CS3500 - Operating System, August 2022

Lab 3: Inter Process Communication

Due Date: Friday, 1th September 11.59PM in Moodle.
Evaluation: Friday, 1th September 2-5pm at DCFE (CSE) Lab, timing -Batchwise

1. Client Server communication using message queues.
   The server and client communicate using two message queues: C2S and S2C. The client program must accept a valid system call from the user as a string and pass it to the server using C2S message queue and wait for response on S2C. Upon receiving the message, the server will take it and make a system call, the output is sent back to the client via S2C which is to be displayed on the client side and then wait for the next input. When the client gets 'End' as input the client should send it to the server and terminate. When the server receives the message, it should also terminate.
   You can have these two programs, running on two different terminals at the same time. And the user always gives input to the client program only.

   Your submission: A folder containing the following -
   1. client.c
   2. server.c
   3. Makefile - This will build your client and server. And also contains an explicit clean rule.

2. In this problem, 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 (so, here n is equal to 25).

   We employ a distributed version of the Selection Algorithm to do this (refer to the first hint given at the end of this problem).

The Parent 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)
READY (#define READY 500)

These queries are sent along the respective parent->child or child->parent pipes. 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 fulfill that command (the behavior of each command is explained in the algorithm).

The detailed algorithm is explained as follows:

a. Child:
    i. The child waits upon the parent->child pipe to receive its id i.
    ii. It then reads an array of 5 integers from its corresponding file (data_i.txt).
    iii. Upon doing so, it sends the code READY along the child->parent pipe.
    iv. It then enters a while loop (broken by a user defined signal - which is sent by the parent to terminate the child process).
    v. In each iteration it waits on the parent->child pipe to respond according to the codes it gets.
    vi. If it receives the command REQUEST from parent:
        1. If it's array is empty, write -1 on the child->parent pipe
        2. Else chose a random element from its array and write it to the child->parent pipe
    vii. If it receives the command PIVOT from parent:
        1. It waits to read another integer (and store it as pivot)
        2. 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.
    viii. If it receives the command SMALL from parent:
        1. It deletes the elements smaller than the pivot and updates the array.
    ix. If it receives the command LARGE from parent:
        1. It deletes the elements larger than the pivot and updates the array.

b. Parent:
    I. The parent forks five child processes along with their respective pipes. It goes on to *exec* the child program in each of the children.
    II. It allots ids 1-5 to each of the children and sends the same along the parent->child pipe.
    III. 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.
    IV. The parent instantiates k = n/2 (we find the kth smallest element in the array - to find the median, we require k = n/2).
    V. The parent selects a random child and queries it for a random element.
    VI. The parent sends the command REQUEST to a random child.
    VII. 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.
    VIII. The first non-negative value forms our pivot element.
    IX. 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.
    X. It then reads the response from each child. This represents the number of elements larger than the pivot in that child.
    XI. 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).
    XII. 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).
    XIII. 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. (Since the median would lie on the left). It also updates k = k - m. (Think why?)
    XIV. It then repeats the REQUEST until it finds the median.
    XV. 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 node.

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 (You can add few more meaningful print statements if you want to)

```
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
```

Hints and Resources:

- Distributed Selection Algorithm to find median: https://www.quora.com/What-is-the-distributed-algorithm-to-determine-the-median-of-arrays-of-integers-located-on-different-computers
- Piping in C: https://users.cs.cf.ac.uk/Dave.Marshall/C/node23.html