0	04:16	pts/0	00:00:00	./ZP
romana	3814	3813	0	04:16	pts/0	00:00:00	[ZP] <defunct>
romana	3854	3140	0	04:16	pts/1	00:00:00	ps -eaf

After 60 sec

```
romana@ubuntu:~/Desktop/Codes$ g++ ZombieProcess.cpp -o ZP
romana@ubuntu:~/Desktop/Codes$ ./ZP
Hey, I am a child and my pid is : 3814
I am parent and my pid is : 3813
romana@ubuntu:~/Desktop/Codes$
```

root	3104	2	0	04:10	?	00:00:00	[kworker/u256:1-]
romana	3130	1274	0	04:10	?	00:00:02	/usr/lib/gnome-terminal/gnome
romana	3140	3130	0	04:10	pts/1	00:00:00	bash
root	3148	2	0	04:10	?	00:00:00	[kworker/1:1-mm_]
root	3855	2	0	04:16	?	00:00:00	[kworker/u256:0-]
romana	3887	3140	0	04:18	pts/1	00:00:00	ps -eaf

```
romana@ubuntu:~$
```

Why do we need to prevent the creation of the Zombie process?

There is one process table per system. The size of the process table is finite. If too many zombie processes are generated, then the process table will be full. That is, the system will not be able to generate any new process, then the system will come to a standstill. Hence, we need to prevent the creation of zombie processes.

A Zombie process creation can be Prevented Using wait () system call: When the parent process calls wait (), after the creation of a child, it indicates that, it will wait for the child to complete and it will reap the exit status of the child. The parent process is suspended (waits in a waiting queue) until the child is terminated. It must be understood that during this period, the parent process does nothing just wait.

What is exec () system call?

In computing, exec is a functionality of an operating system that runs an executable file in the context of an already existing process, replacing the previous executable. This act is also referred to as an overlay. As no new process is created, the process identifier (PID) does not change, but the machine code, data, heap, and stack of the process are replaced by those of the new program.

The exec call is available for many programming languages including compilable languages and some scripting languages. Standard names of such functions in C are execl, execl, execlp, execv, execve, and execvp, but not "exec" itself.

1. Execl

In execl() system function takes the path of the executable binary file as the first and second argument. Then, the arguments that you want to pass to the executable followed by NULL. Then execl() system function runs the command and prints the output. If any error occurs, then execl() returns -1. Otherwise, it returns nothing.

Syntax

int execl (const char *path, const char *arg, ..., NULL);

Example

```
//Save program as a.cpp

#include <iostream>
#include <unistd.h>
using namespace std;
int main (int argc, char const *argv[])
{
    cout<< "\n Hi, I am program a. \n";

    for(int i = 0; i<argc ; i++)
    {
        cout << argv[i];
    }
    cout<<endl;
}
```

Run program as

g++ a.cpp -o a

```
#include <iostream>
#include <unistd.h>
#include <stdlib.h>
using namespace std;
int main ()
{
    const char * binaryPath = "/home/romana/Desktop/Codes/a";
    const char * arg1 = "\nHello, I am from Exec function.\n";

    execl(binaryPath,binaryPath,arg1,NULL);

    cout << "I must be removed. \n";
    return 0;
}
```

Run program as

g++ ExecE1.cpp -o e1

Run the executable as

./e1

Output

```
romana@ubuntu:~/Desktop/Codes$ g++ ExecE1.cpp -o e1
romana@ubuntu:~/Desktop/Codes$ ./e1

Hi, I am program a.
/home/romana/Desktop/Codes/a
Hello, I am from Exec function.

romana@ubuntu:~/Desktop/Codes$
```

2. execv()

In execl() function, the parameters of the executable file is passed to the function as different arguments. With execv(), you can pass all the parameters in a NULL terminated array **argv**. The first element of the array should be the path of the executable file. Otherwise, execv() function works just as execl() function.

Syntax:

```
int execv(const char *path, char *const argv[]);
```

Example:

```
#include <iostream>
#include <unistd.h>
#include <stdlib.h>
using namespace std;
int main ()
{
    char * args[] = {"/home/romana/Desktop/Codes/a",
    "Hello, I am from Exec function.\n",NULL};

    execv(args[0], args);

    cout << "I must be removed. \n";
    return 0;
}
```

Run program as

```
g++ ExecE2.cpp -o e2
```

Run the executable as

```
./e2
```

Output

```
romana@ubuntu:~/Desktop/Codes$ ./e2
Hi, I am program a.
/home/romana/Desktop/Codes/a
Hello, I am from Exec function.
```

Fork () vs Exec () System Call?

Fork () creates new child process while exec () replaces the current running process with a new process.

Eaxmple:

Let's create a simple program which prints first 5 natural numbers named as NaturalNumbers.cpp as shown below.

```

1  #include <iostream>
2  #include <unistd.h>
3  using namespace std;
4
5  int main ()
6  {
7      cout << "\nI prints 5 natural numbers" << endl;
8      cout << "my pid is : " << getpid() << endl;
9
10     for (int i = 0 ; i <5 ; i++)
11     {
12         cout << i+1 << "    ";
13     }
14     cout << endl;
15
16 }

```

Compile this program as

g++ naturalNumbers.cpp -o nn

After creating the executable, create a new file ExecE3.cpp where use fork (). Use exec system call and replace child body with naturalNumbers program as shown below.

```

#include <iostream>
#include <sys/types.h>
#include <unistd.h>
using namespace std;
int main ()
{
    int pid = fork();
    if (pid > 0)
    {
        cout << "I am a parent process and my PID is: " << getpid();
        cout << endl;
    }
    else if(pid==0)
    {
        cout << "I am a child process and my PID is : " << getpid();
        cout << endl;
        const char * BinaryPath = "./nn";
        execl(BinaryPath,BinaryPath,NULL);
    }
    return 0;
}

```

```

romana@ubuntu:~/Desktop/Codes$ g++ ExecE3.cpp -o e3
romana@ubuntu:~/Desktop/Codes$ ./e3
I am a parent process and my PID is: 6811
I am a child process and my PID is : 6812
romana@ubuntu:~/Desktop/Codes$
I prints 5 natural numbers
my pid is : 6812
1    2    3    4    5

```

Select Encoding

Lab Task

1. Create a program OddNumber which displays odd number from 1 to n.
2. Create a new program which creates a child process. Parent process must print its pid. Child process must call program oddProcess. Also provide the value of n through arguments.

Note: use function `atoi(char *a)` or `atoi (char a[])` to convert char array to int. include `stdlib.h` for this function.