Operating Systems

Assignment 1: Median Finding through fork()

Ben Jollymore

A00400128

Oct 6th 2018

**Pseduo-code Structure used to Approach**

**main(){**

pipe(parent to child); for all children

pipe(child to parent); for all children

for (all children)

fork();

break;

if(child)

execute child process;

else if(parent)

execute parent process;

else(error in fork)

terminate in error conditon

terminate

**}**

**child(){**

int ID;

read(ID);

read(file data, to array);

send(READY);

read(control);

loop switch while alive

switch(control)

REQUEST:

send random value to parent

SMALL:

drop values smaller than pivot

LARGE:

drop values larger than pivot

PIVOT:

send values larger than pivot

KILL:

kill child

**}**

**parent(){**

for(all children)

send(ID);

read(command);

if(READY)

continue;

send(REQUEST to random child)

if(PIVOT!= -1)

continue;

else

send(REQUEST to random child) repeating as neccesary until value other than -1 recieved

if m = k

median found as pivot

terminate

if m > k

send LARGE;

k -=m;

try procedure again

else

send SMALL;

try procedure again

**}**

**Adaptations and Problem Solving**

Initially, I wanted to have a parent/child pipe for each child. It would then be possible to read/write each child easily and keep track of pipes.

//declare arrays for all child to parent pipes

int c1p[2], ... c5p[2];

//declare arrays for all parent to child pipes

int pc1[2], ... pc5[2];

//turn all arrays into pipes

pipe(c1p);

...

pipe(c5p);

pipe(pc1);

...

pipe(pc5);

//send information to arbitrary child, say id=4

close(pc4[READ]),

write(pc4[WRITE], &content, sizeof(content))

This approach, however, was very clunky, and wouldn't allow me to read/write each child through a loop or condition statement without repeated chunks of the same code in cascading if/else for each pipe. So I instead dynamically initiated each of my pipes based on a multidimensional array for each parent to child pipe and child to parent pipe. This allowed me to easily access each pipe by simply passing a child ID to the pipe's first dimension.

//define arrays to be converted to pipes

int childToParent[NUM\_CHILDREN][2];

int parentToChild[NUM\_CHILDREN][2];

//initalize pipes. Child 1 will use childToParent[1] and parentToChild[1], and so on.

void intializePipes() {

for (int i = 1; i < NUM\_CHILDREN + 1; i++) {

pipe(childToParent[i]);

pipe(parentToChild[i]);

}

}

From here, I was able to proceed to the fork process initialization segment. Instead of using a string of if/else statements to fork() each process, I insteaded used a for loop. I initially ended up spawning far too many child processes, as I did not break the loop after each process began – so each child would continue in the loop spawing more childrem. This was easily remedied through the break command.

IDs were then sent to each child down it’s respective parent-child pipe based on it’s ID, signifying that each child was ready to read from the input file. However, since each child was spawned without an ID, it would not know which pipe to read from. As such, a global counter was declared to help each child find it’s proper pipe. Prior to forking in the for loop, the counter is incremented, such that the first child’s counter value in memory will be 1, the second’s counter value will be 2, and so on. That counter value is passed to the pipe array to recieve the ID.

//in parent process

for (int i = 1; i < NUM\_CHILDREN + 1; i++) {

childCounter++;

child = fork();

//in child process

close(parentToChild[childCounter][WRITE]);

read(parentToChild[childCounter][READ], &id, sizeof(int));

When reading the file, I wanted a single function that would be able to recieve a request from a child and read the information for that child’s corresponding file into the contents of the child’s number array. To do this, function readFile(int id, int \* array[]) was created, which takes an integer ID and a pointer to a number array as parameters. The function uses that ID to generate a file name unique to that ID (keeping with the file naming standards), and then reads the contents from the file to the addresses of the passed array.

void readFileToArray(char fileID, int numArray[]) {

//generate name

char fileName[] = "input\_i.txt";

fileName[6] = fileID;

//Access the file

FILE \*inputFile;

inputFile = fopen(fileName, "r");

Because in C, early bound arrays are lists of fixed size containing pointers and have no real "empty" value, I couldn't decrease the size of the array nor check if it had a size of zero (indicating empty). Instead of dereferencing the contents of the array during drop operations, I simply set the value of a dropped index to -1, indicating an invalid entry. If all values in the array are -1, the array is deemed empty.

int isEmpty(int numArray[]) {

int empty = 1;

for (int i = 0; i < 5; i++) {

if (numArray[i] != -1) {

empty = 0;

break;

}

}

return empty;

}

**Results**

After several passes, the program always finds the median successfully. Although there have been numerous fault checks put in place, through nearly 100 test runs it never triggered any fault catches.

**Discussion**

My code could be tidied up by declaring a readMessage(pipe[], int \* message location) and writeMessage(pipe[], int message) function to send messgaes via pipes in a single line instead of having repeated multi-line statements. I began implementing this, however the writeMessage() function would always cause the program to hang, so it was eventually discarded as a loss.

I tried to model my program output after the sample output as best as I could. However, I am not entirely sure what the sample output line: “-- Parent: m=0+0+2+5+5=12. 12 = 25/2. Median found!” is trying to communicate, as it only displays once (since the sample output finds the median on the first pass through).

As such, I assumed the 25 represented the number of elements, and 25/2 indicated the midpoint (as the numbers are known to be sorted). Given that 12 is not the median in the sample output and 12 != 25/2 in regard to the midpoint(it is the index before the midpoint), I was confused and assumed it was a typo, meant to be 13 (which would be the midpoint and median). If this assumption were wrong, I could modify my program to, through pipes, update a counter that holds the number of remaining elements after drops, and ouput that in the final output line.

**References**

Creating Pipes in C - The Linux Documentation Project

<https://www.tldp.org/LDP/lpg/node11.html>

pipe() System call - Geeks for Geeks

<https://www.geeksforgeeks.org/pipe-system-call/>

int to char in c - StackOverflow

<https://stackoverflow.com/questions/2279379/how-to-convert-integer-to-char-in-c>

boolean values in c - StackOverflow

<https://stackoverflow.com/questions/1921539/using-boolean-values-in-c>

read numbers from text files to an array in c - StackOverflow

<https://stackoverflow.com/questions/20378430/reading-numbers-from-a-text-file-into-an-array-in-c>

Fork inside for loops - StackOverflow

<https://stackoverflow.com/questions/26793402/visually-what-happens-to-fork-in-a-for-loop>

Random number in range in c - StackOverflow

<https://stackoverflow.com/questions/1202687/how-do-i-get-a-specific-range-of-numbers-from-rand>

C++, fork(), IPCs using pipe. Multiple Forks, Multiple pipes - SFU

<http://www.sfu.ca/~reilande/>

fork() and exec() - U-Waterloo

<https://ece.uwaterloo.ca/~dwharder/icsrts/Tutorials/fork_exec/>

Variable declaration inside switch statement / Scope of switch statement -StackOverflow

<https://stackoverflow.com/questions/92396/why-cant-variables-be-declared-in-a-switch-statement>