**CSCI 3431: OPERATING SYSTEM**

**Winter 2022**

**Project 1: Inter Process Communication Using Pipes**

**Report by: Nabin Bhandari and Tushar Mahat**

**Task Definition:**

In this project, our task was to implement distributed median finding algorithm using pipes.

The parent process spawns K(5) identical child processes using *fork()* system call along with 2\*K (10) pipes using *pipe()* system call - two for each parent-child pair (one sends messages from parent to child, the other sends messages from child to 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 *“input\_1.txt”, “input\_2.txt”, “input\_3.txt”, “input\_4.txt” and “input\_5.txt”.*

Communication Codes:

Parents and Children communicate using codes which are integers that are assigned predefined values. In this algorithm, we will use six commands which are described in the approach below.

The 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 to 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.

Reading Input:

The parent allocates ids (*1, 2, 3, 4, 5*) to the children and communicates them 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 1 and 100 (inclusive). Each file contains sorted numbers, and the median should be found of the 25 random numbers (n=25).

**Approach with pseudocode:**

*Everything was written from scratch following the project instructions.*

1. #include <unistd.h>, #include <stdio.h>, #include <stdlib.h>
2. Define number of children and number of inputs: #define NUMCHILD 5, #define INPUT\_NUMBERS 25
3. Define the required global values: #define REQUEST 100, #define PIVOT 200, #define LARGE 300, #define SMALL 400, #define READY 500, #define KILL 600
4. Define read and write heads along with empty signal: #define READ 0, #define WRITE 1, #define EMPTY -1
5. Declare required variables:

int selectedChild; //reandomly seled child

int p\_c [NUMCHILD] [2]; //parent->child pipes

int c\_p [NUMCHILD] [2]; //child->parent pipes

char fileName[] = "input\_i.txt"; //name of the text file

1. Create pipes using for loop: 5 *parent 🡪 child* pipes and 5 *chid 🡪 parent* pipes are created.
2. Use *fork()* system call to create 5 children.
3. Define parent process and child processes.
4. Execute parent process:

* Set k as 25/2 = 12.
* Assign *id* to every child using for loop.
* Wait for children processes to be ready.
* When all children are ready, parent is also ready and enters *while loop*:
* In the *while loop*, send *REQUEST* signal to get a random number from a random child and set it as *pivot*.
* Broadcast *pivot* to all child processes.
* If *k = m* (number of items greater than the pivot), *median* is found, send *KILL* signal to all child processes.
* Else if, *m > k*, send *SMALL* signal to all child processes.
* Else if, *m < k*, send *LARGE* signal to all child processes and update *k = k - m.*

1. Execute child processes:

* Read the assigned *id* from parent.
* Read the numbers from input files.
* Every child sends a *READY* signal to the parent.
* Enter *while loop*:
* Wait for signal from parent.
* If *signal == REQUEST*,
  + If list is empty, send *EMPTY(-1)* signal to parent to indicate that the list is empty.
  + Else, select a random element from the list and write it to *child 🡪 parent pipe.*
* If *signal == PIVOT*, wait and read the *PIVOT* value, count the number of elements greater than the pivot element using *for loop* and write it to *child 🡪 parent pipe*.
* If *signal == LARGE*, remove all the elements greater than the *pivot* element from the list and update the list.
* If *signal == SMALL*, remove all the elements smaller than the *pivot* element from the list and update the list.
* If *signal == KILL*, terminate all child processes.

**Output and Observation:**

input\_1.txt: 26 29 38 48 50

input\_2.txt: 69 71 72 75 78

input\_3.txt: 79 80 81 82 83

input\_4.txt: 88 89 90 91 92

input\_5.txt: 93 94 95 96 97

If there is no input file present or the input file name is different than expected, the program will send an error message and instruct the user to terminate the program.

Text

Description automatically generated

Graphical user interface, text

Description automatically generatedInitially, *k* is 12. At first, all the child processes read the input and send a *READY* signal to the parent. Now, the parent is ready and sends *REQUEST* signal to a random child 3. Child 3 selects a random element 80 and sends it to the parent. The parent then broadcasts 80 as the *pivot* to all child processes. The child processes count and return the number of elements greater than the *pivot* element. The parent sums all the returned values and stores it as *m* and compares it with *k*. Since *k(12) <m(13)*, it sends the *SMALL* signal to all child processes, and all the elements smaller than 80 are removed from the list. The parent again sends *REQUEST* signal to a random child 3 which returns a random element 82. Now the process is repeated and *k(12) > m(11)*, so *LARGE* signal is sent to child processes and *k* is updated as *k-m=1*, and the elements bigger than 82 are removed from the list.

Again, the parent sends a *REQUEST* signal to a random child 2 for value but since there is no element left in the list of child 2, the parent again sends *REQUEST* signal to another random child 1 which is also empty and then the request is randomly sent to the child 3. Child 3 returns 81 to parent which is sent to all child processes as the pivot. Now only child 3 has an element greater than 81, so m =1 = k.

Hence, the median is 81 as it was the pivot element, the parent finally sends the KILL signal to all child processes to terminate them.

**References:**

1. Delete all array elements that match a user input value: <https://www.youtube.com/watch?v=1uf75EJsveU&t=417s> (Modified according to our project requirement)
2. Project1.pdf (Project 1 instruction)