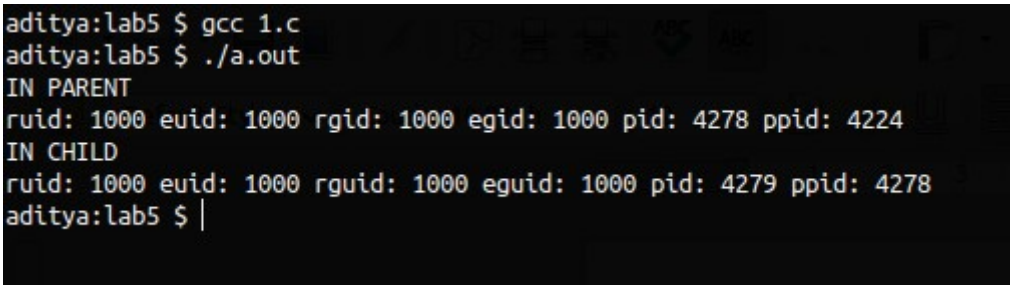
LAB 5
111403003 : AKASH SARDA
111403023 : ADITYA MALU
111403019 : JASSIM ABDUL REHMAN

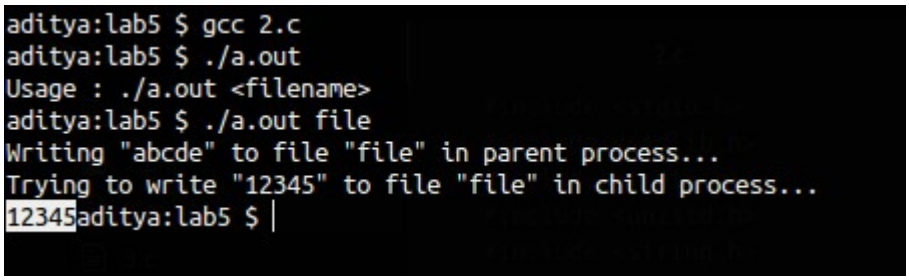
1. A child process inherits real user id, real group id, effective user id and effective group id of the parent process, while process id and parent process id are not. Demonstrate.




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

int main(){
//   pid_t ruid, euid, rguid, eguid, ppid, pid;
     if(fork() != 0){
          printf("IN PARENT\nruid: %d euid: %d rgid: %d egid: %d
pid: %d ppid: %d\n", getuid(), geteuid(), getgid(), getegid(),
getpid(), getppid());
     }else{
          printf("IN CHILD\nruid: %d euid: %d rguid: %d eguid: %d
pid: %d ppid: %d\n", getuid(), geteuid(), getgid(), getegid(),
getpid(), getppid());
     }
}
```

---

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



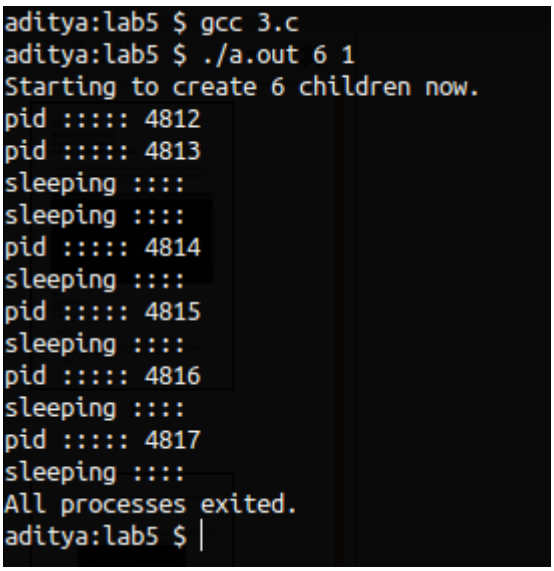
Answer : We open the file in the parent process, write “abcde” to it. While trying to write “12345” to the same file in child process, it gets written to STDOUT. Because, the fd in parent is not accessible to child.

Code :

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
#include <string.h>

int main(int argc, char *argv[]){
     if(argc != 2){
          printf("Usage : %s <filename>\n", argv[0]);
          exit(0);
     }
     char *filename = argv[1];
     char buffer1[] = "abcde", buffer2[] = "12345";
     int n, fd1;

     int pid = fork();
     if(pid){
          fd1 = open(filename , O_CREAT | O_WRONLY | O_TRUNC);
          if(fd1 == -1){
               perror("Error ");
               exit(1);
          }
          printf("Writing \"%s\" to file \"%s\" in parent
process...\n", buffer1, filename);
          write(fd1, buffer1, strlen(buffer1));
          wait(NULL);
     }
     else {
          sleep(2);
          printf("Trying to write \"%s\" to file \"%s\" in child
process...\n", buffer2, filename);
          write(fd1, buffer2, strlen(buffer2));
     }
     close(fd1);

     return 0;
}
```

---

3. 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<unistd.h>
#include<sys/wait.h>
#include<sys/types.h>

int main(int argc, char *argv[]){
     int pid, sleep_time, num_of_proc, i;
     sleep_time = atoi(argv[2]);
     num_of_proc = atoi(argv[1]);
     printf("Starting to create %d children now.\n", num_of_proc);
     for(i = 0; i < num_of_proc; i++){
          pid = fork();
          if (pid == 0){
               printf("sleeping ::::\n");
               sleep(sleep_time);
               exit(0);
          }
          else if(pid != -1){
               printf("pid ::::: %d \n", pid);
               waitpid(pid - 1, NULL, 0);
          }else{
               printf("Error in fork \n");
          }
     }
     wait(0);
     printf("All processes exited. \n");
}
```
