# Florida Polytechnic University – Computer Science Department
# Programming Assignment 1 Spring 2020 - Process API for UNIX

## Submission deadline: Jan 23rd 11:59 pm

## Deliveries:

- **Code**: a .c program following the format described below.
- **Report**: Brief report explaining how you address the problem, and screenshots of the output. Include this paragraph in your report, and sign (failing in include the paragraph and signing will result and total grade of zero in this homework.

**Requirements:** The program should use the POSIX api from unix/linux, and should run in the university UNIX system (ember).
**Note**: Use your student ID number as your program name.

_____
I certify that I coded this program by myself and this code doesn't correspond to the intellectual work of someone else.
Signature: _____ _____

Rubric

| Item | Value |
|---|---|
| Program | 75 |
| Report | 25 |
| **Total** | **100** |

### Description (Part A)

This program should use the fork() instruction to create 1 or 2 children. If the parent creates just a child, the parent will send the array to the child. The child will receive the array and it will add its elements. Then the child will send the summation to the parent who will print that summation. Now if the parent creates two children, it will send the first half of the array to one child and the second half to the another one. Each child will receive the corresponding half of the array, and it will compute it sum. Each child will send the partial sums to the parent who will collect and will add them. Finally, the parent will print the final sum. (Note: Use arrays of even length).

**Program format and expected output:** Here, some examples:

Example 1

./luis 1 2 5 3 10

Here, the parent (luis) create just a child (first number after the gram name, 1). The next the next elements in the command line correspond to elements of the array (2, 5, 3, 10).

Example 2

./luis 2 1 2 3 4 5 6

Here, the parent (luis) create two children (first number after the gram name, 2). The next the next elements in the command line correspond to elements of the array (1, 2, 3, 4, 5, 6).

Expected output

Example 1
./luis 1 2 5 3 10

Assuming parent has pid 2, and child 1 has pid: 3. We want to see the following output

>>I am parent with pid: 2 sending the array: 2 5 3 10 to child with id: 3.
>>I am child with pid: 3, adding the array 2 5 3 10 and sending partial sum 20.
>>I am the parent pid: 2 receiving partial sum 20 and printing 20.

Example 2
./luis 2 1 2 3 4 5 6

Assuming parent has pid 2, child 1 has pid: 3, and child 2 has pid: 4.

>>I am parent with pid: 2 sending the array: 1 2 3 to child with pid 3, and 4 5 6 to child with pid 4.
>>I am child with pid: 3, adding the array 1 2 3 and sending partial sum 6.
>>I am the parent pid: 4, adding the array 4 5 6 and sending partial sum 15.
>>I am the parent pid: 2 receiving partial sum 6 and 15 and printing 20.

**Description (Part B) Just for enthusiastic (10 points extra – transferable to any assignment)**

Here, the first parameter after the program's name (number of children) could be any number between 1 and 10. And the number of elements of the array should be a multiple of the of number of children. Thus, if for example the number of children is 3, then the number of elements is multiple of three (3, 6, 9, …), and parent will send an array with (array length /number of children) elements to each child.

Example

./luis 3 1 2 3 4 5 6 7 8 9

**Resources.**

Here, a link about parsing the command line arguments in C.
https://www.cs.swarthmore.edu/~newhall/unixhelp/C_commandlineargs.php

Here, a good resource about inter process communication.

## Inter Process Communication (IPC)

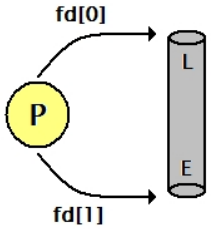
There are several mechanisms for IPC those include: Sharing Memory models, Message Passing model, Sockets, and pipes among others. For this programming assignment we will use pipes.

Tutorial for pipes 1: https://users.cs.cf.ac.uk/Dave.Marshall/C/node23.html
Tutorial for using pipes 2: http://tldp.org/LDP/lpg/node11.html

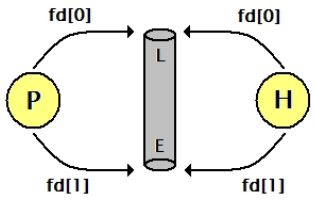
## Basic ideas about pipes.

1. When the program is executed a process is created (parent process)


2. The parent process creates a pipe ( fd) instruction where fd is an array of two descriptors, fd [0] points to the read end of the pipe and fd [1] to the write end of the pipe).

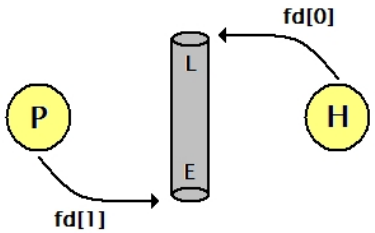



3. Then the parent must create the child process (fork instruction). When the child is created this "inherits" the same descriptors of the parent.

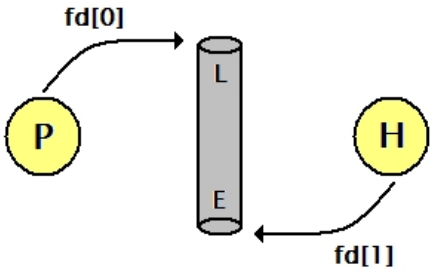



4. This way you can begin to establish communication between both processes.

5. If the PARENT wants to send data to the CHILD ti must close his descriptor fd [0] and the CHILD close its descriptor fd [1] .




6. If the CHILD wants to send data to the PARENT it must close its descriptor fd [0] and the PARENT close its descriptor fd[1] .




**CODE EXAMPLE USING: ONE-DIRECCTIONAL AND BIDIRECTIONAL COMMUNICATION** (Copy paste this link in your browser)

https://translate.google.com/translate?sl=auto&tl=en&js=y&prev=_t&hl=en&ie=UTF-8&u=https%3A%2F%2Fwww.programacion.com.py%2Fescritorio%2Fc%2Fpipes-en-c-linux&edit-text=&act=url