Process Concept Using C in Linux

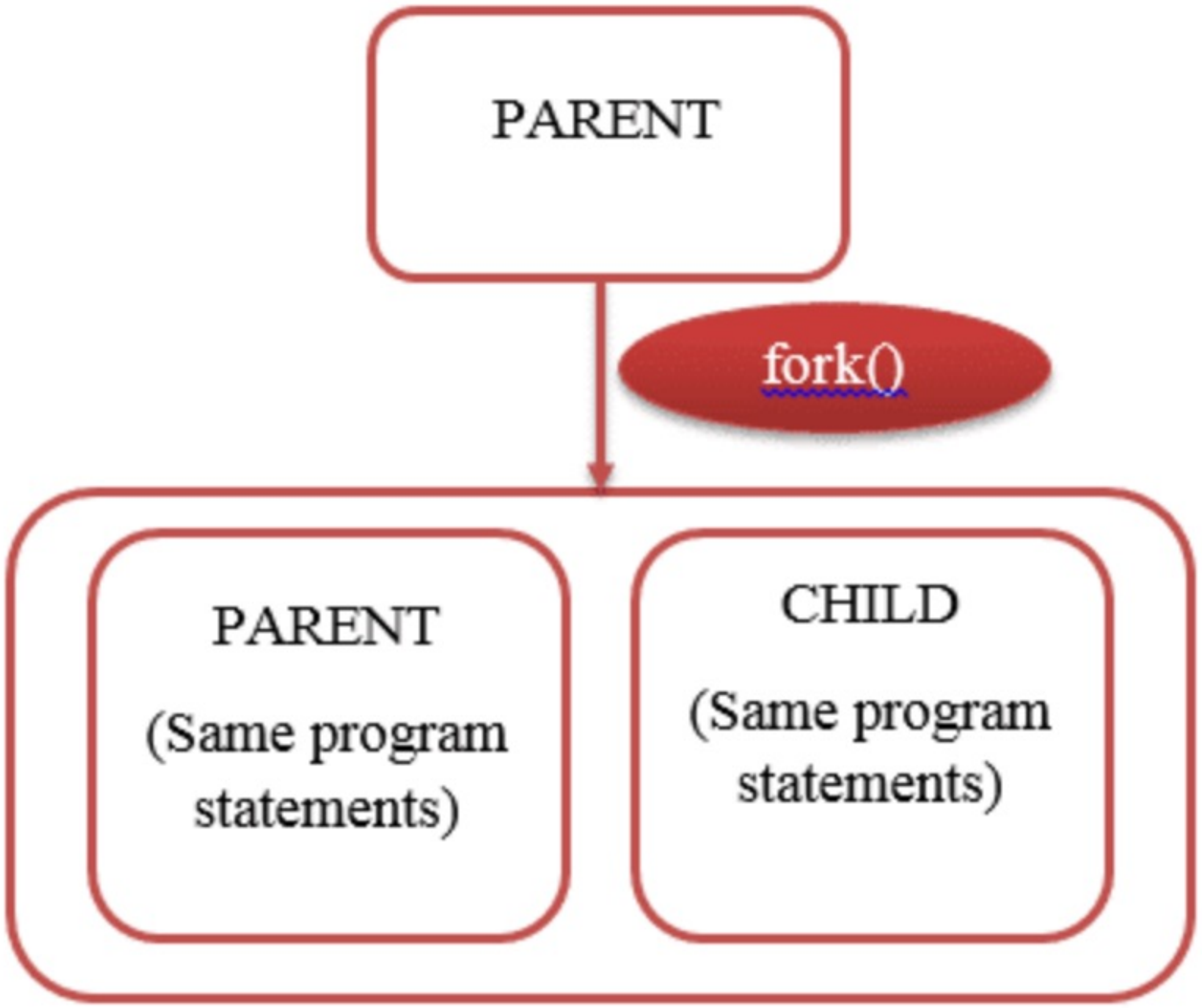
Prepared by:

Dr. Susama Bagchi

Asst. Prof., CUIET, Chitkara University, Punjab

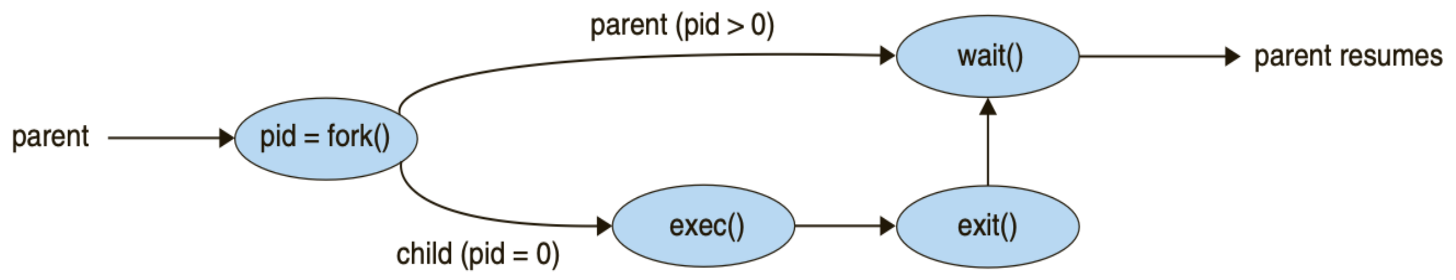
Fork() System Call

- It is a system call.
- Parent process calls fork() to create a child process.
- The child and parent processes are executed concurrently although they reside on different memory spaces. These memory spaces have same content and whatever operation is performed by one process will not affect the other process.
- Both the parent and child processes have the same Program Counter (PC) and hence, both these processes will point to the same next instruction. The files opened by the parent process will be the same for child process.
- In general, parent process waits for the termination of the child process first. In some special cases, child process may wait when the parent process may complete its execution and terminate. In this case, child process may get another arbitrary parent PID so that memory can be freed after its termination.



Use of Fork() …1

- When the child process is created, both the parent process and the child process will point to the next instruction due to the same Program Counter.
- Therefore, the remaining instructions or C statements will be executed a total of **2^n** times, where n is the number of fork() system calls.
- So, when the fork() call is used one time as above, the output will be ($2^1 = 2$) 2 times.



```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

int main()
{
    fork();
    printf("Using fork() system call\n");
    return 0;
}
```

Header files

- ➢ stdio.h - for printf() function
- ➢ sys/types.h - for pid_t type
- ➢ unistd.h - it is used for fork() function

COMPILE and RUN:

```
$ gcc fork.c -o fork
$ ./fork
```

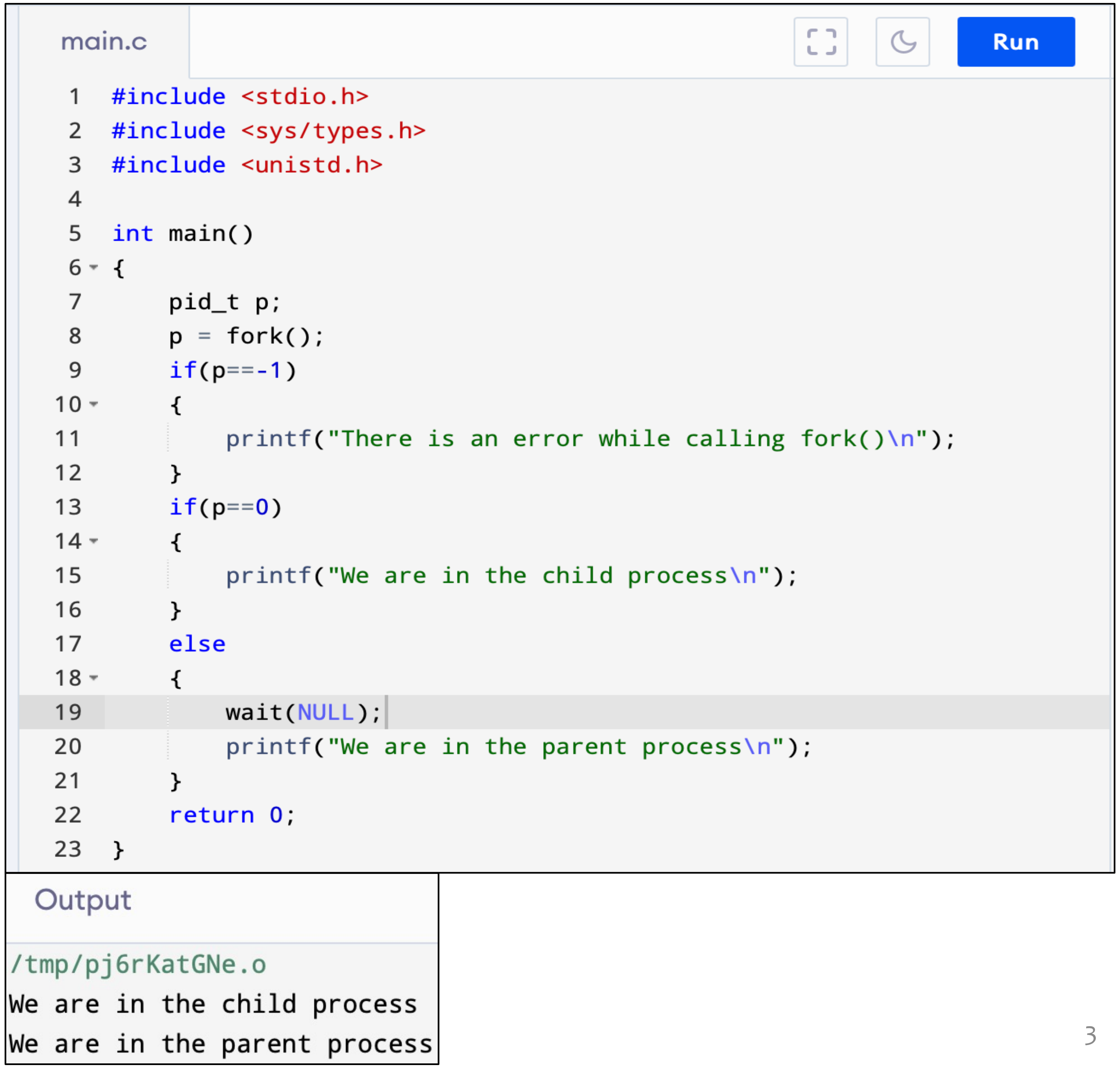
OUTPUT:

```
Using fork() system call
Using fork() system call
```

Use of Fork() …2

- When the child process is successfully created, the PID of the child process is returned in the parent process and 0 will be returned to the child process itself. For parent process, it is 1.

- If there is any error, then -1 will be returned to the parent process and the child process is not created.



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

int main()
{
    pid_t p;
    p = fork();
    if(p==-1)
    {
        printf("There is an error while calling fork()\n");
    }
    if(p==0)
    {
        printf("We are in the child process\n");
    }
    else
    {
        wait(NULL);
        printf("We are in the parent process\n");
    }
    return 0;
}
```

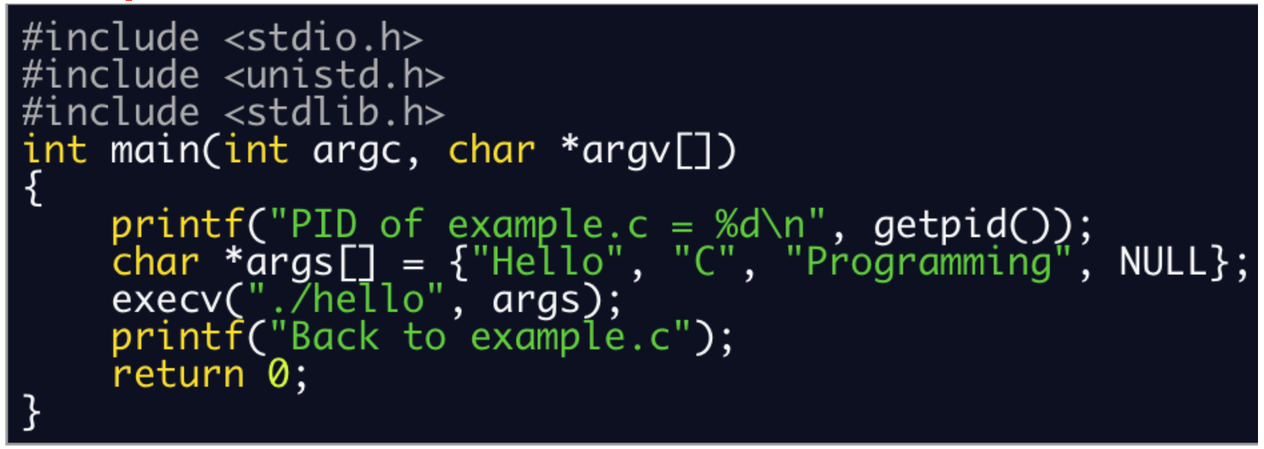
Output

```
/tmp/pj6rKatGNe.o
We are in the child process
We are in the parent process
```

Exec() System Call

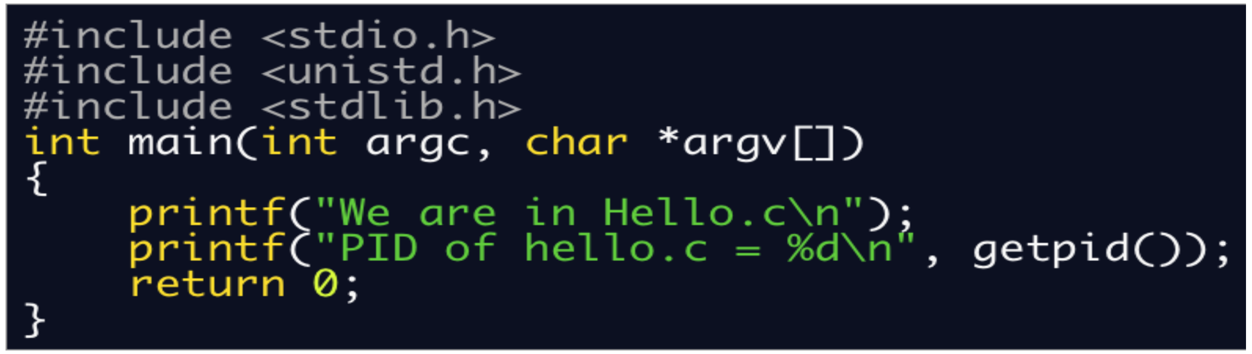
- The exec system call is used to execute a file which is residing in an active process.
- When exec is called the previous executable file is replaced and new file is executed where the new process carries the old PID.

example.c file



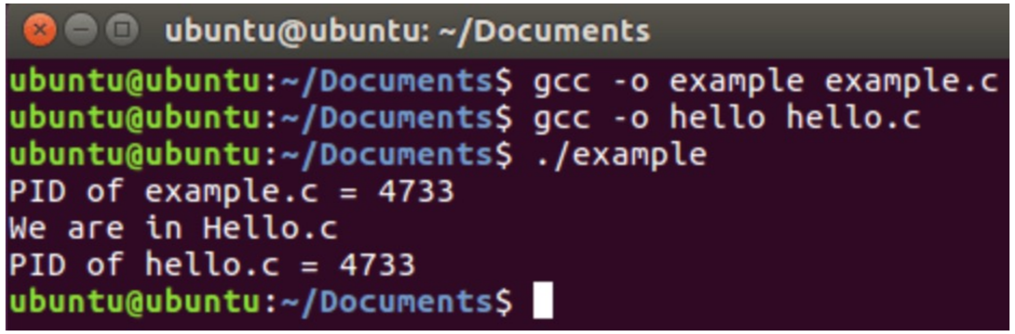
```c
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
    printf("PID of example.c = %d\n", getpid());
    char *args[] = {"Hello", "C", "Programming", NULL};
    execv("./hello", args);
    printf("Back to example.c");
    return 0;
}
```

hello.c file



```c
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
    printf("We are in Hello.c\n");
    printf("PID of hello.c = %d\n", getpid());
    return 0;
}
```

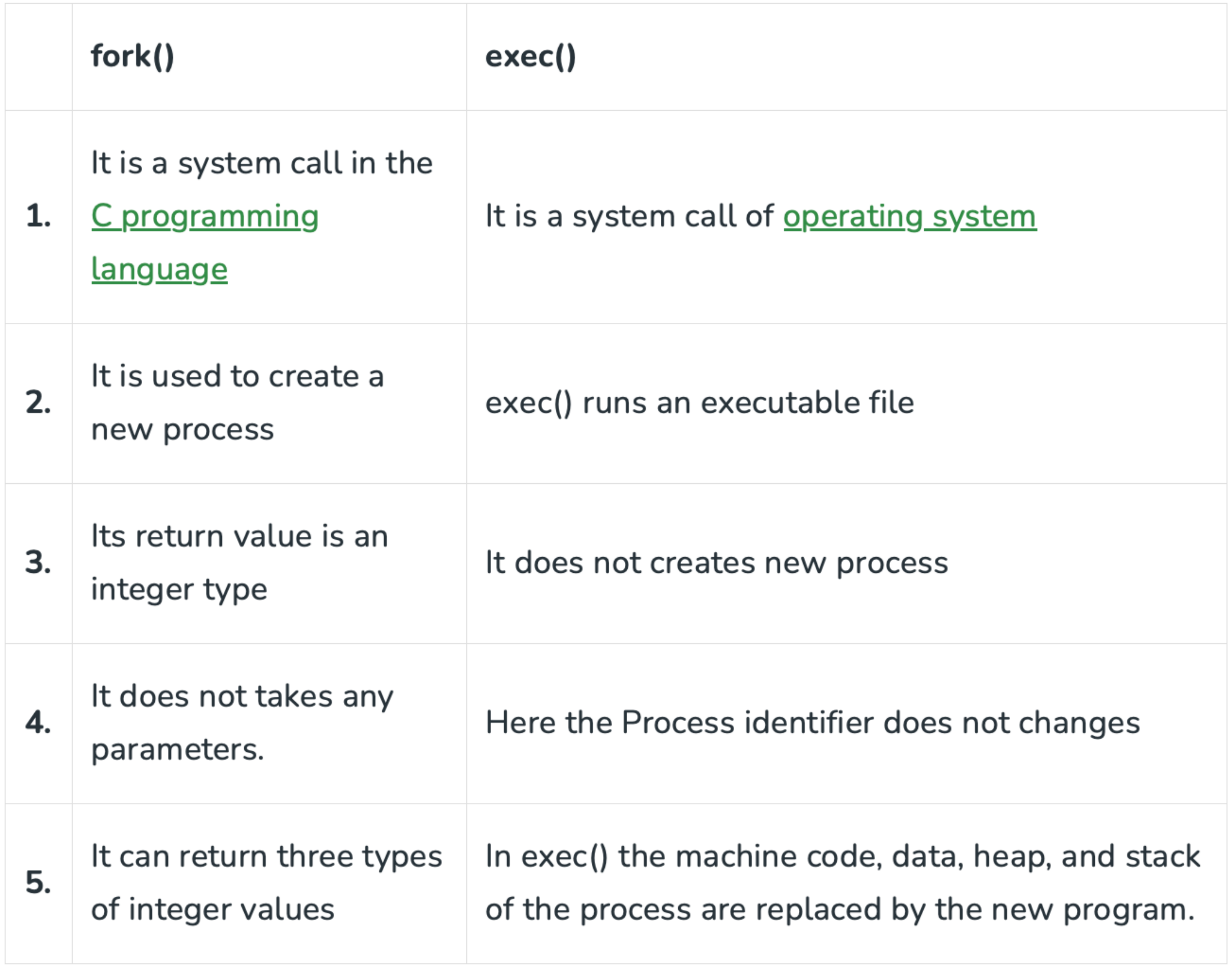
OUTPUT:



```
ubuntu@ubuntu: ~/Documents
ubuntu@ubuntu:~/Documents$ gcc -o example example.c
ubuntu@ubuntu:~/Documents$ gcc -o hello hello.c
ubuntu@ubuntu:~/Documents$ ./example
PID of example.c = 4733
We are in Hello.c
PID of hello.c = 4733
ubuntu@ubuntu:~/Documents$ █
```

- Both example.c and hello.c files should be in the same location.
- Before executing the example.c file, this file along with hello.c should be compiled first.
- In example.c, the last printf statement is coming as output because before that line, execv() was called and hence, in the child process the remaining part of example.c was replaced with the new processes from hello.c.

Difference between fork() & exec()



	fork()	**exec()**
1.	It is a system call in the C programming language	It is a system call of operating system
2.	It is used to create a new process	exec() runs an executable file
3.	Its return value is an integer type	It does not creates new process
4.	It does not takes any parameters.	Here the Process identifier does not changes
5.	It can return three types of integer values	In exec() the machine code, data, heap, and stack of the process are replaced by the new program.

Properties of Child Process

- The CPU counters and the resource utilizations are initialized to reset to zero.
- When the parent process is terminated, child processes do not receive any signal because PR_SET_PDEATHSIG attribute in prctl() is reset.
- The thread used to call fork() creates the child process. So, the address of the child process will be the same as that of parent.
- The file descriptor of parent process is inherited by the child process. For example, the offset of the file or status of flags and the I/O attributes will be shared among the file descriptors of child and parent processes. So, file descriptor of parent class will refer to the same file descriptor of the child class.
- The open message queue descriptors of parent process are inherited by the child process. For example, if a file descriptor contains a message in parent process the same message will be present in the corresponding file descriptor of child process. So, we can say that the flag values of these file descriptors are same.
- Similarly open directory streams will be inherited by the child processes.
- The default Timer slack value of the child class is same as the current timer slack value of parent class.

getpid() and getppid()

- When any process is created, it gets a unique process id.
- getpid() function returns the process id of the calling function.
  - Syntax:
    - pid_t getpid();
- getppid() function returns the process id of the parent function.
  - Syntax:
    - pid_t getppid();

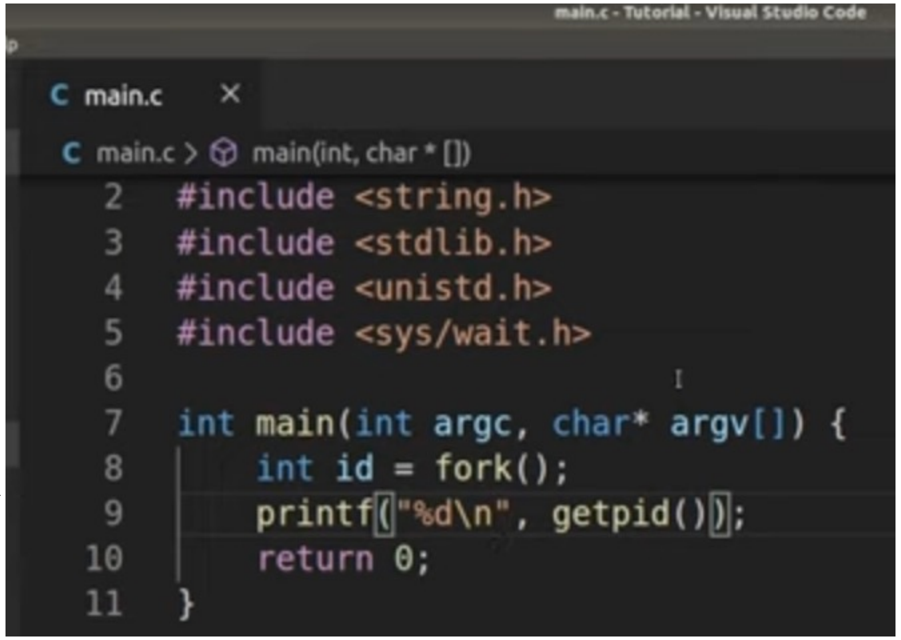
```c
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

int main(void)
{
        //variable to store calling function's process id
        pid_t process_id;
        //variable to store parent function's process id
        pid_t p_process_id;

        //getpid() - will return process id of calling function
        process_id = getpid();
        //getppid() - will return process id of parent function
        p_process_id = getppid();

        //printing the process ids
        printf("The process id: %d\n",process_id);
        printf("The process id of parent function: %d\n",p_process_id);

        return 0;
}
```

Header files

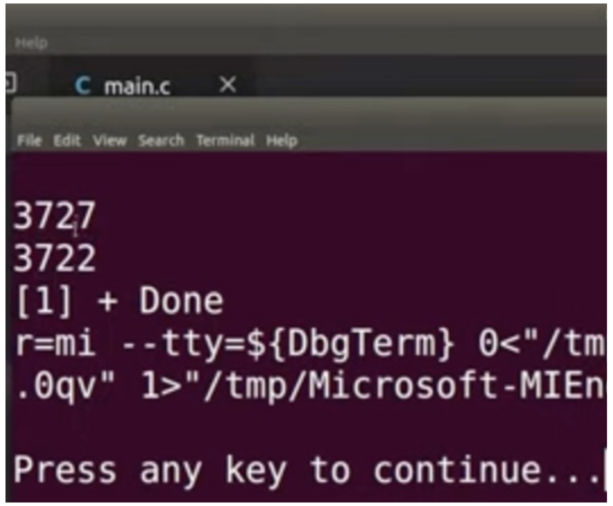
- ➢ stdio.h - it is used for printf() function
- ➢ sys/types.h - it is used for pid_t type, that is the data type of the variables which are using to store the process ids.
- ➢ unistd.h - it is used for getpid() and getppid() functions

Difference of getpid() and getppid()

- First the child PID will be displayed and then the parent ID as can be seen in Fig. 1 output.

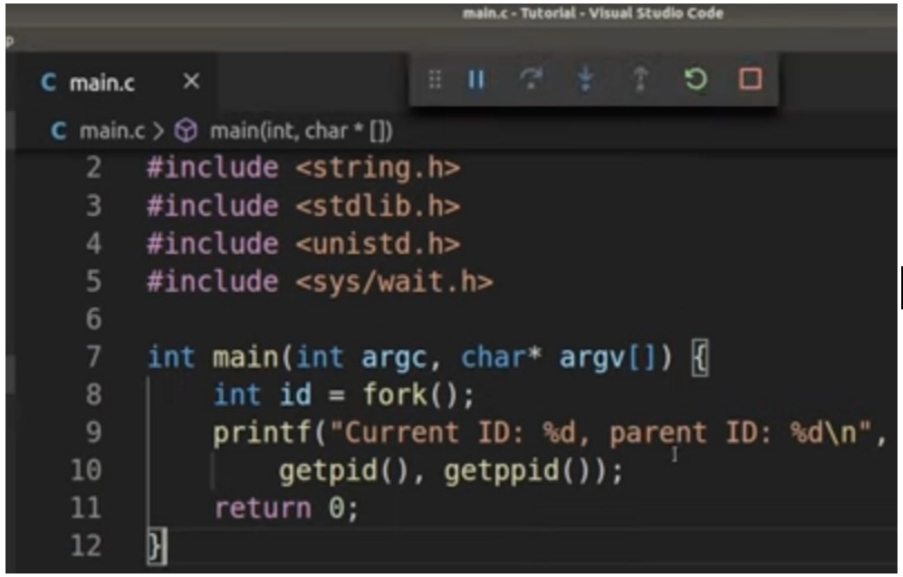

main.c - Tutorial - Visual Studio Code
main.c
main.c > main(int, char * [])

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

int main(int argc, char* argv[]) {
    int id = fork();
    printf("%d\n", getpid());
    return 0;
}
```

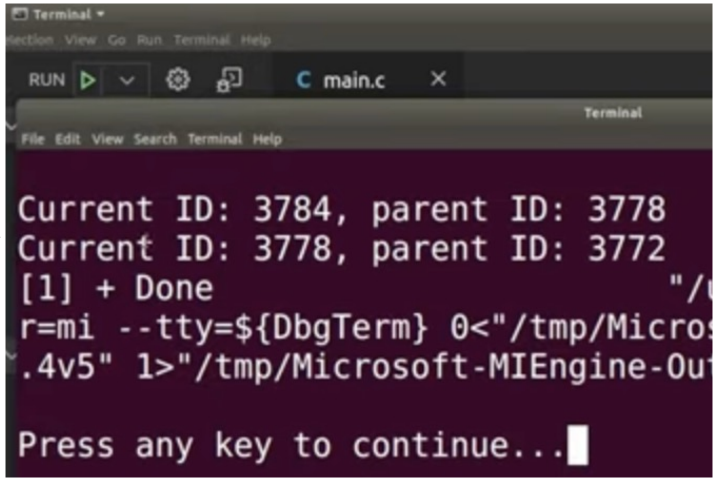


```
3727
3722
[1] + Done
r=mi --tty=${DbgTerm} 0<"/tm
.0qv" 1>"/tmp/Microsoft-MIEn

Press any key to continue...
```

- For 1st time printf execution, Current ID = child PID and Parent ID = the PID of the calling process of fork().
- For 2nd time printf execution, Current ID = the PID of the calling process of fork() and Parent ID = the parent ID of the calling process fork(). For this reason, the 1st line parent ID = 2nd line current ID.


main.c - Tutorial - Visual Studio Code
main.c
main.c > main(int, char * [])

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

int main(int argc, char* argv[]) {
    int id = fork();
    printf("Current ID: %d, parent ID: %d\n",
        getpid(), getppid());
    return 0;
}
```


Terminal
Selection View Go Run Terminal Help
RUN
main.c
Terminal
File Edit View Search Terminal Help

```
Current ID: 3784, parent ID: 3778
Current ID: 3778, parent ID: 3772
[1] + Done                      "/
r=mi --tty=${DbgTerm} 0<"/tmp/Micros
.4v5" 1>"/tmp/Microsoft-MIEngine-Ou

Press any key to continue...
```