CE 4348: Operating Systems Concepts
Section 501
Programming Project 2

Instructor: Neeraj Mittal

Assigned on: April 11, 2024
Due date: May 3, 2024 (at midnight)

This is a group assignment; you can work in groups of size at most two. You can discuss the assignment with other groups but must write the code on your own. Code sharing among groups is strictly prohibited and will result in disciplinary action being taken. The project should be done in C++.

1 Objective

This assignment requires you to use threads and semaphores to implement a pipelined grep application. Your pipelined grep will search all files in the current directory for a given string. The user will also be able to filter files out of the search based on file size, file user id, and file group id. You will construct your application as a five stage pipeline where there will be one thread per stage. Threads between consecutive stages will use the *producer-consumer paradigm* to communicate and synchronize their actions.

A sample solution for producer consumer problem, also known as bounded buffer problem, using semaphores in C++ will be provided to you as a starting point.

2 Resources

You should read the following manual pages:

- sem_init, sem_destroy: initialize and destroy semaphores
- sem_post, sem_wait: increment and decrement operations on semaphores
- strstr, strlen, strcpy, strcat: Useful string functions for implementing grep
- opendir, readdir: Operations for opening a directory and reading its entries one-by-one
- lstat, stat: Get file status

You can also look at the output of grep to verify your matches and line numbers.

3 Assignment

You will implement a simplified, pipelined version of the UNIX command grep. Your command, pipegrep, will take five command-line arguments[1]. Here is a description of each argument:

1. ⟨*buffsize*⟩: buffsize gives the size of the buffers between pipeline stages.
2. ⟨*filesize*⟩: filesize indicates the maximum size (in bytes) of files your program will ignore. For example,if ⟨*filesize*⟩ is 100, then you would ignore all files less than or equal to 100. If ⟨*filesize*⟩ is -1, then you ignore this field.
3. ⟨*uid*⟩: you should only search files whose owner is given by ⟨*uid*⟩. If ⟨*uid*⟩ is -1, then ignore this field.
4. ⟨*gid*⟩: you should only search files whose group id is given by ⟨*gid*⟩. If ⟨*gid*⟩ is -1, then ignore this field.
5. ⟨*string*⟩: This is the string you are searching for in the lines of the files.

Here is a standard UNIX usage string for the command:

Usage: pipegrep ⟨buffsize⟩ ⟨filesize⟩ ⟨uid⟩ ⟨gid⟩ ⟨string⟩

3.1 Pipelined Architecture

Your program will be organized into the following architecture:

Stage 1 → buff1 → Stage 2 → buff2 → Stage 3 → buff3 → Stage 4 → buff4 → Stage 5

The buffers between the stages are "bounded buffers" that will hold intermediate results between stages. You should ensure proper, synchronized access to these buffers. Remember, the size of your buffers is determined by the command-line argument ⟨*buffsize*⟩. The stages are described next.

3.2 Stage 1: Filename Acquisition

The thread in this stage recurses through the current directory and adds filenames to the first buffer (*buff1*).

3.3 Stage 2: File Filter

In this stage, the thread will read filenames from *buff1* and filter out files according to the values provided on the command-line for ⟨*filesize*⟩, ⟨*uid*⟩, and ⟨*gid*⟩ as described above. Those files not filtered out are added to *buff2*.

3.4 Stage 3: Line Generation

The thread in this stage reads each filename from *buff2* and adds the lines in this file to *buff3*.

[1] You do not have to implement any of the options of the UNIX command grep.

3.5 Stage 4: Line Filter

In this stage, the thread reads the lines from *buff3* and determines if any given one contains ⟨*string*⟩ in it. If it does, it adds the line to *buff4*.

3.6 Stage 5: Output

In the final stage, the thread simply removes lines from *buff4* and prints them to stdout. Also, you need to figure out when to exit the program. How do you know when you got the last line? Hint: you can use a "done" token (would not work if you had multiple threads in a stage).

4 Examples

Here is an example of the output:

```
[neerajm@cs1 Solution]$ ./pipegrep 10 -1 -1 -1 endif
./boundedBuffer.h(49): #endif
./pipelineGrepStages.h(16): #endif
***** You found 2 matches *****
[neerajm@cs1 Solution]$ ./pipegrep 10 -1 -1 -1
Usage: pipegrep ⟨bufesize⟩ ⟨filesize⟩ ⟨uid⟩ ⟨gid⟩ ⟨word⟩
[neerajm@cs1 Solution]$
```

5 Additional Notes

The last line of your output will be a line that gives the total number of matches (see above example).

There will be no limit on the number of files, number of matches, or the line length.

Note: There may be multiple critical sections in your code that you must protect with a mutex. You must minimize the size of these critical sections. You will be penalized points if your critical sections contain unnecessary code.

Deliverables

You need to submit the following:

1. All your source files that contain your C/C++ implementation of pipegrep. You also need to supply a properly constructed Makefile that can be used to compile your code (by just typing "make"). The main executable file should be named pipegrep.
2. A README file that describes (1) what you got working, (2) a description of your critical sections, (3) the buffer size that gave optimal performance for 30 or more files, (4) tell me in which stage you would add an additional thread to improve performance and why you chose that stage, and (5) Any bugs in your code.

Notes:

1. You must have the names of your group members at the top of the README file.
2. In your description of your critical section you need to include a justification for code contained in it. You also need to describe the race condition you are protecting against.
3. There are questions you need to answer and experiments you need to run and include in your README file. These will be graded and makeup a non-trivial portion of your grade for this project.

Extra Credit

You can earn up to 20% additional points by recursively searching files in subdirectories as well for the given string.