BLG312E Computer Operating Systems

Spring 2024 - Homework 2

Due: April 26, 2024 @23:59

By turning in this assignment, I agree by the ITU honor code and declare that all of this is my own work.

Important Notes

- Read the instructions and the restrictions carefully and implement the .C code for the homework.
- Please do your project on your own. Team working is not allowed. Cheating will be punished by a negative grade. Also disciplinary actions will be taken.
- You are **only allowed** to use **mmap** for dynamic memory allocation in your code and you need to search for the call function for releasing the memory allocated using **mmap**. You are **not allowed to directly** use 'malloc', 'realloc' or 'free' in this homework. The purpose of this homework is to write your own procedures/functions to replace 'malloc()' and 'free()'.
- You should **comment** your code to clarify its functionality and purpose (Not for all lines but at least comment the important code lines to assist in code examination.).
- You need to write a **report** using IEEE LaTeX Template. It should convey the problem and the implementation details of your code and the dataset you used. In the report, give the result/answer for each question.
- **Submit** a .zip file including 1) the .C file and all code related files, and 2) the report file (.pdf) before the deadline.
- If you have any questions, please contact T.A. Ugur Ayvaz via ayvaz18@itu.edu.tr.

1 Description of the Homework

In this assignment, we want you to write a new modified version of the malloc() and free() calls that a **user-level** process uses that call to manage its heap. When a process wants to enlarge its heap size, the **memory allocation process** first requests memory space from the system using sbrk() it's a system call or **"mmap()"** calls (that is, it is added to the virtual address space). Next, using free calls and malloc, memory is allocated to the process heap, and this allocated memory is used to meet requests received during execution time. The memory allocator keeps the currently empty spaces in the heap in a list **(free-list)**, and incoming requests are met by selecting the most suitable free spaces in this list.

To simplify it, the procedures/functions you write will request memory space from the system once at startup using **mmap**. In other words, a fixed size will be determined for the heap and it will not go beyond that. (In normal use, as the heap grows, the process initially requests memory space from the system, so that all malloc operations can be met.)

2 Implementation Details

2.1 int InitMyMalloc (int HeapSize)

- This procedure/function will be called when the program first starts to run, **creating a fixed size heap**. Herein, the unit size that can be requested from memory is the **page size**.
- Therefore, the heap size to be created must be any **multiple** of the page size of your system (If the user does not enter the page size, you must round up).
- You can find out the page size on your system with the *getpagesize()* system call.
- You must create a header section at the beginning of the heap and the data structures or variables you need to keep there.
- InitMyMalloc should return **0 (zero)** if it completes the memory allocation successfully (that is, mmap completes memory allocation successfully).
- InitMyMalloc should return **-1** if mmap does not succeed, Mem_Init is called more than once, or the Mem_Init argument is invalid (0 or less).

2.2 void *MyMalloc (int size, int strategy)

- It is the function that will replace *"malloc()"*. This procedure takes the size of the heap object to be created in Bytes and returns a pointer that points to the created object.

- When allocating memory, the Bestfit (BF), Worstfit (WF) and Firstfit (FF) and NextFit (NF) memory allocation **strategies** should be used. For the "strategy" parameters: 0 must indicate **BF** method, 1 must indicate **WF** method, 2 must indicate **FF** method, and 3 must indicate **NF** method.
- If there is not enough space on the heap during allocation, it should return NULL.

2.3 int MyFree (void *ptr)

- It is the procedure that will replace *"free()"*.
- It releases the pointer object passed as argument (The pointer object that point out the address of allocated memory.).
- If the pointer is NULL it does nothing.
- After this process, the free memory spaces are **coalesced**. This creates free spaces where larger objects/processes can be allocated.

2.4 void DumpFreeList()

- It is for debugging purposes.
- It shows the available memory spaces in the heap.
- For each block, the address of the block, its size in **bytes**, and its full or empty status information should be printed, with the address of the first block being **0 (zero)**.
- A sample output:

```
Addr   Size   Status
0      48      Full
48     1024    Empty
1072   3096    Full
```

3 Required Outputs/ Questions

- **Code outputs**:
  - Declare 4 different processes (P1, P2, P3, P4) such that each of them requires a specific size of memory to run.
  - Use memory allocation strategies mentioned above to place each process in your heap.

- You can get the strategy type from the user input in the console (i.e. please enter the strategy type: 0 1 2 3).
- Output strategy type for memory allocation, status of allocation for each process (succeed, failed), the addresses of allocated memory if allocation is successful. Give the reason if it fails.
- Print out the DumpFreeList() function output.
- Mention all these output in your report.

- **Questions to be answered in the report** :
  - What are the main differences between "malloc" and "mmap"? If you had to make a decision between these two, which would you choose? Why?
  - What was the method/call you used in MyFree() instead of free()? Do you think it is as successful as free() at correctly and safely releasing the memory back? Why?
  - **Give the outputs/screenshots of Linux command line:**
    * Show the amount of free memory space before and after allocation.
    * Show the memory allocated processes (PID) and their memory addresses.
    * Prove that when you release the allocated memory in your code using MyFree(), the memory is actually freed.

4 Submission

- All of the procedures/functions must be implemented in a **.c** file.
- Supply the **.h** header file for your **.c** code. This header file should contain declarations of each procedure given above in Section 2.
- Supply compiled **.o** object file.
- Explain how can we compile your code? Which command we should enter in Linux command line to successfully compile your code?
- Submit a **.zip** file that includes: **.c** file, **.h** file, **.o** file (all code related files), and the **.pdf** file for the report.

Good Luck!