159.342 - Operating Systems and Networks

Assignment #3 (due on 11/Jun)

**Memory Management Using the Buddy System Strategy**

**OBJECTIVE:**

The main objective in this assignment is to implement the **Buddy System memory management strategy** by writing two functions that implement the same functionalities as the C Standard library’s **malloc()** and **free()**. You should apply your knowledge of memory management concepts, so the functions are as efficient as possible. The buddy system will be characterised and compared against the existing C Standard library functions based on **ACTUAL RUNNING TIME** and **MEMORY USAGE**.

Given a large block of contiguous memory to manage, your functions should be able to fulfil a series of memory allocation and deallocation requests. In your implementation, you are **not** allowed to use the built-in malloc, free, new, delete or any other memory allocators that come from other third-party libraries.

**START-UP CODES:**

The start-up codes provide a **test routine** that makes successive calls for memory allocations and deallocations, involving a replaceable implementation of **MALLOC** and **FREE**. Functions for measuring the actual running time and total memory usage of the memory management strategy are also provided.

*///////////////////////////////*

*// TEST ROUTINE*

*///////////////////////////////*

#ifdef RUN\_COMPLETE\_TEST

cout << "**\n\n** executing " << NO\_OF\_ITERATIONS << " rounds of combinations of memory allocation and deallocation..." << endl;

**for**(i=0;i<NO\_OF\_ITERATIONS;i++) {

k=myrand() % NO\_OF\_POINTERS; *// pick a pointer*

**if**(n[k]) { *// if it was allocated then free it*

*// check that the stuff we wrote has not changed*

**if** ( (n[k][0]) != (unsigned char) k)*//(n[k]+s[k]+k) )*

printf("Error when checking first byte! in block %d **\n**",k);

**if**(s[k]>1 && (n[k][s[k]-1])!=(unsigned char) k )*//(n[k]-s[k]-k))*

printf("Error when checking last byte! in block %d **\n**",k);

**FREE**(n[k]);

}

size=randomsize(); *// pick a random size*

#ifdef DEBUG\_MODE

cout << "**\t**Pick random size to allocate: " << size << endl;

#endif

n[k]=(unsigned char \*)**MALLOC**(size); *// do the allocation*

**if**(n[k] != NULL){

#ifdef DEBUG\_MODE

cout << "**\t**allocated memory of size: " << size << endl;

#endif

s[k]=size; *// remember the size*

n[k][0]=(unsigned char) k; *// put some data in the first and*

**if**(s[k]>1) n[k][s[k]-1]=(unsigned char) k; *// last byte*

} **else** {

cout << "**\t**Failed to allocate memory of size: " << size << endl;

}

}

#endif

You can switch between different memory management strategies by changing the definition of **MALLOC** and **FREE**, as defined inside **main.cpp** (see below). The first two options are already provided, but the third option corresponds to the **Buddy System** and therefore, needs to be implemented in this assignment:

//---------------------------------------

**// WHICH MEMORY MANAGEMENT STRATEGY?**

//---------------------------------------

// enable the following compiler directives to test the real malloc and free

//**(1) use built-in C functions**

const string strategy = "malloc";

#define MALLOC malloc

#define FREE free

//---------------------------------------

//enable the following compiler directives to test a simple implementation of malloc and free

/**/(2) use user-defined functions**

//const string strategy = "mymalloc";

// #define MALLOC mymalloc

// #define FREE myfree

//---------------------------------------

//enable the following compiler directives to test your implementation of the Buddy system strategy

//**(3) use Buddy System**

// const string strategy = "Buddy System";

// #define USE\_BUDDY\_SYSTEM

// #define MALLOC **buddyMalloc**

// #define FREE **buddyFree**

//---------------------------------------

**CORE FUNCTIONS REQUIRED:**

Skeleton functions for buddy system’s malloc and free are already provided inside **buddysys.cpp**. Write your implementation inside them. You are allowed to add any supporting functions, data structures and variables in the program.

void \***buddyMalloc**(int request\_memory){

//write your buddy system’s memory allocation codes here

//this function should provide the same functionality as the malloc() function

}

int **buddyFree**(void \*p){

//write your buddy system’s memory deallocation codes here

//this function should provide the same functionality as the free() function

}

**AUXILIARY FUNCTIONS PROVIDED:**

The Auxiliary functions, constants and data structures declared and implemented inside auxiliary.cpp and auxiliary.h should all be kept as is.

**MINIMUM CONTIGUOUS MEMORY SIZE**

As part of characterising the performance of your implementation, find the minimum contiguous memory size that allows the **Buddy System strategy** to accommodate the succession of memory allocation and deallocation requests in the **test routine**. To accomplish this, you need to experiment with different settings for the **NUMBEROFPAGES** until the test routine could be fully completed.

**Example:**

As an example, if you set **NUMBEROFPAGES** equal to 1024, then the MEMORYSIZE could be computed as follows:

// inside main.cpp

MEMORYSIZE = (long long int) ((long long int)**NUMBEROFPAGES** \* (long long int)PAGESIZE);

Note that in the start-up codes, the PAGESIZE is set to 4096. Therefore, MEMORYSIZE will be equal to 4,194,304 bytes (1024 \* 4096) or **4.19 MegaBytes**.

Next, with your MEMORYSIZE setting, run the test routines to see if all requests for memory allocations and deallocations can be satisfied. If not, increase the MEMORYSIZE.

**Table 1. MEASURE OF PERFORMANCE**

Characterise your Buddy System’s implementation and compare it against the C Standard library’s malloc and free by filling-up the following table.

|  |  |  |
| --- | --- | --- |
| **BUDDY SYSTEM STRATEGY** | | |
| Characteristic |  | Unit |
| MINIMUM NUMBEROFPAGES | 8192 | *Pages* |
| MINIMUM MEMORY SIZE | 2030 | *MegaBytes* |
| RUNNING TIME | 198195 | *MicroSeconds* |
| **C Standard Library’s malloc and free** | | |
| Characteristic |  | Unit |
| MINIMUM NUMBEROFPAGES | 8192 | *Pages* |
| MINIMUM MEMORY SIZE | 2030 | *MegaBytes* |
| RUNNING TIME | 1554636 | *MicroSeconds* |

#### Other Notes

0 – zip all the files comprising your codes by compressing the entire folder. The structure of the folder and subfolders should be exactly the same as that of the start-up codes given.

1 - Submit your files electronically via Stream. You should submit the complete set of files, required to compile your codes, including the makefile, **Measure of Performance** (**Table 1**) **Checklist** (see **Table 2**) and snapshots of simulation results. Bundle all your files together in a \*.zip file.

2 - This assignment is worth **15 marks**.

3 - Marks will be subtracted for obvious copying and/or for delays without justification.

**Table 2. Checklist/Other technical details**

Please accomplish the following check list in order to allow for accurate marking of your assignment.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Item | your assignment details | | Comments |
| 1 | Names and ID numbers of Group Members |  | | (maximum of 3 members in a group) |
| 2 | Operating System(s) used for testing your codes |  | | Note that your programs should run correctly under Windows 10. |
| 3 | Compiler used | g++ 12.2.0 | | Note that g++ 12.2.0 is required |
| 4 | IDE used | visual studio 2019 | | (e.g. SublimeText 3, ScITE) |
| 5 | **Files**: Have you submitted all the files required to compile your codes and test it? Include the **makefile**, source codes and other support files: /CMakeLists.txt, \*.cpp, \*.h, \*.sln, .vcxproj, etc. | Yes | | Indicate ‘**Yes**’ or ‘**No**’. Make sure that you submit all the supporting files if you developed your codes using CLion or Visual Studio. The **makefile** should suffice to build your codes. |
| 6 | Required Functions | Function name | Indicate implementation level | Indicate ‘**full’**, if you have successfully completed and tested the required functionality, ‘**partial’**, if you are only submitting a partial implementation, or ‘**none’**, if not accomplished. |
| buddyMalloc | full |
| buddyFree | full |
| 7 | Snapshot of simulation results | Yes | | indicate ‘Yes’ or ‘No’, if you have submitted the required snapshot. |
| 8 | Number of iterations successfully run using **RUN\_COMPLETE\_TEST**, USE\_SIMULATION\_2 | All the iterations run successfully. | | Specify what problems occurred if the program was not able to complete the simulation. |
| 9 | If your implementation cannot complete **RUN\_COMPLETE\_TEST**, can it run on **RUN\_SIMPLE\_TEST**? | my implementation can complete  **RUN\_COMPLETE\_TEST** | | Note: This setting can be changed inside **auxiliary.h** |