CS342 Operating Systems – Fall 2022
Project #1 – Processes, IPC, and Threads

Assigned: Oct 9, 2022.
Due date: Oct 24, 2022, 23:59. Document version: 1.0

This project will be done in groups of two students. You can do it individually as well. You will program in C/Linux. Programs will be tested in Ubuntu Linux. No deadline extension will be provided. Therefore start as soon as possible.

Part A (50 pts): In this project you will develop an application that will find word frequencies in a given input data set. In part A, the application will use multiple child processes to process the data set. In part B, it will use multiple threads. Hence, you will develop two programs, that will essentially to the same thing. The data set will contain N input files. N can be at most 8. Child processes will use a message queue to pass information to the parent.

In part A, the program will be named as **`pword`** and it will have the following command line arguments.

pword <msgsize> <outfile> <N> <infile1> .... <infileN>

The <msgsize> is size of a message that a child will send to the message queue. Possible values of <msgsize> will be 128, 256, 512, 1024, 2048, 4096 (in bytes). The <outfile> parameter is the name of an output file which will contain the result. <N> is the number of input files. Then comes the names of <N> input files.

An input file will be an an ascii text file containing words (strings) separated with white space characters. 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. The maximum size of a word can be 64 characters, including the '\0' (NULL) character at the end (that means a word can have at most 63 non-white printable characters). We will use such input files in our tests. You can assume that a word will always contain printable characters. After reading a word form a file, you will convert all lower-case letters (a-z) in the word to upper-case (A-Z). For example, a word `123abcBd#` will be converted to `123ABCBD#`. Other characters will remain as they are.

There program will create <N> child processes to process <N> input files. The results obtained by the children will be passed to the parent via a message queue. Therefore, when started, the program (the parent process) will first create a message queue. The name of the message queue should be stored in a global variable, so that child processes can access the same message queue using that name. The parent will also put the names of input files into a global table (an array, for example). Then, the parent process will

create <N> child processes via the `fork()` system call. Since the address space (memory) of the parent is copied to the children initially, all children will obtain the filenames table and the name of the message queue. Using the message queue name, all child processes will open and access the same message queue. Each child process will use one of the filenames in the filenames table and open that file for reading and processing. Each child will operate on a different file.

A child process will read its input file word by word and will process these words to generate information about their counts (frequencies), i.e., how many times each word appears in its input file. This information will be generated into an in-memory data structure that you will design. After finishing reading and processing the input file, the child process will pass the frequency information to the parent process via the message queue. The information will be passed with messages of length <msgsize>.

You should put as many (word, count) pairs as possible into a single message. Note that words are of variable size. Some words may be very short, some may be quite long. In the beginning of a message, you may also include information about the number of word-count pairs in the message. The last message may have fewer words. After the last data message, a child can send another message to indicate the end of the message stream. Then the child will terminate.

The parent process, after creating the child processes, will wait for the arrival of messages that will include (word, count) pairs. This can be done by invoking a blocking receive operation on the message queue. While receiving word information sent by multiple child processes, the parent process will process and store the information into a data structure in memory. Counts for the same word will be added together. When finished, the parent will sort the (word, count) pairs in ascending order. You can use the `strcmp()` function to decide which one of two words should come first. After sorting, the information must be written to the output file in ascending order. Below is a sample output. Your output format must be the same with this. Each line of output will contain information about a word. First, the word will come, then its count, separated by a space character. Each line will start with a word. There will no empty lines.

```
ABC 150
B#25DF 50
HCD 300
THE 400
```

An example invocation of the program can be:

```
pword 1024 outfile.txt 3 in1.txt in2.txt in3.txt
```

All children will run concurrently. While one or more children are sending data to the message queue, the parent process should concurrently retrieve the data. When parent process finishes writing the result to the output file, it will terminate. Before termination it must remove the message queue.

Not that we can use automated tools to test the output. Therefore, your output format must strictly follow the specification here. We will also look to your source code, to see the processes and threads created and used properly.

Your submitted program should print nothing to the screen. It should produce just the output file. However, you ca nprint information to the screen while developing and testing your program. But before submission, you should remove all these extra outputs to screen or to a file.

Part B (25 pts): In this part, do the same project by using threads, instead of child processes. The name of the program will be **`tword`** in this case. The program will have the following parameters. Their meaning is the same with pword.

tword <outfile> <N> <infile1> .... <infileN>

Each input file will be processed by another thread. If there are N input files, there will be N new threads (worker threads) created by the main thread. In this part, you do not need to use a message queue. You will use global variables to pass information from worker threads to the main thread.

Part C – Experiments (25 pts):

Do some measurement experiments, and record, plot and interpret your data. Measure the time it takes to run program A for various values of N (1, 2, 3, 4, 5). Measure the time it takes to run program A for various message sizes (fix N to 5, for example). Compare and interpret the results. Measure the time it takes to run the program B for various values of N. Compare your results. Also compare the results of B with that of A. Try to interpret your results.

Submission:

Put all your files into a directory named with your Student ID. If the project is done as a group, the IDs of both students will be written, separated by a dash '-'. In a README.txt file, write your name, ID, etc. (if done as a group, all names and IDs will be included). The set of files in the directory will include README.txt, Makefile, and program source files. We should be able to compile your programs by just typing make. No binary files (executable files) will be included in your submission. Then tar and gzip the directory, and submit it to Moodle.

For example, a project group with student IDs 21404312 214052104 will create a directory named "21404312-214052104" and will put their files there. Then, they will tar the directory (package the directory) as follows:

tar cvf 21404312-214052104.tar 21404312-214052104

Then they will gzip the tar file as follows:

gzip 21404312-214052104.tar

In this way they will obtain a file called 21404312-214052104.tar.gz. Then they will upload this file into Moodle. For a project done individually, just the ID of the student will be used as file or directory name.

Tips and Clarifications:

- POSIX message queues will be used. Not System V message queues.
- Start early, work incrementally.
- "man mq_overview" will give you information about Linux message queues. You can find more information on the Web.
- Be careful about *memory alignment* while putting and accessing multi-byte scalar data type, like integers, into an array of bytes by use of pointers. For example, an integer value should start at an address that is multiple of sizeof(int).