CS3104 Operating Systems 2025-26

Practical 2 - Buddy Memory Allocator

1 Introduction

This coursework is worth 40% of the practical component of this module, and 24% of your grade overall.

Submission Deadline: Monday 3rd November 21:00 - Week 8

1.1 Main Objectives

- To implement a Buddy Memory Allocator for physical page allocation.
- To explore memory management techniques in an Operating System.

2 Task

Your second coursework task is to implement a physical memory allocator using the **buddy algorithm** for the **StACSOS** operating system.

Important note: Make sure you have read the **StACSOS** supporting document before you attempt this task. It contains a lot of information that will be useful for getting started. You can find it on StudRes here:

`https://studres.cs.st-andrews.ac.uk/CS3104/Coursework/stacsos.pdf`

You should also make sure you have an up-to-date copy of the **StACSOS** repository. If you have already checked it out, run `git pull` to receive any updates. If you're working from a private fork, make sure you have merged in any changes from the main repo.

2.1 Background

Normally, when a program requests memory it will simply ask for a particular amount. It expects the memory allocator to find space for this, and does not care where that memory is in the address space. But, memory allocators need to put this memory somewhere, and at a very low-level, this memory has to exist in *physical* memory.

Physical memory allocation is the act of allocating real physical memory pages to places that require them. Typically, higher-level memory allocators (such as the **StACSOS** object allocator) request physical pages, which in turn are used for storage of smaller objects.

A particular algorithm for managing this physical memory is the **buddy allocation algorithm**, and this is what you are required to implement in **StACSOS**.

Like the scheduler, **StACSOS** comes with a built-in physical memory allocation algorithm, but it is very simple and inefficient. This task requires you to implement the buddy allocation algorithm as described in the lectures and from various resources online[1].

2.2 Memory Allocation

A page of memory is the most fundamental unit of memory that can be allocated by the page allocator. In **StACSOS**, the page size is 4096 bytes (which matches the x86-64 architectural page size). Pages are **always** aligned to their size, and can be referred to with either:

1. Their *page frame number* (PFN), or
2. their *base address*.

It is trivial to convert between the two, when you know the page size.

PFNs are zero-indexed, so for example, the second page in the system has a PFN of `1`. Since pages are aligned, the base address of PFN `1` is `0x1000`. Likewise, given a page base address of `0x20000`, a simple division by `4096` (or right-shift by `12`) yields a PFN of `32`.

For every physical page of memory available in the system, **StACSOS** maintains a *page descriptor* object, which holds meta information about that page. These *page descriptors* are held as a *contiguous* array, and so can be efficiently indexed, given the physical address or PFN of a page.

The `page` class, which represents a page descriptor, is defined in:

`kernel/inc/stacsos/kernel/mem/page.h`

This property will become important in your implementation, as it means that given a *pointer* to a particular *page descriptor*, you can look at the adjacent *page descriptor* by simply incrementing the pointer.

Your allocator implementation should not modify **any** fields in the page descriptors, as the memory management core is responsible for these.

The physical page allocator does not allocate by memory size, or even by number of pages. Instead it allocates by *order*. The *order* is the power-of-2 number of pages to allocate. So, an allocation of order `zero` is an allocation of $2^0 = 1$ page. An allocation of order `four` is an allocation of $2^4 = 16$ pages. Allocating by *order* makes it significantly easier to implement the buddy allocator.

The buddy allocator maintains a list of free areas for each order, up to a maximum order. The maximum order should be configurable, and is specified for you already in the skeleton:

```
static const int LastOrder = 16;
```

Use this definition when implementing your algorithm.

[1]For example, `https://en.wikipedia.org/wiki/Buddy_memory_allocation`

2.3 Important Files and Interaction with StACSOS

The memory management subsystem is quite complex, and so has its own top-level directory (`kernel/src/mem`). You should explore this section of the code, starting with the *memory manager*, to see how it all fits together. Remember that each source-code file (`.cpp`) has a corresponding header file, which might contain a lot of code too.

- `address-space-region.cpp`: Contains code for dealing with a particular region in an address space.
- `address-space.cpp`: Contains code for managing a logical address space.
- `large-object-allocator.cpp`: Allocates objects that are larger than the pre-defined slab cache sizes.
- `memory-manager.cpp`: The memory management *core* routines.
- `new.cpp`: The implementation of the `new` and `delete` operators for C++.
- `object-allocator.cpp`: Allocates individual objects for use by the kernel. This uses slab caches for small objects, and the large object allocator for large objects.
- `page-allocator-buddy.cpp`: The skeleton code for the Buddy Allocator algorithm. You will be filling this in.
- `page-allocator-linear.cpp`: A naïve implementation of a page allocator, so that the OS can boot without a working buddy allocator.
- `page-allocator.cpp`: The core page allocation routines.
- `page-table-allocator.cpp`: An allocator specifically for allocating pages used by page tables.
- `slab-cache.cpp`: An implementation of a slab-based object allocator.

Remember: You must tell **StACSOS** to use the buddy allocator when you start it up – otherwise it will use the built-in linear allocator.

2.4 Skeleton

You are provided with a skeleton memory allocation algorithm interface, in which you must write your code to implement the buddy allocator. The skeleton files live here:

- `kernel/inc/stacsos/kernel/mem/page-allocator-buddy.h`
- `kernel/src/mem/page-allocator-buddy.cpp`

You can change either of these files in any way you see fit - so long as the class still implements the `page_allocator` interface. You **must not** change any other part of **StACSOS** to make your algorithm work.

Some of the ground-work for buddy allocation has already been done for you, and placed in the skeleton files. Implementing the page algorithm interface requires implementing these methods:

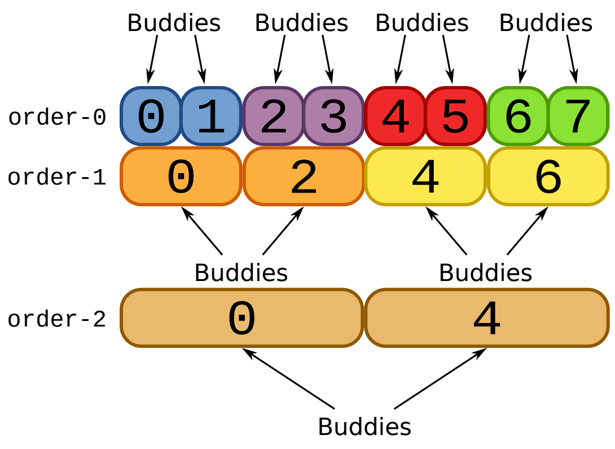

Figure 1: An illustration of block buddy relationships, in different allocation orders.

- `insert_free_pages`**:** This is called by the memory mangement core to insert a range of allocatable pages.
- `allocate_pages`**:** This is called when a request is being made to allocate some pages.
- `free_pages`**:** This is called when a request is being made to free some pages.
- `split_block`**:** This is a helper routine that splits a free block in one order, and places the two halves in the order below, e.g. takes an Order-2 block, and splits it into two Order-1 blocks.
- `merge_buddies`**:** This is a helper routine that takes either buddy in a pair, in a particular order, and merges them together, inserting the merged block into the order above. For example, takes two Order-0 blocks (that are buddies), removes them from Order-0 and places the merged block into Order-1. Either buddy can be specified, but both buddies MUST exist in the free list.

It is guaranteed that `insert_free_pages` will be called by the memory management core *only* with pages that are free to be allocated, i.e. with kernel and system pages excluded.

See the `initialise_page_allocator` function inside `memory-manager.cpp`.

Remember: When implementing your algorithm, you cannot use dynamic memory allocation in your memory allocator! This means you cannot use the `new` operator, and you cannot use containers such as `list<T>` and `map<T, U>`, since these rely on dynamic memory allocation.

You can use the `next_free` field in the page metadata structure to build linked lists—this should be all you need. Page metadata is held in the free pages themselves, and can be accessed with the `metadata()` helper function. See some of the pre-implemented functions for how to use `metadata()`.

The skeleton file contains the following helper methods, which you should use in your implementation. You should also use these functions as a guide:

- `pages_per_block`**:** This function returns the number of pages that make up a block of memory for a particular *order*. For example, in order 1, the number of pages that make up a block is 2.

- `block_aligned`: This function determines if the supplied page is correctly aligned for a particular *order*. For example, page zero is correctly aligned in order 1, but page one is not.
- `insert_free_block`: Given a particular page, being correctly aligned for the given order, this function inserts that block into the free list.
- `remove_free_block`: Given a particular page, being correctly aligned for the given order, this function removes that block from the free list.

2.5 Allocating Pages

The `allocate_pages` method is called when a contiguous number of pages needs to be allocated. The caller does not care *where* in memory these pages are, just that the pages returned are contiguous. Because of this guarantee, if the caller asks (for example) for an order 1 allocation (i.e. two pages), the routine simply needs to return the **first** page descriptor of a sequence of **two** page descriptors that are available for allocation (by following the buddy allocation algorithm).

This works because the pages are contiguous, and because the page descriptor array is contiguous.

2.6 Testing

To test your algorithm, run `make` with the `kernel-args` parameter selecting the `buddy` page allocation algorithm:

```
$ make run kernel-args="pgalloc=buddy"
```

It is VERY important that you put the `pgalloc=buddy` option on the command-line, otherwise the built-in allocator will be used instead.

When you run `make`, your source-code will be built, and if there are any errors, these will be displayed to you and the OS will not load.

To double-check that **StACSOS** is using your algorithm, scroll back up the terminal window and looking for the following output:

```
mem: *** using the 'buddy' page allocator
```

If the message says "`buddy`" then your implementation is being used. When you run this for the first time (i.e. on a newly checked out repository), you should get a kernel panic.

Because memory allocation is such a fundamental operation, it is quite likely that during the course of you implementing your algorithm the system will either:

1. Not boot at all.
2. Triple fault, and continually restart.
3. Behave very strangely.

Therefore, a good test is: *does the system boot to the shell?* and *can I run programs?*. Interestingly, you don't need to free memory for the system to work—only allocation is the critical component.

If the system boots up to the **StACSOS** shell, then you can try running some test programs that will exercise the memory allocator:

```
> /usr/sched-test
> /usr/sched-test2
> /usr/mandelbrot
```

2.7 Self Test

In order to more accurately quantify the success of your implementation, a *self-test* mode is available that will test the memory allocator during start-up. This self-test mode will make a series of allocations and use the `dump()` method of the allocation algorithm to print out the state of the buddy system. Although it's called self-test, it doesn't actually verify correctness—you'll need to do this manually via inspection.

You should use this output to make sure your buddy system is behaving correctly when allocating and freeing pages. The `dump()` method will display the free list for each order. See Appendix A for example output of the self-test mode, which you should use to make sure your own implementation is behaving in a similar fashion.

You can activate this facility by setting the "`pgalloc-selftest`" option to "`yes`":

```
$ make run kernel-args="pgalloc=buddy pgalloc-selftest=yes"
```

The self-test mechanism creates a fake memory layout, so it won't boot up to the prompt, but it runs a series of tests to check that you're allocating blocks correctly. See the appendix for reference output, but also look at how the self-test works (in `kernel/src/mem/page-allocator.cpp`) to see exactly what is going on.

Feel free to modify the self-test routines, and add in your own testing code if you want to try different scenarios/inputs. You are not required, however, to submit these modified tests.

3 Report

You are required to produce a single page PDF report for this assignment.

The report should contain a brief description of the implementation of your algorithm, any assumptions you have made, any clever optimisations you've decided to implement, and anything else worthy of note. You should include a description of the challenges involved in implementing buddy allocation, and any insights you can provide about physical page allocation.

Please do not exceed one page of A4.

4 Deliverables

You must submit a `.tar` file containing your allocator source-code (`page-allocator-buddy.h`, `page-allocator-buddy.cpp`), and single-page PDF report (`report.pdf`) to the P2 slot in MMS.

5 Marking

See the standard mark descriptors in the School Student Handbook:

`https://info.cs.st-andrews.ac.uk/student-handbook/learning-teaching/feedback.html#Mark_Descriptors`

You will be assessed on the correctness of the implementation of the algorithm, and the contents of your report. You can achieve a maximum grade of 14, if you only implement allocation, and make no reasonable attempt at free.

A *passing submission* (7+) will capture the basic requirements of the buddy allocator. The allocator will work, although not all functionality may be implemented, e.g. freeing may not be implemented, or splitting/merging may not be implemented.

A *good submission* (11+) will implement page allocation with proper splitting and merging, but not freeing.

An *excellent submission* (15+) will implement page allocation and freeing correctly, but the report may be lacking in detail. A 17 will be awarded for a well-written piece of software, which correctly implements everything in this specification, accompanied by an excellent report.

To achieve marks in the 18–20 range, you will need to demonstrate exceptional software engineering and insight into implementing the allocator. Your code-base should be highly readable, well commented, and nicely structured. Your report should include a lot of detail, with excellent justifications for your implementation choices, and any assumptions you have made, but without exceeding one page.

Make sure that you comment your code well, as this will make it much easier to mark when reasoning about your thoughts when implementing the algorithm.

5.1 Lateness

The standard penalty for late submission applies (Scheme A: 1 mark per 24 hour period, or part thereof):

`https://info.cs.st-andrews.ac.uk/student-handbook/learning-teaching/assessment.html`

5.2 Good Academic Practice

The University policy on *Good Academic Practice* applies: `https://www.st-andrews.ac.uk/education/handbook/good-academic-practice/`

A Page Allocator Self-Test Output

```
******************************************
*** PAGE ALLOCATOR SELF TEST ACTIVATED ***
******************************************
(1) Initial state
*** buddy page allocator - free list ***
[00]
[01]
[02]
[03]
[04]
[05]
[06]
[07]
[08]
[09]
[10]
[11]
[12]
[13]
[14]
[15]
[16]
(2) Insert power-of-two block (PFN=0, COUNT=8)
*** buddy page allocator - free list ***
[00]
[01]
[02]
[03] 0--7fff
[04]
[05]
[06]
[07]
[08]
[09]
[10]
[11]
[12]
[13]
[14]
[15]
[16]
(3) Insert odd block (PFN=13, COUNT=3)
*** buddy page allocator - free list ***
```

```
[00] d000--dfff
[01] e000--ffff
[02]
[03] 0--7fff
[04]
[05]
[06]
[07]
[08]
[09]
[10]
[11]
[12]
[13]
[14]
[15]
[16]
(4) Insert another block (PFN=1300, COUNT=7)
*** buddy page allocator - free list ***
[00] d000--dfff 51a000--51afff
[01] e000--ffff 518000--519fff
[02] 514000--517fff
[03] 0--7fff
[04]
[05]
[06]
[07]
[08]
[09]
[10]
[11]
[12]
[13]
[14]
[15]
[16]
(5) Allocate page (ORDER=0)
  allocated pfn=d, base=d000
*** buddy page allocator - free list ***
[00] 51a000--51afff
[01] e000--ffff 518000--519fff
[02] 514000--517fff
[03] 0--7fff
[04]
[05]
```

```
[06]
[07]
[08]
[09]
[10]
[11]
[12]
[13]
[14]
[15]
[16]
(6) Free page (PFN=d, ORDER=0)
*** buddy page allocator - free list ***
[00] d000--dfff 51a000--51afff
[01] e000--ffff 518000--519fff
[02] 514000--517fff
[03] 0--7fff
[04]
[05]
[06]
[07]
[08]
[09]
[10]
[11]
[12]
[13]
[14]
[15]
[16]
(7) Insert pages (PFN=20, COUNT=20)
*** buddy page allocator - free list ***
[00] d000--dfff 51a000--51afff
[01] e000--ffff 518000--519fff
[02] 14000--17fff 514000--517fff
[03] 0--7fff 18000--1ffff 20000--27fff
[04]
[05]
[06]
[07]
[08]
[09]
[10]
[11]
[12]
```

```
[13]
[14]
[15]
[16]
(8) Allocate page (ORDER=1)
  allocated pfn=e, base=e000
*** buddy page allocator - free list ***
[00] d000--dfff 51a000--51afff
[01] 518000--519fff
[02] 14000--17fff 514000--517fff
[03] 0--7fff 18000--1ffff 20000--27fff
[04]
[05]
[06]
[07]
[08]
[09]
[10]
[11]
[12]
[13]
[14]
[15]
[16]
(9) Free page (PFN=e, ORDER=1)
*** buddy page allocator - free list ***
[00] d000--dfff 51a000--51afff
[01] e000--ffff 518000--519fff
[02] 14000--17fff 514000--517fff
[03] 0--7fff 18000--1ffff 20000--27fff
[04]
[05]
[06]
[07]
[08]
[09]
[10]
[11]
[12]
[13]
[14]
[15]
[16]
(10) Insert one page (PFN=2, ORDER=0)
*** buddy page allocator - free list ***
```

```
[00] 2000--2fff d000--dfff 51a000--51afff
[01] e000--ffff 518000--519fff
[02] 14000--17fff 514000--517fff
[03] 0--7fff 18000--1ffff 20000--27fff
[04]
[05]
[06]
[07]
[08]
[09]
[10]
[11]
[12]
[13]
[14]
[15]
[16]
(11) Allocate page (ORDER=3)
  allocated pfn=0, base=0
*** buddy page allocator - free list ***
[00] 2000--2fff d000--dfff 51a000--51afff
[01] e000--ffff 518000--519fff
[02] 14000--17fff 514000--517fff
[03] 18000--1ffff 20000--27fff
[04]
[05]
[06]
[07]
[08]
[09]
[10]
[11]
[12]
[13]
[14]
[15]
[16]
(12) Free one page in middle of allocation (PFN=1, ORDER=0)
*** buddy page allocator - free list ***
[00] 1000--1fff 2000--2fff d000--dfff 51a000--51afff
[01] e000--ffff 518000--519fff
[02] 14000--17fff 514000--517fff
[03] 18000--1ffff 20000--27fff
[04]
[05]
```

```
[06]
[07]
[08]
[09]
[10]
[11]
[12]
[13]
[14]
[15]
[16]
(13) Free one page at start of allocation (PFN=0, ORDER=0)
*** buddy page allocator - free list ***
[00] 2000--2fff d000--dfff 51a000--51afff
[01] 0--1fff e000--ffff 518000--519fff
[02] 14000--17fff 514000--517fff
[03] 18000--1ffff 20000--27fff
[04]
[05]
[06]
[07]
[08]
[09]
[10]
[11]
[12]
[13]
[14]
[15]
[16]
(14) Insert page to trigger higher merge (PFN=40, ORDER=3)
*** buddy page allocator - free list ***
[00] 2000--2fff d000--dfff 51a000--51afff
[01] 0--1fff e000--ffff 518000--519fff
[02] 14000--17fff 514000--517fff
[03] 18000--1ffff
[04] 20000--2ffff
[05]
[06]
[07]
[08]
[09]
[10]
[11]
[12]
```

```
[13]
[14]
[15]
[16]
(15) Allocate too big
good!! allocation failed
*** buddy page allocator - free list ***
[00] 2000--2fff d000--dfff 51a000--51afff
[01] 0--1fff e000--ffff 518000--519fff
[02] 14000--17fff 514000--517fff
[03] 18000--1ffff
[04] 20000--2ffff
[05]
[06]
[07]
[08]
[09]
[10]
[11]
[12]
[13]
[14]
[15]
[16]
*** SELF TEST COMPLETE - SYSTEM TERMINATED ***
```