CS307 - Fall 2020-2021
Memory Management API - Phase 1

Date Assigned: 30.11.2020
Due Date Time: 12.12.2020 at 23:55 (sharp, according to server's time)
No Late Submission

1 Introduction

In this project, you are going to implement a memory management API, and we will provide a code for you to test your API. You are expected to create a shared memory, in order to manage memory requests coming from multiple threads and grant or decline requests based on free space available. You are also expected to manage all mutexes and semaphores for shared memory access.

2 Program Flow

The main program, that we will give you, will call various functions in order to initialize, access, change and dump the memory. Also there will be an array of threads, an array of semaphores, a mutex, a memory server thread and a char array representing the memory given.
The init() function that is also provided will initialize the memory array and the request queue and any related data structures. Afterwards it will start the memory server thread.
After initialization, you are expected to start all of threads, which will run in a function called **thread_function**. This function will generate a random size and call the **my_malloc** function, which you will also implement.
Function **my_malloc** will utilize the semaphores and mutex for synchronization of access to shared data structures. The **thread_function** will wait for the answer and -if memory is granted- access the memory and set all the bytes allocated to its **id** value. The size of the memory will be randomly generated by the **thread_function** function.
The **server thread** is expected to handle all requests for memory access. All access requests will be appended to the shared **queue**. The **server thread** will read from this queue and depending on the size and availability will return an answer to the requesting thread.

- If there is enough space it should return the start address of the memory array.
- If there is not enough space it should return -1.

For this phase of the assignment you will only maintain an index of the next available space. You are not expected to manage the memory allocations. The access will be based on the index and the space remaining in the memory array. If the requested size is smaller than (**memory_size - index**), then you should grant access.

The thread which makes the request will be blocked till the memory server processes its request

- If space is allocated: Gains access to the shared memory and set all the bytes allocated to it to character value of its **id**.
- If space is not allocated: Prints an error message “Thread ID: Not enough memory” and exits.

Given Functions and Variables

my_malloc(int thread_id, int size):

This function should be called by a thread with a size value. It will gain access to the shared queue and write the requesting thread_id and the amount.

dump_memory():

This function is used to test memory allocation. It will print the entire contents of the memory array onto the console.

thread_function(int id):

This function should run as a thread. The function should first create a random memory size. The generated number should be between **1** and (**MEMORY_SIZE / 6**).
It will then call the **my_malloc(id, size)** function and block until the memory request is handled by the server thread. Once the thread is unblocked by the server thread it will take appropriate action. The server thread will store the value in a shared array called **thread_message**. The value is -1 indicates there is no available memory, hence only an error message must be printed. If the value is bigger than -1 then it will be interpreted as the starting point in the memory. It will set all the bytes allocated to it to character value of its own id.

server_function():

This thread will run in a loop until all threads are done. It will check the queue for the requests and depending on the memory size it should either grant or decline them. The answers to the requests will be written to the **thread_message** array and the requesting thread should be unblocked. It should return either -1 for declining the request due to no available size, or the start point of the memory location that will be allocated to the requesting thread. When all thread requests are handled, server thread should terminate.

thread_id

This value will be incrementally generated and passed to the thread_function as a parameter starting from 0 (zero), while the threads are being created.

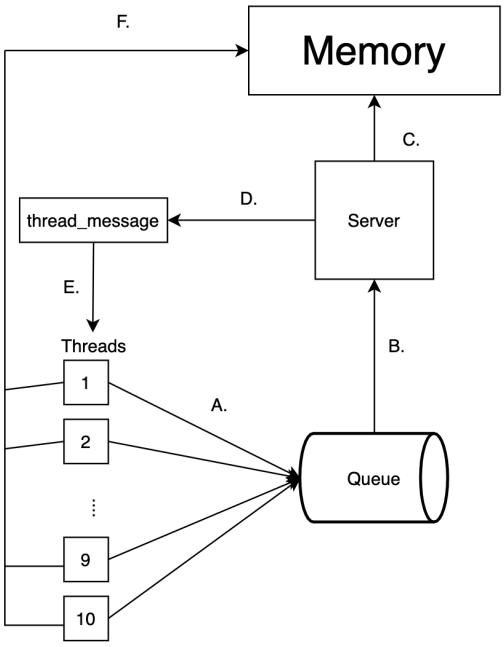

Figure 1: Example Flow of the System

3 Example Pipeline

Figure 1 shows the general structure and steps are explained as :

(A) Threads will generate a struct which has the requested memory size and id. Then struct will be pushed to shared queue. Threads will be blocked on their semaphores.

(B) When there is an item in the queue, server thread will pop the item.

(C) This popped item contains requesting thread and requested memory size. Server will control the memory to check if there is available space.

(D) Depending on the previous memory space check, server thread will update thread_message array and unblock the proper threads semaphore.

(E) Unblocked thread will check thread_message array and depending on the value it read, it will update the memory.

Submission

Please include a report that describes the reasoning and flow of your program. Please briefly explain how your code is working in your report. Your report should be at **MOST** one page long and your report should be in **PDF** format.

You need to use POSIX Threads and test your code on the flow server.
Your **CPP** file should be submitted in a **ZIP** archive, name your zip archive as: ***YourNameSurname_ID_hw3.zip*** and submit to **SUcourse**.

You need to add comments to the your code to make it more understandable. You will lose points if there are problems in your submission, so please write your code on a **CPP** file. After that create a **ZIP** file that contains your report and **CPP** file.

Note that, your system time and SUCourse server's time may not be synchronized so do not wait the last minutes to submit your solution. Only the solutions in the SUCourse system will be graded. Other submissions, such as emailing to instructor or assistants, will not be graded.

Good Luck,
Berkant