OS 2025 MP2
Memory Management: Kernel Memory Allocation (slab)

TAs: 楊子慶、黃敬瑋

Due Date: 2025/04/03 (Thursday) 23:59:59

Email: ntuos@googlegroups.com

TA hours: Wed. 13:00~14:00, Fri. 11:00~12:00 **CSIE B04**

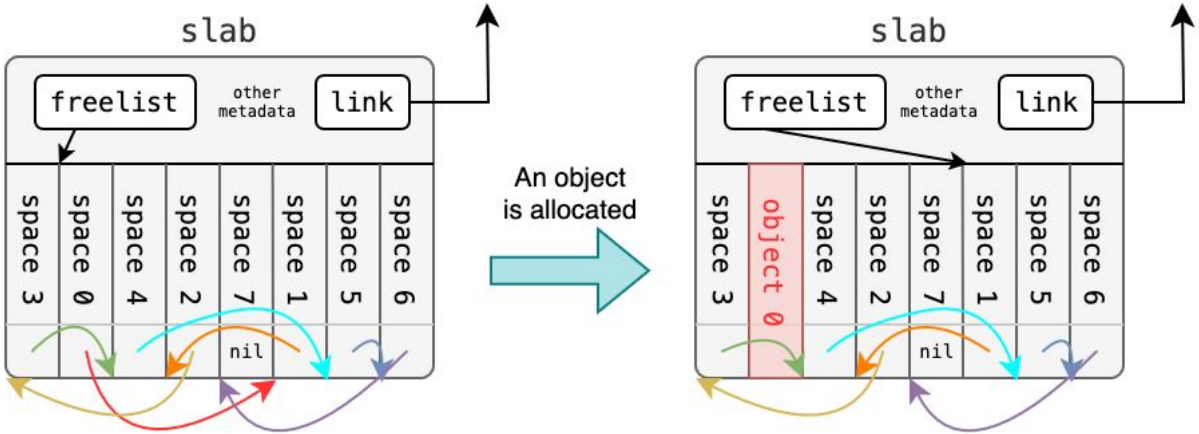
Guideline

- Introduction
- Environment Setting
- Code Structure
- More about Git
- Slab Design
- Implementation Requirement
- Testing
- Reference

Introduction

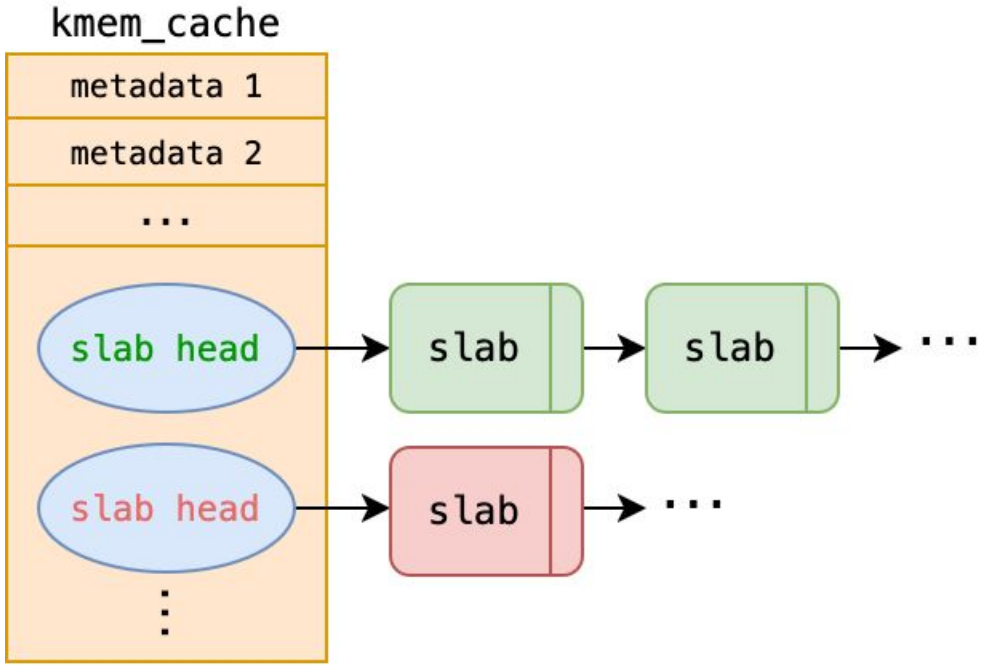
- Imaging kernel metadata (e.g., file) need to be allocated in kernel
- If just simply allocate a page, too many space will be wasted
- Slab is a solution of kernel memory management
  - The space is split to segments, which size is equal to object size
  - It maintain a "`freelist`", which is a linked list of free space

```
struct file {
  enum { FD_NONE,
    FD_PIPE,
    FD_INODE,
    FD_DEVICE } type;
  int ref;
  char readable;
  char writable;
  struct pipe *pipe;
  struct inode *ip;
  uint off;
  short major;
};
```



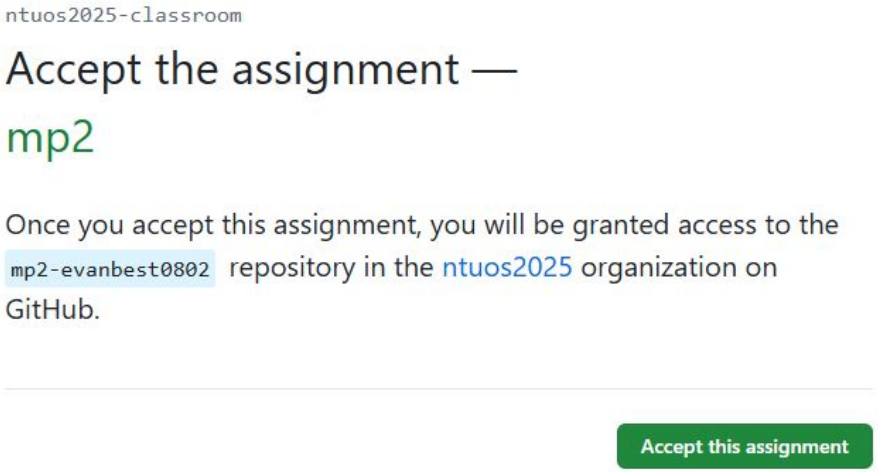
Introduction

- Another structure, kernel memory cache (`kmem_cache`) is used to manage all slabs containing the same type of object
- For example, the system may create a separate `kmem_cache` for `struct file` or `struct proc`



Environment Setting

- Install Git
- Register a GitHub account
- Access the MP2-specific Github Classroom Link and click “Accept this assignment”. The system will create a dedicated repository for you: **mp2-<USERNAME>**.



Environment Setting

- Clone repository: `git clone https://github.com/ntuos2025/mp2-<USERNAME>`
- Fill in your student ID in `student_id.txt` in the repository, e.g., **b12345678**
- Run the `mp2.sh` MP2 script tool: `./mp2.sh setup`
- After development, run specific tests: `./mp2.sh test [case]`
- For further usage of `mp2.sh`, please refer to specification

Environment Setting (Optional)

- Set up VS Code development environment inside the container
  - Install the Docker and Dev Containers extensions.
  - Open VS Code, go to the Docker sidebar, and locate ntuos/mp2.
  - Right-click and select Attach Visual Studio Code.
  - Choose ntuos/mp2. VS Code will open a new development environment, allowing direct development within the container.

Code Structure

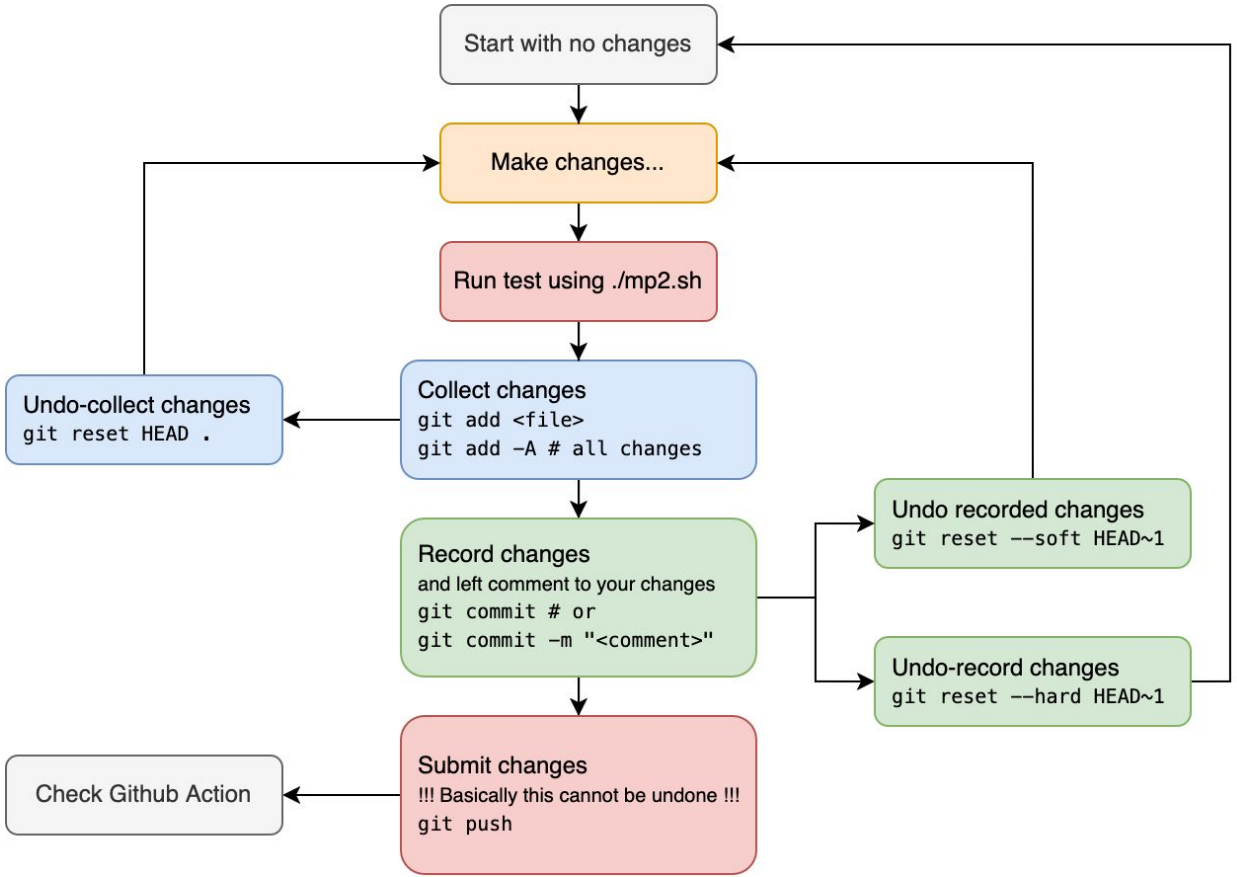
- The repository contains the following files and directories:
  - `student_id.txt`: A file to assist with grading; please ensure you fill in your student ID.
  - `mp2.sh`: A tool for managing the MP2 assignment.
  - `doc/`: Specification requirements for this MP2 assignment in both Chinese and English (Markdown + PDF).
  - `scripts/`: Various helper scripts.
  - `test/`: A testing system based on the original source code Makefile.
  - `.github/`: Code related to GitHub Actions.
  - `update.md`: Update logs for students.
  - `update.sh`: A script that can update restricted files and documents.
    Usage: `./update.sh`

More about Git

- After `git clone` the repository...
  - Modify code
  - Add your changes to repository: `git add <file name>`
  - Commit the change: `git commit`
  - Push the change to Github: `git push`
  - If you want to refine your code, go back to first step
- Other utilities
  - Check which files change: `git status`
  - Check commit history: `git log`
- For other details of Git, you can refer to docs

More about Git

- A development flow using Git



Slab design: struct slab

- We encourage students to minimize memory footprint so as to maximize memory utilization.
- `freelist`: linked list of free space
- `next` and `prev`: Let slab be a linked list

```
struct slab {
    <ptr> freelist;

    {   // Slab list linkage pointers
        <ptr> <next>;
        <ptr> <prev>; // Optional, depending on situational needs
    }
    ... // Other fields can be extended as needed
};
```

Slab design: struct kmem_cache

- Usually contains 3 linked lists of slabs
  - `full`: All objects are allocated.
  - `partial`: Some objects remain available.
  - `free`: Unused slabs.
- `full` and `free` are actually optional members (please refer to specification)

```
struct kmem_cache {
    char name[MP2_CACHE_MAX_NAME];
    uint object_size;

    <ptr> full;    // Points to the list of full Slabs
    <ptr> partial; // Points to the list of partially used Slabs
    <ptr> free;    // Points to the list of free Slabs

    ... // Other fields can be extended as needed
};
```

Slab design: struct kmem_cache

- `kmem_cache` use spinlock to ensure thread safety.
- The right figure is the usage when `kmem_cache` needs to be modified

```
struct kmem_cache {
    ...
    struct spinlock lock;  // For synchronizing kmem_cache management
};

void some_func() {
    struct kmem_cache *cache = kmem_cache_create(...);

    // Enter critical section to prevent race conditions
    acquire(&cache->lock);

    // Critical operation that may cause contention
    cache->partial = NULL;

    // Release lock, exit critical section
    release(&cache->lock);
}
```

Slab design: struct kmem_cache

- In API developing, use conservative approach: acquire `lock` at the beginning and release `lock` before return.

```
void some_api(struct kmem_cache *cache, ...)
{
    acquire(&cache->lock);

    if (...) {
        ...
        release(&cache->lock);
        return;
    }
    ...

    release(&cache->lock);
    return;
}
```

Slab design: Functions Overview

- Function signature:

```
// Initialize a slab allocator to manage system objects named "name" with size
"object_size"
struct kmem_cache *kmem_cache_create(char *name, uint object_size);

// Allocate a system object and return it
void *kmem_cache_alloc(struct kmem_cache *cache);

// Free a previously allocated system object "obj"
void kmem_cache_free(struct kmem_cache *cache, void *obj);

// Destroy the kmem_cache (not included in grading)
void kmem_cache_destroy(struct kmem_cache *cache);
```

Slab design: Functions Overview

- Usage:

```
// Initialize a slab allocator for struct file
struct kmem_cache *file_cache = kmem_cache_create("file", sizeof(struct file));

// Allocate a struct file object
struct file *file_allocated = (struct file *) kmem_cache_alloc(file_cache);

// Free a struct file object
kmem_cache_free(file_cache, file2release);

// Destroy the file_cache when no longer needed
kmem_cache_destroy(file_cache);
```

Slab design: kmem_cache_create()

```
struct kmem_cache *kmem_cache_create(char *name, uint object_size);
```

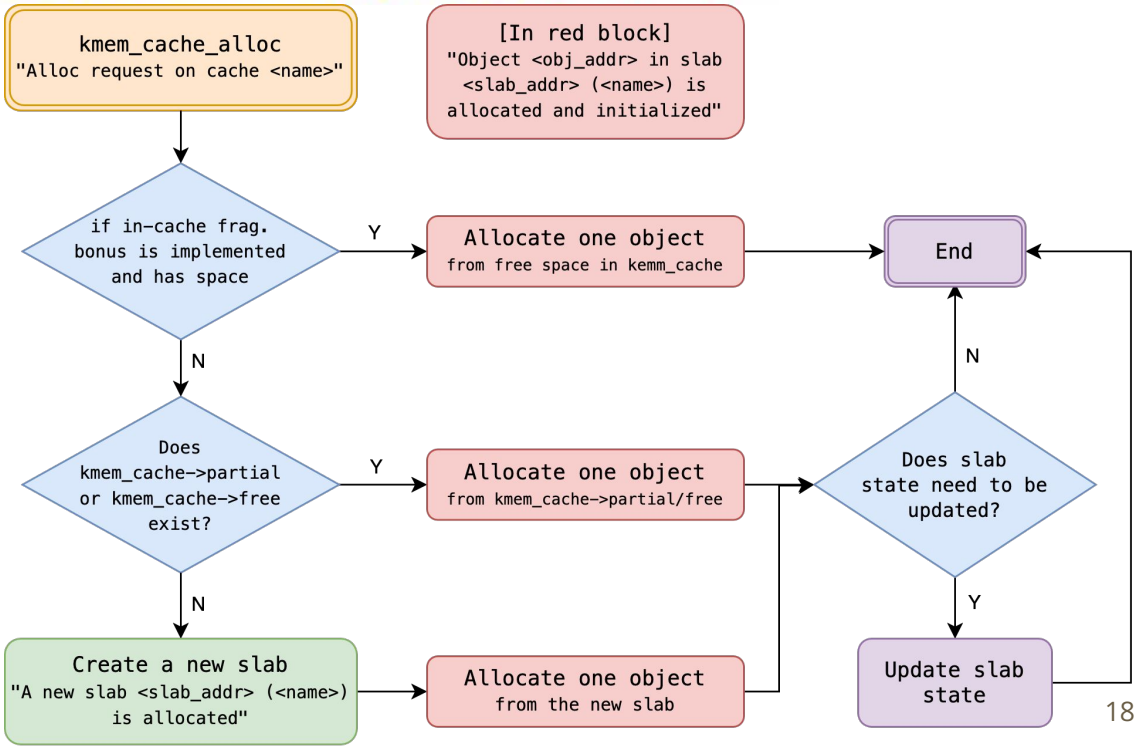
- **`name`**: Name of the new `kmem_cache` (`kmem_cache.name`).
- **`obj_size`**: Size of objects within the kmem_cache (`kmem_cache.object_size`, in bytes).
- Before successfully creating and returning `kmem_cache`, print:

`[SLAB] New kmem_cache (name: <name>, object size: <obj_size> bytes, at: <kmem_cache_addr>, max objects per slab: <max_objs>, support in cache obj: <in_cache_obj>) is created`

Slab design: kmem_cache_alloc()

```
void *kmem_cache_alloc(struct kmem_cache *cache);
```

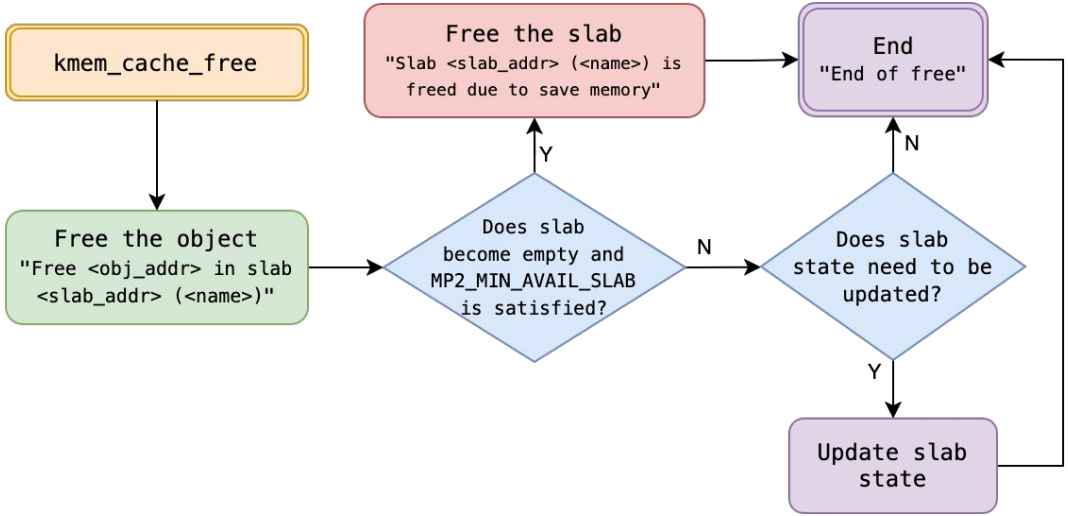
- **`name`**: Name of the `kmem_cache` (`cache.name`)
- **`slab_addr`**: Memory address of the slab containing the object
- **`obj_addr`**: Memory address of the allocated object



Slab design: kmem_cache_free()

```
void kmem_cache_free(struct kmem_cache *cache, void *obj);
```

- **`name:`** Name of the kmem_cache (`kmem_cache.name`)
- **`slab_addr:`** Memory address of the slab containing the object
- **`obj_addr:`** Memory address of the object to be freed
- **`before:`** State of the object's slab before freeing (full/partial/free/cache)
- **`after:`** State of the object's slab after freeing (full/partial/free/cache)



Slab design: sys_printfslab() and print_kmem_cache()

- `void print_kmem_cache(struct kmem_cache *, void (*)(void *));`
  - Print `struct kmem_cache` information
  - For format details, please refer to specification.
- `sys_printfslab()` is a system call
  - Usage:

```c
#include "kernel/types.h"
#include "user/user.h"

int main(int argc, char *argv[])
{
  printfslab();
}
```

Implementation Requirement

- Ensure your student ID is filled in the `student_id.txt` file.
- Modification of these files is prohibited:
  - `mp2.h`
  - `scripts/action_grader.h, scripts/pre-commit`
  - `kernel/main.c, kernel/mp2_checker.h, kernel/file.h, kernel/list.h, kernel/param.h`
  - All code within the `.github/` directory
  - All files within the `test/` directory, except `test/custom/mytest.txt`
  - All existing files within the `user/` directory; students may additionally implement other user programs for testing purposes
- Students are free to add new files and modify other code.
- Using `./mp2.sh setup` can automatically prevent students from attempting to submit problematic changes.

Implementation Requirement: debug

- We offer debugging tool
  - In xv6, use `debugswitch` to turn on/off debug messages (default mode is “on”)
  - The debug message API is below. The format is the same as `printf`
  - All output must be handled using the debug API to ensure compatibility with the testing and grading system
  - For detailed usage instructions, please refer to kernel/debug.h

```
// in file.c
#include "debug.h" // Include the kernel debugging tool
// Formatted output
debug("tp: %d, ref: %d, readable: %d, ...",
      file->type, file->ref, file->readable, ...);
// Standard string output
debug("[FILE] filealloc");
```

```
$ debugswitch
Switch debug mode to 0
$ ls
.               1 1 1024
..              1 1 1024
README          2 2 2292
cat             2 3 34728
...
console         3 22 0
$ debugswitch
Switch debug mode to 1
$ ls
[FILE] filealloc
[FILE] filealloc
[FILE] fileclose
.               1 1 1024
[FILE] filealloc
[FILE] fileclose
..              1 1 1024
[FILE] filealloc
[FILE] fileclose
README          2 2 2292
...
[FILE] filealloc
[FILE] fileclose
console         3 22 0
[FILE] fileclose
```

Testing

- Slab structure test (5% + 5% bonus)
- Functionality test (75%)
  - If the functionality test score exceeds 66 points, the following two bonus tests will be conducted; otherwise, the bonus item evaluation will be skipped:
    - List API Test (+10%)
    - In-Cache Test (+10%)
  - Note: This logic differs from the execution method of `./mp2.sh test all`. The latter is designed to allow students to test all cases in one go, regardless of their score.
- Hidden test (20%)
- Check "mp2.sh Script Usage Guide" in specification for other testing command options

Reference

- xv6: a simple, Unix-like teaching operating system
- ISO/IEC 9899:2024 (C Language Standard)
- linux/mm/slab.h
- linux/mm/slub.c
- sysprog21/lab0-c
- linux/include/linux/list.h
- Slab Memory Allocator

Q&A