CS39002: Operating Systems Lab
Spring 2015
Assignment 2(b): Interprocess Communication
Due: February 7, 2015

Multiple processes may exist in an environment. In numerous applications, there is clearly a need for these processes to communicate with each exchanging data or control information. This may be achieved using pipes, signals etc.

In this assignment, we will use pipes to implement a distributed median finding algorithm.

The parent process spawns 5 identical child processes along with 10 pipes - two for each *parent-child* pair (one sends messages from *parent->child*, the other sends messages from *child->parent*).

Each child reads an array of 5 integers. The numbers are read from 5 files (one for each child process). The files are named as "*data_1.txt*", "*data_2.txt*", … , "*data_5.txt*". This is done as follows:

Reading Input Data
The parent allots ids (1,2,...5) to the children and communicates it via the *parent->child* pipe.
The child receives the id and gets the input from the corresponding data file.

The numbers are **distinct** and lie **between 0 and 50** (inclusive). Our task is to find the median of the 25 random numbers (*n=25*).

We employ a distributed version of the Selection Algorithm to do this (have a look at the first hint given below for a summarized description).

Parents and Children communicate in the form of codes. Codes are simply integers which are assigned predefined values. In this algorithm we will use five kinds of commands - each represented by a unique code: REQUEST (*#define REQUEST 100*), PIVOT (*#define PIVOT 200*), LARGE (*#define LARGE 300*), SMALL (*#define SMALL 400*) and READY (*#define READY 500*).

These queries are sent along the respective *parent->child* or *child->parent* pipes. i.e., if a parent needs to send the first type of command to a particular child, it simply sends the integer 100 along their *parent->child* pipe. Since the child already knows that this integer corresponds to the command type REQUEST, it then goes on to behave in the required fashion to fulfil that command (the behavior of each command is explained in the algorithm).

The detailed algorithm is explained as follows:

- **Child:**
    - The child waits upon the *parent->child* pipe to receive its id *i*.

    - It then reads an array of 5 integers from its corresponding file (*data_i.txt*).
    - Upon doing so, it sends the code READY along the *child->parent* pipe.
    - It then enters a while loop (broken by a user defined signal - which is sent by the parent to terminate the child process).
    - In each iteration it waits on the *parent->child* pipe to respond according the codes it gets.
    - If it receives the command REQUEST from parent:
        - If its array is empty, write -1 on the child->parent pipe
        - Else chose a random element from its array and write it to the *child->parent* pipe
    - If it receives the command PIVOT from parent:
        - It waits to read another integer (and store it as pivot).
        - It then writes the number of integers greater than pivot on the *child->parent* pipe. If it has an empty array, the number would be 0.
    - If it receives the command SMALL from parent:
        - It deletes the elements smaller than the pivot and updates the array.
    - If it receives the command LARGE from parent:
        - It deletes the elements larger than the pivot and updates the array.

- **Parent:**
    - The parent forks five child processes along with their respective pipes. It goes on to *exec* the child program in each of the children.
    - It allots ids 1-5 to each of the children and sends the same along the *parent->child* pipe.
    - The parent waits on all the *child->parent* pipes until it receives the code READY from all child processes. This ensures that the algorithm initiates only after all child processes have read the input completely.
    - The parent instantiates k = n/2 (we find the kth smallest element in the array - to find median, we require k=n/2).
    - The parent selects a random child and queries it for a random element.
    - The parent sends the command REQUEST to a random child.
    - It then reads the response from the child along the corresponding *child->parent* pipe. If the response is -1, it repeats the same again. If not, it continues.
    - The first non-negative value forms our *pivot element.*
    - The parent subsequently broadcasts this *pivot element* to all its child processes. To do the same, it first writes the code PIVOT along each of the *parent->child* pipes and subsequently writes the value of the *pivot element*.
    - It then reads the response from each child. This represents the number of elements *larger* than the pivot in that child.
    - It sums up the total from all its children, call it *m.* if *m* = k, there are n/2 elements larger than *pivot* in the data set. Thus *pivot* is the median. (Make sure you handle even values correctly).

- If m>k, it sends the command SMALL to all its children which signifies that the children should drop all elements smaller than the pivot element. (Since the median would lie on the right)
- if m<k, it sends the command LARGE to all its children which signifies that the children should drop all elements larger than the pivot element. (Since the median would lie on the left). It also updates k = k - m. (*Think why?*)
- It then repeats the REQUEST until it finds the median.
- Once the median is found, the parent reports it and sends a user-defined signal to all its children - and the child processes exit after handling the signal.

All steps must be supplemented by appropriate print statements.

You have to make two files: **parent.c** and **child.c**. Ensure that input is taken in the format specified (will be used for testing the code). Parent can ***fork*** and then ***exec*** the child code.

Sample Input
data1.txt: 1 2 3 4 5
data2.txt: 6 7 8 9 10
data3.txt: 11 12 13 14 15
data4.txt: 16 17 18 19 20
data5.txt: 21 22 23 24 25

Sample Output
```
--- Child 1 sends READY
--- Child 5 sends READY
--- Child 3 sends READY
--- Child 4 sends READY
--- Child 2 sends READY
--- Parent READY
--- Parent sends REQUEST to Child 3
--- Child 3 sends 13 to parent
--- Parent broadcasts pivot 13 to all children
--- Child 1 receives pivot and replies 0
--- Child 1 receives pivot and replies 0
--- Child 1 receives pivot and replies 2
--- Child 1 receives pivot and replies 5
--- Child 1 receives pivot and replies 5
--- Parent: m=0+0+2+5+5=12. 12 = 25/2. Median found!
--- Parent sends kill signals to all children
--- Child 1 terminates
--- Child 2 terminates
--- Child 3 terminates
--- Child 4 terminates
```

--- Child 5 terminates

Optional (you will not be marked for this**):** Instead of choosing a random pivot, each process may find its own median (among its five integers) and report that.

Concepts to know for viva:

- Why were two pipes needed per child process?
- Explain how the algorithm works.
- Describe the complexity of the algorithm. Would the optional part above have any effect on the same?
- What happens if we remove the READY command?

In addition, you should be able to describe various aspects of the code.

Hints and Resources:

- *Distributed Selection Algorithm to find median:* *http://www.quora.com/What-is-the-distributed-algorithm-to-determine-the-median-of-arrays-of-integers-located-on-different-computers*
- *Piping in C*: http://www.cs.cf.ac.uk/Dave/C/node23.html
- *Interprocess communication*: http://www.advancedlinuxprogramming.com/alp-folder/alp-ch05-ipc.pdf