CS3103: Operating Systems

Spring 2021

Programming Assignment 2

1 Goals

The purpose of this assignment is to help you:

- get familiar with multi-threaded programming using pthread
- get familiar with mutual exclusion using mutexes
- get familiar with synchronization using semaphores

2 Background

Sentiment analysis, which is a powerful technique based on natural language processing, has a wide range of applications, including consumer reviews analysis, recommender system, political campaigning, stock speculation, etc. A sentiment analysis model requires a large text corpus, which consists of classified articles grabbed from the internet using web crawlers.

In the simplest scenario, a text corpus can be built by two components: a web **crawler** and a **classifier**. The crawler browses through web pages and grabs articles from websites. The grabbed articles are stored in a **buffer**, from which the classifier processes articles and classifies them.

Considering the complexity of modern websites, it usually takes a long time for a crawler to locate and grab an article from the web page. So, the speed of crawlers is usually too slow for the classifier. Thus, multiple crawlers would be a better choice.

3 Components and Requirements

You are required to design and implement **three crawlers**, **a buffer** and **a classifier in C/C++ on Linux** (other languages are **not** allowed). Mutual exclusion and synchronization **must** be done with mutex and semaphore provided in libraries <pthread.h> and <semaphore.h>.

3.1 ~~crawler~~

Each crawler **thread** is created to ~~grab articles from websites~~ and ~~load them into the buffer.~~ It keeps doing grabbing and loading job, which ~~takes time~~ ~~*interval_A*~~, until the buffer is full. And then it ~~starts waiting~~ until the classifier deletes an article from the buffer.

A function `char* str_generator(void)`, is provided to generate articles for the crawler to grab and each article is represented by a string of 50 characters.

3.2 ~~buffer~~

The buffer **structure** is a ~~first-in-first-out (FIFO) queue.~~ It is used to store the grabbed articles from crawlers temporarily, until they are taken by the classifier. It can store up to ~~12 articles~~ at the same time. You need to implement your own queue. You are **not** allowed to use standard c++ library (e.g., queue or other container provided by standard template library) or third-party libraries.

3.3 classifier

A classifier **thread** is created to classify the articles grabbed by the crawlers in FIFO order. Specifically, there are two steps in the procedure:

1. **Pre-processing**: the classifier ~~makes a copy~~ of the article at the head of the buffer, changes ~~all the uppercase letter ('A'-'Z') to lowercase letter ('a'-'z')~~ and ~~deletes any symbol that is not a letter~~.

2. **Classification**: the classifier classifies the article into one of the 13 classes based on the first letter, ***x***, of the processed article as follows.

~~Class label = int(***x*** – 'a')%13 + 1~~

Next, an ~~auto-increasing key starting from 1~~ will be given to the classified article. (So, the keys of classified articles are 1, 2, 3, …). At last, ~~**the key, the class label** and the **original article**~~, are stored to the ~~**text corpus**~~ in a ~~text file~~. Then, the classifier ~~deletes the classified article in the buffer~~. The whole procedure takes time of ~~*interval_B*~~.

3.4 termination

~~The articles are divided into 13 classes. Denote the number of articles in each class~~ as C_1, C_2, … C_{13}, and $p = min\{C_1, C_2, \ldots C_{13}\}$. When ~~$p \geq 5$, the classifier notifies all crawlers to quit~~ **after** finishing the current job at hand, and ~~then the program terminates~~.

3.5 input arguments

Your program has to accept the following two arguments in input order:

~~*interval_A*, *interval_B*:~~ integer, unit: microsecond.

3.6 sample outputs

The outputs of your program are:

- A table with multiple columns shown on the screen, ~~each column shows the activities of a single thread in time order~~, and ~~each row shows only one single activity of a thread.~~
- The text corpus, each line consists of ~~a key, a class label and an article separated by a space~~.

All activities that need to be recorded for each thread are listed below, together with their abbreviations.

Crawler:

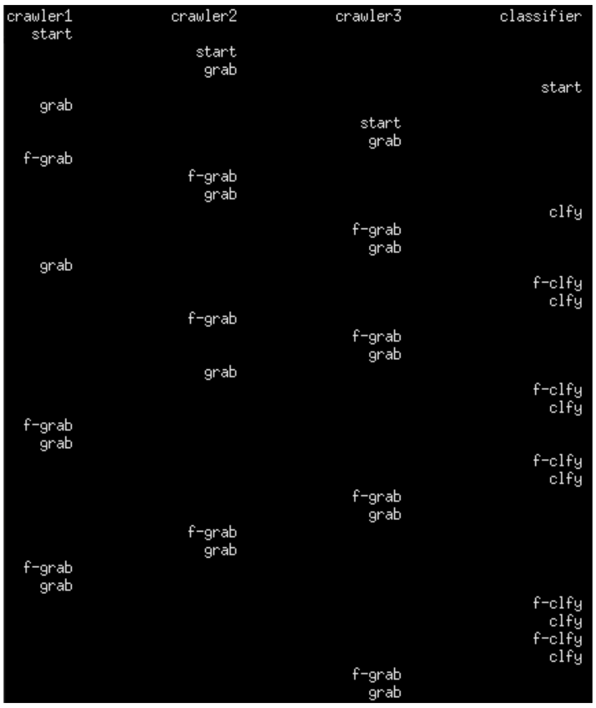
~~*start* – crawler starts.~~
~~*grab* – crawler starts to grab an article.~~
~~*f-grab* – an article has been grabbed and loaded into the buffer.~~
~~*wait* – crawler starts waiting for available space in the buffer.~~
~~*s-wait* – crawler stops waiting.~~
~~*quit* – crawler finished all job and about to quit.~~

Classifier:

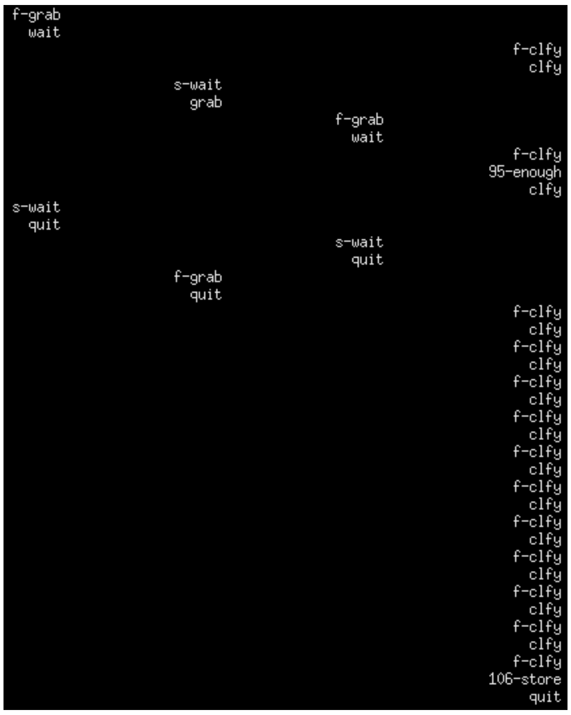
~~*start* – classifier starts.~~
~~*clfy* – classifier starts to classify an article.~~
~~*f-clfy* – the article has been classified and deleted from the buffer.~~
~~*k-enough* – *k* number of articles have been classified and the classifier notifies all threads to quit.~~

~~*n-stored* – a total *n* articles have been stored in the text corpus.~~
~~*quit* – classifier finished all job and about to quit.~~

Below are sample output of the table on the screen and the text corpus. For example, in the table, crawler1 started at t1, then, crawler2 started at t2 and grabbed at t3, and so on.

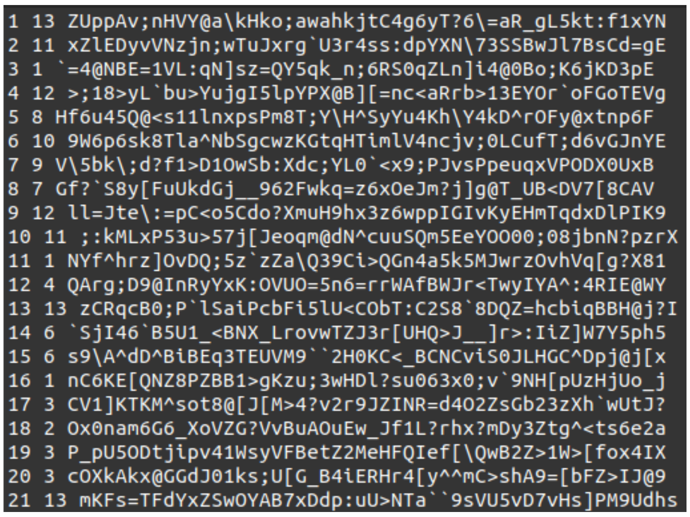


```
crawler1            crawler2            crawler3            classifier
   start
                       start
                        grab
                                                                 start
    grab
                                           start
                                            grab
  f-grab
                      f-grab
                        grab
                                                                  clfy
                                          f-grab
                                            grab
    grab
                                                                f-clfy
                                                                  clfy
                      f-grab
                                          f-grab
                                            grab
                        grab
                                                                f-clfy
                                                                  clfy
  f-grab
    grab
                                                                f-clfy
                                                                  clfy
                                          f-grab
                                            grab
                      f-grab
                        grab
  f-grab
    grab
                                                                f-clfy
                                                                  clfy
                                                                f-clfy
                                                                  clfy
                                          f-grab
                                            grab
```



```
  f-grab
    wait
                                                                f-clfy
                                                                  clfy
                      s-wait
                        grab
                                          f-grab
                                            wait
                                                                f-clfy
                                                             95-enough
                                                                  clfy
  s-wait
    quit
                                          s-wait
                                            quit
                      f-grab
                        quit
                                                                f-clfy
                                                                  clfy
                                                                f-clfy
                                                                  clfy
                                                                f-clfy
                                                                  clfy
                                                                f-clfy
                                                                  clfy
                                                                f-clfy
                                                                  clfy
                                                                f-clfy
                                                                  clfy
                                                                f-clfy
                                                                  clfy
                                                                f-clfy
                                                                  clfy
                                                                f-clfy
                                                                  clfy
                                                                f-clfy
                                                                  clfy
                                                                f-clfy
                                                             106-store
                                                                  quit
```

Beginning of the table   End of the table



```
1 13 ZUppAv;nHVY@a\kHko;awahkjtC4g6yT?6\=aR_gL5kt:f1xYN
2 11 xZlEDyvVNzjn;wTuJxrg`U3r4ss:dpYXN\73SSBwJl7BsCd=gE
3 1 `=4@NBE=1VL:qN]sz=QY5qk_n;6RS0qZLn]i4@0Bo;K6jKD3pE
4 12 >;18>yL`bu>YujgI5lpYPX@B][=nc<aRrb>13EYOr`oFGoTEVg
5 8 Hf6u45Q@<s11lnxpsPm8T;Y\H^SyYu4Kh\Y4kD^rOFy@xtnp6F
6 10 9W6p6sk8Tla^NbSgcwzKGtqHTimlV4ncjv;0LCufT;d6vGJnYE
7 9 V\5bk\;d?f1>D1OwSb:Xdc;YL0`<x9;PJvsPpeuqxVPODX0UxB
8 7 Gf?`S8y[FuUkdGj__962Fwkq=z6xOeJm?j]g@T_UB<DV7[8CAV
9 12 ll=Jte\:=pC<o5Cdo?XmuH9hx3z6wppIGIvKyEHmTqdxDlPIK9
10 11 ;:kMLxP53u>57j[Jeoqm@dN^cuuSQm5EeYOO00;08jbnN?pzrX
11 1 NYf^hrz]OvDQ;5z`zZa\Q39Ci>QGn4a5k5MJwrzOvhVq[g?X81
12 4 QArg;D9@InRyYxK:OVUO=5n6=rrWAfBWJr<TwyIYA^:4RIE@WY
13 13 zCRqcB0;P`lSaiPcbFi5lU<CObT:C2S8`8DQZ=hcbiqBBH@j?I
14 6 `SjI46`B5U1_<BNX_LrovwTZJ3r[UHQ>J__]r>:IiZ]W7Y5ph5
15 6 s9\A^dD^BiBEq3TEUVM9``2H0KC<_BCNCviS0JLHGC^Dpj@j[x
16 1 nC6KE[QNZ8PZBB1>gKzu;3wHDl?su063x0;v`9NH[pUzHjUo_j
17 3 CV1]KTKM^sot8@[J[M>4?v2r9JZINR=d4O2ZsGb23zXh`wUtJ?
18 2 Ox0nam6G6_XoVZG?VvBuAOuEw_Jf1L?rhx?mDy3Ztg^<ts6e2a
19 3 P_pU5ODtjipv41WsyVFBetZ2MeHFQIef[\QwB2Z>1W>[fox4IX
20 3 cOXkAkx@GGdJ01ks;U[G_B4iERHr4[y^^mC>shA9=[bFZ>IJ@9
21 13 mKFs=TFdYxZSwOYAB7xDdp:uU>NTa``9sVU5vD7vHs]PM9Udhs
```

text corpus

4 Challenge

This challenge is for those students wish to get an A+ grade in this programming assignment and to take one more step to the real-world application.

Most modern websites are under anti-crawler protection. Thus, crawlers should be updated with new IP addresses and cookies periodically to get through the barrier.

A strategy manager **thread** is created to update the crawlers with a new IP and cookies. Each crawler notifies the strategy manager to update its IP and cookies after every *M* articles are grabbed. The update takes time of *interval_C*. The input and extra output are listed below.

Your program has to accept the following arguments in input order:

interval_A, *interval_B*, *interval_C*: integer, unit: microsecond, *M*: integer.

Crawler: two more activities have to be recorded:
rest – crawler starts resting.
s-rest – crawler stops resting.

Strategy-Manager:
start – manager starts.
get-crx – manager gets a notification from crawler *x*.
up-crx – manager updated crawler *x* with new IP and cookies.
quit – manager finished all job and about to quit.

5 Helper Program and Hint

5.1 generator.cpp

The function `char* str_generator(void)` is provided in the file `generator.cpp`. It returns a string (char array) of length 50. Use it by declaring a prototype in your code and compiling it along with your source code.

5.2 hint

Multi-threading needs careful manipulation. A specious program may show correctness in several tests at the beginning, but collapses at the later tests. Thus, testing your program multiple times would be a good choice. Testing it with different arguments would be even better.

6 Marking Scheme

Your program will be tested on our CSLab Linux servers (cs3103-01, cs3103-02, cs3103-03). You should describe clearly how to compile and run your program as comments in your source program file. **If an executable file cannot be generated and running successfully on our Linux servers, it will be considered as unsuccessful.**

A. Design and use of multi-threading (15%)
- Thread-safe multithreaded design and correct use of thread-management functions
- Non-multithreaded implementation (0%)

B. Design and use of mutexes (15%)
- Complete, correct and non-excessive use of mutexes
- Useless/unnecessary use of mutexes (0%)

C. Design and use of semaphores (30%)
- Complete, correct and non-excessive use of semaphores
- Useless / unnecessary use of semaphores (0%)

D. Degree of concurrency (15%)
- A design with higher concurrency is preferable to one with lower concurrency.
  - An example of lower concurrency: only one thread can access the buffer at a time.
  - An example of higher concurrency: various threads can access the buffer

    but works on different articles at a time.
- No concurrency (0%)

E. Program correctness (15%)
- Complete and correct implementation of other features including:
  - correct logic and coding of thread functions
  - correct coding of queue and related operations
  - passing parameters to the program on the command line
  - program output conform to the format of the sample output
  - successful program termination
- Fail to pass the g++ complier on our Linux servers to generate a runnable executable file (0%)

F. Programming style and documentation (10%)
- Good programming style
- Clear comments in the program to describe the design and logic
- Unreadable program without any comment (0%)

7 Submission

- This assignment is to be done individually or by a group of two students. You are encouraged to discuss the high-level design of your solution with your classmates but you **must implement the program on your own**. Academic dishonesty such as copying another student's work or allowing another student to copy your work, is regarded as a serious academic offence.
- Each submission consists of **two** files: a source program file (.cpp file) and a text file (.txt file) containing the table outputted by your program and the text corpus.
- Write down your name(s), eid(s), student ID(s), the command line to compile and run your program in the beginning of your program as comments.
- Use your student ID(s) to name your submitted files, such as 5xxxxxxx.cpp and 5xxxxxxx.txt for individual submission, or 5xxxxxxx_5yyyyyyy.cpp and 5xxxxxxx_5yyyyyyy.txt for group submission. You may ignore the version number appended by Canvas to your files. Only **one** submission is required for each group.
- Submit the files to Canvas. As far as you follow the above submission procedure, there is no need to add comment to repeat your information in Canvas.
- The deadline is **11:00am, 11-MAR-2021** (Thu). No late submission will be accepted.

8 Questions?

- This is not a programming course. You are encouraged to debug the program on your own first.
- If you have any question, please submit your question to Mr Wu Wei via the Discussion board "Programming Assignment #2" on Canvas.
- To avoid possible plagiarism, do not post your source code on the Discussion board.
- If necessary, you may also contact Mr Wu Wei at weiwu56-c@my.cityu.edu.hk.