




School of Computing Technologies
COSC1114 Operating Systems Principles

Project 2 – Memory Allocation

	Assessment Type: Group project (maximum of two students) Submit online via Canvas → Assignments → Project 2 Clarifications/updates may be made via announcements and relevant discussion forums.
	Due Date: Monday, October 7, 2024, at 23:59.
	Weighting: 60 marks that contributes 30% of the total assessment.

1. Overview

This assignment aims to deepen your understanding of memory management techniques in operating systems through practical implementation. You will develop a simulation that allocates memory blocks using the First-Fit and Best-Fit allocation strategies. This will help you comprehend the efficiencies and drawbacks of each method in different scenarios.

You will implement your solution to this task in C++ or C but note that the required function calls are C system calls that are specific to Linux, so you won't be able to compile/run code outside of Linux. This does however include WSL but does not include MacOS, which while having a terminal has a different approach to memory allocation.

2. Learning outcomes

This assessment is relevant to the Course Learning Outcomes CLOs , 2, 3, 4, 5 & 6.

3. Assessment details

This assessment will determine your ability to

1. Understand the concepts taught over the weeks 8, 9, and 10 materials of the course.
2. Work independently in self-directed study to research the identified issues.

4. Academic integrity and plagiarism

It is understood by us that many of the algorithms used in this course have common implementations. You are welcome to look at online code examples to understand possible solutions to the set problems. However, what you submit must be your own work and your submission will be checked and compared with other solutions. **Submitting material generated by an AI as your own work constitutes plagiarism and academic dishonesty. DO NOT simply copy other people's work, it is not difficult for us to detect copied work and we will pursue such cases.**

5. Group work

The group work should be completed in pairs, that is, with no more than two students per group (you may form groups with any student in the course).

You have the option to complete this assignment on your own.

Students are required to contribute to and demonstrate collaborative effort on all tasks. All group members will receive the same grade by default.

To ensure effective management of your group work and to demonstrate ongoing contributions to your project, **you will be required to maintain contribution logs that detail your involvement in both tasks.** These should include records of tasks completed, hours spent, and notable interactions within the group. This document can be reviewed if there are disputes about contributions or for grading adjustment purposes. **Should the group disband, each member must be capable of completing the project on their own.**

If there are any issues that affect your work, such as being sick, you must keep your group informed in a timely manner. Your final grade will be based on the work you and your partner complete throughout the duration of the assignment. We will treat everybody fairly, and account for times when issues arise when marking your group work and contributions. However, you must be upfront and communicate with us. If you fail to inform us of issues in a timely fashion and we deem that your actions significantly harm your group's performance, we may award you a reduced grade. It is academic misconduct if you are deliberately dishonest, lie to, or mislead your group or teaching staff in a way that harms your group.

Notify your tutor immediately if you encounter any issues working with your group at any time. We will do our best to help manage group issues, so that everybody receives a fair grade for their contributions.

6. Submission

Your assignment should follow the requirement below and submit via [Canvas](#) → [Assignments](#) → [Project 2](#).

You will submit two things:

1. Your C++/C source code in [a zip file](#), which includes
 - the code you have developed, and
 - a README file in which **please specify exactly how to run your code on the provided servers.**

When working in pairs, please include both student numbers in the zip filename, for example, [s1234567_s4567890.zip](#); otherwise, use your own student number only.

2. Besides the zip file, **organize the source code of tasks in a separate text file [s1234567_s4567890.txt](#)** (copy & paste them into a text editor and then save it). **This text file is for Turnitin plagiarism check.**

Your submission will not be graded if you fail to submit your txt or if your txt does not pass the Turnitin plagiarism check.

It is your responsibility to correctly submit your files. Please verify that your submission is correctly submitted by downloading what you have submitted to see if your submitted file includes the correct content.

Never leave submission to the last minute – you may have difficulty uploading files. **However, if your final submission is after the due time, late penalties will apply.**

7. Late submission policy

A penalty of 10% per day of the total available marks will apply for each day being late. **After 10 days, you will receive zero marks for the assignment.**

If you want to seek an extension of time for assignment submission, you must have a substantial reason for that, such as unexpected circumstances. Reasons such as, unable to cope with study load, is not substantial. Also, you must apply for an extension as soon as possible. Last minute extensions cannot be granted unless it attracts special consideration.

Please find out how to apply for special consideration online at <https://www.rmit.edu.au/students/student-essentials/assessment-and-results/special-consideration/eligibility-and-how-to-apply>.

Any student wishing an extension must go through the official procedure for applying for extensions and must apply at least a week before the due date. Do not wait till the submission due date to apply for an extension.

8. Project tasks

Task 1 – Solution Design (10 marks)

You will use two linked lists to manage memory – a linked list of occupied chunks (memory allocations) and a linked list of free chunks (to be used for new memory allocations). **You are not required to create linked lists from scratch.** At this stage, you should understand how linked lists function and be capable of utilizing an existing library. Therefore, no marks will be given for the linked list implementation itself.

You will however need to determine and keep track of some accounting information as well as the memory allocations themselves (**at a minimum, the location and the size of each allocation**). This means the data structures for keeping track will need to be **global variables**. **These global variables should not be referenced outside of the two function calls below.**

The core requirements are the two functions:

```
void * alloc(std::size_t chunk_size);
```

```
void dealloc(void * chunk);
```

and functions to set the allocation strategy (more on this later).

These two functions are essentially a simpler version of `malloc` and `free` (you are encouraged to get familiar with these two functions using man pages on titan). You may not have come across the `size_t` type before but it's just an unsigned long integer that is guaranteed to be able to hold any valid size of memory. So, it's the size of the memory chunk that has been requested. The pointer passed into `dealloc` is the chunk to be “freed” and so it must be a chunk that was previously allocated with the `alloc` function.

Note: `dealloc` does not return the chunk to OS. It simply makes the chunk free/available to new requests.

It is suggested that you represent an `allocation` using a class or struct like the following:

```
struct allocation {  
    std::size_t size;  
    void *space;  
};
```

where `space` is initially allocated using the C function the `sbrk()` function. **It is also suggested that you store each allocation in its own pointer rather than having a list of allocation structs.** This is because the `pop()` operations in the standard library call the destructor which may free the memory, depending upon your design.

Task 2 – Memory Allocation (10 marks)

Memory allocation will be implemented in the function `alloc()` as specified above.

This function will first look in the free list for a chunk big enough to meet the allocation request (based on the previously set strategy). If a chunk large enough cannot be found, a new request will be sent to the operating system using `sbrk()` system call to grow the heap.

In either case, the chunk and its metadata will be added to the allocated list before returning the pointer to the memory chunk itself back to the caller that requested the memory allocation. **Memory should be allocated strictly in fixed partition sizes of 32, 64, 128, 256, and 512 bytes.**

Task 3 – Allocation Strategies (25 marks)

As part of memory allocation, you will need to implement a search for a chunk using an allocation strategy. The allocation strategies you need to implement are:

- First fit: the first chunk that you find big enough is returned to the user.
- Best fit: you find the chunk whose size is the closest match to the allocation need. This will match the size needed by the user, thus eliminating internal fragmentation. **(Here we assume the perfect scenario to simplify the task.)**

Task 4 – Memory Deallocation (10 marks)

Search for the pointer passed in the allocated list (the linked list of occupied chunks). This will be performed using the `dealloc()` function.

- If it is not found, the program should terminate as this is a fatal error – you can't free memory that was never allocated.
- If the chunk is found, simply remove it from the allocated list and place it at the end of the free list (the linked list of free chunks).

How to Test Your Program

You should make a sequence of interleaved `alloc` and `dealloc` calls to test out your allocation strategies. We have provided you a generator `p2_gen.sh` to create these sequences. **Please find the bash script in the assignment page.** To generate a file called `datafile` with a sequence of 20 interleaved `alloc` and `dealloc` calls as shown by the screenshot below:

```
[e52483@csitprdap01 ~]$ chmod +x p2_gen.sh
[e52483@csitprdap01 ~]$ ./p2_gen.sh 20 > datafile
[e52483@csitprdap01 ~]$ cat datafile
alloc: 114
alloc: 509
alloc: 150
alloc: 271
alloc: 112
alloc: 408
dealloc
dealloc
dealloc
dealloc
alloc: 55
alloc: 310
dealloc
alloc: 469
alloc: 275
dealloc
dealloc
alloc: 358
alloc: 253
alloc: 433
[e52483@csitprdap01 ~]$ █
```

An `alloc` call allocates a suitable free chunk.

A `dealloc` call simply deallocates the last memory chunk allocated, **specifically, in a LIFO (last in, first out) order.**

Note: In Task 4, the actual implementation of `void dealloc(void * chunk)` needs to be able to `dealloc` any chunk specified by the user, just like `free()` function in C. **In this task, `dealloc` the last memory chunk allocated is for the testing and marking purpose only.** You simply use the `dealloc()` that you developed in Task 4.

Your programs are expected to parse command line arguments according to the following format:

- First fit memory allocation: `firstfit datafile`
- Best fit memory allocation: `bestfit datafile`

where the datafile is the one we generated using the given `alloc_generator`.

Your programs are expected to generate the following output on the screen (only print once in the end):

- The allocated list (the linked list of occupied chunks)
For each chunk, print out the address of the chunk, the total size of the chunk, and the used size of the chunk.
- The free list (the linked list of free chunks)
For each chunk, print out the address of the chunk and the total size of the chunk.

Technical Specifications

- Programming Language: Implement the system in C++ or C.
- Error Handling: Your program must gracefully handle and report errors such as failed memory allocation or file access issues.
- Makefile: Create a `makefile` to compile your programs.

- `make all` will build 2 programs, `firstfit` and `bestfit`, at once.
- `make clean` will get rid of object and executable files.

Build Environment

Please note that it is expected that your program will compile and run cleanly on the provided servers:

titan.csit.rmit.edu.au

jupiter.csit.rmit.edu.au

saturn.csit.rmit.edu.au

Your code must execute on the designated servers. A 20% penalty will be applied if the markers are unable to run your program on these servers. Make sure you compile with the following flags and fix the errors before submitting your assignment:

`-Wall -Werror`

Code Quality

Please note that while code quality is not specifically marked for, it is part of what is assessed as this course teaches operating systems principles partly through the software you develop. **As such lack of comments, lack of error checking, good variable naming, avoidance of magic numbers, etc., may be taken into consideration by your marker in assigning marks for the components of this assignment, although operating systems principles do take priority.**

9. Rubric and marking guidelines

The rubric can also be found in **Canvas → Assignments → Project 2.**

For each assessment requirement, the rubric is designed to be general and the details of what you can improve on will be outlined by your marker in the comments. Nevertheless, this should be treated as a guide to what to aim for.

Requirement	No Attempt (0%)	Poor (25%)	OK (50%)	Good (75%)	Excellent (100%)
Solution design (10 marks)	No attempt	A reasonable attempt but it does not compile.	Code compiles but is an incomplete/incorrect implementation.	Mostly complete implementation but there are minor problems.	Excellent job. Well done.
General allocation algorithm: (10 marks)	No attempt.	Some attempt but did not get the code working or does not compile.	Code compiles but is an incomplete/incorrect implementation.	Mostly complete implementation but there are minor problems.	Excellent job. Well done.
General deallocation algorithm (10 marks)	No attempt.	Some attempt but did not get the code working or does not compile.	Code compiles but is an incomplete/incorrect implementation.	Mostly complete implementation but there are minor problems.	Excellent job. Well done.
The first-fit algorithm for finding blocks of memory: (10 marks)	No attempt.	Some attempt but did not get the code working or does not compile.	Code compiles but is an incomplete/incorrect implementation.	Mostly complete implementation but there are minor problems.	Excellent job. Well done.
The best-fit algorithm for finding blocks of memory: (15 marks)	No attempt.	Some attempt but did not get the code working or does not compile.	Code compiles but is an incomplete/incorrect implementation.	Mostly complete implementation but there are minor problems.	Excellent job. Well done.
Makefile, Arguments, and Output (5 marks)	No attempt.	Some attempt but did not get the code working or does not compile.	Code compiles but is an incomplete/incorrect implementation.	Mostly complete implementation but there are minor problems.	Excellent job. Well done.