SWEN 563/CMPE 663/EEEE 663
REAL-TIME & EMBEDDED SYSTEMS
ROCHESTER INSTITUTE OF TECHNOLOGY

PROJECT-4: MULTI-THREADED BANK SIMULATION ON QNX

Submitted to: Larry Kiser(llkiee@rit.edu)

Submitted by:

Mokshan Shettigar (msmokshan1@rit.edu)

Nicolas Delanou (nxd7208@rit.edu)

D.O.S: 11/02/2016

Areas of Focus

Writing of the main code:

- Nicolas Delanou: 100%

Writing of the graduate student extension:

- Nicolas Delanou: 100%

Writing of the report:

- Nicolas Delanou: 100%

Analysis / Design

In this project, we used the concept of multi-threading with the pthread library. The project has been developed for QNX material in the QNX Momentics IDE.

In order to continue the development at home, I installed the pthread library for Visual Studio on my Windows laptop.

Test Plan

The Main Function

The main function of this project is pretty simple:

```
int main(int argc, char *argv[]) {

    Thread_Init();  // Creating all the threads

    Thread_Exit();  // Waiting for all the threads to exit

    printfMetrics();
    clean_memory();

    system("pause");
    return EXIT_SUCCESS;
}
```

First, we initialize all the thread we are creating.

Then, we are simply waiting for all the threads to end. This is the end of the simulation.

We then proceed to print the asked metrics.

Finally, we clear the memory allocated for the client's generation.

The Bank thread

First, we created a structure type for the bank called “bank_t”. This structure contains a variable telling if the bank is open or closer. This variable is protected in the code by a mutex. This structure also contains an array of tellers, which is another structure type we created. Finally, the structure holds a variable relative to the thread in charge of the bank management.

```
typedef struct
{
    int isOpen;
    pthread_mutex_t mutex_isOpen;

    teller_t tellers[NB_TELLERS];

    pthread_t thread_bank;
}
bank_t;
```

The “teller_t” type used in the bank structure represented here:

```
typedef struct
{
    int time_working;
    int time_waiting;
    int time_in_break;

    pthread_t thread_teller;
}
teller_t;
```

It contains several variables used for the metrics. Each teller has an associated thread created in the “Thread_Init” function.

The bank structure created in the program is a global variable directly accessible by every function of the main.c file.

The bank thread is really simple. It basically consists of waiting the right time for opening the bank then waiting for the closing and finally exiting thread

The teller’s threads created are a bit more complex. They handle the clients while the bank is open, writing data for the metrics in the teller’s structures and in the client structures explained in the next part. Every client is handled by the teller with a random time varying between 30sec and 6min. The teller thread also handle the fact that tellers have to take a break (1 to 4 minutes) every 30 to 60 minutes.

The Client Queue thread

We had to create a client queue for the program. We then decided to create a structure type for this. This structure contains a variable storing the index of the next client that will added in the queue. This variable is protected by a mutex. There are also other variables used for the metrics.
The structure also contains a semaphore variable. We post a new semaphore every time a new client enter the queue.
This structure contains an array of pointer on clients' structures. This is where all clients are stored. The last variable of this structure stores the thread of the client generator in charge of the clients' creation.

```
typedef struct
{
    int index_next_client;
    pthread_mutex_t mutex_index_next_client;

    int max_client_waiting;
    int nb_client_generated;
    sem_t sem_client_queue;

    struct client_t* queue[QUEUE_SIZE];

    pthread_t thread_client_generator;
}
client_queue_t;
```

The "client_t *" variable used in the "client_queue_t" structure refers to a structure type we have created with this code:

```
typedef struct
{
    int time_entering_queue;
    int time_begin_teller;
    int time_end_teller;
    int index_in_queue;
}
client_t;
```

The main purpose of this structure is to store data for the metrics

The "client_queue_t" structure created in this program is a global variable.

The client_generator thread created simply wait for the bank to open to generate new clients with a random time between every client (from 1min to 4min).

The Time Management thread

A global variable is created storing the time of the day in minutes. A thread is created in order to increment this variable every 100ms. This variable is used across the program in order to store timing, for example, in order to know how long a client waited in the queue.

Project Results

We began the project by creating the different threads. After a successful generation of a random number of clients without any data stored in the structures, we started to measure some timings and stored them into the clients' structures. We then confronted the results we got with the expected results and everything seems to be correct.

Here is a screenshot of the result obtained with the final program:

```
##########################################
Customers served today :        171
Average time in queue :         0.05 minutes
Average time with the teller :  3.30 minutes
Average time waiting customer:  15.00 minutes
Max wait time in queue :        1 minutes
Max wait time for a teller :    9.00 minutes
Max time working for tellers :  5.99 minutes
Max time in break for tellers : 0.00 minutes
Max depth of the queue :        1
##########################################
```

Then we proceed to write the graduate extension.

Here is the result we got with the graduate extension:

```
##########################################
Customers served today :        168
Average time in queue :         0.08 minutes
Average time with the teller :  3.23 minutes
Average time waiting customer:  11.33 minutes
Max wait time in queue :        2 minutes
Max wait time for a teller :    7.00 minutes
Max time working for tellers :  6.00 minutes
Max time in break for tellers : 3.98 minutes
Max depth of the queue :        1
##########################################
```

In conclusion, 100% of the features asked in the subject are working correctly. The graduate extension is also working. We can note in the result that for approximately the same number of client, the average waiting time for a client is higher and the average waiting time for tellers is lower for the pause extension. We can also note that the maximum wait time for a teller is smaller for the extension.

Lessons learned

In this project, we learned how to use the QNX environment in order to realize a real-time application.

We are now able to create real-time multi-threaded programs.