Operating Systems

Practical Coursework 2

Tom Spink
tspink@inf.ed.ac.uk

February 2019

Coursework Task 1

Task 1 was to implement a round-robin scheduler

Coursework Task 1

Answer!

Coursework Task 1

Notes:

- `UniqueIRQLock`
- `scl enable devtoolset-7`
- Other

Coursework Task 1

Notes:

- `UniqueIRQLock`
- `scl enable devtoolset-7`
- Other

Coursework Task 1

Notes:

- `UniqueIRQLock`
- `scl enable devtoolset-7`
- Other

Task 2

Buddy Memory Allocator

Due: Thursday 7th March, 2019 @ 4PM GMT

Worth 60 marks

Task 2: Buddy Memory Allocator

- Two types of memory allocators in InfOS:
  - Page Allocator
  - Object Allocator
- InfOS has an interface for physical memory allocation called the page allocation algorithm
- Your job is to implement this interface, by creating a buddy memory allocator

Task 2: Buddy Memory Allocator

- (mm/mm.cpp)
- mm/page-allocator.cpp
- mm/simple-page-alloc.cpp
  - Simple, and inefficient, linear scan.
  - Does not use the next_free pointer.
- include/infos/mm/page-allocator.h
  - Contains PageDescriptor structure.
  - You do not (and should not) modify the type field.

Task 2: Buddy Memory Allocator

- Provided skeleton is `buddy.cpp`
- You are given these useful methods:
  - `insert_block`
  - `remove_block`
- Implement these six methods:
  - `split_block` (helper)
  - `merge_block` (helper)
  - `alloc_pages`
  - `free_pages`
  - `reserve_page`
  - `init`

Task 2: Buddy Memory Allocator

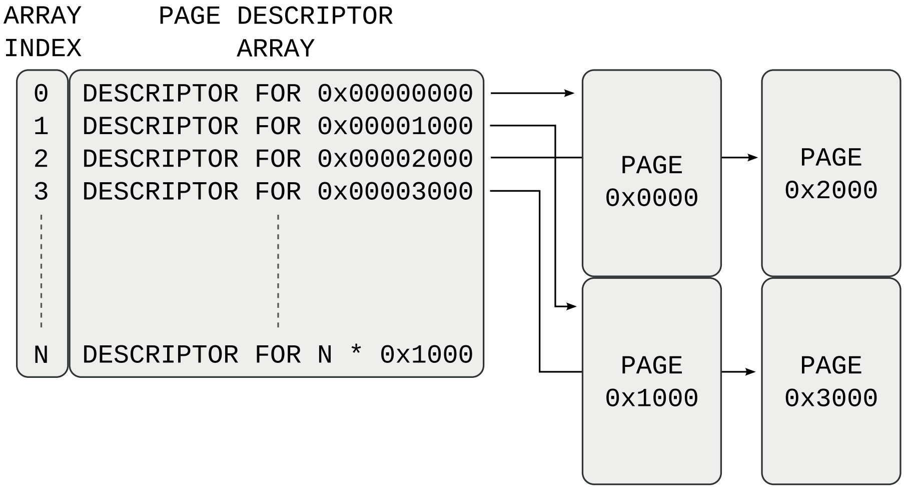
- Provided skeleton is `buddy.cpp`
- You are given these useful methods:
  - `insert_block`
  - `remove_block`
- Implement these six methods:
  - `split_block` (helper)
  - `merge_block` (helper)
  - `alloc_pages`
  - `free_pages`
  - `reserve_page`
  - `init`

Task 2: Buddy Memory Allocator

- Provided skeleton is `buddy.cpp`
- You are given these useful methods:
  - `insert_block`
  - `remove_block`
- Implement these six methods:
  - `split_block` (helper)
  - `merge_block` (helper)
  - `alloc_pages`
  - `free_pages`
  - `reserve_page`
  - `init`

Task 2: Buddy Memory Allocator

- Page allocator returns page descriptors NOT pointers.
- One page descriptor for every physical page.
- Page descriptors held in a contiguous array.
- Page descriptors in the array have a one-to-one mapping to contiguous physical pages.
- If you have a pointer to a page descriptor, then advancing the pointer moves to the next page descriptor, and hence the next physical page.


ARRAY INDEX
PAGE DESCRIPTOR ARRAY
0
1
2
3
N
DESCRIPTOR FOR 0x00000000
DESCRIPTOR FOR 0x00001000
DESCRIPTOR FOR 0x00002000
DESCRIPTOR FOR 0x00003000
DESCRIPTOR FOR N * 0x1000
PAGE 0x0000
PAGE 0x2000
PAGE 0x1000
PAGE 0x3000

Task 2: Buddy Memory Allocator

- Page Descriptor structure contains a `next_free` pointer.
- Use this to build linked-lists.
- You cannot use the `List<>` or `Map<>` containers, and you cannot allocate memory.

`alloc_pages`

- Allocates by order, not by size or count.
- Always returns contiguous pages, by returning first page descriptor in a sequence.
- Order 0 allocation means $2^0 = 1$ pages.
- Order 4 allocation means $2^4 = 16$ pages.
- Use `split_block` here.

free_pages

- Counter-part to alloc_pages
- Frees by order, not by size or count.
- Always frees contiguous pages, by accepting first page descriptor in a sequence.
- Use merge_block here.

`reserve_page`

- Called by the kernel to mark a specific page as allocated.
- Your allocator sees the entire physical memory as one big blob.
- Therefore, your allocator must be told which pages contain the kernel, so you do not allocate those pages!
- Accepts a single page descriptor, you must remove it from your free lists (following the buddy algorithm)
- Use `split_block` here.

`init`

- Your opportunity to initialise the free lists.

Task 2: Buddy Memory Allocator

- Test by using the `build-and-run.sh` script
  - `./build-and-run.sh pgalloc.algorithm=buddy`
- If your implementation is broken, it's likely that the system will hang.
  - Although you could get away with not implementing free_pages, the self-test will fail if this doesn't work.
- Use the self-test mode to test the memory allocator.
  - `./build-and-run.sh pgalloc.algorithm=buddy pgalloc.self-test=1`
- There are no shell test commands, but being able to run any command in the shell is a good indication that your allocator is working.
- Modify the skeleton however you want, but you should only need to implement the six functions (technically four if you don't want to implement the helpers).

Task 2: Buddy Memory Allocator

- Test by using the `build-and-run.sh` script
  - `./build-and-run.sh pgalloc.algorithm=buddy`
- If your implementation is broken, it's likely that the system will hang.
  - Although you could get away with not implementing free_pages, the self-test will fail if this doesn't work.
- Use the self-test mode to test the memory allocator.
  - `./build-and-run.sh pgalloc.algorithm=buddy pgalloc.self-test=1`
- There are no shell test commands, but being able to run any command in the shell is a good indication that your allocator is working.
- Modify the skeleton however you want, but you should only need to implement the six functions (technically four if you don't want to implement the helpers).

Task 2: Buddy Memory Allocator

- Test by using the `build-and-run.sh` script
  - `./build-and-run.sh pgalloc.algorithm=buddy`
- If your implementation is broken, it's likely that the system will hang.
  - Although you could get away with not implementing free_pages, the self-test will fail if this doesn't work.
- Use the self-test mode to test the memory allocator.
  - `./build-and-run.sh pgalloc.algorithm=buddy pgalloc.self-test=1`
- There are no shell test commands, but being able to run any command in the shell is a good indication that your allocator is working.
- Modify the skeleton however you want, but you should only need to implement the six functions (technically four if you don't want to implement the helpers).

Task 2: Buddy Memory Allocator

- Test by using the `build-and-run.sh` script
  - `./build-and-run.sh pgalloc.algorithm=buddy`
- If your implementation is broken, it's likely that the system will hang.
  - Although you could get away with not implementing free_pages, the self-test will fail if this doesn't work.
- Use the self-test mode to test the memory allocator.
  - `./build-and-run.sh pgalloc.algorithm=buddy pgalloc.self-test=1`
- There are no shell test commands, but being able to run any command in the shell is a good indication that your allocator is working.
- Modify the skeleton however you want, but you should only need to implement the six functions (technically four if you don't want to implement the helpers).

Task 2: Buddy Memory Allocator

- Test by using the `build-and-run.sh` script
  - `./build-and-run.sh pgalloc.algorithm=buddy`
- If your implementation is broken, it's likely that the system will hang.
  - Although you could get away with not implementing free_pages, the self-test will fail if this doesn't work.
- Use the self-test mode to test the memory allocator.
  - `./build-and-run.sh pgalloc.algorithm=buddy pgalloc.self-test=1`
- There are no shell test commands, but being able to run any command in the shell is a good indication that your allocator is working.
- Modify the skeleton however you want, but you should only need to implement the six functions (technically four if you don't want to implement the helpers).

Self-test Output

```
notice: mm: PAGE ALLOCATOR SELF TEST - BEGIN
notice: mm: ------------------------
  info: mm: * INITIAL STATE
 debug: mm: BUDDY STATE:
 debug: mm: [0]
 debug: mm: [1]
 debug: mm: [2]
 debug: mm: [3]
 debug: mm: [4]
 debug: mm: [5]
 debug: mm: [6]
 debug: mm: [7]
 debug: mm: [8]
 debug: mm: [9]
 debug: mm: [10]
 debug: mm: [11]
 debug: mm: [12]
 debug: mm: [13]
 debug: mm: [14]
 debug: mm: [15]
 debug: mm: [16] 0 10000 20000 30000 40000 50000 60000 70000 80000 90000 a0000 b0000 c0000 d0000 e0000 f0000 100000 110000
   120000 130000 140000
```

Self-test Output

```
 info: mm: ------------------------
 info: mm: (1) ALLOCATING ONE PAGE
 info: mm: ALLOCATED PFN: 0x0
debug: mm: BUDDY STATE:
debug: mm: [0] 1
debug: mm: [1] 2
debug: mm: [2] 4
debug: mm: [3] 8
debug: mm: [4] 10
debug: mm: [5] 20
debug: mm: [6] 40
debug: mm: [7] 80
debug: mm: [8] 100
debug: mm: [9] 200
debug: mm: [10] 400
debug: mm: [11] 800
debug: mm: [12] 1000
debug: mm: [13] 2000
debug: mm: [14] 4000
debug: mm: [15] 8000
debug: mm: [16] 10000 20000 30000 40000 50000 60000 70000 80000 90000 a0000 b0000 c0000 d0000 e0000 f0000 100000 110000
  120000 130000 140000
```

Questions/Clarifications?