

CECS 326-01

Operating Systems

Dylan Dang (ID 026158052)

Assignment 1

Due Date: 09/22/2022

Submission Date: 09/22/2022

Program Description

The purpose of this assignment is to incorporate the `fork()`, `exec()`, and `wait()` system calls, within our two C++ programs named *parent.cc* and *child.cc* whilst using command-line arguments on Linux. `Fork()` allows a new process to be created and consists of a copy of the address space of the original process. This mechanism allows the parent process to communicate easily with its child process. `Exec()` system calls replaces the process's memory space with a new program. The `exec()` system call loads a binary file into memory and starts its execution. `Wait()` moves the parent process itself off the ready queue until the termination of the child.

The *parent.cc* program is designed to take a list of gender-name pairs from the command-line arguments on Linux, create as many child processes as there are in the gender-name pairs and passes to each child process a child number and gender-name pair, wait for all child processes to terminate, and finally output "All child processes terminated. Parent exits." And terminates. To keep track of when *child.cc* program is finished with its task, we give the child a process id (PID).

The *child.cc* program is designed to receive a child number and one gender-name pair arguments from parent, output "Child # x, I am a boy (or girl), and my name is xxxxx". The output data from *child.cc* is dependent on the data received from *parent.cc*. After that the processes are terminated.

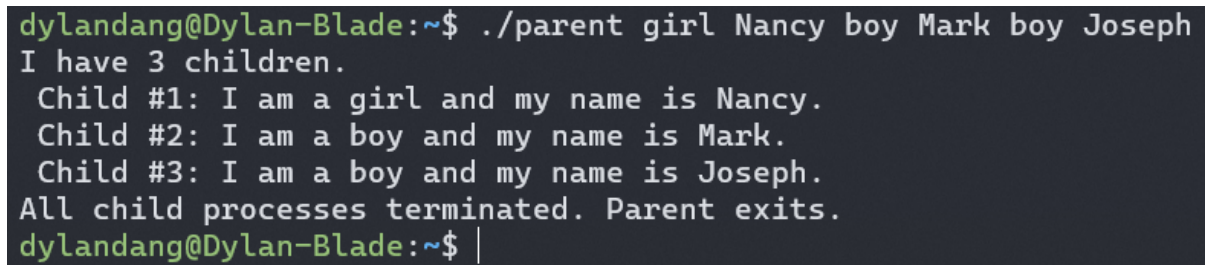
Together, the *parent.cc* and *child.cc* programs are to communicate with each other in asking and providing children data such as the child number and gender-name pairs. The *parent.cc* programs receive data from the user inputted command-line arguments, then passes on that data to the *child.cc* programs which then outputs the data into a nice and clean formatted statement.

parent.cc & child.cc compilation screenshot:

A screenshot of a terminal window with two tabs. The active tab is titled 'dylandang@Dylan-Blade: ~'. The terminal shows the following commands and output:

```
dylandang@Dylan-Blade:~$ g++ parent.cc -o parent
dylandang@Dylan-Blade:~$ g++ child.cc -o child
dylandang@Dylan-Blade:~$ ls
child  child.cc  parent  parent.cc
dylandang@Dylan-Blade:~$
```

running parent executable:

A screenshot of a terminal window showing the execution of the parent executable. The terminal output is as follows:

```
dylandang@Dylan-Blade:~$ ./parent girl Nancy boy Mark boy Joseph
I have 3 children.
Child #1: I am a girl and my name is Nancy.
Child #2: I am a boy and my name is Mark.
Child #3: I am a boy and my name is Joseph.
All child processes terminated. Parent exits.
dylandang@Dylan-Blade:~$ |
```

parent.cc code:

```
GNU nano 4.8 parent.cc
1 #include <iostream>
2 #include <sys/wait.h>
3 #include <unistd.h>
4
5 using namespace std;
6
7 int main (int children, char* children_data[]) {
8
9     // since passed in data are in name-gender pairs, we divide by 2.
10    int numOfChildren = children/2;
11
12    // printing total number of children.
13    printf("I have %d children.", numOfChildren);
14    fflush(stdout);
15
16    // creating process id for child and child wait status
17    pid_t child_pid, childStatus_pid;
18
19    // childIndex keeps track of number of children
20    char childIndex[3];
21
22    for (int i = 0; i < numOfChildren; i++) {
23
24        // label new process copy as the child
25        child_pid = fork();
26
27        // make sure child process exists
28        if ((child_pid == 0)) {
29            sprintf(childIndex, "%d", i+1);
30
31            // prepping the data to be sent to child program.
32            char *dataForChild[] = {childIndex, children_data[i*2+1], children_data[i*2+2], NULL};
33
34            // executing the child program with the data provided.
35            execv("./child", dataForChild);
36
37            // exit child program
38            exit(0);
39        }
40    }
41
42    // waiting for all child processes to finish
43    int childStatus = 0;
44    while ((childStatus_pid = wait(&childStatus)) > 0);
45    printf ("\nAll child processes terminated. Parent exits.\n");
46
47 }
48 |
```

child.cc code:

```
GNU nano 4.8 child.cc
1 #include <iostream>
2 #include <stdio.h>
3
4 using namespace std;
5
6 int main (int children, char* children_data[]) {
7     // output children with child number, gender, and name
8     printf("\n Child #s: I am a %s and my name is %s.", children_data[0], children_data[1], children_data[2]);
9     return 0;
10 }
11
```