LAB #05
Process Creation and Execution

Spring 2023

CSE-204L Operating Systems Lab

Submitted by: MUHAMMAD SADEEQ

Registration No.: 21PWCSE2028

Section: C

“On my honor, as a student of the University of Engineering and Technology, I have neither given nor received unauthorized assistance on this academic work”

Submitted to:

Engr. Madiha Sher

(31 Mar 2023)

Department of Computer systems engineering
University of Engineering and Technology,
Peshawar

CSE 302L: Operating Systems Lab

LAB ASSESSMENT RUBRICS

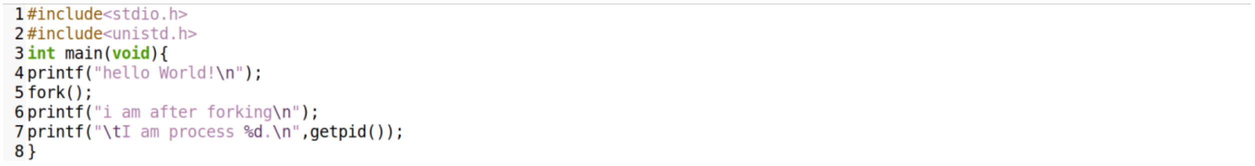
Marking Criteria	**Exceeds expectation (2.5)**	**Meets expectation (1.5)**	**Does not meet expectation (0)**	**Score**
1. **Correctness**	Program compiles (no errors and no warnings). Program always works correctly and meets the specification(s). Completed between 81-100% of the requirements.	Program compiles (no errors and some warnings). Some details of the program specification are violated, program functions incorrectly for some inputs. Completed between 4180% of the requirements.	Program fails to or compile with lots of warnings. Program only functions correctly in very limited cases or not at all. Completed less than 40% of the requirements.	
2. **Delivery**	Delivered on time, and in correct format (disk, email, hard copy etc.)	Not delivered on time, or slightly incorrect format.	Not delivered on time or not in correct format.	
3. **Coding Standards**	Proper indentation, whitespace, line length, wrapping, comments and references.	Missing some of whitespace, line length, wrapping, comments or references.	Poor use of whitespace, line length, wrapping, comments and references.	
4. **Presentation of document**	Includes name, date, and assignment title. Task titles, objectives, output screenshots included and good formatting and excellently organized.	Includes name, date, and assignment title. Task titles, objectives, output screenshots included and good formatting.	No name, date, or assignment title included. No task titles, no objectives, no output screenshots, poor formatting.	

Instructor:

Name: __Engr. Madiha Sher___ Signature: MUHAMMAD SADEEQ

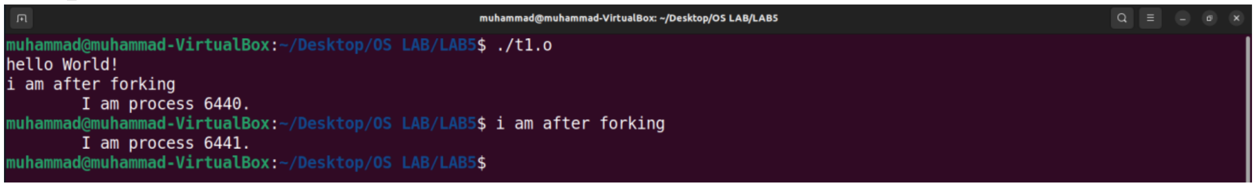
Example 1

CODE:



```c
#include<stdio.h>
#include<unistd.h>
int main(void){
printf("hello World!\n");
fork();
printf("i am after forking\n");
printf("\tI am process %d.\n",getpid());
}
```

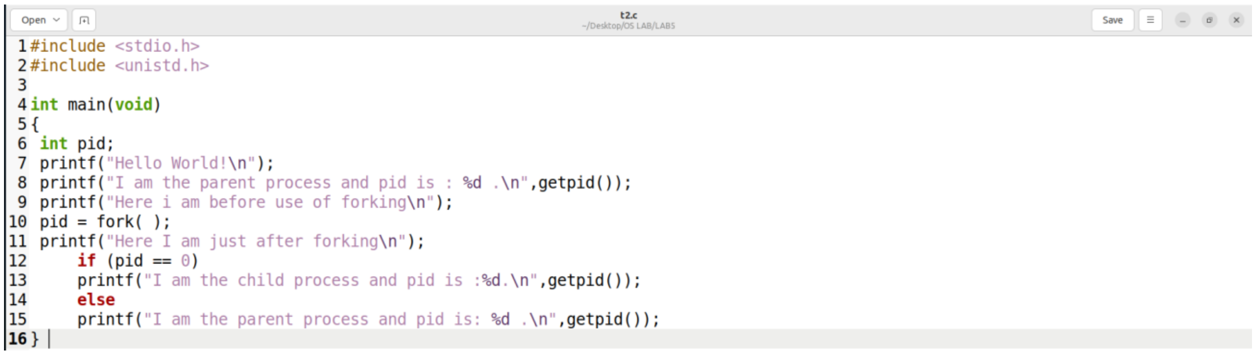
Output:



```
muhammad@muhammad-VirtualBox:~/Desktop/OS LAB/LAB5$ ./t1.o
hello World!
i am after forking
        I am process 6440.
muhammad@muhammad-VirtualBox:~/Desktop/OS LAB/LAB5$ i am after forking
        I am process 6441.
muhammad@muhammad-VirtualBox:~/Desktop/OS LAB/LAB5$
```

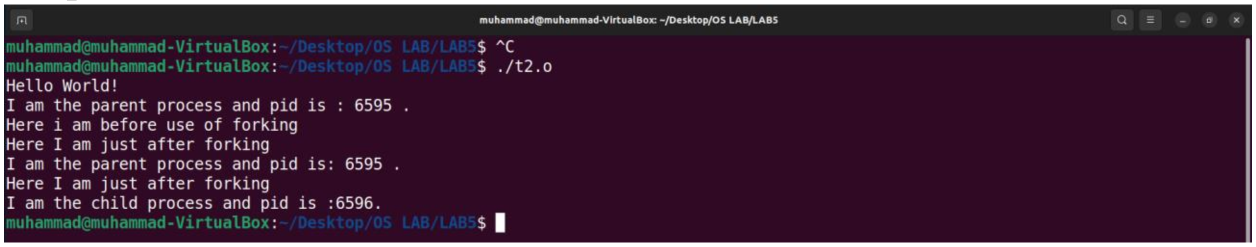
Example 2

CODE:



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

int main(void)
{
 int pid;
 printf("Hello World!\n");
 printf("I am the parent process and pid is : %d .\n",getpid());
 printf("Here i am before use of forking\n");
 pid = fork( );
 printf("Here I am just after forking\n");
     if (pid == 0)
     printf("I am the child process and pid is :%d.\n",getpid());
     else
     printf("I am the parent process and pid is: %d .\n",getpid());
}
```

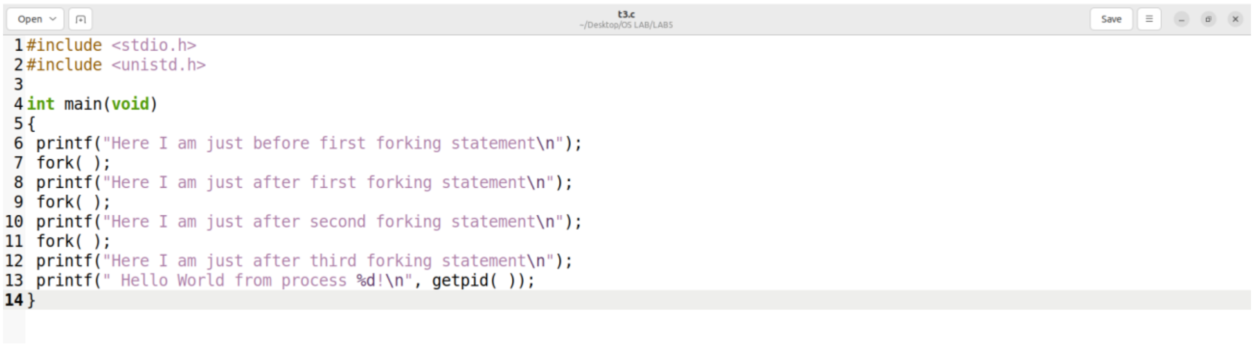
Output:



```
muhammad@muhammad-VirtualBox:~/Desktop/OS LAB/LAB5$ ^C
muhammad@muhammad-VirtualBox:~/Desktop/OS LAB/LAB5$ ./t2.o
Hello World!
I am the parent process and pid is : 6595 .
Here i am before use of forking
Here I am just after forking
I am the parent process and pid is: 6595 .
Here I am just after forking
I am the child process and pid is :6596.
muhammad@muhammad-VirtualBox:~/Desktop/OS LAB/LAB5$
```

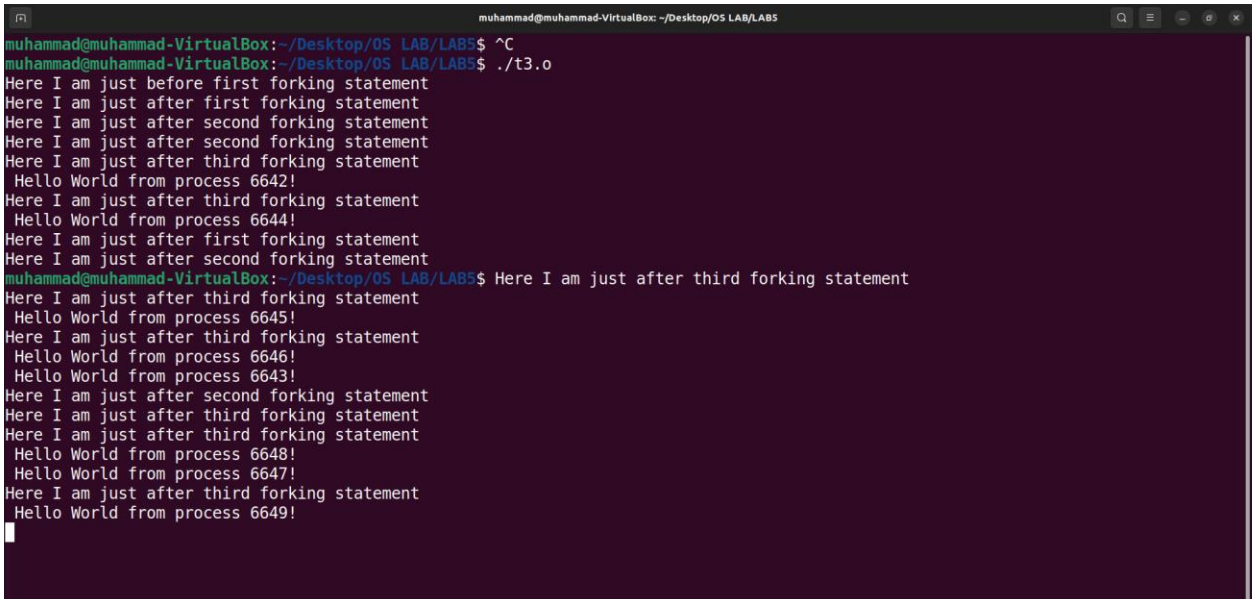
Example 3

CODE:



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

int main(void)
{
 printf("Here I am just before first forking statement\n");
 fork( );
 printf("Here I am just after first forking statement\n");
 fork( );
 printf("Here I am just after second forking statement\n");
 fork( );
 printf("Here I am just after third forking statement\n");
 printf(" Hello World from process %d!\n", getpid( ));
}
```

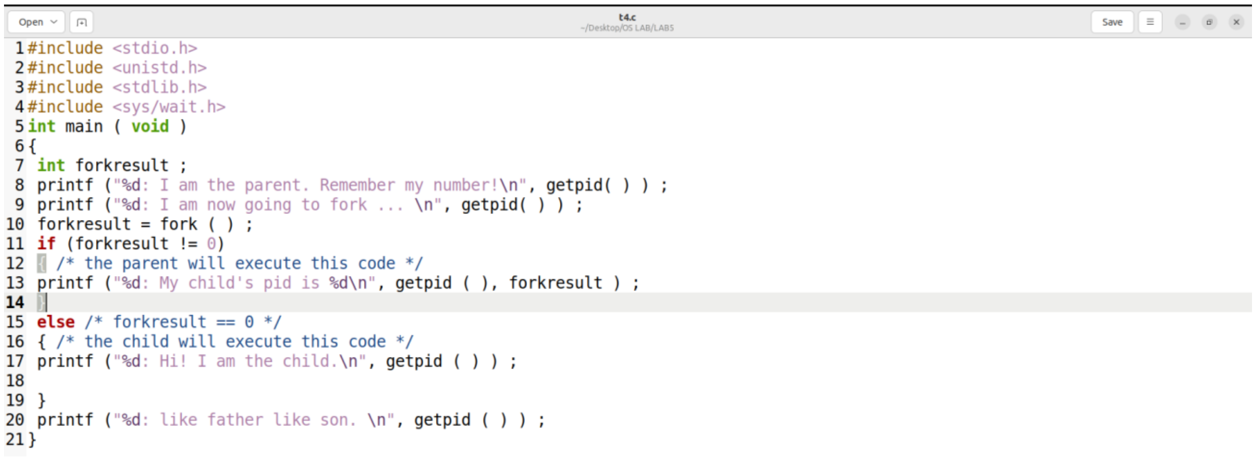
Output:



```
muhammad@muhammad-VirtualBox:~/Desktop/OS LAB/LAB5$ ^C
muhammad@muhammad-VirtualBox:~/Desktop/OS LAB/LAB5$ ./t3.o
Here I am just before first forking statement
Here I am just after first forking statement
Here I am just after second forking statement
Here I am just after second forking statement
Here I am just after third forking statement
 Hello World from process 6642!
Here I am just after third forking statement
 Hello World from process 6644!
Here I am just after first forking statement
Here I am just after second forking statement
muhammad@muhammad-VirtualBox:~/Desktop/OS LAB/LAB5$ Here I am just after third forking statement
Here I am just after third forking statement
 Hello World from process 6645!
Here I am just after third forking statement
 Hello World from process 6646!
 Hello World from process 6643!
Here I am just after second forking statement
Here I am just after third forking statement
Here I am just after third forking statement
 Hello World from process 6648!
 Hello World from process 6647!
Here I am just after third forking statement
 Hello World from process 6649!
```

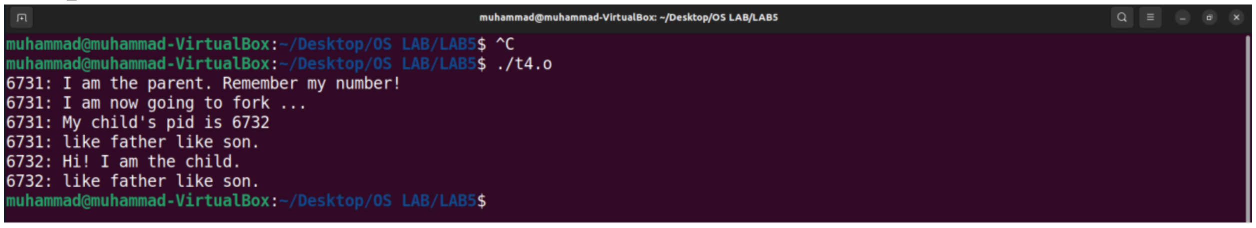
Example 4

CODE:



```c
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/wait.h>
int main ( void )
{
 int forkresult ;
 printf ("%d: I am the parent. Remember my number!\n", getpid( ) ) ;
 printf ("%d: I am now going to fork ... \n", getpid( ) ) ;
 forkresult = fork ( ) ;
 if (forkresult != 0)
 { /* the parent will execute this code */
 printf ("%d: My child's pid is %d\n", getpid ( ), forkresult ) ;
 }
 else /* forkresult == 0 */
 { /* the child will execute this code */
 printf ("%d: Hi! I am the child.\n", getpid ( ) ) ;

 }
 printf ("%d: like father like son. \n", getpid ( ) ) ;
}
```

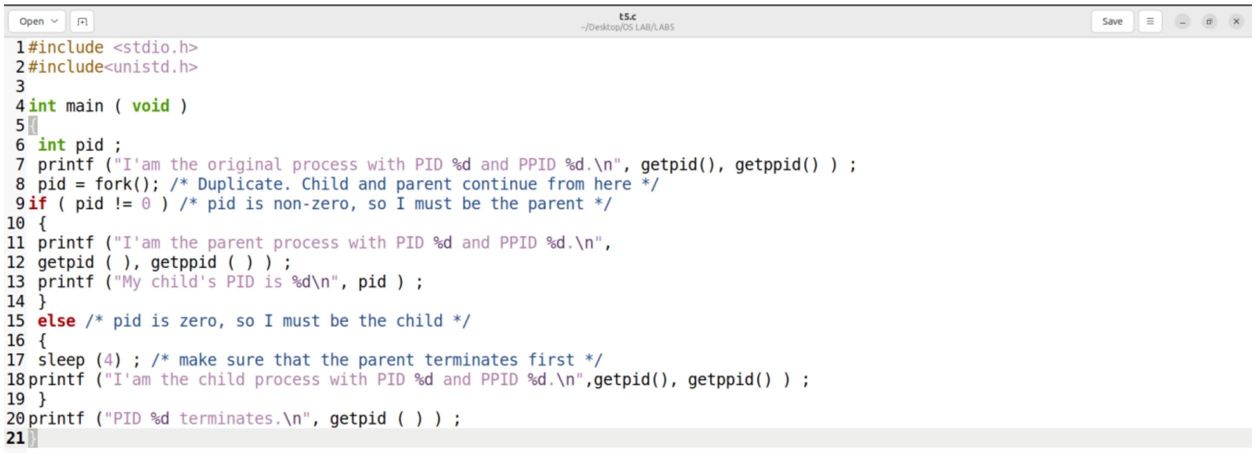
Output:



```
muhammad@muhammad-VirtualBox:~/Desktop/OS LAB/LAB5$ ^C
muhammad@muhammad-VirtualBox:~/Desktop/OS LAB/LAB5$ ./t4.o
6731: I am the parent. Remember my number!
6731: I am now going to fork ...
6731: My child's pid is 6732
6731: like father like son.
6732: Hi! I am the child.
6732: like father like son.
muhammad@muhammad-VirtualBox:~/Desktop/OS LAB/LAB5$
```

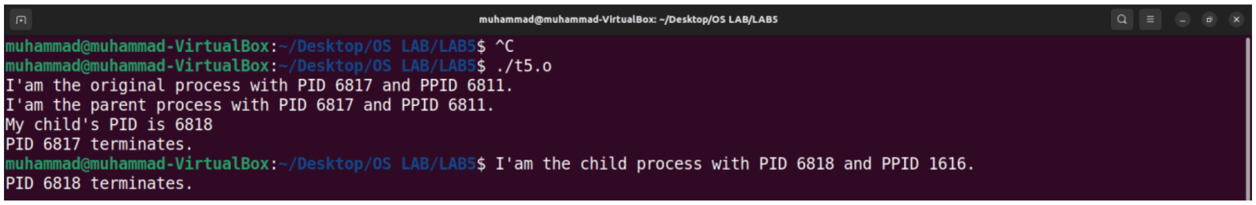
Example 5

CODE:



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

int main ( void )
{
 int pid ;
 printf ("I'am the original process with PID %d and PPID %d.\n", getpid(), getppid() ) ;
 pid = fork(); /* Duplicate. Child and parent continue from here */
if ( pid != 0 ) /* pid is non-zero, so I must be the parent */
 {
 printf ("I'am the parent process with PID %d and PPID %d.\n",
 getpid ( ), getppid ( ) ) ;
 printf ("My child's PID is %d\n", pid ) ;
 }
 else /* pid is zero, so I must be the child */
 {
 sleep (4) ; /* make sure that the parent terminates first */
printf ("I'am the child process with PID %d and PPID %d.\n",getpid(), getppid() ) ;
 }
printf ("PID %d terminates.\n", getpid ( ) ) ;
}
```

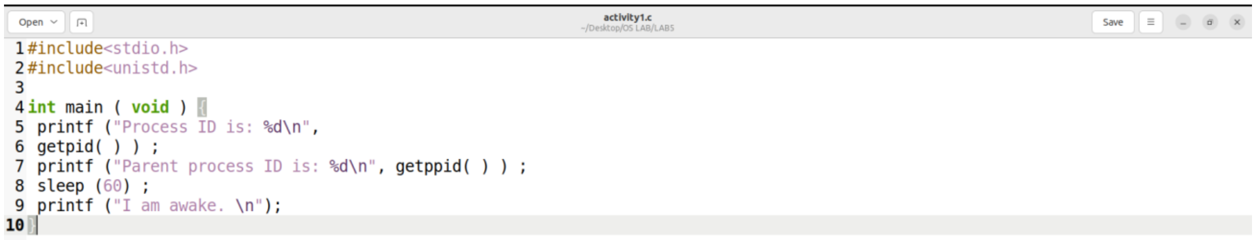
Output:



```
muhammad@muhammad-VirtualBox:~/Desktop/OS LAB/LAB5$ ^C
muhammad@muhammad-VirtualBox:~/Desktop/OS LAB/LAB5$ ./t5.o
I'am the original process with PID 6817 and PPID 6811.
I'am the parent process with PID 6817 and PPID 6811.
My child's PID is 6818
PID 6817 terminates.
muhammad@muhammad-VirtualBox:~/Desktop/OS LAB/LAB5$ I'am the child process with PID 6818 and PPID 1616.
PID 6818 terminates.
```

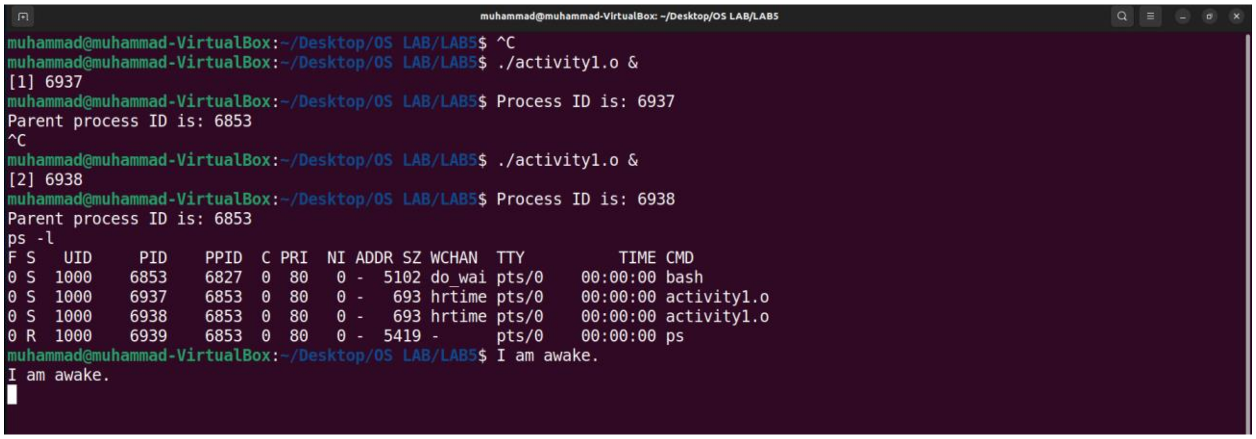
Task #01

CODE:



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

int main ( void ) {
 printf ("Process ID is: %d\n",
 getpid( ) ) ;
 printf ("Parent process ID is: %d\n", getppid( ) ) ;
 sleep (60) ;
 printf ("I am awake. \n");
}
```

Output:



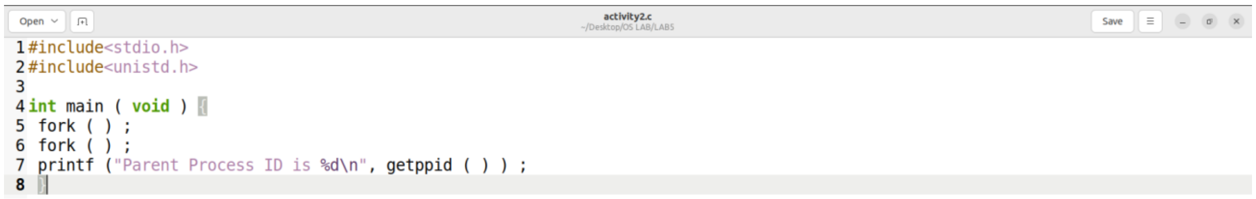
```
muhammad@muhammad-VirtualBox:~/Desktop/OS LAB/LAB5$ ^C
muhammad@muhammad-VirtualBox:~/Desktop/OS LAB/LAB5$ ./activity1.o &
[1] 6937
muhammad@muhammad-VirtualBox:~/Desktop/OS LAB/LAB5$ Process ID is: 6937
Parent process ID is: 6853
^C
muhammad@muhammad-VirtualBox:~/Desktop/OS LAB/LAB5$ ./activity1.o &
[2] 6938
muhammad@muhammad-VirtualBox:~/Desktop/OS LAB/LAB5$ Process ID is: 6938
Parent process ID is: 6853
ps -l
F S   UID     PID    PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  1000    6853    6827  0  80   0 -  5102 do_wai pts/0    00:00:00 bash
0 S  1000    6937    6853  0  80   0 -   693 hrtime pts/0    00:00:00 activity1.o
0 S  1000    6938    6853  0  80   0 -   693 hrtime pts/0    00:00:00 activity1.o
0 R  1000    6939    6853  0  80   0 -  5419 -      pts/0    00:00:00 ps
muhammad@muhammad-VirtualBox:~/Desktop/OS LAB/LAB5$ I am awake.
I am awake.
```

Observation:

When i run the program twice as a background process, two instances of the program run independently in the background, and I saw two processes with different process IDs in the process table.
Initially, both processes were in the "S" (Sleeping) state as they have invoked the "sleep" system call and are waiting for the specified amount of time (60 seconds) to elapse.
After 60 seconds, the processes waked up and print "I am awake" on the console. Then, they terminate, and their status change to "Z" (Zombie) until their parent process collects their exit status using the "wait" system call.
When i run the program again, we will see two new processes with different PIDs and PPIDs. The process state and behavior will be the same as before.
In summary, we can observe that each instance of the program runs independently as a separate process with its own PID and PPID. The process remains in the Sleeping state for 60 seconds and then terminates, leaving behind a Zombie process until the parent process collects its exit status.

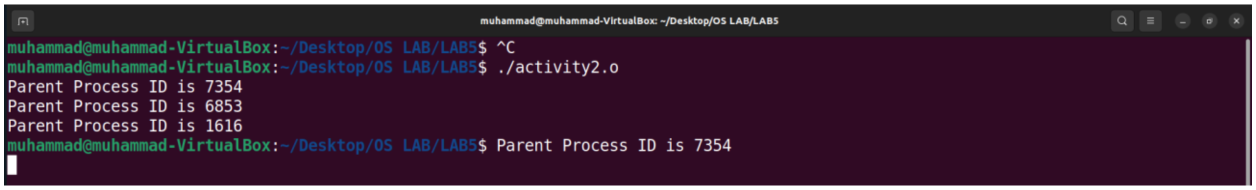
Task #02

CODE:



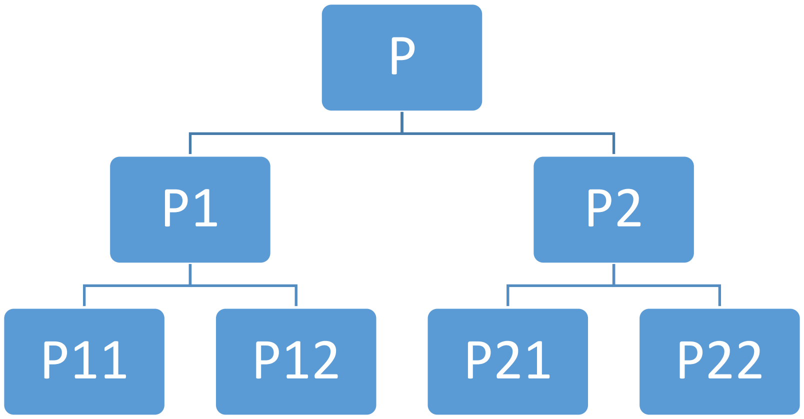
```
#include<stdio.h>
#include<unistd.h>

int main ( void ) {
 fork ( ) ;
 fork ( ) ;
 printf ("Parent Process ID is %d\n", getppid ( ) ) ;
}
```

Output:



```
muhammad@muhammad-VirtualBox:~/Desktop/OS LAB/LAB5$ ^C
muhammad@muhammad-VirtualBox:~/Desktop/OS LAB/LAB5$ ./activity2.o
Parent Process ID is 7354
Parent Process ID is 6853
Parent Process ID is 1616
muhammad@muhammad-VirtualBox:~/Desktop/OS LAB/LAB5$ Parent Process ID is 7354
```

Family Tree

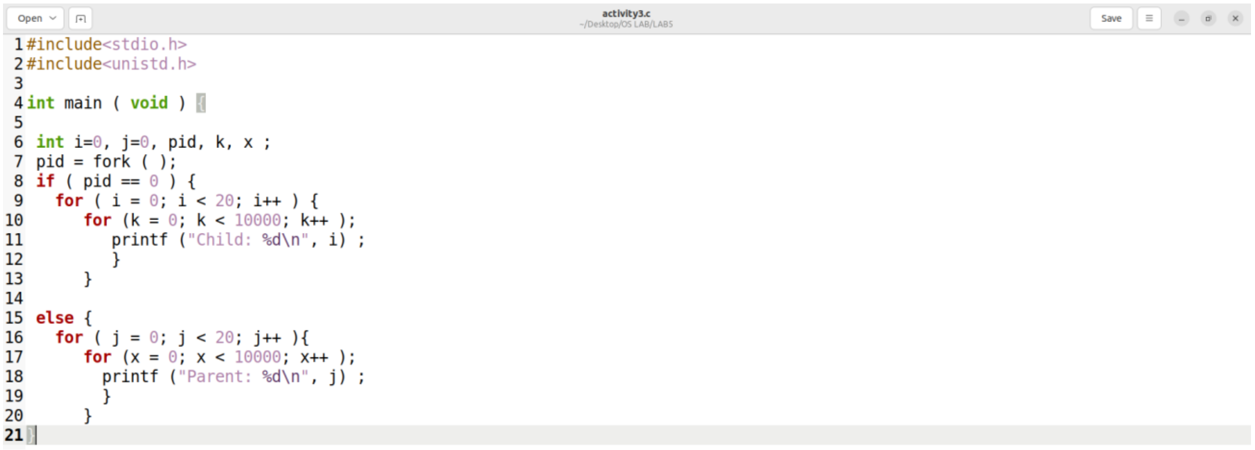


Observation:

The program will create four processes, and each process will execute the printf() statement, outputting the ID of the original parent process. The order in which the statements are executed may not follow the order in which the fork() statements were executed.

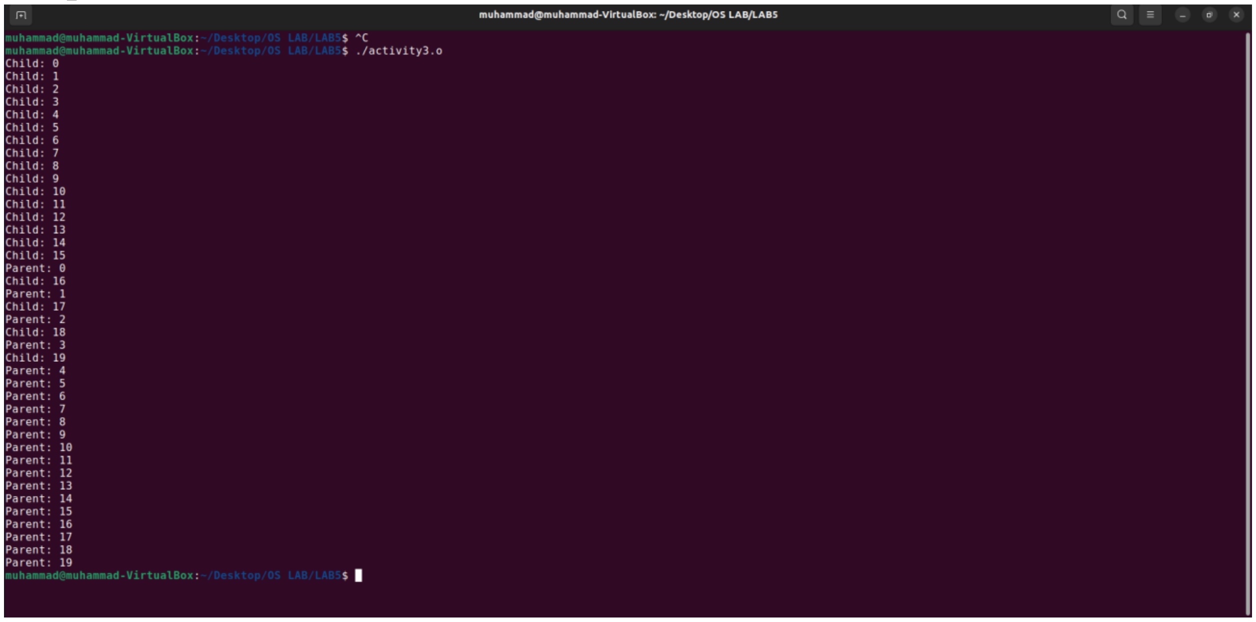
Task #03

CODE:



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

int main ( void ) {

 int i=0, j=0, pid, k, x ;
 pid = fork ( );
 if ( pid == 0 ) {
   for ( i = 0; i < 20; i++ ) {
      for (k = 0; k < 10000; k++ );
         printf ("Child: %d\n", i) ;
         }
      }

 else {
   for ( j = 0; j < 20; j++ ){
      for (x = 0; x < 10000; x++ );
        printf ("Parent: %d\n", j) ;
        }
      }
}
```

Output:



```
muhammad@muhammad-VirtualBox:~/Desktop/OS LAB/LAB5$ ^C
muhammad@muhammad-VirtualBox:~/Desktop/OS LAB/LAB5$ ./activity3.o
Child: 0
Child: 1
Child: 2
Child: 3
Child: 4
Child: 5
Child: 6
Child: 7
Child: 8
Child: 9
Child: 10
Child: 11
Child: 12
Child: 13
Child: 14
Child: 15
Parent: 0
Child: 16
Parent: 1
Child: 17
Parent: 2
Child: 18
Parent: 3
Child: 19
Parent: 4
Parent: 5
Parent: 6
Parent: 7
Parent: 8
Parent: 9
Parent: 10
Parent: 11
Parent: 12
Parent: 13
Parent: 14
Parent: 15
Parent: 16
Parent: 17
Parent: 18
Parent: 19
muhammad@muhammad-VirtualBox:~/Desktop/OS LAB/LAB5$ 
```

Observation:

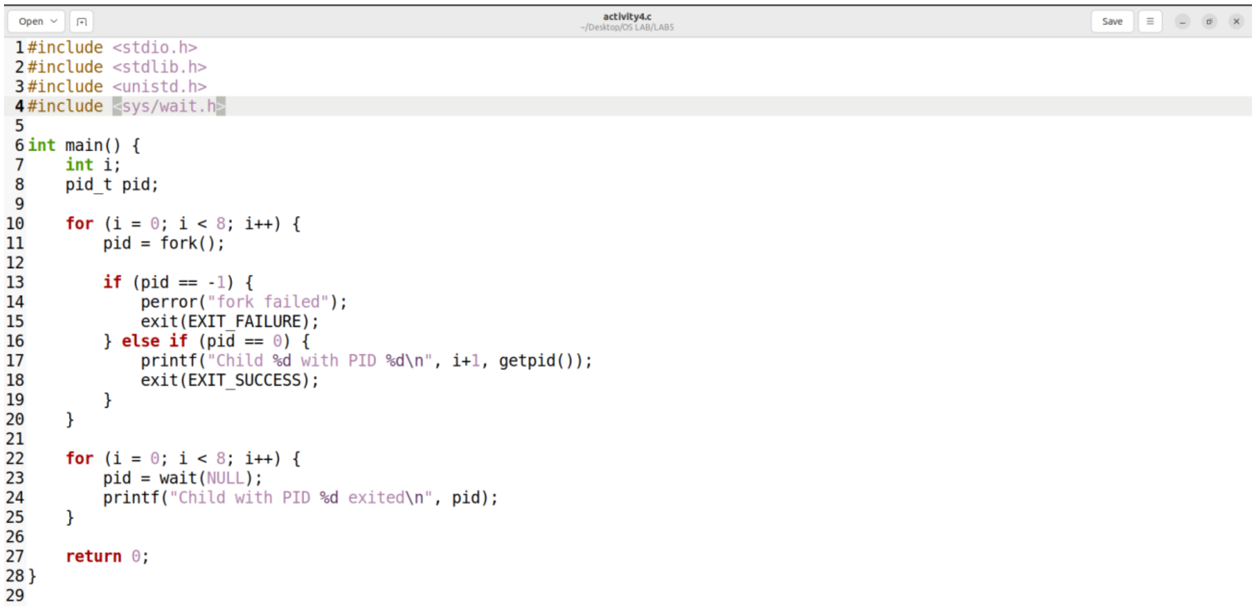
This program creates a parent and child process using the fork() system call. The child process executes the code inside the if block, and the parent process executes the code inside the else block.

Both the child and parent processes have a loop that iterates 20 times and performs a delay of 10000 iterations for each iteration of the loop using the for loop. Inside the loop, a message is printed to the console indicating whether the message was printed by the child or parent process and the current iteration number.

Since both the parent and child processes are running concurrently, the output from the two processes will be interleaved, and we can observe the time slicing used by UNIX.

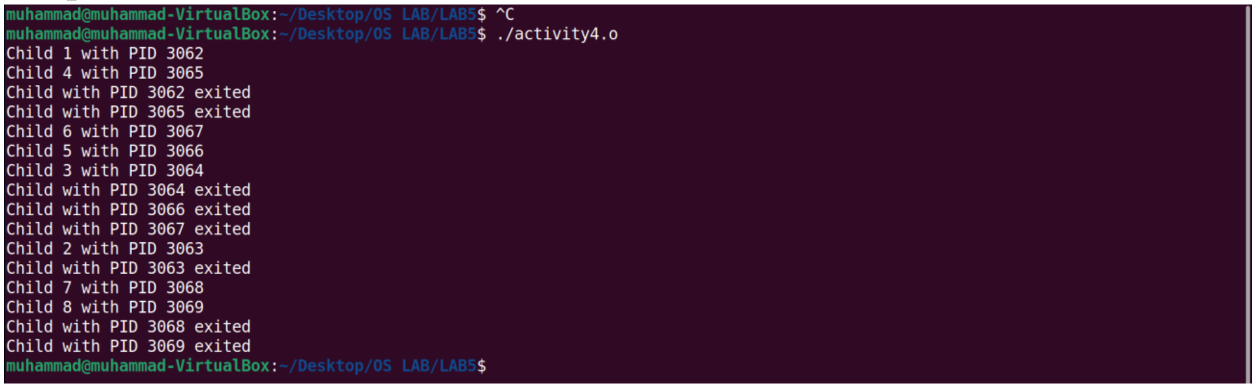
Task #04

CODE:



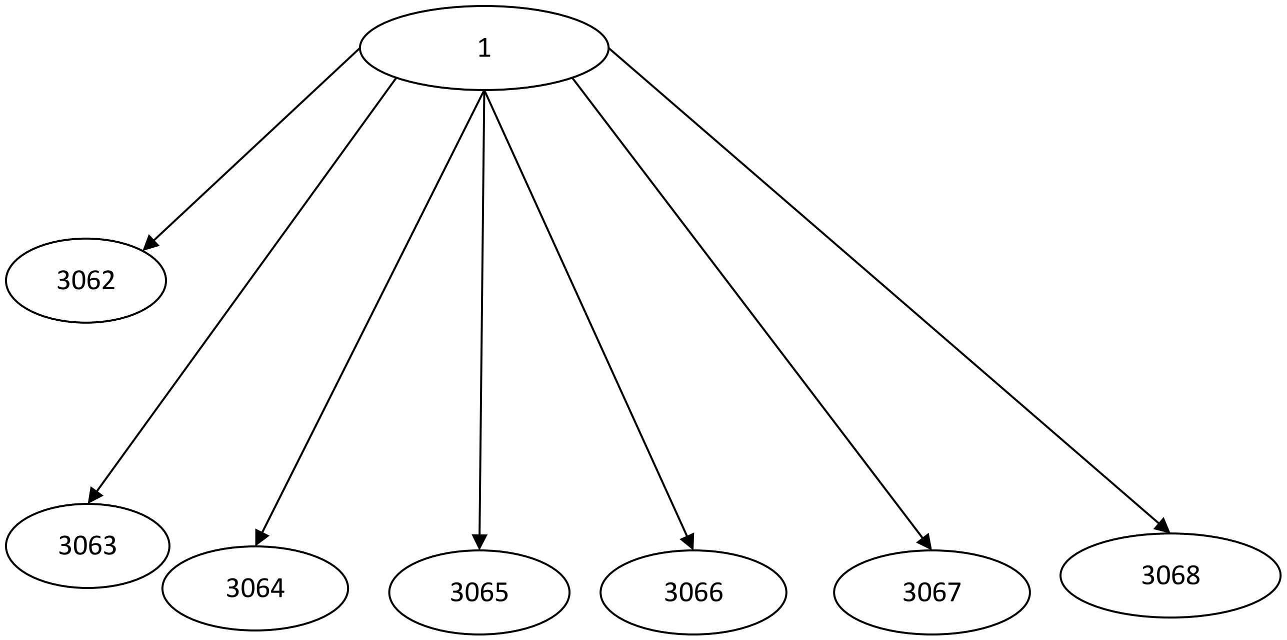
```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>

int main() {
    int i;
    pid_t pid;

    for (i = 0; i < 8; i++) {
        pid = fork();

        if (pid == -1) {
            perror("fork failed");
            exit(EXIT_FAILURE);
        } else if (pid == 0) {
            printf("Child %d with PID %d\n", i+1, getpid());
            exit(EXIT_SUCCESS);
        }
    }

    for (i = 0; i < 8; i++) {
        pid = wait(NULL);
        printf("Child with PID %d exited\n", pid);
    }

    return 0;
}

```

Output:



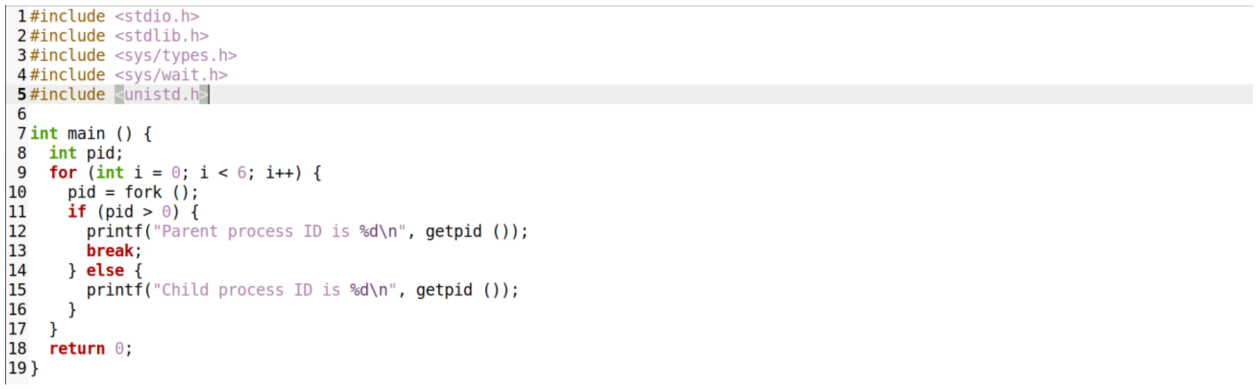
```
muhammad@muhammad-VirtualBox:~/Desktop/OS LAB/LAB5$ ^C
muhammad@muhammad-VirtualBox:~/Desktop/OS LAB/LAB5$ ./activity4.o
Child 1 with PID 3062
Child 4 with PID 3065
Child with PID 3062 exited
Child with PID 3065 exited
Child 6 with PID 3067
Child 5 with PID 3066
Child 3 with PID 3064
Child with PID 3064 exited
Child with PID 3066 exited
Child with PID 3067 exited
Child 2 with PID 3063
Child with PID 3063 exited
Child 7 with PID 3068
Child 8 with PID 3069
Child with PID 3068 exited
Child with PID 3069 exited
muhammad@muhammad-VirtualBox:~/Desktop/OS LAB/LAB5$
```

Figure:



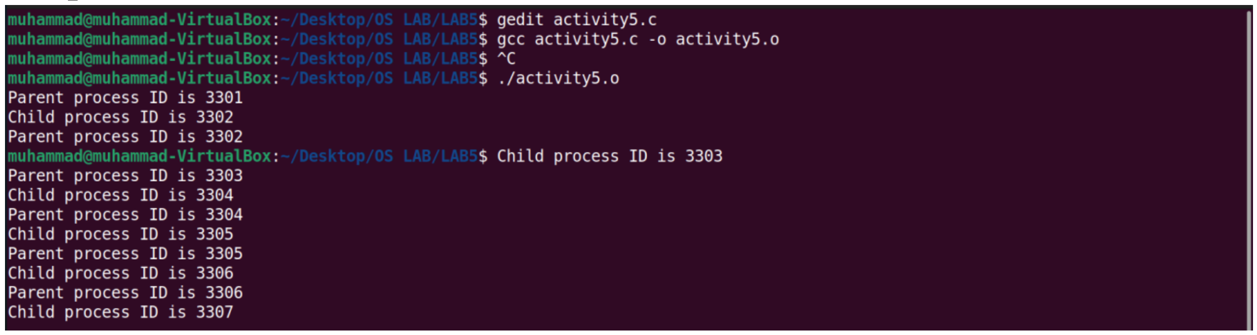
Task #05

CODE:



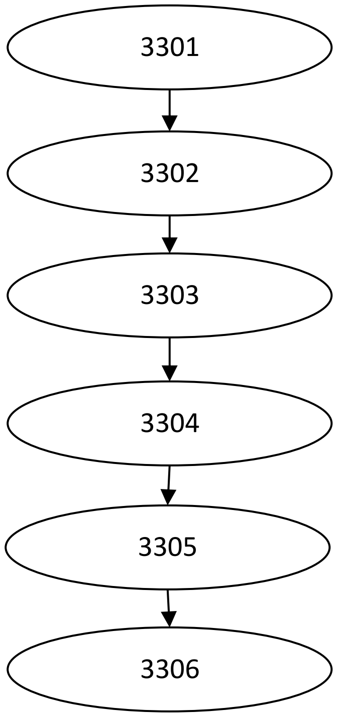
```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>

int main () {
  int pid;
  for (int i = 0; i < 6; i++) {
    pid = fork ();
    if (pid > 0) {
      printf("Parent process ID is %d\n", getpid ());
      break;
    } else {
      printf("Child process ID is %d\n", getpid ());
    }
  }
  return 0;
}
```

Output:



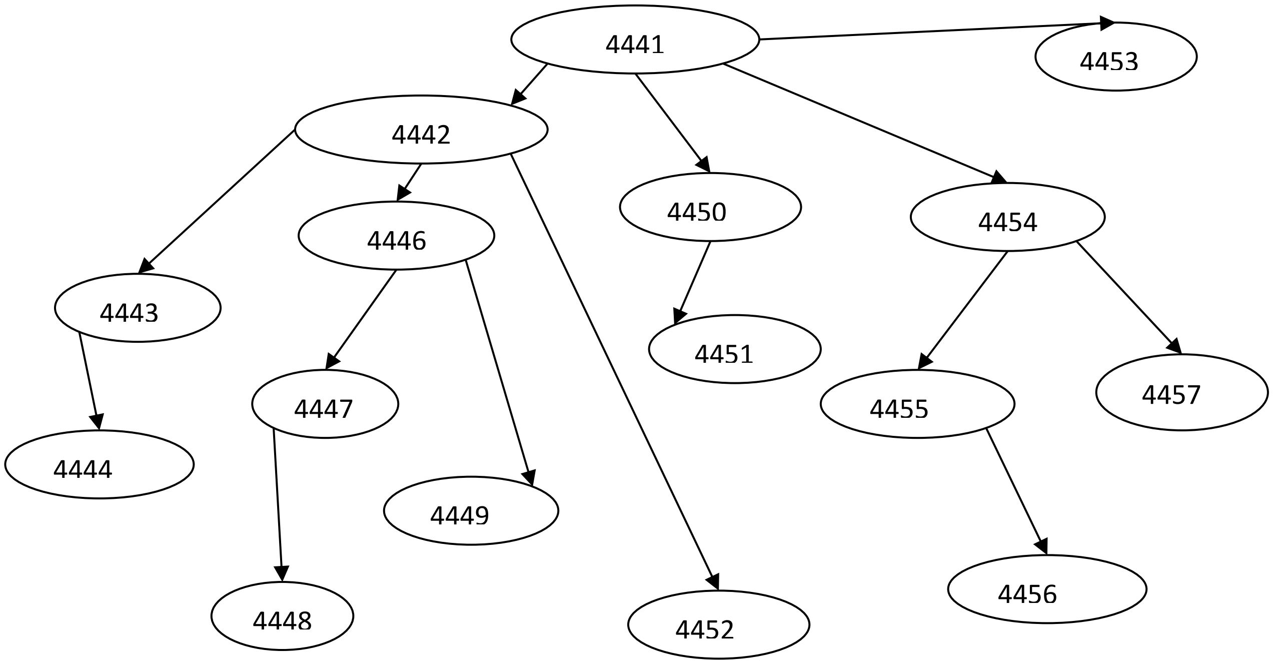
```
muhammad@muhammad-VirtualBox:~/Desktop/OS LAB/LAB5$ gedit activity5.c
muhammad@muhammad-VirtualBox:~/Desktop/OS LAB/LAB5$ gcc activity5.c -o activity5.o
muhammad@muhammad-VirtualBox:~/Desktop/OS LAB/LAB5$ ^C
muhammad@muhammad-VirtualBox:~/Desktop/OS LAB/LAB5$ ./activity5.o
Parent process ID is 3301
Child process ID is 3302
Parent process ID is 3302
muhammad@muhammad-VirtualBox:~/Desktop/OS LAB/LAB5$ Child process ID is 3303
Parent process ID is 3303
Child process ID is 3304
Parent process ID is 3304
Child process ID is 3305
Parent process ID is 3305
Child process ID is 3306
Parent process ID is 3306
Child process ID is 3307
```

Figure:



Task #06

Figure:


4441
4453
4442
4450
4454
4446
4443
4451
4447
4455
4457
4444
4449
4448
4452
4456