Computing Lab – I

Assignment 6

Multi-Threaded Programming and Thread Synchronization

05 October, 2017

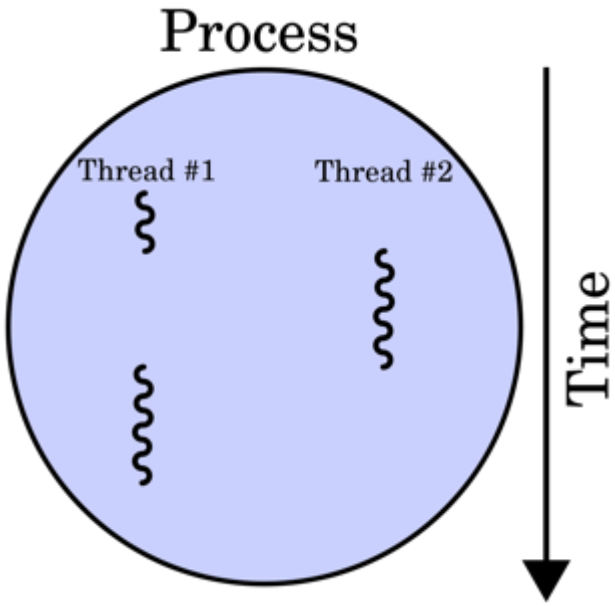
Objective of this Assignment

- To understand multi-threaded programming using POSIX threads or Pthreads.
- To understand thread synchronization using semaphores.
- To understand the difference between processes and threads.
- This is an group assignment.

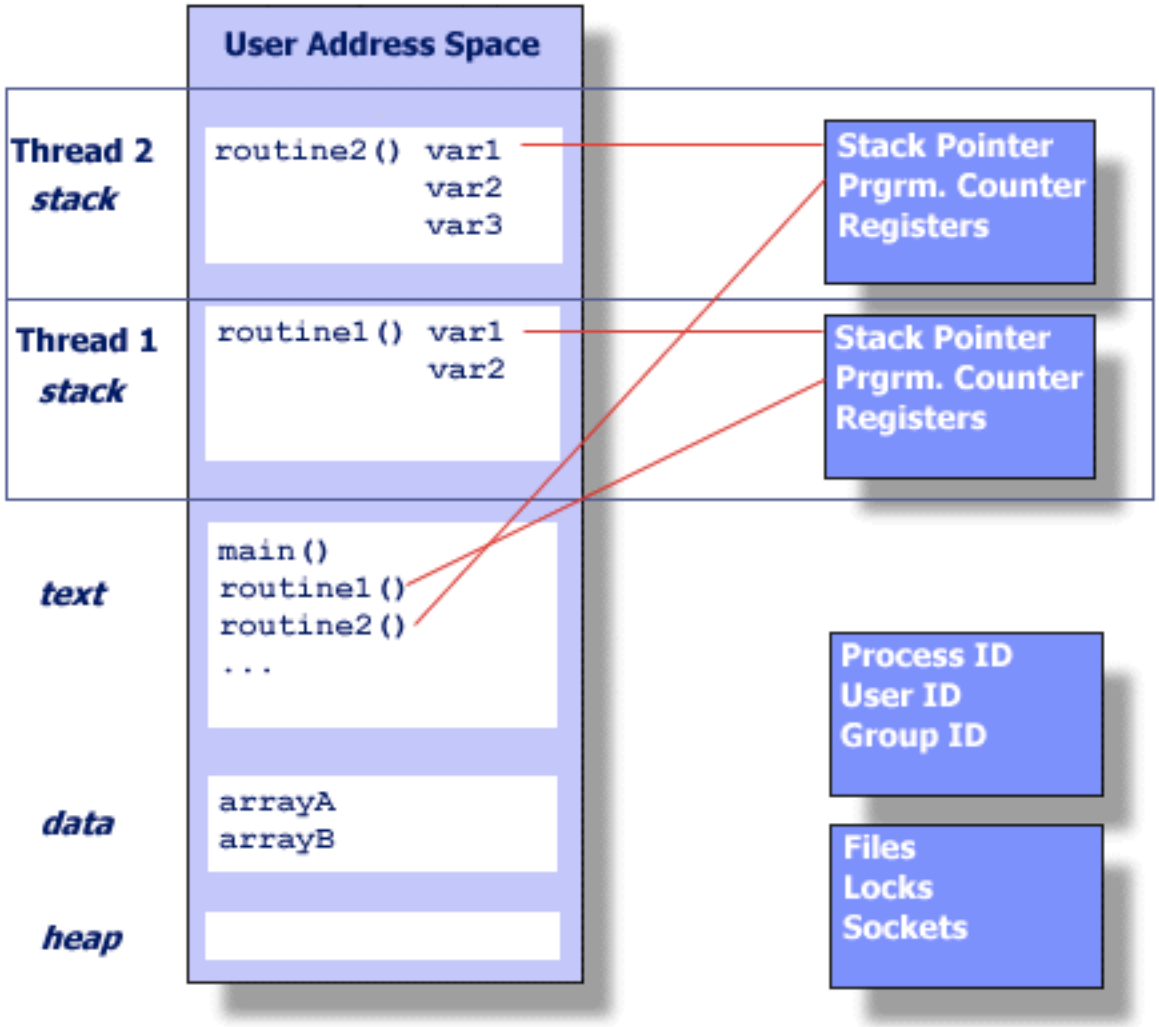
Multi-Threaded Programming

- In shared memory multiprocessor architectures, threads can be used to implement parallelism (an alternate of process for implementing parallelism).

- A **thread** of execution is the smallest sequence of programmed instructions that can be managed independently by a scheduler, which is typically a part of the operating system.



Process and Threads



Process and Threads

- A thread maintains its own
  - Stack pointer
  - Registers
  - Scheduling properties (such as policy or priority)
  - Set of pending and blocked signals
  - Thread specific data

Process and Threads

- Because threads within the same process share resources
  - Changes made by one thread to shared system resources (such as closing a file) will be seen by all other threads.
  - Two pointers having the same value point to the same data.
  - Reading and writing to the same memory locations is possible, and therefore requires explicit synchronization by the programmer.

Pthread Example

```
void *inc_x(void *x_void_ptr)
{
    /* increment x to 100 */
    int *x_ptr = (int *)x_void_ptr;
    while(++(*x_ptr) < 100);
    printf("x increment finished\n");
      /* the function must return
something - NULL will do */
    return NULL;
}
```

```
int main()
{
      int x = 0, y = 0;
      /* show the initial values of x and y */
      printf("x: %d, y: %d\n", x, y);
      /* this variable is our reference to the second thread */
      pthread_t inc_x_thread;
      /* create a second thread which executes inc_x(&x) */
      if(pthread_create(&inc_x_thread, NULL, inc_x, &x)) {
          fprintf(stderr, "Error creating thread\n");
          return 1;
       }
       /* increment y to 100 in the first thread */
       while(++y < 100);
        printf("y increment finished\n");
      /* wait for the second thread to finish */
      if(pthread_join(inc_x_thread, NULL)) {
            fprintf(stderr, "Error joining thread\n");
            return 2;
      }
       /* show the results - x is now 100 thanks to the second
thread */
      printf("x: %d, y: %d\n", x, y);
       return 0;
}
```

Input to the System

- We want to build up a system for getting summarized information from Twitter based on the personal interests of the user.

- Assume that you have **multiple threads** that can crawl Twitter to collect different tweets and store them in a database (say, in a file)
  - You don't need to implement the actual crawler. You can assume that you already have multiple files having multiple tweets. The crawler processes read those files and put the tweets to a single database file at random times.

Database (File) Format

#PMOIndia Violence in the name of faith is unacceptable.
Deep Learning can tell a gentleman by his shoes **#DeepLearning #TensorFlow #MachineLearning #ArtificialIntelligence**
India's stature in the world is rising. The world is with us in fighting the menace of terror. I thank all nations helping us doing so: PM **#PMOIndia**
Like <3 or ReTweet this tweet & we will inform you when your fav Nokia 6 is launching on Amazon.in **#UniteFor #Fun #AmazonExclusive**
Microsoft builds autonomous **#AI**-controlled sailplane—able to identify, predict & act on multiple, complex variables http://msft.social/z6dcnz
Warm greetings & good wishes to all my fellow citizens, especially my Parsi brothers & sisters, on the Parsi New Year **#PresidentKovind**
Govt to conduct assessment of learning outcomes on Nov 13 **#QualityEducation #NCERT #Transformingeducation**

- The tweets are associated with some hashtags.

User Input and System Output

- Every user connects to the system, and gives a set of hashtags as an input to the system.
- The **summarizer thread** corresponding to every user finds out the tweets related to every hashtag.
- The summarizer thread returns a summarization of those tweets corresponding to every hashtag, and writes this information in a **single output file** for all the users.

User Input and System Output

- You have already implemented this problem using process synchronization (Assignment 5). Now compare the running statistics of these two programs:
  - Process Synchronization as done in Assignment 5
  - Thread Synchronization as done in Assignment 6

- Compare the two programs in terms of
  - Total time of execution for same input
  - Total memory usage for same input

User Input and System Output

- Change the input as follows, by keeping all other parameters same for both the program, and find out execution time and memory usage for the following cases:
  - Number of crawler process/threads – vary between 2,3 and 4 (keep user and hashtags same)
  - 2 user, 2 hashtags per user; 3 user, 2 hashtags per user (number of crawler process/threads fixed at 3)

Sample Output

>Enter User:
user123
>Enter hashtags (one at every line, put STOP to stop):
#PMOIndia
#AI
STOP
>Enter User:
User456
>Enter hashtags (one at every line, put STOP to stop):
#MachineLearning
#AI
#EducationForAll

Summary of hashtags for user123 is written at file summary_all.txt
Summary of hashtags for user456 is written at file summary_all.txt

Sample Output

User123 **(Current timestamp: 24/08/2017 11:50 AM)**
#PMOIndia
<summary of tweets>
#AI
<summary of tweets>
==========
User456 **(Current timestamp: 24/08/2017 11:53 AM)**
#MachineLearning
<summary of tweets>
#PMOIndia
<summary of tweets>
#EducationForAll
<summary of tweets>
==========

Note that summary for different users for the same hashtag may be different

Sample Output

- Compute the program running statistics, and prepare a report mentioning the data obtained.

- You can compute the memory usage using the Unix tool valgrind.

Thread Flow

- The **crawler threads** crawl (read) the tweet files at random time and write the data in a database file tweetdata.txt. **The number of crawler threads will be an input from the user.**

- The summarization process (*this is the single process that creates the user specific threads*) keeps on listening for the inputs (user name and hashtags) from the users.

- Once it gets the input from the user, it creates a **summarizer thread** to handle the summarization.

- The user specific summarizer thread filters our relevant tweets, summarize them, and generates the output file.

Submission Instructions

- The code needs to be written in C or C++ by using POSIX thread (Pthread) library. You can use the solution for Assignment 3 for summarization. Submit the C/C++ source **along with the report in a PDF file**, in a single compressed file. The file must contain the name and roll number of the student.

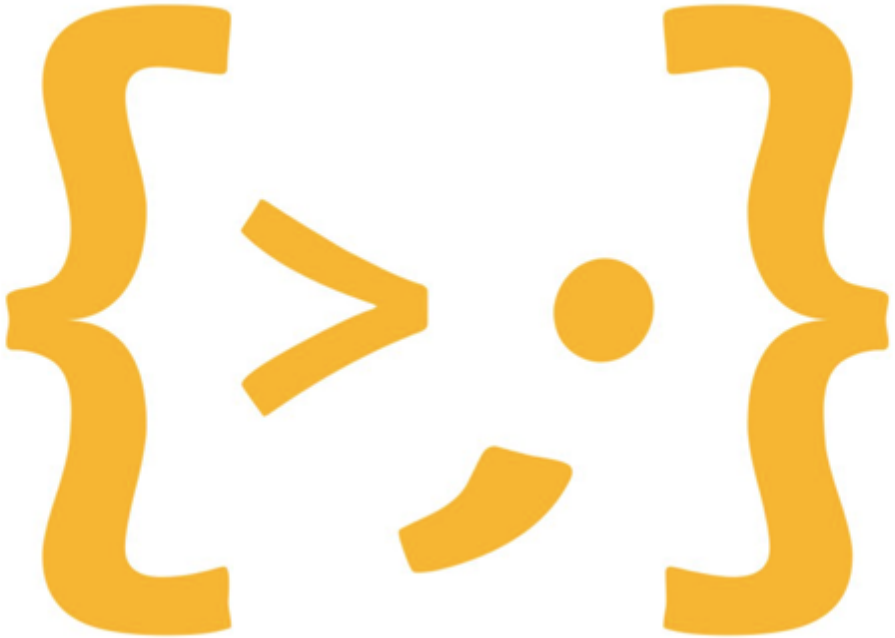
- **Submission Deadline – October 12, 2:00 PM IST**

Marking scheme

- Crawler thread synchronization: 20%
- Reader writer synchronization between crawler thread and the summarization thread: 20%
- Handling synchronization at user specific summarizer threads during output production: 20%
- Report containing comparison between process synchronization and thread synchronization: 30%
- Documentation, coding, understandability: 10%

Some Useful Links

- https://computing.llnl.gov/tutorials/pthreads/#Pthread
- https://linux.die.net/man/7/sem_overview
- http://www.csc.villanova.edu/~mdamian/threads/posixsem.html
- http://www.minek.com/files/unix_examples/semab.html



HAPPY

CODING