CSE 430: Operating Systems

Instructor: Dr. Violet R. Syrotiuk

An Online Social Networking Service: Tweeter

Design and Analysis Report

Project 2: Final Report

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
				strncat(userFileToken[iThreadID].inputBuffer, line, strlen(line)-1);

				//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]);

                while(1)
                {
                        //User Thread Signals the Streamer Thread to start following the tweet
                the user wants to follow.
                        sem_post(&sem_u_to_s_follow);
                        //Wait for the Signal from the Streamer for the output buffer to be
                populated
                        sem_wait(&sem_s_to_u_output);

                        //Display the outputBuffer contents
                }
                //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);

			while(1)
			{
				//Wait for Signal from User to follow a tweet
				sem_wait(&sem_u_to_s_follow);

				//Signal the Tweeter to follow a topic.
				sem_post(&sem_s_to_t_follow);

				//Wait for Signal from the Tweeter that it has added the follow tweet
		into the output buffer
				sem_wait(&sem_t_to_s_output);

				//Signal the User that the follow tweet has been added into the buffer
		and it can pick it up from there.
				sem_post(&sem_s_to_u_output);
			}
			//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)
                {
                        //Streamer: Tweeter No Work for you.
                }
                else if(gStateOfUserThreads[i] == STATE5)
                {

                        //Streamer: 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)
{
        while(1)
        {
                //Wait for Signal from the Streamer Thread for Add or follow command
                sem_wait(&sem_streamer_to_tweeter);

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

```
		//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)
{
		//Wait for Signal from the Streamer Thread to start following a tweet topic
		sem_wait(&sem_s_to_t_follow);

		// Searching the topic in repository
		local_tweet_temp_node = isFollowTagInRepository();
		if(local_tweet_temp_node == NULL)
		{
			//If no related tweets are present then follow complete flag can be set.
			iFollowCompleteFlag = 1;
			//Signal the Streamer that output message is available in buffer.
			sem_post(&sem_t_to_s_output);
		}
		Else
		{
			While(1)
			{
				//Copy the data into output buffer.
				//Signal the Streamer that message has been put into output
			buffer.
			sem_post(&sem_t_to_s_output);
			//Wait for Signal from the Streamer to follow the next tweet from the
		repository.
			sem_wait(&sem_s_to_t_follow);
			}
		}
		//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.
sem_s_to_t_follow	Streamer Thread	Tweeter Thread	0	This semaphore is used inside a while loop when one user is following a particular tweet topic. It is used to maintain synchronization between Streamer Thread and Tweeter Thread during a follow command from a user. Here the Streamer signals the Tweeter to send the next related tweet of the topic being followed by the user.
sem_u_to_s_follow	User Thread	Streamer Thread	0	This semaphore is used inside a while loop when one user is following a particular tweet topic. It is used to maintain synchronization between User Thread and Streamer Thread during a follow command from a user. Here the User signals the Streamer to send the next related tweet of the topic being followed by the user.
sem_t_to_s_output	User Thread	Streamer Thread	0	This semaphore is used inside a while loop when one user is following a particular tweet topic. It is used to maintain synchronization between Tweeter Thread and Streamer Thread during a follow command from a user. Here Tweeter signals the Streamer that

				it has populated the output buffer related to the follow topic.
sem_s_to_u_output	Streamer Thread	User Thread	0	This semaphore is used inside a while loop when one user is following a particular tweet topic. It is used to maintain synchronization between Streamer Thread and User Thread during a follow command from a user. Here the Streamer Signals the User that the output buffer has been populated by the tweeter, you can read the message and display it to the user frontend.

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.

```
/**
 * Global Semaphores for resource sharing between Threads
 */
sem_t sem_user_to_streamer[MAX_NUMBER_OF_USERS];
sem_t sem_streamer_to_user[MAX_NUMBER_OF_USERS];
sem_t sem_streamer_to_tweeter;
sem_t sem_tweeter_to_streamer;
sem_t sem_s_to_t_follow;
sem_t sem_u_to_s_follow;
sem_t sem_t_to_s_output;
sem_t sem_s_to_u_output;
```

```
/*
 * Initialize all the semaphores with initial values of 0, as we are using them for signaling
 */
sem_init(&sem_streamer_to_tweeter, 0, 0);
sem_init(&sem_tweeter_to_streamer, 0, 0);
sem_init(&sem_s_to_t_follow, 0, 0);
```

```
sem_init(&sem_u_to_s_follow, 0, 0);
sem_init(&sem_t_to_s_output, 0, 0);
sem_init(&sem_s_to_u_output, 0, 0);

for(i=0; i<iNoOfThreads; i++)
{
        sem_init(&sem_user_to_streamer[i], 0, 0);
        sem_init(&sem_streamer_to_user[i], 0, 0);
}
```

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.

Methodology to implement pseudo code

As per the pseudo code submitted for milestone, I have created 3 different thread functions, one each for user thread, one for streamer thread and one for tweeter thread. All the user threads make use of the same function. So totally n+2 threads are spawned by the main function.

To read from the files I used getline function which reads one line at a time. After reading the line I parse it to determine which command the user is trying to execute. Based on the type of the command a global variable is set. This global variable is then read by the streamer thread to determine what the operation performed by the user thread. Along with this I made of the semaphores to signal between the user, streamer and tweeter threads. For details regarding the semaphores that have been declared and initialed to values and their function have already been discussed in the previous sections. For start# and end# I used a while loop and I let a particular user read one line and pass control to the next user, but that particular user resumes control from the same point inside the while loop to continue reading the line one by one. After the end tag has been reached I created semaphore to signal the streamer that state has been updated and you can continue to next user after completing the work that I have assigned you. For the follow tag also, similar approach has been used, but in this the user is not switched by the streamer till all the tweets that user wants to follow has been passed to him. Only once he has received all the related tweets, tweeter signals the streamer which in turn signals the user that it has reached the end of the follow tweets from the repository. Only at this point user thread signals the

streamer thread to continue with other users. For inserting the data into repository and retrieving data from the repository I have created two separate functions. These have been explained later in this report.

Errors that arose from the pseudo code

The pseudo code that was supplied for the milestone did not take care of the controlling signals between the user, streamer and tweeter thread for follow tag completely. The control should not be released from one user to the next user, till all the related tweets present in the repository had been sent to user. This synchronization I faced problem and later I solved by introducing semaphores inside the while loop and control the signal between 3 threads till the end of the repository is reached and then finally signaling that all the tweets have been transferred to the user and streamer can now schedule the next user which is ready to run or get executed.
One more issue that I faced was terminating streamer and the tweeter thread after all the user threads have reached their exits in the respective files. For this I created another state in the code and used to count the number of the user threads that have exited. Once the number of the user threads that have exited is equal to the number of user threads that were spawned then it is signal to the streamer to exit and streamer before exiting signals the tweeter that all the user threads have exited and it can exit using the pthread_exit command. So, these were the two main issues that I had faced during the development of the code that was given for milestone submission to the final code submission.

Data Structure to maintain Tweets in Repository.

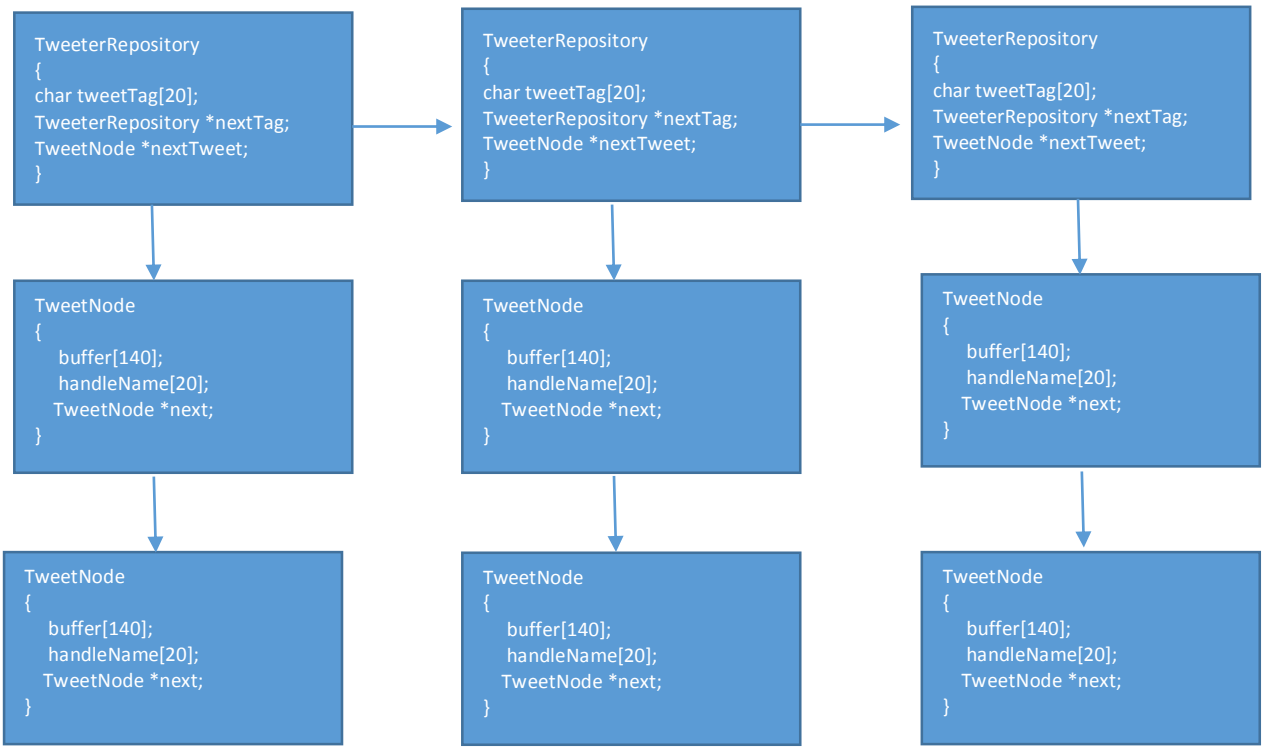
I have defined 2 data structures to maintain tweets in the repository. The first data structure is struct TweeterRepository which is used to maintain the linked list. This Linked List is used to point to a new similar data structure which holds the tweet name at each node. Each of this node points to the next node, which contains next tweet name. Each of this node also has an extra pointer, which points to another data structure which again forms a Linked List of nodes which contains information regarding the handle name and the tweet comments between the start and end tags.

```
/**
 * Tweet Repository Data Structure
 */
struct TweeterRepository
{
        char tweetTag[20];
        struct TweeterRepository *nextTag;
        struct TweetNode *nextTweet;
};
struct TweeterRepository *tr_head_node = NULL, *tr_temp_node = NULL;
/**
 * Tweet Data Structure inside Repository
 */
struct TweetNode
{
        char buffer[140];
        char handleName[20];
        struct TweetNode *next;
};
```

*struct TweetNode *tweet_temp_node = NULL;*

So basically, I have used two data structures, first one is a Linked List which points to the next tweet topic in the repository. Then each of this tweet topic node, points to the next another data structure which contains a pointer which forms another Linked List to hold handle names and the tweets from the users. This is like Linked List has a pointer to another Linked List. The data structure is similar to the data structure used to maintain the HashMap data structure which consists of the array of elements and each element is a bucket, used to store the objects that produce the same hash keys. The reason of me using such a data structure is that it satisfies the project requirements, which says that tweets in the data structure used needs to be categorized by their hash tags.

Data Structure Overview Block Diagram



Code to insert data into repository.

```
/**
 * Function called by Tweeter thread to add the tweets into the repository.
 */
void insert_into_Repository()
{
        struct TweeterRepository *new_tag_node = NULL;
        struct TweetNode *new_tweet_node = (struct TweetNode *)malloc(sizeof(struct TweetNode));
```

```
	//Populate the New Tweet Node
	strcpy(new_tweet_node->buffer, userFileToken[iCurrentThreadInProgress].inputBuffer);
	strcpy(new_tweet_node->handleName, userFileToken[iCurrentThreadInProgress].handleName);
	new_tweet_node->next = NULL;

	//Check if the repository is empty
	if(tr_head_node == NULL)
	{
		new_tag_node = (struct TweeterRepository *)malloc(sizeof(struct TweeterRepository));

		strcpy(new_tag_node->tweetTag, userFileToken[iCurrentThreadInProgress].tweetTag);
		new_tag_node->nextTag = NULL;

		new_tag_node->nextTweet = (struct TweetNode*)new_tweet_node;
		tr_head_node = new_tag_node;
	}
	else
	{
		int iNewTweetNode = 0;
		tr_temp_node = tr_head_node;
		while(1)
		{
			//Check if any of the existing Tweet Tag names matches with the Tweet Tag to
		be inserted.
			if(0 == strcmp(tr_temp_node->tweetTag,
userFileToken[iCurrentThreadInProgress].tweetTag))
			{
				tweet_temp_node = (struct TweetNode *)tr_temp_node->nextTweet;
				while(tweet_temp_node->next != NULL)
				{
					tweet_temp_node = (struct TweetNode *)tweet_temp_node-
>next;
				}
				tweet_temp_node->next = (struct TweetNode*)new_tweet_node;
				iNewTweetNode = 1;
				break;
			}
			if(tr_temp_node->nextTag != NULL)
			{
				tr_temp_node = (struct TweeterRepository*)tr_temp_node->nextTag;
			}
			else
			{
				break;
			}
		}
//Check if any of the existing Tweet Tag names doesn't match with the Tweet Tag to be inserted then
new TweetRepository Node needs to be created.
```

```
                if(iNewTweetNode == 0)
                {
                        new_tag_node = (struct TweeterRepository*)malloc(sizeof(struct
TweeterRepository));

                        strcpy(new_tag_node->tweetTag,
userFileToken[iCurrentThreadInProgress].tweetTag);
                        new_tag_node->nextTag = NULL;

                        new_tag_node->nextTweet = (struct TweetNode*)new_tweet_node;

                        tr_temp_node->nextTag = (struct TweeterRepository*)new_tag_node;
                }
        }
}
```

Code to retrieve data from Repository.

```
/**
 * This function is called by the Tweeter Thread and is used to determine
 * if the follow topic is present in the repository or not.
 * It return null if the follow topic is not present in repository
 * and it return the tweet node pointer if it present in the repository.
 */
struct TweetNode* isFollowTagInRepository()
{
        print_Repository();
        //If the Head Node is null, meaning the repository is still empty.
        if(tr_head_node == NULL)
        {
                char *emptyMessage = "Repository is Empty";
                strcpy(userFileToken[iCurrentThreadInProgress].outputBuffer, emptyMessage);
                return NULL;
        }
        else
        {
                tr_temp_node = tr_head_node;
                //Loop untill the end of the repository is reached.
                while(tr_temp_node->nextTag != NULL)
                {
                        //Check of the follow topic tag matches with the repository.
                        if(0 == strcmp(tr_temp_node->tweetTag,
userFileToken[iCurrentThreadInProgress].followTag))
                        {
                                tweet_temp_node = (struct TweetNode *)tr_temp_node->nextTweet;
                                return tweet_temp_node;
                        }
```

```
                tr_temp_node = (struct TweeterRepository*)tr_temp_node->nextTag;
            }

            char *noTweetMessage = "No Related Tweets in Repository";
            strcpy(userFileToken[iCurrentThreadInProgress].outputBuffer, noTweetMessage);
            return NULL;
        }
}
```