Advanced Unix Programming Lab 5

Abhijeet Kale	111403050
Sagar Panchmatia	111403067
Nikunj Singh	111403076

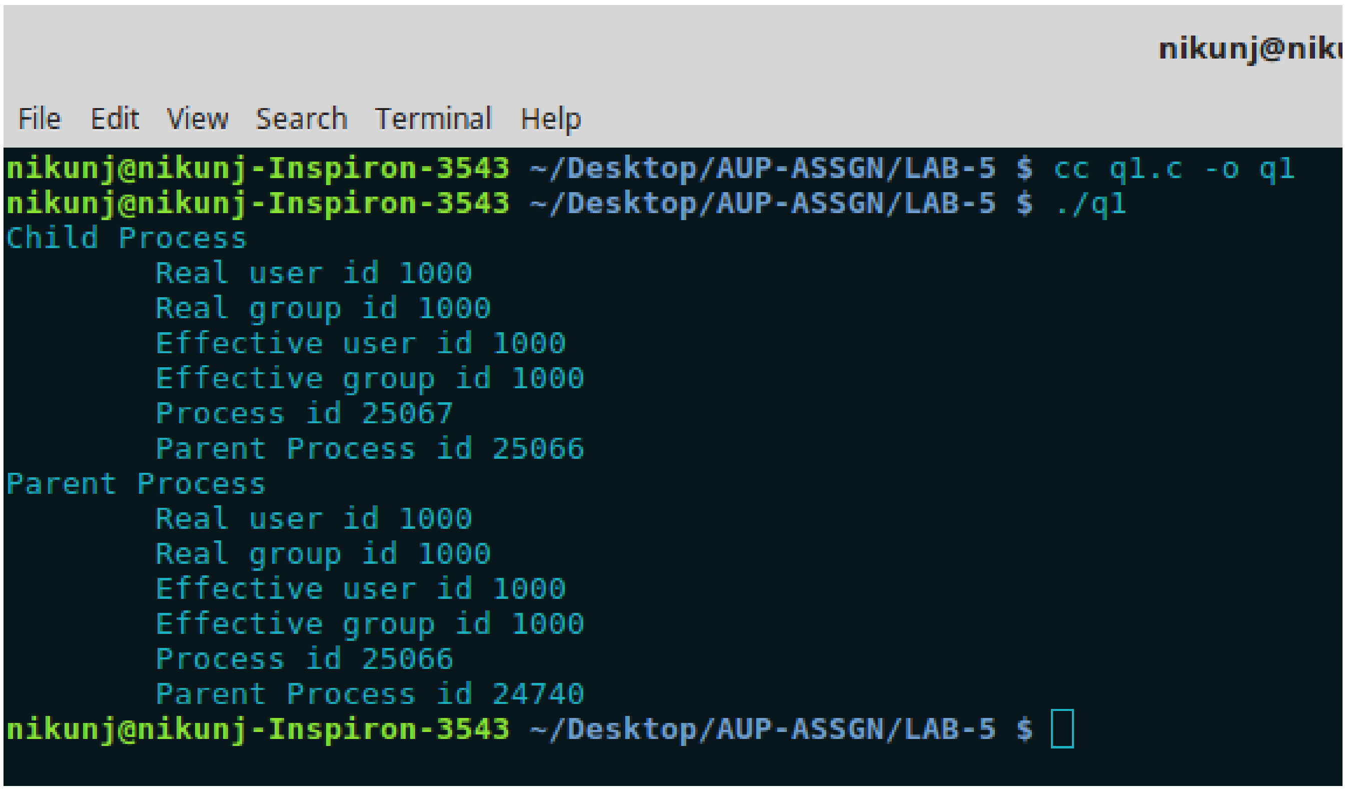
Question 1)
A child process inherits real user id, real group id, effective user id and effective group id of the parent process, while process id and parent process id are not. Demonstrate

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

int main(int argc, char *argv[]) {
	pid_t pid;
	if((pid = fork())< 0) {
		perror("Fork Failed");
		return -1;
	}
	else if(pid == 0) {
		// child process
		printf("Child Process\n");
		printf("\tReal user id %d\n\tReal group id %d\n\tEffective user
		id %d\n\tEffective group id %d\n", getuid(), getgid(), geteuid(),
		getegid());
		printf("\tProcess id %d\n\tParent Process id %d\n", getpid(),
		getppid());
	}
	else if(pid > 0) {
		// parent process
		sleep(5);
		printf("Parent Process\n");
		printf("\tReal user id %d\n\tReal group id %d\n\tEffective user
		id %d\n\tEffective group id %d\n", getuid(), getgid(), geteuid(),
		getegid());
```

```
			printf("\tProcess id %d\n\tParent Process id %d\n", getpid(),
			getppid());
		}
		return 0;
	}
```

Execution :



```
nikunj@nikunj-Inspiron-3543 ~/Desktop/AUP-ASSGN/LAB-5 $ cc q1.c -o q1
nikunj@nikunj-Inspiron-3543 ~/Desktop/AUP-ASSGN/LAB-5 $ ./q1
Child Process
        Real user id 1000
        Real group id 1000
        Effective user id 1000
        Effective group id 1000
        Process id 25067
        Parent Process id 25066
Parent Process
        Real user id 1000
        Real group id 1000
        Effective user id 1000
        Effective group id 1000
        Process id 25066
        Parent Process id 24740
nikunj@nikunj-Inspiron-3543 ~/Desktop/AUP-ASSGN/LAB-5 $ 
```

As it can be seen from the execution of the above program that
Real UserID, Real Group Id, Effective UserId, Effective GroupID is
inherited by the child process
But the process Id and ParentProcessId aren’t.

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

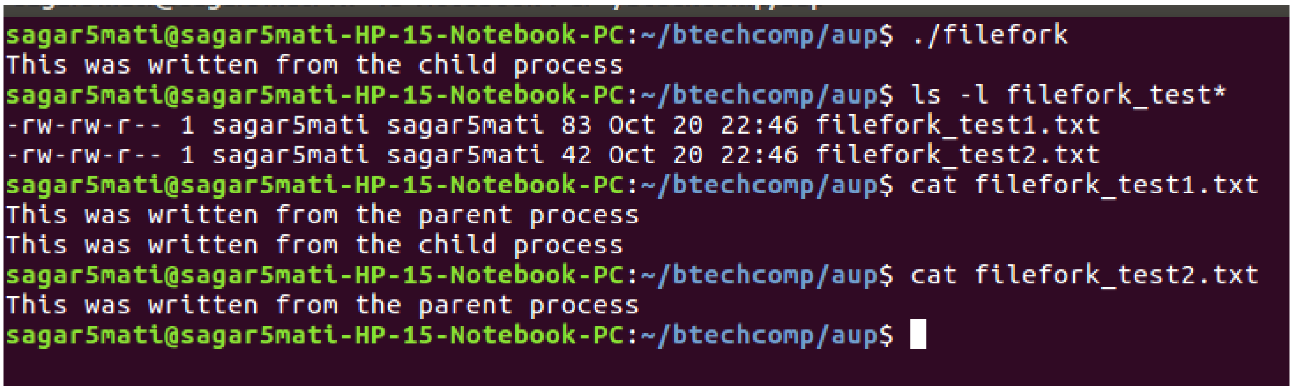
Code:

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

int main() {
        pid_t child;
        int fd1, fd2, status;
        char c[] = "This was written from the parent process\n";
        if((fd1 = open("filefork_test1.txt", O_WRONLY | O_CREAT,
S_IRUSR |
                        S_IWUSR | S_IRGRP | S_IWGRP | S_IROTH)) == -1) {
                perror("filefork.c: Error while opening file");
                return 1;
        }
        if((write(fd1, c, sizeof(c))) == -1) {
                perror("filefork.c: Error while writing to file");
                return 1;
        }
        if((child = fork()) < 0) {
                perror("filefork.c: Error while forking");
                return 1;
        }
        if(child == 0) {
                char d[] = "This was written from the child process\n";
                sleep(3);
                if((write(fd1, d, sizeof(d))) == -1) {
                        perror("filefork.c: Error while writing from child
process");
                        return 1;
                }
                if((write(fd2, d, sizeof(d))) == -1) {
```

```
                perror("filefork.c: Error while writing from child
process");
                return 1;
            }
            return 0;
        }
        else {
            if((fd2 = open("filefork_test2.txt", O_WRONLY | O_CREAT,
S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP | S_IROTH)) == -1) {
                perror("filefork.c: Error while opening file");
                return 1;
            }
            if((write(fd2, c, sizeof(c))) == -1) {
                perror("filefork.c: Error while writing to file");
                return 1;
            }
            wait(&status);
            close(fd1);
            close(fd2);
            if(status > 0) {
                printf("filefork.c: Some error occured. Please refer logs
above\n");
                return 1;
            }
            else {
                return 0;
            }
        }
        return 0;
}
```

Execution:



```
sagar5mati@sagar5mati-HP-15-Notebook-PC:~/btechcomp/aup$ ./filefork
This was written from the child process
sagar5mati@sagar5mati-HP-15-Notebook-PC:~/btechcomp/aup$ ls -l filefork_test*
-rw-rw-r-- 1 sagar5mati sagar5mati 83 Oct 20 22:46 filefork_test1.txt
-rw-rw-r-- 1 sagar5mati sagar5mati 42 Oct 20 22:46 filefork_test2.txt
sagar5mati@sagar5mati-HP-15-Notebook-PC:~/btechcomp/aup$ cat filefork_test1.txt
This was written from the parent process
This was written from the child process
sagar5mati@sagar5mati-HP-15-Notebook-PC:~/btechcomp/aup$ cat filefork_test2.txt
This was written from the parent process
sagar5mati@sagar5mati-HP-15-Notebook-PC:~/btechcomp/aup$ 
```

Conclusion:

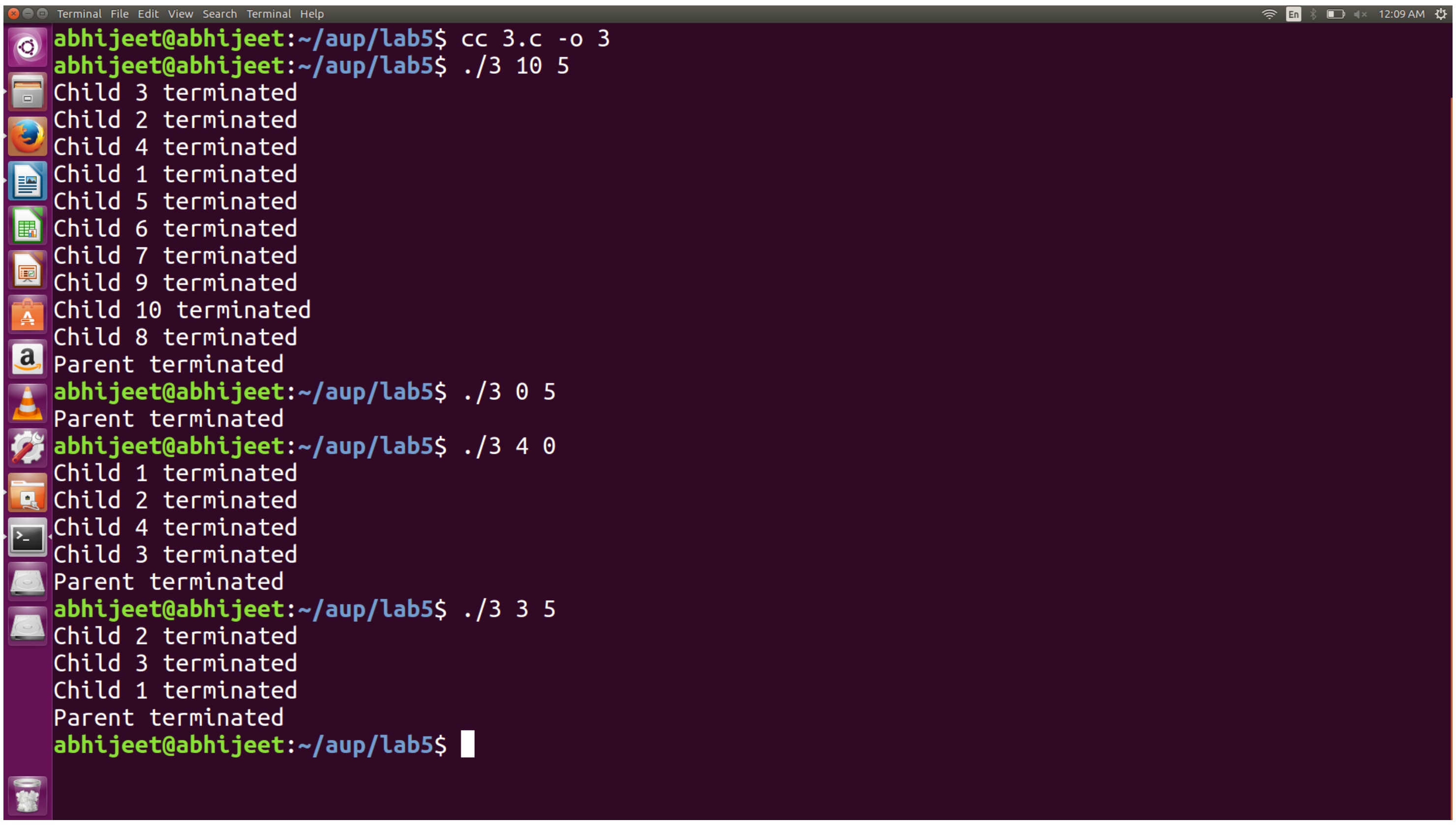
A child process is passed all the open file descriptors from its parent. Therefore any files opened by the parent process before calling fork can be accessed by the child as well. This is evident from the file filefork_test1.txt.
After the parent has called fork if it opens any file then this data is not shared by the child so it cannot these files. This can be seen by the behaviour of the filefork_test2.txt file and stdout.

Question 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 <ctype.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <limits.h>
int charToInt(char *str) {
	int result = 0, count = 0;
```

```c
	while(str[count]) {
		if(isdigit(str[count])) {
			result = result * 10 + (str[count] - '0');
		}
		else {
			return INT_MIN;
		}
		count++;
	}
	return result;
}
int main(int argc, char *argv[]) {
	if(argc < 3) {
		printf("Arguments not valid\n");
		return 1;
	}
	int children_, sleep_, count, pid;
	children_ = charToInt(argv[1]);
	sleep_ = charToInt(argv[2]);
	if(children_ == INT_MIN || sleep_ == INT_MIN) {
		printf("Incorrect arguments\n");
		return 1;
	}
	for(count = 0; count < children_; count++) {
		pid = fork();
		if(!pid) {
			break;
		}
	}
	if(!pid) {
		sleep(sleep_);
		printf("Child %d terminated \n", count + 1);
	}
	else {
		while(wait(NULL) > 0);
		printf("Parent terminated\n");
	}
	return 0;
}
```

Execution:



```
abhijeet@abhijeet:~/aup/lab5$ cc 3.c -o 3
abhijeet@abhijeet:~/aup/lab5$ ./3 10 5
Child 3 terminated
Child 2 terminated
Child 4 terminated
Child 1 terminated
Child 5 terminated
Child 6 terminated
Child 7 terminated
Child 9 terminated
Child 10 terminated
Child 8 terminated
Parent terminated
abhijeet@abhijeet:~/aup/lab5$ ./3 0 5
Parent terminated
abhijeet@abhijeet:~/aup/lab5$ ./3 4 0
Child 1 terminated
Child 2 terminated
Child 4 terminated
Child 3 terminated
Parent terminated
abhijeet@abhijeet:~/aup/lab5$ ./3 3 5
Child 2 terminated
Child 3 terminated
Child 1 terminated
Parent terminated
abhijeet@abhijeet:~/aup/lab5$ 
```

Number of child processes to be forked is sent in as the first argument to the program and the time to sleep for the processes as the second argument. As you can see the parent waits in a while loop until all the processes are over.