Advanced Unix Programming Lab 8

111403050   Abhijeet Kale
111403067   Sagar Panchmatia
111403076   Nikunj Singh

Q1)
Create a new system call wait2, which extends the wait system call.

int wait2(int *wtime, int *rtime, int *iotime);

Where the three arguments are pointers to integers to which the wait2 function will assign:

a. The aggregated number of clock ticks during which the process waited (was able to run but did not get CPU)

b. The aggregated number of clock ticks during which the process was running

c. The aggregated number of clock ticks during which the process was waiting for I/O (was not able to run).

The wait2 function shall return the pid of the child process caught or -1 upon failure

Code :

```
#include<stdio.h>

#include<unistd.h>

#include <sys/acct.h>

#include<stdlib.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <fcntl.h>


pid_t wait2(int *wtime, int *rtime, int *iotime);

unsigned int long compt2ulong(comp_t comptime);

struct acct acdata;


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

        int i, wtime, rtime, iotime;

        pid_t pid;

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

                printf("...Fork failed...\n");
```

```c
                return -1;
        }
        else if(pid > 0){          // parent process
                printf("Pid of caught process is %d\n", pid);
                pid = wait2(&wtime, &rtime, &iotime);
        }
        else {                      // some computation in child process
                i = 10;
                while(i > -1000) {
                        i = i - 1;
                }
                exit(1);
        }
        return 0;
}


pid_t wait2(int *wtime, int *rtime, int *iotime) {
        int fd, val, ret, pid;
        char *pathname = "/home/nikunj/Desktop/AUP-ASSGN/LAB-8/info.txt";
        fd = open(pathname, O_RDWR | O_APPEND, 0777);
        acct(pathname);                 // process accounting turned on
        pid = wait(NULL);       // waiting for child process
        acct(NULL);             // process accounting turned off
        ret = read(fd, &acdata, sizeof(acdata));
        if(ret == -1) {
                printf("...Read failed ...\n");
                return -1;
        }
        val = compt2ulong(acdata.ac_utime) + compt2ulong(acdata.ac_stime);
        *rtime = val;
        val = compt2ulong(acdata.ac_etime - acdata.ac_utime - acdata.ac_stime);
```

```
                *wtime = val;

                // calculation of iotime


                printf("rtime in clockticks is %d\nwtime in clockticks is %d\n", pid, rtime, wtime);

                return pid;

        }


unsigned int long compt2ulong(comp_t comptime) {

        int val, exp;

        val = comptime & 0x1fff;

        exp = (comptime >> 13) & 7;

        while (exp-- > 0) {

                val *= 8;

        }

        return val;

}
```

Execution :

Here wait2 function extends the wait system call. The concept of process accounting is used. The struct acct is used to get the process accounting information. "info.txt" file is used to write the accounting info using acct(pathname); function call. Process accounting is turned on just before a call to wait() system call. Process accounting is turned off with NULL as an argument to acct function. It's turned off just immediately after call to wait().

The fields ac_utime, ac_etime, ac_stime are used to fetch the desired values.

Q2) "Call fork. Let the child create a new session. Verify that the child becomes the process group leader and it does not have a controlling terminal

Code:

```c
#include <stdio.h>

#include <unistd.h>

#include <errno.h>

#include <sys/types.h>

#include <sys/wait.h>


int main() {

        pid_t child;

        int status;

        if((child = fork()) < 0) {

                perror("Error while fork");

                return -1;

        }


        if(child == 0) {

                pid_t processid;

                char buf[L_ctermid];

                processid = getpid();

                printf("Process ID: %u\nProcess Group ID: %u\nSession ID: %u\nControlling
Terminal: %s\n", getpid(), getpgid(processid), getsid(processid), ctermid(buf));

                if((setsid()) == -1) {

                        perror("Error while creating session");

                        return -1;

                }

                printf("Process ID: %u\nProcess Group ID: %u\nSesssion ID: %u\nControlling
Terminal: %s\n", getpid(), getpgid(processid), getsid(processid), ctermid(buf));
```

```
                        _exit(0);

                }

        else {

                        wait(&status);

                        if(WIFEXITED(status)) {

                                return 0;

                }

                        else {

                                printf("Some error occured with child process");

                                return -1;

                }

        }


                return 0;

}
```
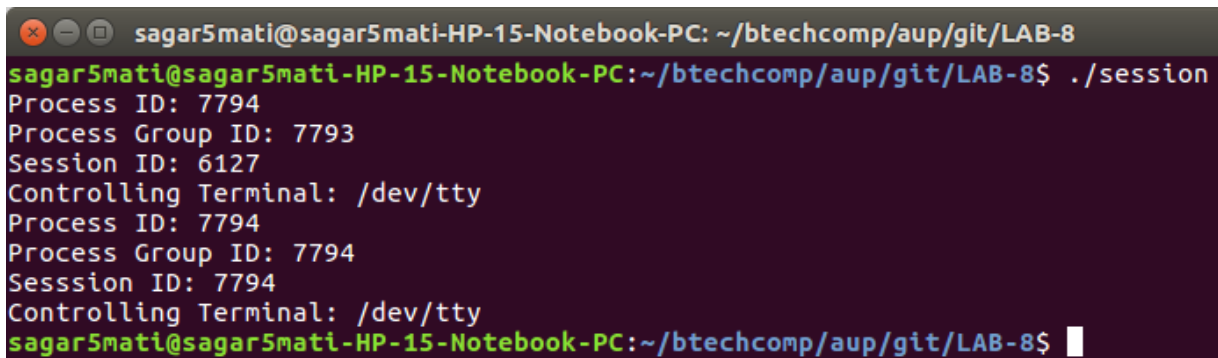
Execution:

Explanation:

It can be seen from the output that after creating the session child becomes the new process group leader and has a new session id.

Q3)

Write a program to verify that a parent process can change the process group ID of one of its children before the child performs an exec(), but not afterward.

Code:

```c
#include <stdio.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <limits.h>
int main(int argc, char *argv[]) {
        int pid = fork();
        if(pid == 0) {
                printf("Child without exec....\n");
                printf("Before pgid of child = %u\n", getpgid(0));
                sleep(5);
                printf("After pgid of child = %u\n", getpgid(0));
        }
        else {
                sleep(2);
                setpgid(pid, getppid());
                wait(NULL);
                pid = fork();
                if(pid == 0) {
                        printf("Child with exec of ls......\n");
                        execv("/bin/ls", argv);
                }
                else {
                        sleep(5);
                        int result = setpgid(pid, getppid());
                        if(result == -1) {
                                perror("Could not change GPID: ");
                        }
                }
        }
}
```
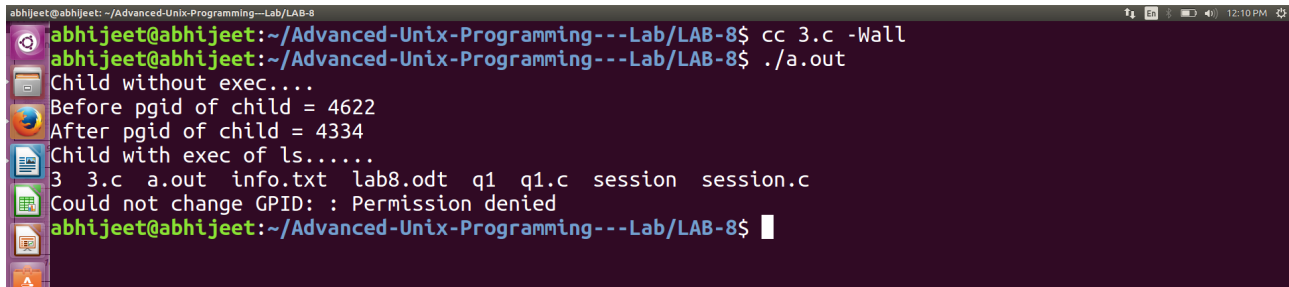
```
        return 0;

}
```

Execution:

The above program first makes a fork and then this child prints the first two lines in the output. The second line of the output is the pgid of the child before it's parent tries to chage it's pgid. Then this child sleeps for 5 seconds, during which time the parent tries and change the pgid of this child process. After the sleep function returns in child, it prints the new pgid set by it's parent. This verifies the first part that a parent can change the pgid of it's child process if the child process has not called exec.

After the first part, the parent again calls fork. This time the child process calls exec for ls. The parent sleeps for 5 seconds just to give enough time for the exec function to execute, after which it tries to change the pgid of the child process. This time -1 is returned by the function stating an error with the message of Permission denied. So the pgid of the child process can not be changed by it's parent after it has called exec.