LAB 3 ASSIGNMENT

Name: Birva Babaria

Roll no.: CE010

ID: 19CEUON064

Aim: Process Creation and Termination (Use of fork, wait, getpid, and getppid system calls).

Description of System calls:

1) ‘fork’ system call

Library: #include<unistd.h>

Syntax: pid_t fork(void);

Description: fork() creates new process that is exact duplicate of the calling process (parent process). The child process has its unique ID. The child’s parent process ID is same as the parent’s process ID. On success, PID of child process is returned to parent and 0 is returned to child. On failure, -1 is returned to the parent and no child process is created.

Example: fork()

2) ‘getpid’ and ‘getppid’ system call

Library: #include<sys/types.h>

#include<unistd.h>

Syntax: pid_t getpid(void);

pid_t getppid(void);

Description: getpid() returns the PID of the current calling process. getppid() returns the PID of the parent of the calling process.

Example: int pid = getpid(); int ppid = getppid();

3) ‘wait’ system call

Library: #include<sys/types.h>

#include <sys/wait.h>

Syntax: pid_t wait(int *status);

Description: wait() system call is used to wait until the child process changes its state. Child process changes its states when it terminates or stops by the signal. If a child has already changes state, then this call returns immediately. On success, PID of the terminated child is returned. On error, -1 is returned.

Example: wait(NULL);

- **Zombie process:** The process which has finished its execution but still has entry in the process table to report to its parent process is called zombie process. Child process always becomes zombie before being removed from the process table.

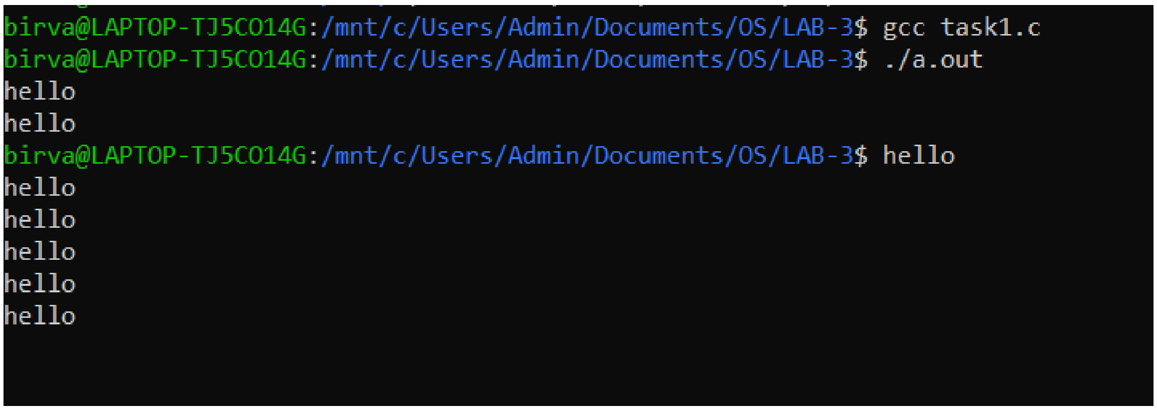
- **Orphan process:** A process whose parent process no more exists. This happens when parent process is either finished or terminated without waiting for the child process to terminate is called the orphan process.

1). Call fork once, twice, thrice and print “Hello”. Observe and interpret the outcomes.

CODE:

```
#include<unistd.h>

#include<stdio.h>

#include <sys/types.h>

void main()

{

    fork();

    fork();

    fork();

    printf("hello\n");

}
```

OUTPUT:



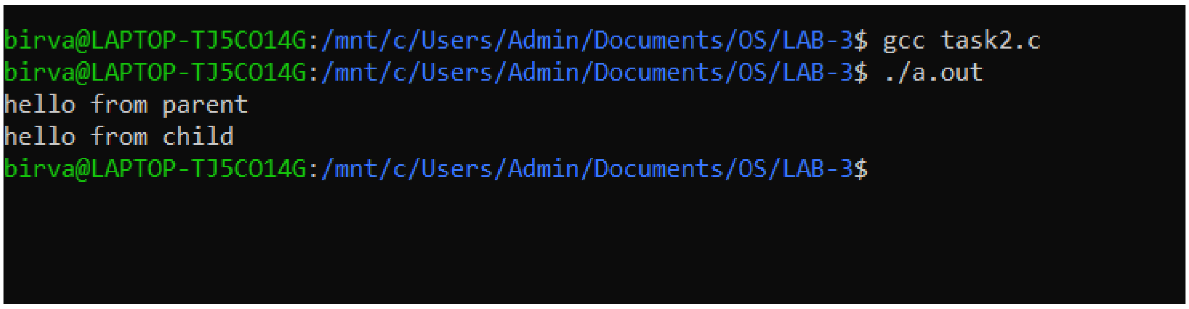
```
birva@LAPTOP-TJ5CO14G:/mnt/c/Users/Admin/Documents/OS/LAB-3$ gcc task1.c
birva@LAPTOP-TJ5CO14G:/mnt/c/Users/Admin/Documents/OS/LAB-3$ ./a.out
hello
hello
birva@LAPTOP-TJ5CO14G:/mnt/c/Users/Admin/Documents/OS/LAB-3$ hello
hello
hello
hello
hello
hello
```

2). Print “Hello from Parent” using the parent process and “Hello from Child” using child process.

CODE:

```
#include<unistd.h>

#include<stdio.h>

#include <sys/types.h>

int main()

{

    //fork will return 0 is the process is child process.

    if(fork() == 0)

    {

        printf("hello from child\n");

    }

    else

    {

        printf("hello from parent\n");

    }

    return 0;

}
```

OUTPUT:



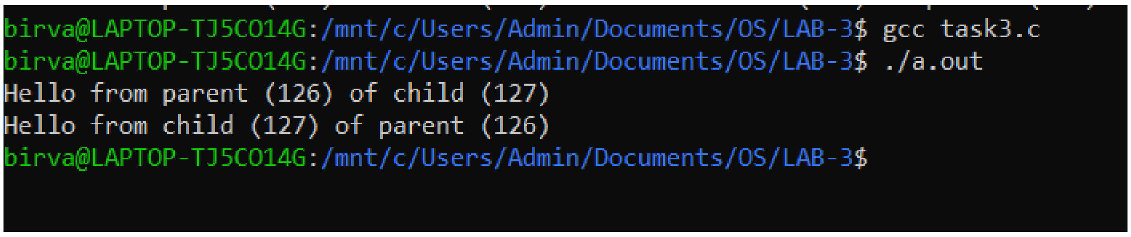
```
birva@LAPTOP-TJ5CO14G:/mnt/c/Users/Admin/Documents/OS/LAB-3$ gcc task2.c
birva@LAPTOP-TJ5CO14G:/mnt/c/Users/Admin/Documents/OS/LAB-3$ ./a.out
hello from parent
hello from child
birva@LAPTOP-TJ5CO14G:/mnt/c/Users/Admin/Documents/OS/LAB-3$
```

3). Print PID and PPID for parent and child processes. Observe and interpret the outcomes.

CODE:

```
#include<unistd.h>

#include<stdio.h>

#include <sys/types.h>

int main()

{

     int pid = fork();

     //pid=0 for child process and pid=child's pid for parent
process.

     if(pid == 0)

     {

          printf("Hello from child (%d) of parent
(%d)\n",getpid(),getppid());

     }

     else

     {

          printf("Hello from parent (%d) of child
(%d)\n",getpid(),pid);

     }

     return 0;

}
```

OUTPUT:



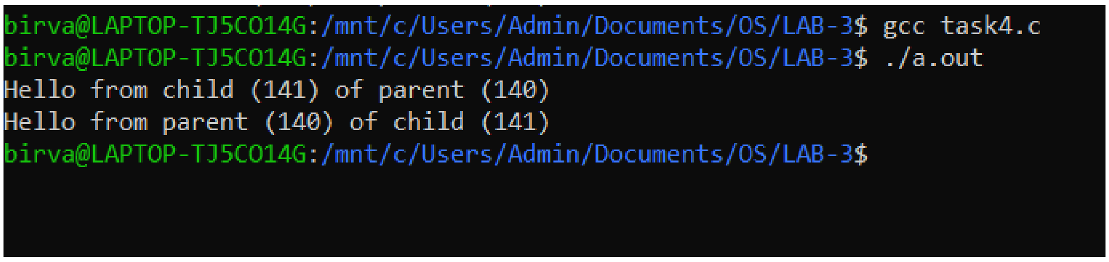
```
birva@LAPTOP-TJ5CO14G:/mnt/c/Users/Admin/Documents/OS/LAB-3$ gcc task3.c
birva@LAPTOP-TJ5CO14G:/mnt/c/Users/Admin/Documents/OS/LAB-3$ ./a.out
Hello from parent (126) of child (127)
Hello from child (127) of parent (126)
birva@LAPTOP-TJ5CO14G:/mnt/c/Users/Admin/Documents/OS/LAB-3$
```

4). Add wait to the code of task 2. Observe and interpret the outcomes.

CODE:

```
#include<unistd.h>

#include<stdio.h>

#include <sys/types.h>

#include<sys/wait.h>

int main()

{

     int pid = fork();

     //pid=0 for child process and pid=child's pid for parent
process.

     if(pid == 0)

     {

          printf("Hello from child (%d) of parent
(%d)\n",getpid(),getppid());

     }

     else

     {

          //wait until child changes its state

          wait(NULL);

          printf("Hello from parent (%d) of child
(%d)\n",getpid(),pid);

     }

     return 0;

}
```

OUTPUT:



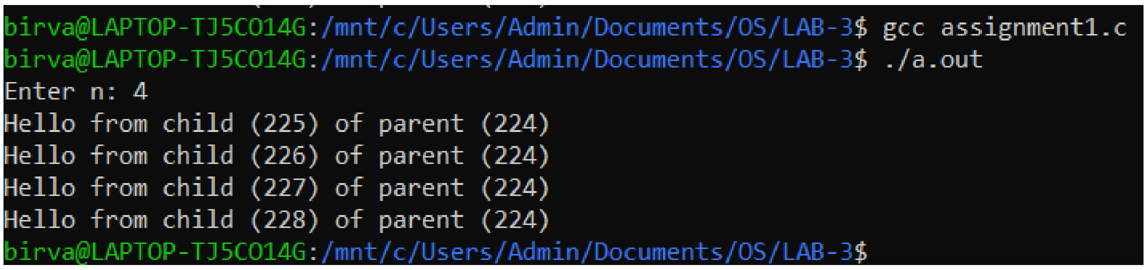
```
birva@LAPTOP-TJ5CO14G:/mnt/c/Users/Admin/Documents/OS/LAB-3$ gcc task4.c
birva@LAPTOP-TJ5CO14G:/mnt/c/Users/Admin/Documents/OS/LAB-3$ ./a.out
Hello from child (141) of parent (140)
Hello from parent (140) of child (141)
birva@LAPTOP-TJ5CO14G:/mnt/c/Users/Admin/Documents/OS/LAB-3$
```

5). Write a program to implement fan of n processes.

CODE:

```
#include<unistd.h>

#include<stdio.h>

#include<stdlib.h>

#include <sys/types.h>

#include<sys/wait.h>

int main()

{

    int n;

    printf("Enter n: ");

    scanf("%d",&n);

    for(int i=0;i<n;i++)

    {

        //if child process is running currently then print the
following statement and terminate the process

        if(fork() == 0)

        {

            printf("Hello from child (%d) of parent
(%d)\n",getpid(),getppid());

            exit(0);

        }

    }

    return 0;

}
```

OUTPUT:



```
birva@LAPTOP-TJ5CO14G:/mnt/c/Users/Admin/Documents/OS/LAB-3$ gcc assignment1.c
birva@LAPTOP-TJ5CO14G:/mnt/c/Users/Admin/Documents/OS/LAB-3$ ./a.out
Enter n: 4
Hello from child (225) of parent (224)
Hello from child (226) of parent (224)
Hello from child (227) of parent (224)
Hello from child (228) of parent (224)
birva@LAPTOP-TJ5CO14G:/mnt/c/Users/Admin/Documents/OS/LAB-3$
```

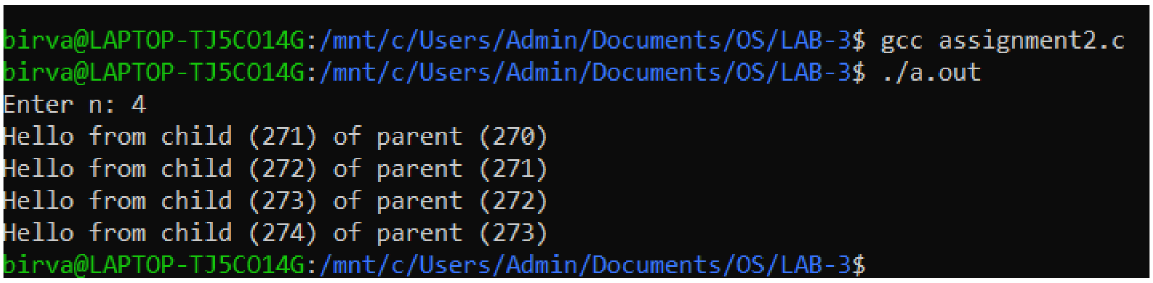
6). Write a program to implement chain of n processes.

CODE:

```
#include<unistd.h>

#include<stdio.h>

#include<stdlib.h>

#include <sys/types.h>

#include<sys/wait.h>

int main()

{

     int n;

     printf("Enter n: ");

     scanf("%d",&n);

     for(int i=0;i<n;i++)

     {

          int pid = fork();

          if(pid == 0)

          {

               printf("Hello from child (%d) of parent
(%d)\n",getpid(),getppid());

          }

          else

          {

               wait(NULL);

               exit(0);
```

```
            }
        }
        return 0;
}
```

OUTPUT:



```
birva@LAPTOP-TJ5CO14G:/mnt/c/Users/Admin/Documents/OS/LAB-3$ gcc assignment2.c
birva@LAPTOP-TJ5CO14G:/mnt/c/Users/Admin/Documents/OS/LAB-3$ ./a.out
Enter n: 4
Hello from child (271) of parent (270)
Hello from child (272) of parent (271)
Hello from child (273) of parent (272)
Hello from child (274) of parent (273)
birva@LAPTOP-TJ5CO14G:/mnt/c/Users/Admin/Documents/OS/LAB-3$
```