**Lab06 Documentation:**

**Lab Assignment:**

**Assessed questions:  
1. Demonstrate C programming Exercise 4.**

**2. Complete Unix Programming Exercise 2.**

**3. Complete Unix Programming Exercise 6.**

**4. Complete Unix Programming Exercise 7.**

**Unix programming exercise:**

1. **How many child processes are created?**

Try to work out the answer of this question mentally: what would happen if you call fork twice in the following way:

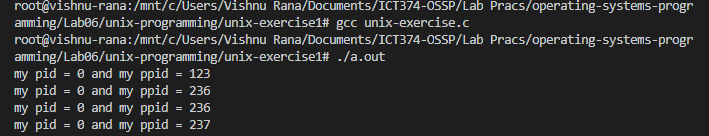
fork();

fork();

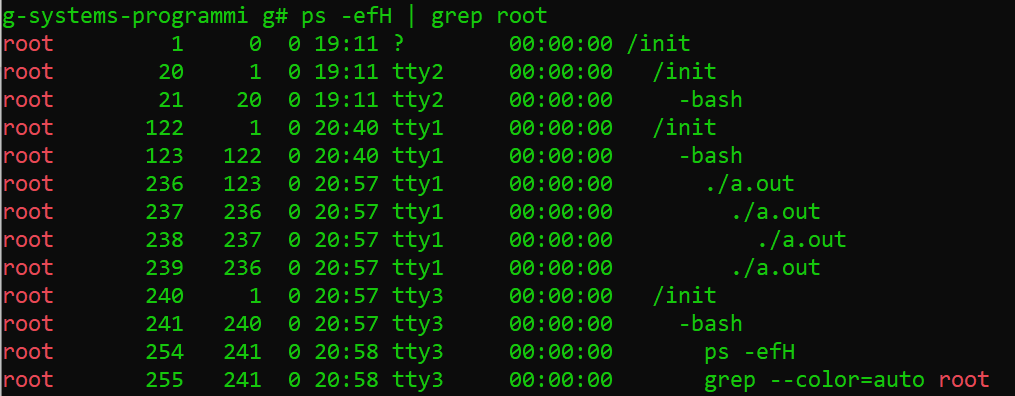
How many child processes would be created? What are the relationships between these processes? Complete the following program:

While running the above program in one terminal, display the processes in another terminal. What ps options would you need to show these processes and their parent-child relationship?

**Program output:**



**Another terminal output:**



1. **Creating multiple direct child processes**

Write a program that creates *n* direct child processes. For *i*th child, it prints "Child i" and its pid and ppid numbers. It then sleeps for *i* minutes before termination.

In creating the child processes you must make sure that all children are the direct children of the original process. Use command ps to verify that this is indeed the case.

Hint: consider the following program structure:

for (i=1; i<=n; ++i) {

pid = fork();

if (pid==0) { // child

print Child i: my pid and my ppid

sleep (i\*60); // sleep for i minutes

exit(0);

}

}

Make sure you genuinely understand how the above code works.

Test your program with different *n* values (e.g., 3, 4, 5) and verify the parent-child relationship with the program outputs as well as the output of the ps command.

**Source code listing:**

#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

int getNumber()

{

int number;

printf("Please enter the number of children you want?\n");

scanf("%d%\*c", &number);

return number;

}

int main(){

pid\_t pid;

int n = getNumber();

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

{

pid = fork();

if(pid == 0)

{

printf("child no. %d, pid = %d & ppid = %d\n",i, pid, getppid());

sleep(i \* 60);

exit(0);

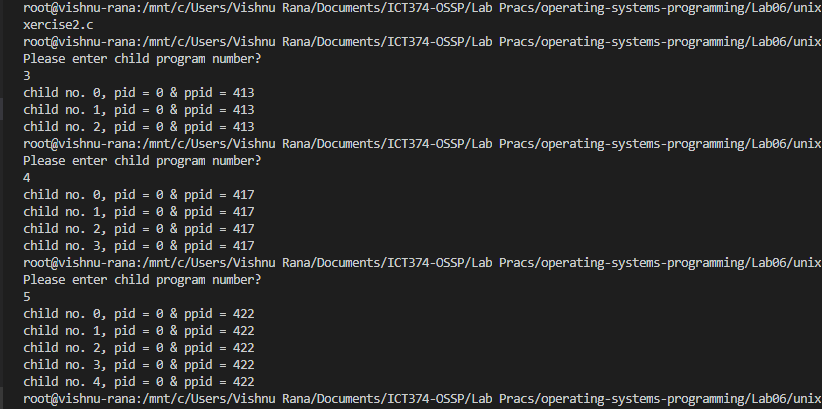
}

}

return 0;

}

**Output Screenshot:**



**Question 6.) Creating child process to execute a command**

Write a program that keeps reading one string at a time from the keyboard. Each time a string is read, it is treated as a command name, and is executed in a child process. The program stops once the user types "bye". Test your program by typing the following strings: ls, ps, date, bye.

You may adopt the following program structure:

**char command[COMMAND\_LINE\_SIZE];**

**pid\_t pid;**

**while (1) {**

**display a prompt such as "%" or "$"**

**read a line from user and store it in array command**

**if command is "bye"**

**break;**

**if ((pid = fork()) < 0) {**

**perror("fork");**

**exit(1);**

**}**

**if (pid==0) { // child**

**load and execute command**

**}**

**wait for the child to finish**

**}**

**Source code listing:**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/wait.h>

#include <string.h>

int main(int argc, char \*argv[]){

if(argc > 1)

{

pid\_t pid[argc];

char \*commandName;

int status;

for(int i = 1 ; i < argc; i++){

pid[i-1] = fork();

if(pid[i-1] < 0)

{

perror("fork unsuccessful ");

exit(1);

}

else

{

if(pid[i-1] == 0)

{

commandName = strchr(argv[i], '/');

if(strcmp(commandName,"bye") == 0)

{

break;

}

else

{

execl(argv[i], commandName, (char\*)0);

perror("execl failed...\n");

exit(1);

}

}

}

printf("%s \n", argv[i]);

if(pid[i-1] > 0)

{

wait((int\*)0);

}

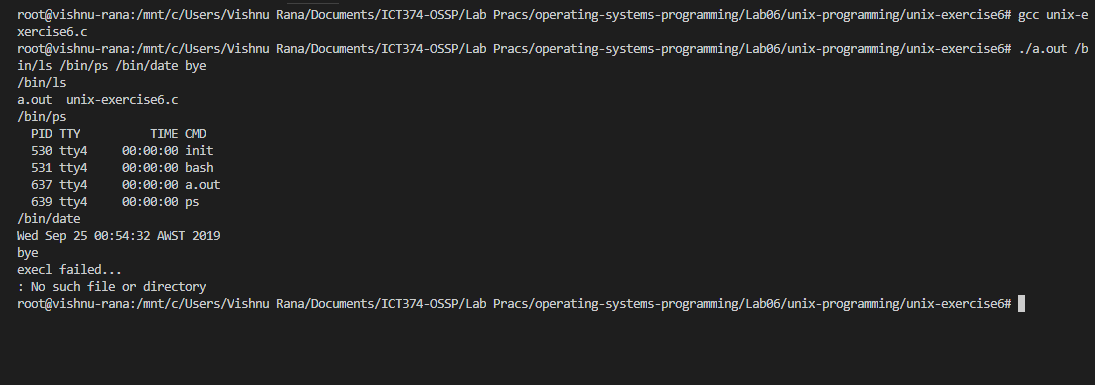
}

}

return 0;

}

**Output screen capture:**



**7. Creating a daemon process**

Write a program that turns itself (actually its child) into a daemon. The daemon process simply writes one line of log "I have nothing to do" to a log file once every 10 seconds. Your program should print the daemon's pid at the beginning.

Check that your daemon process is running using ps command (what options are needed?). Compare the pid from the ps command with that from your program. Check whether your daemon has a control terminal.

Make sure you kill the daemon process using kill command at the end of this exercise.

Hints: read Section 10 of Topic 4. You may use daemon\_init function from that section.

Source code Listing:

#include <stdio.h>

#include <stdlib.h>

#include <sys/types.h>

#include <unistd.h>

int daemon\_init(void)

{

pid\_t pid;

if ( (pid = fork() ) < 0)

{

return (-1);

}

else if (pid != 0)

{

printf("server pid=%d\n", pid);

exit(0);

}

/\* child continues \*/

setsid(); /\* become session leader \*/

chdir("/"); /\* change current directory \*/

umask(0); /\* clear umask \*/

return (0);

}

int main()

{

FILE \*log;

pid\_t pid;

// create a log file

log = fopen(flockfile, "w+");

if (!log)

{

fprintf(stderr, "cannot create log file %s\n", flockfile);

exit(1);

}

// turn the process into a daemon

daemon\_init();

// log daemon pid

pid = getpid();

fprintf(log, "My pid is %d\n", pid);

fflush(log);

while (1)

{

sleep(10);

fprintf(log, "I have nothing to do\n");

fflush(log);

}

}