National University of Computer and Emerging Sciences

Laboratory Manual

for

Operating Systems Lab

(CL-220)

Course Instructor	Ms. Namra Absar
Lab Instructor(s)	Ms. Saba Tariq
Section	CS-4A
Semester	Spring 2024

Department of Computer Science

FAST-NU, Lahore, Pakistan

Objectives

In this lab, students will:

1. practice Basic commands on terminal
2. develop a small program in C for reading/writing files
3. create a process using Fork() system call

Basic Commands

• Clear the console: **clear**
• Changing working Directory: **cd Desktop**
cd Home
• List all files in directory: **ls**
• Copy all files of a directory within the current work directory: **cp dir/***
• Copy a directory within the current work directory: **cp -a tmp/dir1**
• Look what these commands do
cp -a dir1 dir2
cp filename1 filename2

Compiling C and C++ Programs on the Terminal:

For C++:
Command: g++ source_files… -o output_file

For C:
Command: gcc source_files… -o outputfiles

Example:
g++ main.cpp lib.cpp -output

1. Process Creation:

The processes in most systems can execute concurrently, and they may be created and deleted dynamically. Thus, these systems must provide a mechanism for process creation and termination.

I. fork()

- Has a return value
- Parent process => invokes fork() system call
- Continue execution from the next line after fork()
- Has its own copy of any data
- Return value is > 0 //it's the process id of the child process. This value is different from the Parents own process id.
- Child process => process created by fork() system call
- Duplicate/Copy of the parent process //LINUX
- Separate address space

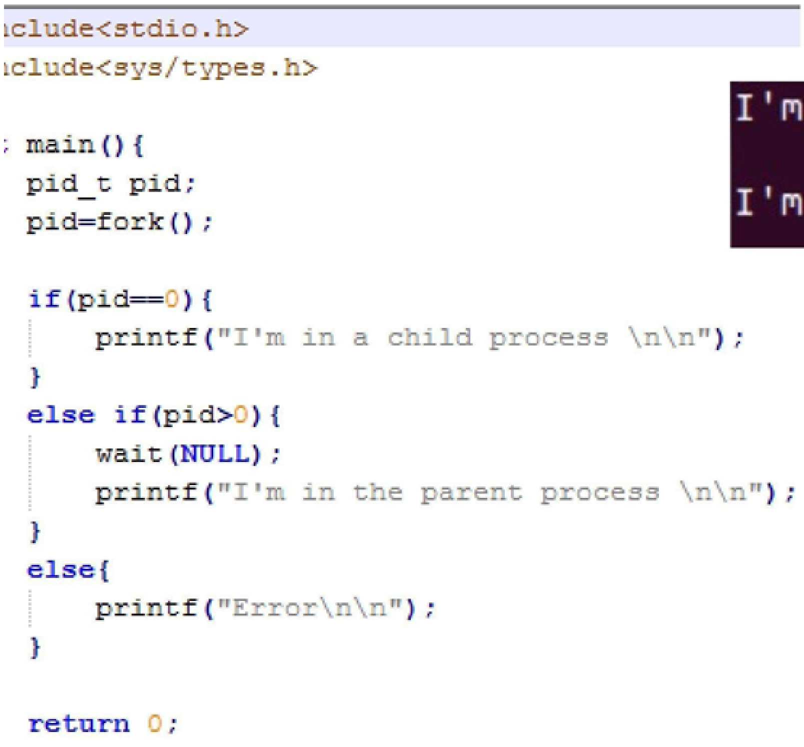
- Same code segments as parent process
- Execute independently of parent process
- Continue execution from the next line right after fork()
- Has its own copy of any data
- Return value is 0

II. wait ()

- Used by the parent process
- Parent's execution is suspended
- Child remains its execution
- On termination of child, returns an exit status to the OS
- Exit status is then returned to the waiting parent process //retrieved by wait ()
- Parent process resumes execution
- #include <sys/wait.h>
- #include <sys/types.h>

III. exit()

- Process terminates its execution by calling the exit() system call
- It returns exit status, which is retrieved by the parent process using wait() command
- EXIT_SUCCESS // integer value = 0
- EXIT_FAILURE // integer value = 1
-
- OS reclaims resources allocated by the terminated process (dead process) Typically performs clean-up operations within the process space before returning control back to the OS
- _exit()
- Terminates the current process without any extra program clean-up
- Usually used by the child process to prevent from erroneously release of resources belonging to the parent process

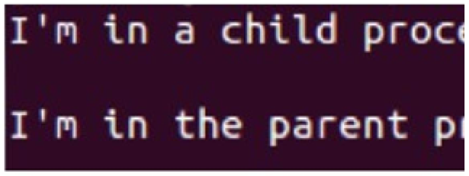


```
clude<stdio.h>
clude<sys/types.h>

 main(){
  pid_t pid;
  pid=fork();

  if(pid==0){
      printf("I'm in a child process \n\n");
  }
  else if(pid>0){
      wait(NULL);
      printf("I'm in the parent process \n\n");
  }
  else{
      printf("Error\n\n");
  }

  return 0;
```



```
I'm in a child proc

I'm in the parent p
```

In Lab Tasks

Question 1:
See the usage of the following commands online. Also run them on the terminal.

1. pwd
2. ls
3. cd
4. cp
5. mkdir & rmdir
6. man
7. sudo
8. apt-get

Question 2:

a. Create a function removeNonAlphabets(char * inputFileName, char * outputFileName) in C or C++ that is passed as parameters: an input file name and an output file name. The function then reads the input file using read system call and removes all non-alphabets. It then writes the data to output file using write system call. You will need to see open, read, write, and close system calls. https://www.geeksforgeeks.org/input-output-system-calls-c-create-open-close-read-write/

Question 3: Create a program which initiate 4 process and each of them prints their process id. Draw a process tree of all initiated programs.