CSE 430: Operating Systems

Instructor: Dr. Violet R. Syrotiuk

An Online Social Networking Service: Tweeter

Design and Analysis Report

Project 2: Milestone

Author: Ankit Rathi

ASU ID: 1207543476

Contents

Pseudo Code for User Thread

```
/**
 * Function called by User Threads.
 */
void *thread_users_fn(void *data)
{
        while(1)
        {
                //Wait for Streamer thread to signal the user thread
                sem_wait(&sem_streamer_to_user[iThreadID]);

                //Read a line from the file useri.txt
                ch = getline(&line, &len, fp);

                //Check if the line Read is a Handle Command
                if(fnLineContains(line, "Handle @") == 1)
                {
                        //Update the state info of the thread
                        gStateOfUserThreads[iThreadID] = STATE1;
                        //Signal the Streamer thread that state has been updated
                        sem_post(&sem_user_to_streamer[iThreadID]);
                }
                //Check if the line Read is a Start #tag Command
                else if(fnLineContains(line, "Start #") == 1)
                {
                        //Signal the Streamer thread that Start# has been found
                        sem_post(&sem_user_to_streamer[iThreadID]);
                        //Wait for Signal from Streamer thread to User Thread
                        sem_wait(&sem_streamer_to_user[iThreadID]);

                        //Loop till End # tag is encountered
                        while(1)
                        {
                                //Read next line
                                ch = getline(&line, &len, fp);
                                //If next line is a End #tag line, break the loop
                                if(fnLineContains(line, "End #") == 1)
                                {
                                        break;
                                }
                                //Copy the line read into the inputBuffer
                                for(iLoopIndex = inputBufferLength[iThreadID]; iLoopIndex <
(inputBufferLength[iThreadID] + len) ; iLoopIndex++)
                                {
                                        inputBuffer[iThreadID][iLoopIndex] = *(line+iLoopIndex-
inputBufferLength[iThreadID]);
                                }
                                //Signal the Streamer thread that one line has been added to buffer
                                sem_post(&sem_user_to_streamer[iThreadID]);

                                //Wait for Signal from Streamer Thread to read next line
                                sem_wait(&sem_streamer_to_user[iThreadID]);
                        }
                        //Update the state info of the thread to indicate the Tweet has ended
                        gStateOfUserThreads[iThreadID] = STATE2;
                        //Signal the Streamer Thread that entore tweet has been added into the
inputBuffer
                        sem_post(&sem_user_to_streamer[iThreadID]);

                        //Wait for the Signal from Streamer Thread for successful addition of tweet into
the repository
                        sem_wait(&sem_streamer_to_user[iThreadID]);

                        //Signal the Streamer Thread to continue with other user thread
                        sem_post(&sem_user_to_streamer[iThreadID]);
                }
                //Check if the line Read is a Follow #tag Command
                else if(fnLineContains(line, "Follow #") == 1)
                {
                        //Update the state info of the thread to indicate Follow # tag has been
                encountered
                        gStateOfUserThreads[iThreadID] = STATE3;
                        //Signal the Streamer Thread about the state update of Follow # tag
                        sem_post(&sem_user_to_streamer[iThreadID]);

                        //Wait for Signal from Streamer Thread to take the follow tweets from the
outputBuffer and display it to the user
                        sem_wait(&sem_streamer_to_user[iThreadID]);
                        //Signal the Streamer Thread to continue with next user
                        sem_post(&sem_user_to_streamer[iThreadID]);
                }
                //Check if the line Read is a Read Command
                else if(fnLineContains(line, "Read") == 1)
                {
                        //Update the state info of the thread to indicate Read command
                        gStateOfUserThreads[iThreadID] = STATE4;

                        //Signal the Streamer Thread to continue with next user
                        sem_post(&sem_user_to_streamer[iThreadID]);

                        //Here the Thread can go to sleep, indicating the Read Operation delay, but it
has already signaled the other users to continue.
```

```
        }
        //Check if the line Read is a Exit Command
        else if(fnLineContains(line, "Exit") == 1)
        {
            //Update the state info of the thread to indicate Exit command
            gStateOfUserThreads[iThreadID] = STATE5;

            //Signal the Streamer Thread to continue with next user and terminate this
thread.
            sem_post(&sem_user_to_streamer[iThreadID]);
            pthread_exit(0);
        }
        sleep(1);
    }
}
```

Pseudo Code for Streamer Thread

```
/**
 * Function called by Streamer threads to send/receive data.
 */
void *thread_streamer_fn(void *data)
{
    //Signal the first User Thread to get started
    sem_post(&sem_streamer_to_user[i]);

    while(1)
    {
        //wait for state update from the User Thread
        sem_wait(&sem_user_to_streamer[i]);
        //If there is no state update
        if(gStateOfUserThreads[i] == STATE0)
        {
            //Streamer: Tweeter No Work for you
        }
        //If state update indicates a Handle update
        else if(gStateOfUserThreads[i] == STATE1)
        {
            //Streamer: Tweeter No Work for you
        }
        //When the user is tweeting
        else if(gStateOfUserThreads[i] == STATE2)
        {
            //Signal the Tweeter Thread to add Tweet to Repository
            sem_post(&sem_streamer_to_tweeter);
            //printf("Streamer: Signal Sent: From=Streamer To=Tweeter: Add Tweet to
Repository\n");
```

```
            //Wait for Signal from Tweeter Thread for Tweeted added to Repository
            sem_wait(&sem_tweeter_to_streamer);

            //Signal the User that the Tweet has been added successfully
            sem_post(&sem_streamer_to_user[i]);

            //Wait for Signal from the User Thread to continue
            sem_wait(&sem_user_to_streamer[i]);
        }
        //When user wants to follow
        else if(gStateOfUserThreads[i] == STATE3)
        {
            //Tweeter Work for you: Follow
            //Signal the Tweeter that it needs to follow the topic
            sem_post(&sem_streamer_to_tweeter);

            //Wait for Signal from Tweeter for update to outputBuffer
            sem_wait(&sem_tweeter_to_streamer);

            //Signal the User Thread that the outputBuffer has the follow tweets and can be
consumed.

            sem_post(&sem_streamer_to_user[i]);

            //Wait for the Signal from the User Thread after consumption
            sem_wait(&sem_user_to_streamer[i]);
        }
        //If the state update is an Read Operation
        else if(gStateOfUserThreads[i] == STATE4)
        {
            //Tweeter No Work for you.
        }
        else if(gStateOfUserThreads[i] == STATE5)
        {

            //Tweeter No Work for you.
            /*
            * Check if all user threads have set state to exit,
            * if yes, then exit the streamer thread.
            * */
        }
        while(1)
        {
            i++;
            i = i % iNoOfThreads;
            if(gStateOfUserThreads[i] == STATE5)
            {
                continue;
            }
```

```
                iCurrentThreadInProgress = i;
                gStateOfUserThreads[i] = STATE0;
                //Signal the next user to continue its work
                sem_post(&sem_streamer_to_user[i]);
                break;
            }
            sleep(1);
        }
}
```

Pseudo Code for Tweeter Thread

```
/**
 * Function called by tweeter threads to send/receive data.
 */
void *thread_tweeter_fn(void *data)
{
    int *tparams = (int*)data;

    while(1)
    {
        //Wait for Signal from the Streamer Thread
        sem_wait(&sem_streamer_to_tweeter);

        //If request is to add to repository
        if(gStateOfUserThreads[iCurrentThreadInProgress] == STATE2)
        {
            //Add Tweet to the Streamer

            //Signal the Streamer Tweet added successfully to repository
            sem_post(&sem_tweeter_to_streamer);
        }
        //If request is to follow a tag from repository
        else if(gStateOfUserThreads[iCurrentThreadInProgress] == STATE3)
        {
            // Searching the topic in repository
            //Put the related tweet into the outputBuffer

            //Signal the Streamer, that outputBuffer has been loaded with the Tweets
            sem_post(&sem_tweeter_to_streamer);
            }
        else
        {
            /*
            * Check if all user threads have set state to exit,
            * if yes, then exit the streamer thread.
            * */
            int j = 0;
```

```
                        int count = 0;
                        for(j=0;j<iNoOfThreads;j++)
                        {
                                    if(gStateOfUserThreads[j] == STATE5)
                                    {
                                                count++;
                                    }
                        }
                        if(count == iNoOfThreads)
                        {
                                    //Terminate the Tweeter Thread and exit
                                    printf("Tweeter: Exit: pthread_exit\n");
                                    pthread_exit(0);
                                    return NULL;
                        }
            }
            sleep(1);
      }
}
```

Semaphore and Its Usage

Semaphore	Signaled From	Signaled To/Waited By	Initial Value	Purpose
sem_user_to_streamer(i)	User Thread	Streamer Thread	0	These semaphores are used to signal from user thread to Streamer Thread. They are used to update the state of reading each line by the user thread to the Streamer Thread.
sem_streamer_to_user(i)	Streamer Thread	User Thread	0	These semaphores are used to signal from Streamer thread to User Thread. They are used to send signal to user thread to read a line. Its main purpose is to schedule the user threads to read from input files in a round robin manner.
sem_streamer_to_tweeter	Streamer Thread	Tweeter Thread	0	This semaphore is used to send signal Streamer Thread to the Tweeter Thread. It is basically used to signal the

				Tweeter thread about the tweet and follow requests from the user.
sem_tweeter_to_streamer	Tweeter Thread	Streamer Thread	0	This semaphore is used to send signals from Tweeter Threads to the User Threads. It is basically used to signal the Streamer Thread that the operation that was assigned to Tweeter Thread has been completed and now you can ahead with the flow you want to.

Semaphore Initialized Values

Each semaphore that I have used has been initialized to 0. Here the main purpose of the semaphore to send signals from one thread to the other threads. They are not used to control the number of users trying to access the same resource but just to indicate that flow of control from one thread to the another thread. Hence it is better to initialize them to 0 as their initial values.

Maximizing concurrency in the Design

In the design that has been used for implementation here, is used to signal the users to take turns and process one line at a time. Normally if we would not have used such a type of implementation, then if one thread was accessing data from a useri.txt which could have been possibly large, then in such case only thread would consume a lot of time and not allowing other threads to run in parallel. By the approach designed here, we are allowing only line to be processes at one time. Hence all the threads are getting equal chance of getting their respective commands from the user file and allowing it to get executed.

One more way, in which I have tried to achieve higher concurrency is by allowing the thread to be skipped from time allocation if they have reached its EXIT point. Hence by allowing this, if one thread has reached it end point, then that thread is not scheduled as we know that scheduling would lead to just waste of cpu cycles as it has nothing to do. So Only those thread that are currently are in need of the cpu cycle allocation will be allowed to run and those completed will not be scheduled. Hence this approach has also helped to increase concurrency in the program developed.