Operating Systems Coursework 2 (30%)

Dr James Stovold

Released: Mon Week 8
Due: Fri Week 9, 6pm German Time

1 General Rules

This assessment is covered under the the relevant sections of Lancaster University's Manual of Academic Regulations and Procedures[1], of particular relevance are the '*General Regulations for Assessment and Award*' and '*Academic Malpractice Regulations and Procedures*'.

Late submissions will be downgraded as per these regulations. This is an **individual** assessment, so your work must be your own. Remember: if you are unsure about whether something constitutes academic misconduct, it is better to ask than to guess.

Questions about the assessment should be submitted to James via email. For fairness, the (anonymised) question and answer will then be sent via email to the entire cohort.

2 Task

Your task is to implement the version of `malloc` described in the lectures. This should be implemented using Java. You should provide a single Java class called `MemoryMgmt` which provides the following methods:

> `malloc(int size)`, which allocates a chunk of memory sized `size`. This function returns an integer representing a pointer to the start of the dynamically-allocated chunk of memory. You can assume the total amount of memory available to the system is 8192 bytes (8KB). Any requests above this point should trigger a call to your implementation of `sbrk()`, detailed below.
>
> `free(int ptr)`, which frees the memory chunk pointed to by `ptr`, coalescing it into the free memory around it. This function does not return anything. Multiple calls to `free` on the same pointer should trigger an exception.
>
> `sbrk(int size)`, which returns a new array of memory (the length should be the smallest power of two larger than `size`), and an integer representing the base pointer of the new chunk of memory. You should not assume that each new chunk of memory will be allocated contiguously.
>
> `print()`, which runs a series of tests on your code and prints the outputs to stdout in an easily-readable manner.

[1] `https://www.lancaster.ac.uk/academic-standards-and-quality/marp/`

2.1 Further information

- Java does not have a pointer type you can use, so you will need to do the pointer arithmetic manually.
- You are not permitted to use `unsafe` methods or built-in dynamic allocation functionality, other than to allocate a new chunk of memory inside your implementation of `sbrk`.
- You are also not permitted to use built-in system calls to the OS: you must implement the memory management by hand (i.e. you cannot just wrap system calls to `malloc`/`sbrk` in a Java file).
- You can assume the code will be run sequentially, so you do not have to consider issues relating to concurrent access to the memory etc.
- Your code will be tested on the SCC VMs, so ensure your code runs correctly from there.
- You are required to provide `build.sh` and `run.sh` shell scripts to build and run your code respectively.
- During testing, I may override one of your methods (for example, `sbrk`) to test the robustness of your implementation.

2.2 Tests

Your `print()` function should run a series of tests on your code. At each stage of the test, you should print out the pointers in your system and what you are doing. At a minimum, these tests need to include the following:

a. Request a chunk of memory of size 28 bytes. Store a string in it. Retrieve the string. Free the memory.

b. Request three chunks of memory (in the following order), sizes: 28 bytes, 1024 bytes, 28 bytes. Free the 1024 bytes. Request another chunk of 512 bytes. Free all the remaining chunks.

c. Request a chunk of memory of size 7168 bytes. Request another chunk of 1024 bytes. Free all the chunks.

d. Request a chunk of memory of size 1024 bytes. Request another of 28 bytes. Free the 28 bytes. Free the 28 bytes again.

2.3 Example output

One possible output from a call to `print()` could be as follows, given a test which requests 1024 bytes twice, frees the first one, then requests another 1024 bytes, before requesting 10KB, freeing all the pointers, then freeing the original pointer for a second time.

```
Running test number 7...

HEAD pointer: 0x0080
```

```
Requesting 1024 bytes of memory... memory allocated.
Pointer: 0x001A

Requesting 1024 bytes of memory... memory allocated.
Pointer: 0x0420

Freeing pointer 0x001A ... memory freed.
Requesting 1024 bytes of memory... memory allocated.
Pointer: 0x0824

Requesting 10240 bytes of memory...
Memory limit exceeded, requesting further memory blocks... memory allocated.
Pointer: 0x209A

Freeing pointer 0x209A ... memory freed.
Freeing pointer 0x0824 ... memory freed.
Freeing pointer 0x0420 ... memory freed.
Freeing pointer 0x001A ... Exception triggered in thread. Exiting.
```

3 Demonstration

During the week 10 lab session, you will be allocated a 5mins slot to give a one-on-one demonstration of your code. During this time you should quickly present your approach to the problem, and demonstrate its functionality on the lab computers. James will then ask a few questions about your implementation.

4 Submission

You should submit *one file* for coursework 2. This is a `zip` or `tar.gz` file containing all your code and scripts for building/running the code. I will extract your compressed archive and run `build.sh`, followed by `run.sh`. If the code does not build/run, you will likely not get any marks for that component.

For your demonstration, you must present the same code that you have submitted to Moodle. If there are any changes between the two pieces of code, you will receive a 0 mark for the assessment.

Do not include your name anywhere on your submission, your work will be marked anonymously. You should, however, include your Lancaster student number as the author in any comments in your code.

The coursework will be marked, and feedback provided, within 4 working weeks of submission. Be sure to familiarise yourself with the marking rubric available at the end of this document. The allocation of marks is dependent on the features implemented in your code.

Component (weight)	**Fail [0, 40)**	**3 [40, 50)**	**2.ii [50, 60)**	**2.i [60, 70)**	**1 [70, 100]**
Dynamic Memory Allocation (70%)	Code fails to provide even basic functionality. Code may have syntax errors, or may not compile properly on the SCC VMs as required. If code compiles correctly, it may not solve the basic allocation problem.	Code compiles correctly, basic allocation code appears to work, but may be very inefficient or fail to use core techniques to manage pointers. There might be some missing functionality (such as `free` or `sbrk`), or might not provide all the requested tests. Code might be difficult to interpret due to poor coding practices.	Code provided appears to solve the problem sufficiently. Some management of memory has been attempted, but approach could be simpler, or fails to maintain appropriate free lists. Unable to handle changes to the underlying memory model. Code might be a little difficult to follow, and some focus on good coding practice would help.	Memory management problem has been approached well, with appropriate free lists and pointer arithmetic implementations. The code is able to handle all the required tests, and some others, but may fail on some edge cases or when the underlying memory model changes.	A well-designed and well-implemented memory management class. The implementation is able to handle changes to the underlying memory model with ease, and is able to handle edge cases. Code practices are at the level that I could be looking at professional code.
Demonstration (30%)	Did not attend demonstration, or was unable to answer questions about their code.	Attended the demonstration, and was able to describe the general approach taken. Can answer basic questions but not in-depth questions about their approach.	Demonstration was reasonable, student was able to describe their approach, but might struggled to justify their rationale for certain design decisions.	A good, logical approach to demonstrating the code. The approach was well described and the student was able to answer most questions with ease.	A good, logical approach to the demonstration, highlighting key areas that are worth focus, while being able to easily expand on points of interest as requested by the marker. Questions were not only answered with ease, but showed a clear and thorough understanding of the underlying concepts.