CS342 Operating Systems - Fall 2020
Project #1 – Multi-Process Application using IPC

Assigned: Oct 10, 2020, Saturday.
Due date: Oct 21, 2020, Wednesday, 23.59 Document version: 1.0

- *Submit through Moodle. Make sure you start submitting one day before the deadline. You can overwrite your submission as many times as you wish. Late submissions will not be accepted (no excuse; no email will be accepted).*
- *The project will be done individually.*

Assignment

Part A) In this project you will write a multi-process application called **pwc** that will read a set of ascii text files containing words and generate word frequency information into an output file. The application will create multiple child processes, and processes will communicate using Linux message queues. You will use POSIX message queues.

The program will be invoked as follows:

pwc N f1 f2 ... fN outfile

f1, f2, fN are N input files. The main process will create N child processes. Each child process will process another input file. Each input file contains a sequence of words. A line of input file may contain multiple words. A word is just a sequence of non-whitespace characters between two whitespace characters. A whitespace character can be space, tab, or newline character (you can use fscanf("%s") to read a word). The maximum size of a word can be 1024 characters including the NULL character at the end. For each child process, the parent will also create a message queue, so that the child process will be able to send information to the parent process. The message queues should better be created earlier than the child processes in the parent.

Each child process will read and process its own input file and will create a sorted linked list distinct words in memory (you can use insertion sort) in ascending order. For each distinct word, the list will have a node that will keep the word count as well, besides the word itself. Since the string-length of the word, can be anything between 1 and 1023, you should dynamically allocate memory for a node of the list using malloc(). After reading the input file completely and generating the list meanwhile, the child process will start sending the words in ascending sorted order to the parent process using the respective message queue. Two words can be compared using the strcmp() function.

The parent process will receive the incoming words and their counts from children as messages, one by one in sorted order, and will write them into the output file. Note that the parent process will not sort again. It should retrieve the words from children in sorted order. This can be achieved by first receiving one word from each

child (through the respective message queues), and then finding the word that is the first in ascending order, and then writing it out to the output file. Then the parent can receive another word from the child whose word has just been written out, and compare it again with the already received words, find the word that is first in ascending order, and write it out. If the words received form multiple messages queues are the same, their counts will be summed up and the word will be written to the output file once. The output file will have one word and its count per a line. The word and its count will be separated with a space character.

The maximum number of child processes will be 5. The maximum number of message queues will be 5 as well.

Part B) This time you will implement the same program using multiple threads. This program will be called **twc**. For each input file, you will create another thread, that can be called a worker thread, that will process the file. A worker thread the put the words read into a sorted linked list (again you can use insertion sort). The main thread will wait until all worker threads finished generating the linked lists. Then the main thread will access the linked lists generated worker threads, and will write the words and their counts to output file in sorted order, as in part A. You will use POSIX threads (also called Pthreads).

Experiments

After developing and testing the programs, you will do some timing experiments. For each program, conduct timing experiments that will measure the time it takes to complete the program for various number of children, sizes of input files, etc. Plot the results in graphs. Compare thread performance with child-process performance. Try to interpret the results. Compare thread performance with child-process performance. Put all these into a report.

Submission

You will submit your report in pdf form. You will also submit your pmc.c file, tmc.c file, and your Makefile. Put all these files into a directory named with your Student Id, and tar and gzip the directory. For example a student with ID 21404312 will create a directory named "21404312" and will put the files there. Then he/she will tar the directory (package the directory) as follows:

```
tar cvf 21404312.tar 21404312
```

Then he will gzip the tar file as follows:

```
gzip 21404312.tar
```

In this way he will obtain a file called 21404312.tar.gz. Then he will upload this file in Moodle.

Late submission will not be accepted (no exception). A late submission will get 0 automatically (you will not be able to argue it). Make sure you make a submission one day before the deadline. You can then overwrite it.

References

[1] man mq_overview will give you help pages for messages queues in Linux.

[2] man pthreads will give you information about POSIX treads in Linux. There are also a lot of resources in Internet for POSIX thread programming.

Tips and Clarifications

- Start early. Do not be late in submitting your project.
- Do not forget to compile your twc program with the -lpthread option, so that Pthreads library can be used.