CS342 Operating Systems – Spring 2023
Project #1 – Processes, IPC, and Threads

Assigned: March 3, 2023.
Due date: March 21, 2023, 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, please start as soon as possible.

Part A (50 pts): In this project you will develop an application that will find the K most frequently occurring words, i.e., top-K words, in a given input data set (K is an positive integer). 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 do the same thing. The data set will contain N input files. Child processes will use **shared memory** to pass information to the parent. The shared memory segment will be created by the parent. Each child process will access and use a portion of the shared memory.

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

proctopk <K> <outfile> <N> <infile1> .... <infileN>

The parameter <K> is the number of words to find, denoted as K. The <outfile> parameter is the name of an output file which will store the result. <N> is the number of input files, denoted as N. Then comes the names of N input files.

An input file will be an an ascii text file containing words (strings) separated by 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 immediately 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.

The program will create N child processes to process N input files. The results obtained by the children will be passed to the parent via shared memory. Therefore, when started, the program (the parent process) will first create a shared memory segment of enough size. The name of the shared memory should be stored in a global variable so that child processes can access the

same shared memory 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 shared memory. Using the shared memory name, all child processes will be able to open and access the same shared memory. 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 unique words and their counts (frequencies), i.e., how many times a word appears in its input file. Then it will select the K most frequently occurring words and will write them to its portion of the shared memory, together with their frequencies (counts). Each child process will use a portion of the shared memory.

The parent process, after creating the child processes, will wait until all children run and put their information to the shared memory. Then, the parent will read the information from the shared memory and combine the information. In total it will read KxN words and their counts from the shared memory (K words from each child). For the same words arriving from different children, the counts will be added. Then the parent will select the top K words and write them to the output file together with their frequency information (in descending sorted order with respect to counts).

You can use the `strcmp()` function to decide if two words are same or not. 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 be no empty lines.

```
ABC 150
B#25DF 100
HCD 75
495 70
300 64
THE 60
```

An example invocation of the program can be:

```
proctopk 100 outfile.txt 3 in1.txt in2.txt in3.txt
```

All children will run concurrently. After a child process writes its result to the shared memory, it can terminate. Before reading from shared memory, the parent process needs to wait until all child processes terminate. When parent

process finishes writing the result to the output file, it will also terminate. Before termination it must remove the shared memory.

Note 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, shared memory, and threads created and used properly.

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

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 **`threadtopk`** in this case. The program will have the following parameters. Their meaning is the same with proctopk.

threadtopk <K> <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 shared memory. You will use global variables to pass information from worker threads to the main thread. Each worker thread can put its result to a different linked list (or any other data structure that you wish to use). After all worker threads terminate, the main thread can read these lists to learn the words each child has selected. Then the main thread can select the top K words and write them to the output file together with their frequency information.

Part C – Experiments (25 pts):

Do some measurement experiments, and record, plot and interpret your data. Measure the time it takes to run the programs A and B for various values of N (fix K to a value). Measure the time it takes to run the programs A and B for various K values (fix N to a value). Compare and try interpret the 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:

- Start early, work incrementally.
- POSIX shared memory will be used. Not System V shared memory.
- "man shm_overview" will give you information about Linux shared memory. You can find more information on the Web.
- You will use POSIX threads (Pthreads). To get information on Linux command line, you can type "man pthreads". You can also find a lot of information in Internet.
- Be careful about *memory alignment* while putting and accessing multi-byte scalar data types, like integers, into an array of bytes by use of pointers. For example, an integer value should start at an address that is a multiple of sizeof(int).
- Min value of K can be 1. Max value can be 1000.
- Min value of N can be 1. Max value can be 10.
- The size of the shared memory that you should create depends on the value of K and N. You will decide. To optimize space usage, you can place words into shared memory to occupy just enough space. For that you need to store the lengths of the words as well.