Brandon Snow tfv024
Meng Wang nji768

WCDesignForMultiprocess.txt

1.Name of collaborators: Please include the ABCid and Name of both collaborators under the title.

Two people worked very hard on this project Brandon Allen Snow and Meng (jason) Wang. We spent probably like 40 hours on this project and made sure to try to account for everything. This project is definatly the hardest thing we have done in this class and it humbled both Jason and Me. What we did was a 50/50 split. Even on the write up we constantly are switching back and fowarth to make sure we can switch any code out. The base for our project was Brandon Snows assign4. We spent a hell of alot of time getting this going and I (brandon snow) are very proud of what we have accomplished and learned.

2.Design Description: This part describes how you implement your program, such as the steps to resolve this problem. You should describe the steps to finish this assignment. For example, how you can make different processes access different parts of the same file. How you define the relationship between the parent and children processes? How to know whether a child process has been exited or not? How to sort those words inside different processes? I should not READ your source code to know how you will finish this project. Please talk in a reasonably detailed, but not putting your code inside.

The basic idea for this project is using multiple processes. For assignment 3, we have c code to wordcount the file. In this one we gonna using fork() to create multiple children, and using those children to read file. The mmap() is a function will cut the file start from offset to the size that you give to the function. One thing I think it will be really helpful is give us some example for this mmap(). How is this working, because at frist we think you are going use the mmap() to cut the file into different memory. But truth is we going to use mmap to map this large file to the memory and divide the workload by passing different starting address. After that we starting the access those memory blocks. We are using the stat to return the size of the file. Afte that we using filesize devided the number of process which was inputed by user. Then we get the how big for each and every memory block. There is one thing, when you try to seprate the file. There is a chance to cut a word into two different part. we don't want that happened. So to avoid that happened, we are using startingIndex and endingIndex. The startingIndex = i* length and the endingIndex = (i * length) + length. If the char is not equal to space or tab or enter. We just gonna plus one to index, also we have to move the two index(starting index and endingindex) sametime. because we want reavarange the index. after memory block is ok, then we starting the child process. We will have a loop. The times of this loop will equals to the number of block. For each and every time we will use pid = fork(). Then this will create a child. if the child equals to -1. which means there is some error in fork. if the fork equals to 0, means we successfually create a child, we are going using this child to reading from the first memory block then print to the file. This is all happened in almost the same time. Each child will write the file. After all the process finished access the file, we will use parents to read to file and print to the terminal. Of course we are going to use the doubly linked list to store and sorted the word.

3. Performance Comparison: This part will get the performance of your program on a middle and large size input, which is provided at http://www.cs.utsa.edu/~tongpingliu/teaching/cs3423/largeinput. You will have to insert some figure as Figure 1. This time, you can evaluate the program by feeding different PROCNUMBER to it. In the end, you should provide a figure with different number of processes, at least including 1 (PROCNUMBER is 0), 2, 4, 8, 16 processes in the figure. You will normalize the performance to 1 process. Here is an example on this, if the runtime of 1 process is 1 second and the runtime of 2 processes is 0.5 second, then the normalized performance of 2 processes is 2 or 200%.

In the meanwhile, you should explain the reason of this performance data. Horizontally, you will have to compare with middle input and large input. Vertically, you will explain the difference with different number of processes. You can get the CPU information by checking the following file: /proc/cpuinfo.

The performance of this sucks. We couldn't get pipe working so what we did was write to a seprate file and read from it so its fucking terrible. This was the wrost c program I have ever had the displeasure of writing. The run time is about the same or worst. It was a very quick and dirty way of getting this done. If we had more time we could of perfected this i believe its just not a good time since classes are winding down and we are getting to the end of the semester and are burnt out.

As for the performances increasing or descreasing with more process's doesn't give the desired performance boost. It does work alittle faster but not enough to put into graph form. If you want just send us a email and we will make a graph but it doesn't seem to give the desired outcome we had hoped.

4. Advantage of Using Multiprocesses: In which condition, do you think that multiple processes can provide you the best performance? You can be open minded here.

ADVANTAGES more like disadvantages. This was a nightmare for me. This was the hardest thing I have had to do. I stayed up late trying to get the performence going but it was terrible. I spent alot of time trying to get pipe to work to no avail.

Brandon Snow tfv024
Meng Wang nji768