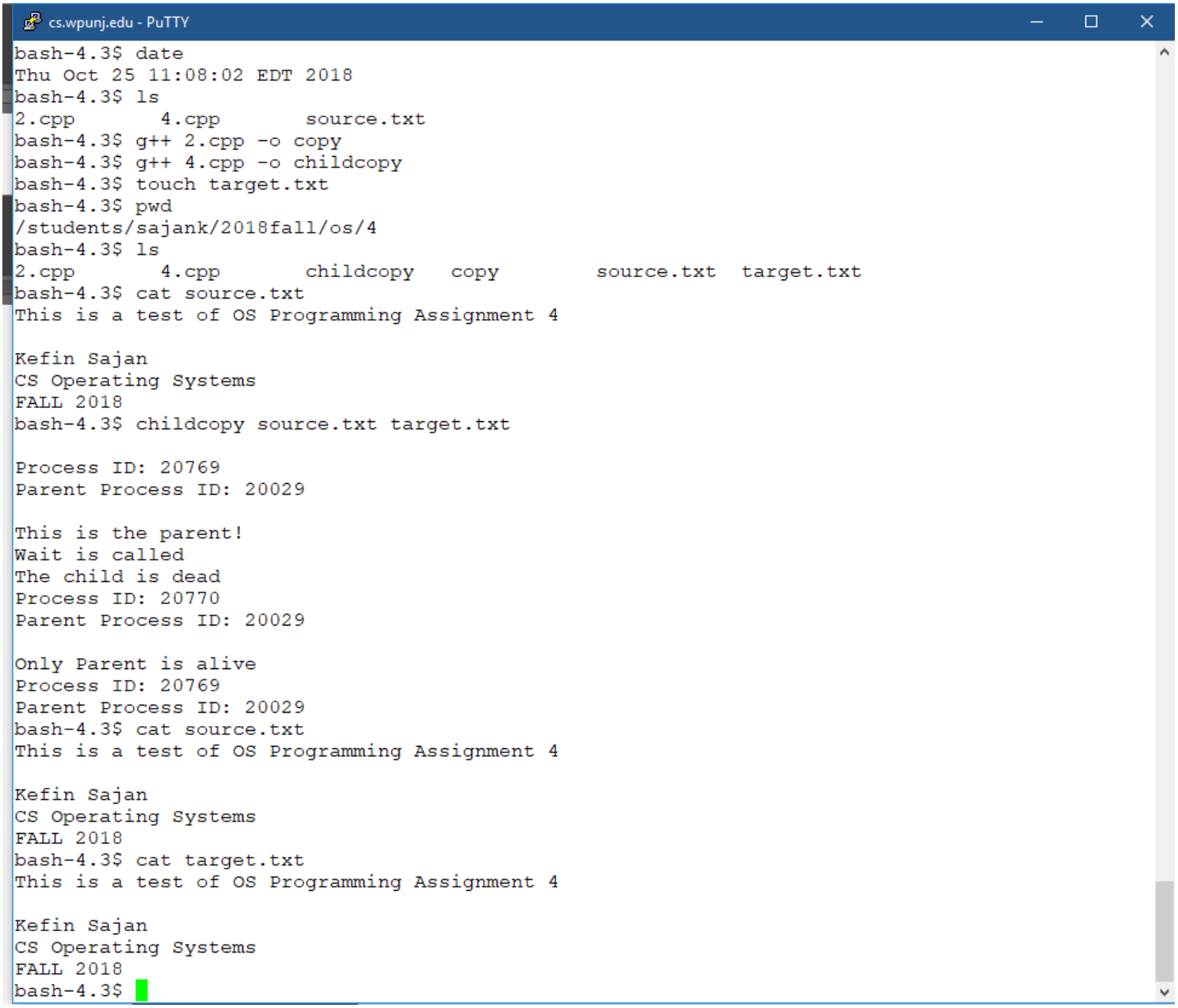
```
//          Course: CS3820-01 Operating System
//            Name: Sajan, Kefin
//      Assignment: Programming Assignment 4
//   Date assigned: 10/11/17
//        Date due: 10/25/18
//  Date handed in: 10/25/18
//          Remark: This program is showcase the Linux system calls fork(), exec() and wait().
//                  The program calls the fork() function and creates a child process.
//                  The Child inherts the program code and calls the exec() function.
//                  This child process is dead when the parent calls the wait() function.

////////////////////////////////////////////////////////////////////////////
//4.cpp
//Source code for “childcopy”
////////////////////////////////////////////////////////////////////////////

#include <iostream>
#include <unistd.h>
#include <errno.h>

#define DEFCHLDRTN 50

using namespace std;

int main(int argc, char *argv[])
{
	int pid;
	int ppid;
	int foutput;
	int error = 0;

	//Show all Process ID
	pid = getpid();
	ppid = getppid();
	cout << "\nProcess ID: " << pid;// OK
	cout << "\nParent Process ID: " << ppid << endl;// OK

	foutput = fork();
	//Fork called, process duplicated

	if (foutput != 0)  // foutput is non-zero, this is the parent
	{
		cout << "\nThis is the parent!";
		pid = wait(NULL);// wait() called
		cout << "\nWait is called";
		cout << "\nThe child is dead";


		ppid = getppid();
		cout << "\nProcess ID: " << pid;
		cout << "\nParent Process ID: " << ppid << endl;
	}

	else

	{
		cout << "*** The child is alive, time to copy contents of file" << endl;
		// foutput is zero, this is the child
		printf("&&& The child is alive, time to copy contents of file\n");

		char* argv[] = { "copy", "source.txt", "target.txt", NULL };
		error = execv("copy", argv);//exec call is called by the child
		if (error == -1) //If error is found
		{
```

```
                cout << "\nThis program needs program 'copy' to run  ";
                cout << "\nerror = " << errno << endl << endl;
                return -1;
        }
        if (error != -1)
                printf("\nCopy function called!");

        pid = getpid();
        ppid = getppid();
        printf("\nProcess ID: ", pid);
        printf("\nParent Process ID: ", pid);
        printf("\n\n");

        return DEFCHLDRTN;
    }

    cout << "\nOnly Parent is alive";//Show all Process ID
    pid = getpid();
    ppid = getppid();
    cout << "\nProcess ID: " << pid;
    cout << "\nParent Process ID: " << ppid << endl;

    return 0;

}

////////////////////////////////////////////////////////////////////////////////
//2.cpp
//Source code for “copy”
////////////////////////////////////////////////////////////////////////////////

//          Remark: This is a system program that copies the contents of an existing ASCII
//                  text file (name it source.txt) to a newly created empty file
//                  (name it target.txt). The program is compiled and linked into
//                  an executable file name copy. So when the user executes the command
//                  copy source.txt target.txt at command line, the contents of the
//                  source file are copied to the target file.

#include <iostream>
#include <fcntl.h>
//fcntl is C header used for file management
//      Ex. opening, closing, changing permissions, etc.

void copy(int, int);
// The file descriptors are passed into the function

char buffer[2048];
//     This Buffer is used to limit the amount of character stored inside the program
//     2048 stands for 2 kilobytes of storage

int main(int argc, char *argv[])
{
    printf("\nCopy function called!\n");

    int fd_source, fd_target;
    //     A file descriptor is an integer value returned by the open() call

    if (argc != 3) {
            // To make sure program has two file to
            printf("Need two arguments!\n");
            return 1;
    }
    fd_source = open(argv[1], O_RDONLY);
    //     "argv[1]" is the file name
```

```
	//	O_RDONLY means that the file contents is to be read only
	//	 and is not allowed to be modified.

	if (fd_source == -1) {
		// make sure "open" call is successful
		printf("Cannot open %s file!", argv[1]);
		return 1;
	}

	fd_target = creat(argv[2], 0666);
	//	"0666" is specific permission that is set to the file
	//	 File is only read and write for user, group and other

	if (fd_target == -1) {
		// make sure "create" call is successful
		printf("Cannot create %s file!", argv[2]);
		// What is argv[2]?
		//  The second file name passed into the program
		return 1;
	}

	copy(fd_source, fd_target);
	//Copy function

	int pid;
	int ppid;

	//Show Process ID
	pid = getpid();
	ppid = getppid();
	printf("\nProcess ID: " << pid;
	printf("\nParent Process ID: " << ppid << endl;

	printf("\nProcess ID: " << pid << endl;

	return 0;
}
void copy(int source, int target)
{
	int count;

	while ((count = read(source, buffer, sizeof(buffer))) > 0)
		write(target, buffer, count);

}
```

```
////////////////////////////////////////////////////////////////////////////
//Output
////////////////////////////////////////////////////////////////////////////
```


cs.wpunj.edu - PuTTY
bash-4.3$ date
Thu Oct 25 11:08:02 EDT 2018
bash-4.3$ ls
2.cpp 4.cpp source.txt
bash-4.3$ g++ 2.cpp -o copy
bash-4.3$ g++ 4.cpp -o childcopy
bash-4.3$ touch target.txt
bash-4.3$ pwd
/students/sajank/2018fall/os/4
bash-4.3$ ls
2.cpp 4.cpp childcopy copy source.txt target.txt
bash-4.3$ cat source.txt
This is a test of OS Programming Assignment 4
Kefin Sajan
CS Operating Systems
FALL 2018
bash-4.3$ childcopy source.txt target.txt
Process ID: 20769
Parent Process ID: 20029
This is the parent!
Wait is called
The child is dead
Process ID: 20770
Parent Process ID: 20029
Only Parent is alive
Process ID: 20769
Parent Process ID: 20029
bash-4.3$ cat source.txt
This is a test of OS Programming Assignment 4
Kefin Sajan
CS Operating Systems
FALL 2018
bash-4.3$ cat target.txt
This is a test of OS Programming Assignment 4
Kefin Sajan
CS Operating Systems
FALL 2018
bash-4.3$