Florida Polytechnic University – Computer Science Department
Programming Assignment 1 - Process API

<u>**Submission deadline**</u>**: September 17th 11:59 pm**

Deliveries:

Code: a .c program following the instruction of the section:Format.
Report: Brief report explaining how you address the problem and screenshots of the results. Include the paragraph in your report, and sign (failing in include the paragraph and signing will result and total grade of zero in this homework.

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

Format: A .c program will read the number of children and number as command-line argument as show in Example 1, and will output the result as in Example 2. The program should work for 1, 2, and 3 children.

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

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

Note: Use your student ID number as your program name.

Example:

./luis 3 12 Here, the first argument 3 is the number of children or helpers the parent will create. The parent finds and prints the list of the divisors of the *second argument*, and place these numbers in an array. In this case, the divisors of 12 are 1, 2, 3, 4, 6, and 12.

Next the parent send the array to the children, and each child computes a partial sum, and prints it. Then each child send the partial sum to the parent who finally computes and prints the result.

Example 1: possible program execution.

```
./luis 1 12
```

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

```
>>I am the parent with pid: 2 sending the array: 1, 2, 3, 4, 6, 12.
>>I am the child with pid: 3, adding the array 1, 2, 3, 4, 6, 12 and sending partial sum 28.
>>I am the parent with pid 2 receiving from child with pid 3 thepartial sum 28, the total sum is 28
```

Example 2: possible program execution.

```
./luis 2 12
```

Assuming parent has pid 2, child 1 has pid: 3, child 2 has pid: 4. We want to see the following output

```
>>I am the parent with pid: 2 sending the the array: 1, 2, 3, 4, 6, 12.
>>I am the child with pid: 3, adding the array 1, 2, 3, and sending partial sum 6
>>I am the child with pid: 4, adding the array 4, 6, 12, and sending partial sum 22
>>I am the parent with pid 2 receiving from child with pid 3 the partial sum 28, and for child with pid 4 the partial sum of 22, the total sum is 28
```

Example 5: possible program execution.

```
./luis 3 12
```

Assuming parent has pid 2, child 1: 3, child 2: 4, child 3: 5. We want to see the following output

```
>>I am the parent with pid: 2 sending the the array: 1, 2, 3, 4, 6, 12.
>>I am the parent with pid: 2 sending the the array: 1, 2, 3, 4, 6, 12.
>>I am the child with pid: 3, adding the array 1, 2, and sending partial sum 3
>>I am the child with pid: 4, adding the array 3, 4, and sending partial sum 7
>>I am the child with pid: 5, adding the array 6, 12, and sending partial sum 18
>>I am the parent with pid 2 receiving from child with pid 3 the partial sum of 3,
for child with pid 4 the partial sum of 7,  and for child with pid 5 the partial sum of
18, the total sum is 28
```

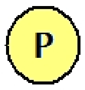
Inter Process Communication (IPC)

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.

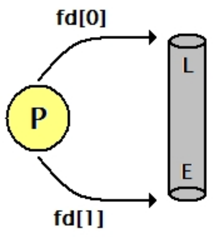
Tutorial for pipes 1: https://users.cs.cf.ac.uk/Dave.Marshall/C/node23.html
Tutorial for using pipes 2: http://tldp.org/LDP/lpg/node11.html

Basic ideas about pipes.

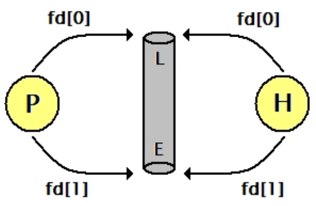
1. When the program is executed a process is created (parent process)



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).

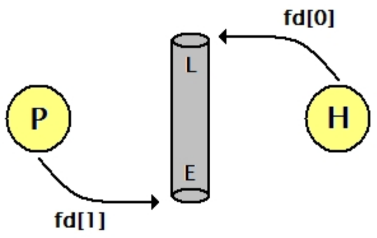


3. Then the parent must create the child process (fork instruction). When the child is created this "inherits" the same descriptors of the parent.

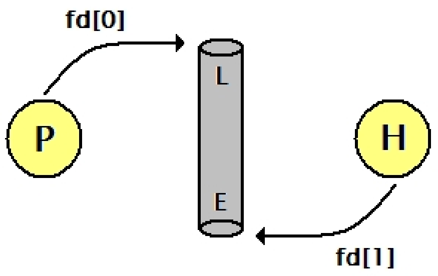


4. This way you can begin to establish communication between both processes.

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