UF UNIVERSITY of FLORIDA
Herbert Wertheim College of Engineering

COP4600

P2: Memory Management & Layering

Overview

Due to your excellent work handling the metadata for project Sky Skink, the great Silurian overload Reptar himself has commissioned your work in designing a new custom Memory Manager application for the mobile version of Reptilian. This new project, deemed ***Project Repto***, is built using functionality from various layers of the OS to avoid human corruption. For extra security, the memory manager will write a log of its state when closed. If all goes well, your manager will be deployed all throughout the great Lizard Legion.

In this project, you will implement a memory manager in C++, whose features include initializing, tracking, allocating, and deallocating sections of memory. You will integrate the memory manager into two programs that we provide – an interactive Android application and a console-based testing program. The console program will provide some simple unit testing to check for correctness but will **not** check for memory leaks - *it is your responsibility to test for leaks and ensure overall correctness!* You'll then create a short video to demonstrate your code and submit the project via Canvas.

Structure

Modern operating systems are often built in layers with specific goals and limited privileges. Layering also facilitates the implementation of recognized standards by separating hardware and OS-specific implementations from generalized API calls. Often these layers are written in different languages and permit access via a binding layer to lower level functionality. In Android, POSIX other standard functions can be called from Java applications via the Native Development Kit (NDK). Your memory manager will make use of these standard functions so that it can be used in console and Android GUI applications.

The project is broken into three main parts:

1) Write a memory manager in C++ using basic system functionality.
2) Integrate the memory manager into the console-based testing program we provide.
3) Bind memory manager invocation, via the NDK, to the Android GUI application provided.

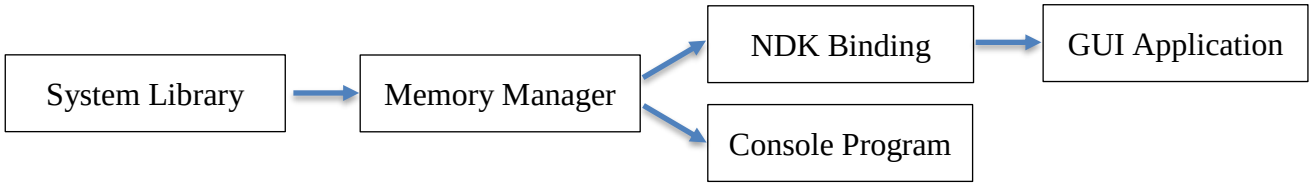


Figure 1: Memory manager is instantiated from a GUI application, and separately, a console program.

While exact implementation may vary, *the library functions must match the signatures laid out in this document.*

Specification

Students will write several sections of code according the following specifications.

Memory Manager Class

The Memory Manager will handle the allocation/deallocation of memory and provide details of its state. How the class keeps track of allocation/deallocation is implementation dependent and is left for the student to decide. **`MemoryManager.h`** and **`MemoryManager.cpp`** will contain declaration and definition, respectively.

```
public MemoryManager(unsigned wordSize, std::function<int(int, void *)> allocator)
```
Constructor; sets native word size (for alignment) and default allocator function for finding a memory hole.

```
public ~MemoryManager()
```
Releases all memory allocated by this object ***without leaking memory***.

```
public void initialize(size_t sizeInWords)
```
Instantiates a new block of memory of requested size, no larger than 65536. **Hint**: you may use ***`new char[…]`***.

```
public void shutdown()
```
Releases memory block acquired during initialization.

```
public void *allocate(size_t sizeInBytes)
```
Allocates a block of memory. If no memory is available or size is invalid, return **`nullptr`**.

```
public void free(void *address)
```
Frees the memory block within the memory manager so that it can be reused.

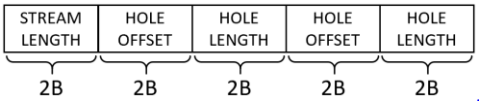
```
public void setAllocator(std::function<int(int, void *)> allocator)
```
Changes the allocation algorithm to identifying the memory hole to use for allocation.

```
public int dumpMemoryMap(char *filename)
```
Uses standard POSIX calls to write hole list to filename **as text**, returning **`-1`** on error and **`0`** if successful.

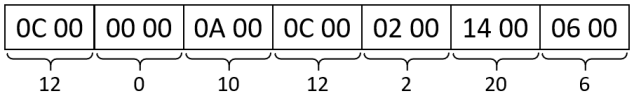
Format: "`[START, LENGTH] - [START, LENGTH] …`", e.g., "**`[0, 10] - [12, 2] - [20, 6]`**"

```
public void *getList()
```
Returns a byte-stream of information (**in binary**) about holes for use by the allocator function (*little-Endian*).

Format:



Example:




```
public void *getBitmap()
```
Returns a bit-stream of bits representing whether words are used (**`1`**) or free (**`0`**). The first two bytes are the size of the bitmap (*little-Endian*); the rest is the bitmap, word-wise.

```
public unsigned getWordSize()
```
Returns the word size used for alignment.

```
public void *getMemoryStart()
```
Returns the byte-wise memory address of the beginning of the memory block.

```
public unsigned getMemoryLimit()
```
Returns the byte limit of the current memory block.

Memory Allocation Algorithms

```
public int bestFit(int sizeInWords, void *list)
```
Returns **word offset** of hole selected by the best fit memory allocation algorithm, and **-1** if there is no fit.

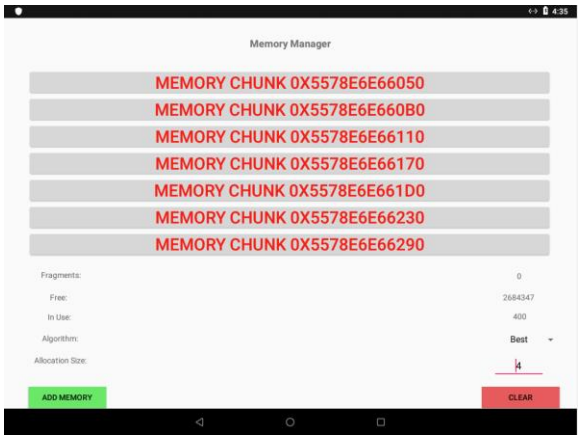
```
public int worstFit(int sizeInWords, void *list)
```
Returns **word offset** of hole selected by the worst fit memory allocation algorithm, and **-1** if there is no fit.

Testing

Your code must also compile at the command line and function with provided sample files. These sample files cover a base level functionality for the functions above. ***It is critical that you test for memory leaks!*** The code we provide will not test for this. Submissions with memory leaks will be marked **zero** for functionality.

GUI Program

The GUI program skeleton provides all the GUI elements you should need (without the necessary bindings). You must to implement the binding calls in **native-calls.cpp** labeled with **TODO** to complete the application.



Students should modify the C++ bindings. A simple example of transmitting a string from C++ to Java is included in the project base code.

NOTE: You will need to add any new C++ source files to the `CMakeLists.txt` build file!

Extra Credit

In the above implementation of `MemoryManager.initialize(...)`, you most likely made use of the **`new`** operator to acquire your initial block of memory. Your job is to now figure out how to allocate your initial block of memory without the use of **`new`** or any stdlib functions, e.g. `malloc, calloc`, etc.

Submissions

You will submit the following at the end of this project:

- Report (p2.7) on Canvas, including a link to the screencast
- Compressed tar archive (**`MemoryManager.tgz`**) containing source/build files for text program on Canvas
- Zip archive (**`nativeapp.zip`**) containing source/build files for the GUI program on Canvas

Report

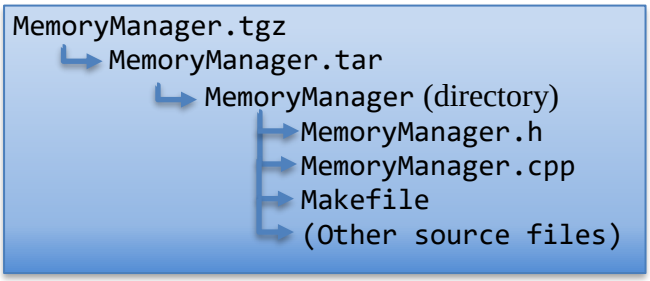
Your report will explain how you implemented all your functions (including the NDK functions) along with an explanation of how each function performs in relation to its layer within the OS. It will describe how the unit test operates, justifying why the output is correct. The report should be no more than 500 words. It should cover all relevant aspects of the project and be organized and formatted professionally – ***this is not a memo!***

Screencast

In addition to the written text report, you should submit a screencast (with audio) walking through each of your functions, highlighting how the various layers interact with each other and showing your unit test (~6 minutes).

Compressed Archive (MemoryManager.tgz)

Your compressed tar file should have the following directory/file structure:



To build your program, we will execute these commands:

```
$ tar zxvf MemoryManager.tgz
$ cd MemoryManager
$ make
$ cd ..
```

To link against the library, we will execute this command:

```
$ c++ -o program_name sourcefile.c -L ./MemoryManager -lMemoryManager
```

Zip Archive (nativeapp.zip)

To create a compact package from your Android project, select **File → Export to Zip File** from the Android Studio menu system. This will package your project for upload.

Please test your functions before submission! If your code does not compile it will result in **zero credit** (0, none, goose-egg) for that portion of the project.

Helpful Links

http://www.idc-online.com/technical_references/pdfs/information_technology/Memory_Management_with_Bitmaps_and_Linked_List.pdf

https://www.cs.rit.edu/~ark/lectures/gc/03_00_00.html

https://www.ibm.com/developerworks/library/pa-dalign/index.html

https://developer.android.com/ndk/guides/

Helpful Testing Links:

http://valgrind.org/

https://github.com/catchorg/Catch2