Advanced Unix Programming
Assignment-5

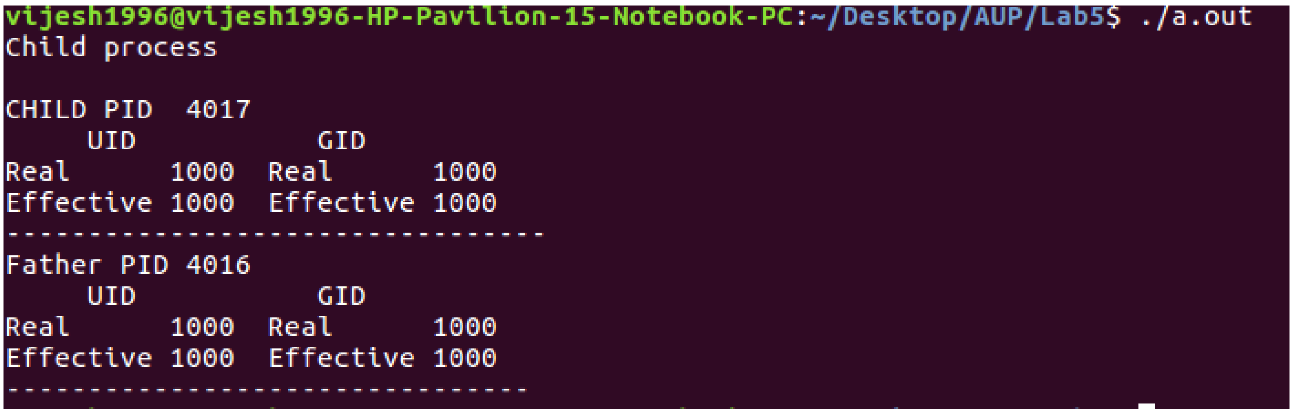
Group Members:
Vijesh Ghandare 111403013
Nikhil Gawande 111408013
Anupam Godse 111408016

Q1.A child process inherits real user id, real group id, effective user id and effective group id of the parent process, while process id and parent process id are not. Demonstrate.

CODE:

```
#include <stdio.h>
#include <unistd.h>
int main () {
        pid_t pid;
        int status = 2;
        pid = fork();

        if (!pid) {
                 puts("Child process\n");
                 printf("CHILD PID  %d \n", getpid());
                printf("     UID           GID  \n"
                    "Real      %d  Real      %d  \n"
                    "Effective %d  Effective %d  \n",
                        getuid (),     getgid (),
                        geteuid(),     getegid()
                 );
                puts("--------------------------------");
                 return;
        }

        wait(status);

        printf("Father PID %d\n", getpid());
        printf("     UID           GID  \n"
             "Real      %d  Real      %d  \n"
             "Effective %d  Effective %d  \n",
                 getuid (),     getgid (),
                 geteuid(),     getegid()
          );
        puts("--------------------------------");

 return 0;
}
```

Demonstration:



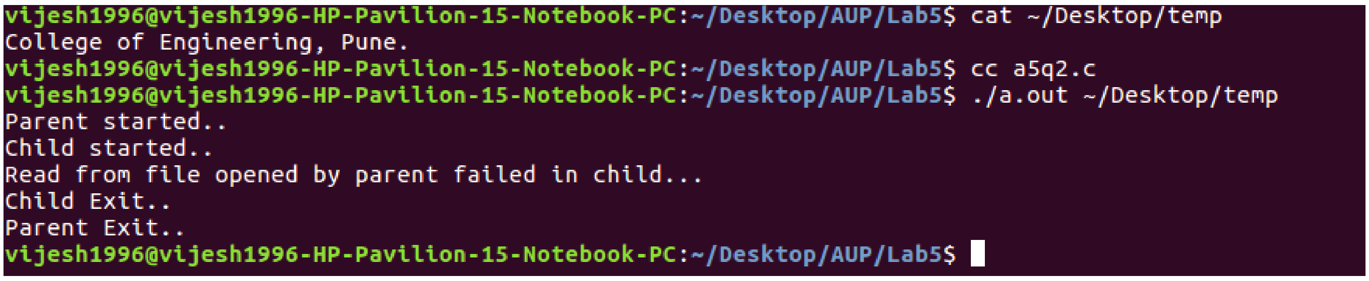
```
vijesh1996@vijesh1996-HP-Pavilion-15-Notebook-PC:~/Desktop/AUP/Lab5$ ./a.out
Child process

CHILD PID  4017
     UID           GID
Real      1000  Real      1000
Effective 1000  Effective 1000
---------------------------------
Father PID 4016
     UID           GID
Real      1000  Real      1000
Effective 1000  Effective 1000
--------------------------------
```

Q2.Verify whether it is possible for a child process to handle a file opened by its parent Immediately after the fork() call?

CODE:

```
#include <stdio.h>
#include<stdlib.h>
#include<sys/types.h>
#include<sys/wait.h>
#include<unistd.h>
#include <sys/stat.h>
#include <fcntl.h>
int main(int argc, char **argv) {
        int i= 0, pid,fd;
        char *str;
        str = (char*)malloc(sizeof(char)*128);
                pid = fork();                                          //fork
                if (pid > 0) {
                        printf("Parent started..\n");
                        fd = open(argv[1],O_RDONLY);            //file opened in parent
                        if(fd < 0)
                                printf("open failed...!!!\n");
                }
                else{
                        sleep(2);
                        printf("Child started..\n");
                        if(read(fd,str,128) < 0)                //reading from file opened by
parent
                                printf("Read from file opened by parent failed in child...\n");
                        else
                                printf("%s",str);
                        printf("Child Exit..\n");
                        return 0;
                }
                wait(NULL);                                                     //wait until child
exits.
                printf("Parent Exit..\n");
        return 0;
}
```

Verification:



```
vijesh1996@vijesh1996-HP-Pavilion-15-Notebook-PC:~/Desktop/AUP/Lab5$ cat ~/Desktop/temp
College of Engineering, Pune.
vijesh1996@vijesh1996-HP-Pavilion-15-Notebook-PC:~/Desktop/AUP/Lab5$ cc a5q2.c
vijesh1996@vijesh1996-HP-Pavilion-15-Notebook-PC:~/Desktop/AUP/Lab5$ ./a.out ~/Desktop/temp
Parent started..
Child started..
Read from file opened by parent failed in child...
Child Exit..
Parent Exit..
vijesh1996@vijesh1996-HP-Pavilion-15-Notebook-PC:~/Desktop/AUP/Lab5$ 
```

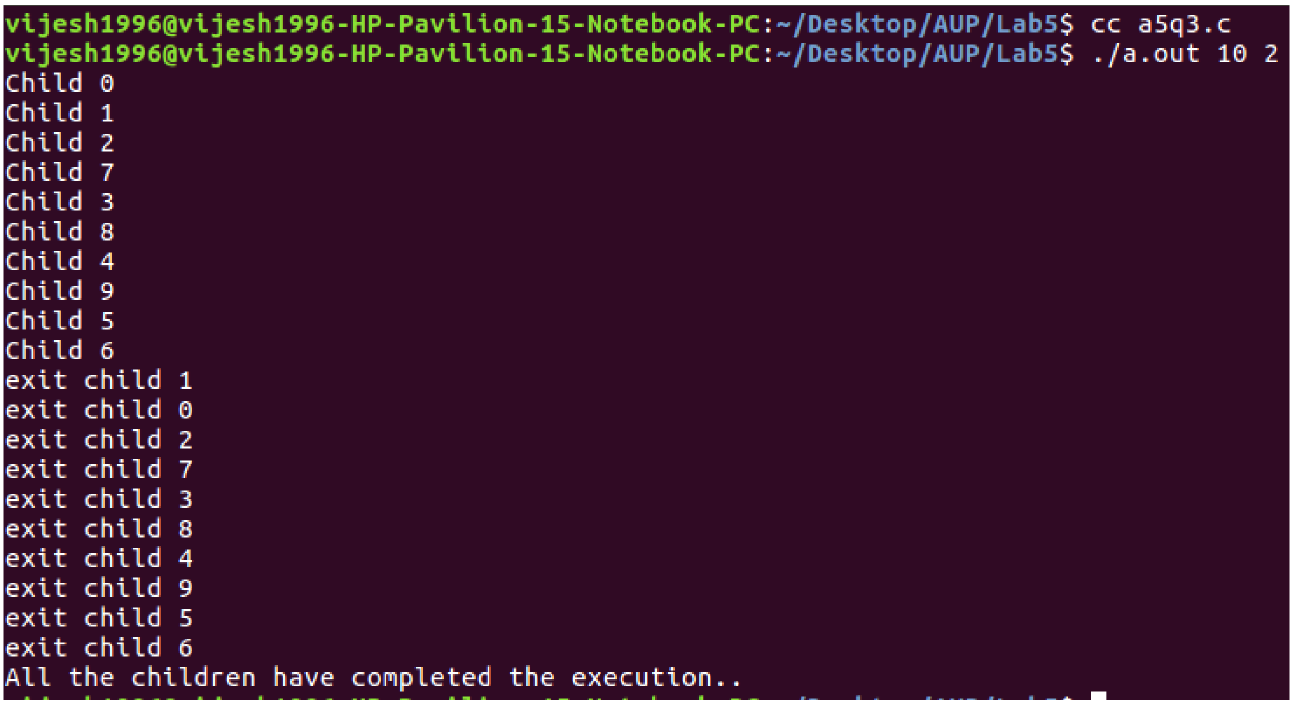
So, it doesn’t allow child to share the file opened by parent.

Q3.The parent starts as many child processes as to the value of its integer command line argument. The child processes simply sleep for the time specified by the argument, then exit. After starting all the children, the parent process must wait until they have all terminated before terminating itself.

CODE:

```
#include <stdio.h>
#include<stdlib.h>
#include<sys/types.h>
#include<sys/wait.h>
#include<unistd.h>
int main(int argc, char **argv) {
        int i= 0, pid;
        int n = atoi(argv[1]);                          //no. of processes
        for(i = 0; i < n; i++) {
                if (pid = fork() == 0) {                        //child
                        printf("Child %d\n",i);
                        sleep(atoi(argv[2]));                   //sleeping
                        printf("exit child %d\n",i);
                        return 0;

                }
        }
        while(wait(NULL) > 0);                                          //wait until all child exits.
        printf("All the children have completed the execution..\n");
        return 0;
}
```

Output:



```
vijesh1996@vijesh1996-HP-Pavilion-15-Notebook-PC:~/Desktop/AUP/Lab5$ cc a5q3.c
vijesh1996@vijesh1996-HP-Pavilion-15-Notebook-PC:~/Desktop/AUP/Lab5$ ./a.out 10 2
Child 0
Child 1
Child 2
Child 7
Child 3
Child 8
Child 4
Child 9
Child 5
Child 6
exit child 1
exit child 0
exit child 2
exit child 7
exit child 3
exit child 8
exit child 4
exit child 9
exit child 5
exit child 6
All the children have completed the execution..
```

here, number of child processes are 10 and sleeping time is 2, so child processes are sleeping for 2 secs and exiting. After all child exit parent is exiting.