Project Description

You will design and implement a keyword search program called ks_bb, which will take two command line arguments from the user as follows:

```
./ks_bb commandFile bufSize
```

commandFile is the name of a file, which includes the search commands to be performed. Each search command is composed of a keyword and a dirPath. A sample content of a commandFile can be as

follows:

/home/user/dirOne is

/home/user/dirOne a

/home/user/dirTwo a

For each command, your program will search the keyword in all the files sitting in the given

directory using multiple threads. You can make the following assumptions about the commandFile:

1. The dirPath is a full path

2. The keyword is a single word, starting with a whitespace character and ending with a whitespace

character, not including any whitespace character in themiddle. A whitespace character is either

a <SPACE>, <TAB>, or <NEWLINE> character. There is no empty keyword specified.

3. Each line of the commandFile will never exceed MAXLINESIZE characters (including the

directory path, keyword, and whitespace(s) in between)

Your program will take the commands from the commandFile one by one. Then it will create a

child process to serve the command. The command info will be passed to the child process. The child

process will then create a number of threads (one thread per input file in the given directory) to find

out the matching lines in the input files. These will be the worker threads. A worker thread will be

responsible from exactly one input file. It will scan the input file and look for matching lines. You

can assume that there will be only files in the given dirPath, no sub-directories.

A match in a line will be an exact match of an entire word only and the maximum length of

a line in an input file will never exceed MAXLINESIZE. When a match is found in a line, it will

prepare an item that includes the following information: the name of the file where match occurred,

the line number, and the line itself (i.e. the line string excluding the newline character at the end).

If the keyword is seen, for example, in 5 separate lines, then 5 separate items will be created. If the

keyword is seen more than once in a line, then a single item will be created for that line. For example,

if we had a line as "a a a", then the number of matches on this line would be 3. However, we have

to generate a single item. Whenever an item is created, then the worker thread will add this item to a

memory buffer: a bounded buffer.

The second argument of your program (bufSize) specifies the size of the bounded buffer to be used

by the threads. Each worker thread will add the produced items to this bounded buffer. The buffer

can hold at most bufSize items. This bounded buffer can be implemented in one of two ways: 1) as a linked list of items; or 2) as a circular array of pointers to items. You can choose either one of these implementation options. A worker thread will add a new item to the end of the buffer. The buffer has to be accessed by all worker threads. While trying to insert an item, if a worker thread finds the buffer full, then the thread has to go into sleep (block) until there is space for one more item. Since all worker threads will access the buffer concurrently, the access must be coordinated. Otherwise we can have race conditions. You can use semaphores/mutexes/condition variables to protect the buffer and to synchronize the threads so that they can sleep and wake-up when necessary.

Beside the worker threads, there will be another thread created by the child process while serving a request. That other thread will be identified as a printer thread. The job of that thread will be to retrieve the items from the bounded buffer and print them to the screen. While the items are added to the buffer by the worker threads, the printer thread can work concurrently and try to retrieve the items from the buffer and print them to the screen. The printer thread will try to retrieve one item from the bounded buffer. If the buffer is empty, the printer thread has to go into sleep until the buffer has at least one item. When the printer thread is successful in retrieving an item from the buffer, it will print it to the screen in the following format:

filename:linenumber:linestring

For example, if the keyword is "a" and the content of the file to be searched (assume the file name is file1.txt) is as follows:

I raised my daughter in the American

fashion; I gave her freedom, but

taught her never to dishonor her

family. She found a boy friend,

not an Italian. She went to the

movies with him, stayed out late.

Two months ago he took her for a

drive, with another boy friend.

They made her drink whiskey and

then they tried to take advantage

of her. She resisted; she kept her

honor. So they beat her like an

animal. When I went to the hospital

her nose was broken, her jaw was

shattered and held together by

wire, and she could not even weep

because of the pain.

Then, the printer thread will print the following output to the screen:

```
file1.txt:4:family. She found a boy friend,
file1.txt:7:Two months ago he took her for a
```

Pay attention to the following issues:

• The output should not contain the name of the directory, but just the name of the file.

• Only the lines that include the exact match of an entire word is printed.

• For each search command read from the commandFile, a specific bounded buffer, a specific printer thread, and bunch of worker threads (one for each file in the given dirPath) are created by the child process that is assigned for that search command.

• All the worker threads for a given command do the same thing on a different files and fill the same bounded buffer.

• Your program passes each received search command to a child process and handle the next search command immediately without waiting for the created child to complete its execution. However, you should also make sure that your program does not terminate before all the output is printed. To achieve this, you can count the number of children created by the parent and after handling all the requests (end of command file is reached), then the parent process can call wait() system call in a loop for count times.

- **You can assume that theMAXLINESIZE is 1024. Also, remember to use thread-safe library functions inside your threads! (for instance strtok_r() instead of strtok()).**
- **Compiler: gcc**
- **Submit file: ks_bb.c**