Lab 2:

Compiling a simple C program

1-create a text document"Hello.c" in the Desktop directory:

- Cd Desktop/
- Touch Hello.c

2-open the Hello.c file, and write the following simple code:

```
#include<stdio.h>

int main()

{

        Printf("Hello World ….Lab 2 \n");

        return 0;

}
```

3- Save the file, to compile the file 'hello.c' with gcc [GNU Compiler Collection], use the following command:

- gcc hello.c –o hello

This compiles the source code in 'hello.c' to machine code and stores it in an executable file 'hello'. The output file for the machine code is specified using the -o option.

4- To run the program, type the path name of the executable like this:

- ./hello

5- Output will be:

- Hello World ….Lab 2.

Fork ():

Every running instance of a program is known as a process. The concept of processes is fundamental to the UNIX / Linux operating systems.
A process has its own identity in form of a PID or a process ID. This PID for each process is unique across the whole operating system. Also, each process has its own process address space.

The fork() function is used to create a new process by duplicating the existing process from which it is called. The existing process from which this function is called becomes the parent process and the newly created process becomes the child process. As already stated that child is a duplicate copy of the parent but there are some exceptions to it.

- The child has a unique PID like any other process running in the operating system.
- The child has a parent process ID which is same as the PID of the process that created it.

Fork() has an interesting behavior while returning to the calling method. If the fork() function is successful then it returns twice. Once it returns in the child process with return value '0' and then it returns in the parent process with child's PID as return value.

Example 1:

```
#include <stdio.h>
#include<unistd.h>
int main ()
{
        pid_t ChildID;
        ChildID = fork();
        //check for Fork
        if ( ChildID >= 0) // fork sucess
        {
                if ( ChildID ==0 ) // Code Of Child which the value will be 0 in case of Child code
                {
                        printf("Child says my ID          %d \n", getpid());
                        printf("Child says my Parent ID %d \n", getppid());

                }
                Else // Code of parent and the value of ChildID = process ID of child in case of Parent's code
```

```
        {
                printf("Parent says my  ID %d \n", getppid());
        }
    }
    else
    {
        printf("Fork is fail \n");
        return 1;
    }
    return 0 ;

}
```

Output will be:

Parent says my ID = 2445

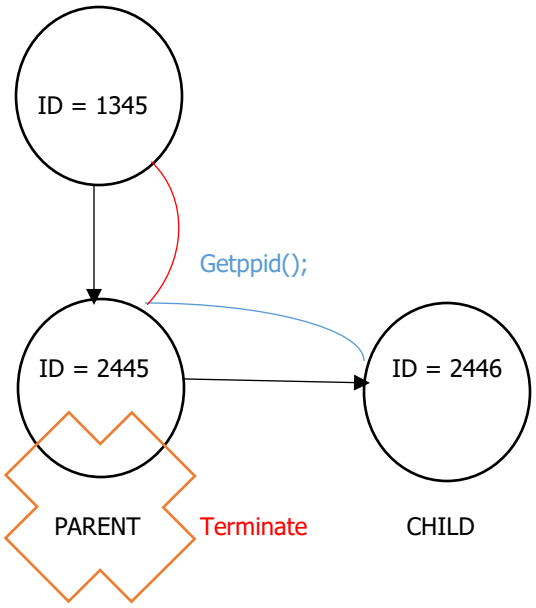
Child says my ID =2446

Child says my Parent ID=1345

WHY:

The two processes run at the same time, parent by like terminate first and print his id = 2445

And child run and print his id = 2446 but when he gets the parent id, it was terminate and get the id of parent of his parent as shown:



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

Example 2:

We can fix this example ???

- Yes by using the **sleep (int seconds)** function waits for seconds.

  If sleep returns because the requested interval is over, it returns a value of zero.

Example 2:

```
#include <stdio.h>
#include<unistd.h>
int main ()
{
        pid_t ChildID;
        ChildID = fork();
        //check for Fork
        if ( ChildID >= 0) // fork sucess
        {
                if ( ChildID ==0 ) // Code Of Child which the value will be 0 in case of Child code
                {
                        printf("Child says my ID          %d \n", getpid());
                        printf("Child says my Parent ID %d \n", getppid());

                }
                Else // Code of parent and the value of ChildID = process ID of child in case of Parent's code
                {
                        printf("Parent says my  ID %d \n", getppid());
                        sleep(50);
                }
        }
        else
        {
                printf("Fork is fail \n");
```

```
                return 1;
        }
        return 0 ;

}
```

Output will be:

Parent says my ID = 2445

Child says my ID =2446

Child says my Parent ID= 2445

Example 3:

What about Global and local Variables???

- Parent and child are two separated copies of code with the same variable declared before calling the fork methods.

```
#include <stdio.h>
#include<unistd.h>
int globalVraiable ;
int main ()
{
        pid_t ChildID;
        int localVraiable =0  ;
        ChildID = fork();
        //check for Fork
        if ( ChildID >= 0) // fork sucess
        {
                if ( ChildID ==0 ) // Code Of Child which the value will be 0 in case of Child code
                {
                        globalVariable++;
                        localVraiable++;
                        printf("Child says my globalVariable %d \n", globalVariable);
                        printf("Child says my localVraiable %d \n", localVraiable);
```

```
                }
                Else // Code of parent and the value of ChildID = process ID of child in case of Parent's code
                {
                        globalVariable = 10;
                        localVraiable = 50;
                        printf("parent says my globalVariable %d \n", globalVariable);
                        printf("parent says my localVraiable %d \n", localVraiable);
                }
        }
        else
        {
                printf("Fork is fail \n");
                return 1;
        }
        return 0 ;

}
```

Output will be:

parent says my globalVariable 10

parent says my localVraiable 50

Child says my globalVariable 1

Child says my localVraiable 1

Example 4:

There are certain situations where when a child process terminates or changes state then the parent process should come to know about the change of the state or termination status of the child process.

The system call wait () is easy to solve this problem.

This function blocks the calling process until **one of its *child* processes exits**, it takes the address of an integer variable and returns 0 in this address if one of its child processes terminate.

```
include <stdio.h>

#include<unistd.h>

int main ()

{

        pid_t ChildID;

        int status;

        ChildID = fork();

        //check for Fork

        if ( ChildID >= 0) // fork sucess

        {

                if ( ChildID ==0 ) // Code Of Child which the value will be 0 in case of Child code

                {

                        printf("Child says Hello \n");

                        printf("Child says Bye  \n");


                }

                Else // Code of parent and the value of ChildID = process ID of child in case of Parent's code

                {

                        printf("parent says Hello \n");

                        wait(&status);

                        printf("parent says Bye after my child terminate \n");

                }

        }

        else

        {

                printf("Fork is fail \n");

                return 1;
```

```
        }
        return 0 ;

}
```

Thanks….,