MISYS

Operating Systems Practice

4: Slab allocator

Kilho Lee

Xv6 Memory Allocator

- `kalloc()` in kalloc.c
  - `freelist` : singly linked list
  - Allocation unit: 4KB page

```
79 // Allocate one 4096-byte page of physical memory.
80 // Returns a pointer that the kernel can use.
81 // Returns 0 if the memory cannot be allocated.
82 char*
83 kalloc(void)
84 {
85   struct run *r;
86
87   if(kmem.use_lock)
88     acquire(&kmem.lock);
89   r = kmem.freelist;
90   if(r)
91     kmem.freelist = r->next;
92   if(kmem.use_lock)
93     release(&kmem.lock);
94   return (char*)r;
95 }
96
```

```
20 struct {
21   struct spinlock lock;
22   int use_lock;
23   struct run *freelist;
24 } kmem;
```

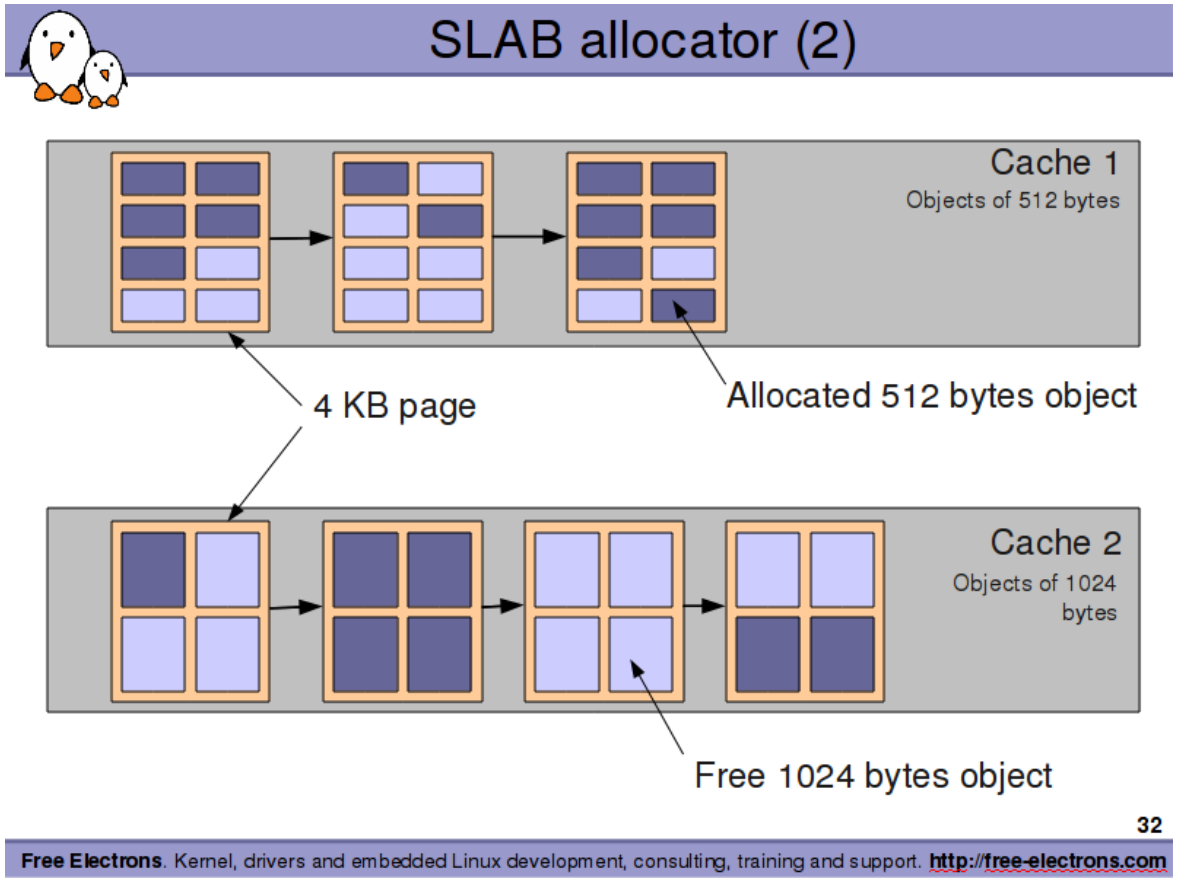
Xv6 Memory Allocator

- kfree in kalloc.c
  - The The free page list is embedded into the empty page

```
54 //PAGEBREAK: 21
55 // Free the page of physical memory pointed at by v,
56 // which normally should have been returned by a
57 // call to kalloc().  (The exception is when
58 // initializing the allocator; see kinit above.)
59 void
60 kfree(char *v)
61 {
62   struct run *r;
63
64   if((uint)v % PGSIZE || v < end || V2P(v) >= PHYSTOP)
65     panic("kfree");
66
67   // Fill with junk to catch dangling refs.
68   memset(v, 1, PGSIZE);
69
70   if(kmem.use_lock)
71     acquire(&kmem.lock);
72   r = (struct run*)v;
73   r->next = kmem.freelist;
74   kmem.freelist = r;
75   if(kmem.use_lock)
76     release(&kmem.lock);
77 }
```

Slab Allocator

- A cache that contains a set of objects of the same size
- Used for data structures that are present in many instances in the kernel
  - `struct dentry`
  - `struct inode`
  - `struct page`
  - `struct task_struct`
  - `struct file`
  - etc.
- Pre-defined fixed size cache is called general cache
  - 32, 128, 512, … bytes



Project 3. Byte-level Memory Allocator

- Implement a byte-level slab memory allocator on xv6
- Individual allocator per fixed size of bytes
  - 16, 32, 64, 128, 256, 512, 1024, and 2048 bytes
- New files
  - slab.c, slab.h, sysslab.c, slabtest.c
- Your target
  - Edit the slab.c file only

Slab Cache Overview

main.c

```
17 int
18 main(void)
19 {
20   kinit1(end, P2V(4*1024*1024)); // phys page allocator
21   kvmalloc();      // kernel page table
22   mpinit();        // detect other processors
23   lapicinit();     // interrupt controller
24   seginit();       // segment descriptors
25   picinit();       // disable pic
26   ioapicinit();    // another interrupt controller
27   consoleinit();   // console hardware
28   uartinit();      // serial port
29   pinit();         // process table
30   tvinit();        // trap vectors
31   binit();         // buffer cache
32   fileinit();      // file table
33   ideinit();       // disk
34   startothers();   // start other processors
35   kinit2(P2V(4*1024*1024), P2V(PHYSTOP)); // must come after startothers(
   )
36   userinit();      // first user process
37   mpmain();        // finish this processor's setup
38 }
```

slab.c

```
 9 struct {
10   struct spinlock lock;
11   struct slab slab[NSLAB];
12 } stable;
13
14 void slabinit(){
15   /* fill in the blank */
16 }
17
18 char *kmalloc(int size){
19   /* fill in the blank */
20   return 0;
21 }
22
23 void kmfree(char *addr, int size){
24   /* fill in the blank */
25 }
26
27 void slabdump(){
28   cprintf("__slabdump__\n");
29
30   struct slab *s;
31
32   cprintf("size\tnum_pages\tused_objects\tfree_objects\n");
33
34   for(s = stable.slab; s < &stable.slab[NSLAB]; s++){
35     cprintf("%d\t%d\t\t%d\t\t%d\n",
36       s->size, s->num_pages, s->num_used_objects, s->num_free_objects);
37   }
38 }
```

It requires some initialization in the “main” function (already added)

Fill up here with your own codes

Project 3. Byte-level Memory Allocator

- slab.h
  - size: size of a slab allocator
  - num_pages: number of pages of a slab allocator
  - num_free_objects: number of free objects of a slab allocator
  - num_used_objects: number of used objects of a slab allocator
  - bitmap: allocation bitmap for a slab allocator (use a page as a bitmap)
  - page: page array for a slab allocator

```
1  #define NSLAB 8 // {16, 32, 64, 128, 256, 512, 1024, 2048}
2  #define MAX_PAGES_PER_SLAB 100
3
4  struct slab {
5      int size;
6      int num_pages;
7      int num_free_objects;
8      int num_used_objects;
9      int num_objects_per_page;
10     char *bitmap;
11     char *page[MAX_PAGES_PER_SLAB];
12 };
13
14 void slabdump();
~
```

DO NOT CHANGE

Project 3. Byte-level Memory Allocator

- slab.c (YOUR TARGET)
  - `stable`: pre-defined slab allocators & lock
  - `void slabinit()`
    - Initialization function
    - Allocate one page for bitmap
    - Allocate one page for slab cache
  - `char *kmalloc(int size)`
    - Byte-level allocation function
    - 0 < size <= 2048
    - Allocate one more page when free space runs out
  - `void kmfree(char *addr, int size)`
    - Free function
    - Memory should be allocated by kmalloc() first
    - IMPLEMENT "page deallocation", if all slabs in each page is released
  - `void slabdump()`
    - Dump function for slab allocator

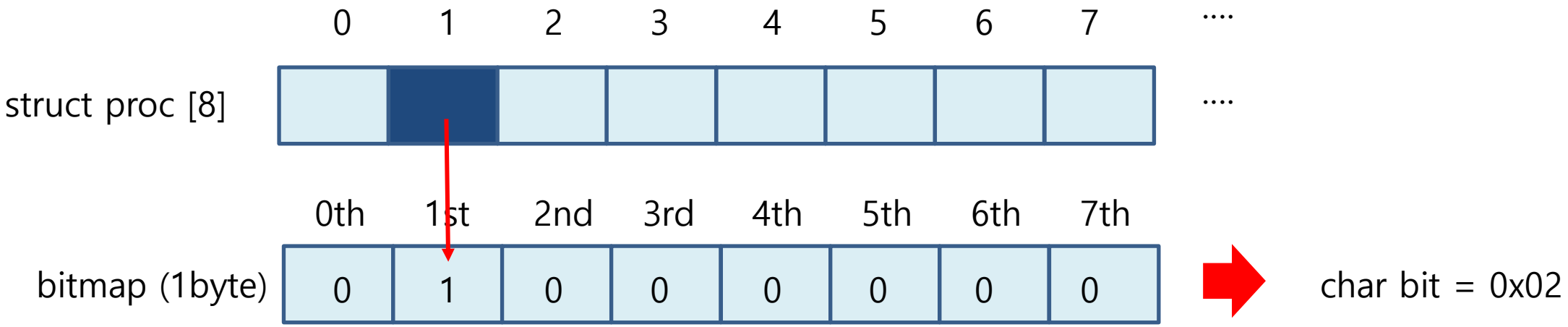
```c
#include "types.h"
#include "defs.h"
#include "param.h"
#include "memlayout.h"
#include "mmu.h"
#include "spinlock.h"
#include "slab.h"

struct {
  struct spinlock lock;
  struct slab slab[NSLAB];
} stable;

void slabinit(){
  /* fill in the blank */
}

char *kmalloc(int size){
  /* fill in the blank */
  return 0;
}

void kmfree(char *addr, int size){
  /* fill in the blank */
}

void slabdump(){
  cprintf("__slabdump__\n");

  struct slab *s;

  cprintf("size\tnum_pages\tused_objects\tfree_objects\n");

  for(s = stable.slab; s < &stable.slab[NSLAB]; s++){
    cprintf("%d\t%d\t\t%d\t\t%d\n",
      s->size, s->num_pages, s->num_used_objects, s->num_free_objects);
  }
}
```

Project 3. Byte-level Memory Allocator

- slabtest system call
  - slabtest.c, sysslab.c
  - We will test the project via various slabtest() functions
    - Please refer to the test code
  - DO NOT MODIFY the slabtest.c

```
/* TEST1: Single slab alloc */
cprintf("==== TEST1 =====\n");
start = counter;
t[0][0] = (int*) kmalloc (TESTSIZE);
*(t[0][0]) = counter;
counter++;

cprintf( (*(t[0][0]) == start && numobj_slab(TESTSLABID) == 1) ? "OK\n":"WRONG\n");
kmfree ((char*) t[0][0], TESTSIZE);

/* TEST2: Single slab alloc: the size not equal to a power of 2. */
cprintf("==== TEST2 =====\n");
start = counter;
t[0][0] = (int*) kmalloc (TESTSIZE-10);
*(t[0][0]) = counter;
counter++;

cprintf( (*(t[0][0]) == start && numobj_slab(TESTSLABID) == 1) ? "OK\n":"WRONG\n");
kmfree ((char*) t[0][0], TESTSIZE);

/* TEST3: Multiple slabs alloc */
cprintf("==== TEST3 =====\n");
start = counter;
for (int i=0; i<NSLAB; i++)
{
    slabsize = 1 << (i+4);
    t[i][0] = (int*) kmalloc (slabsize);
    for (int j=0; j<slabsize/sizeof(int); j++)
    {
        memmove (t[i][0]+j, &counter, sizeof(int));
        counter++;
    }
}

// CHECK
pass = 1;
for (int i=0; i<NSLAB; i++)
{
    slabsize = 1 << (i+4);
    for (int j=0; j < slabsize/sizeof(int); j++)
    {
        //cprintf("%d, %d, %d, %d\n", i, j, *(t[i][0]+j), start);       //YOU MAY USE
        if ( *(t[i][0]+j) != start )
        {
            pass = 0;
            break;
        }
        start++;
    }
}
cprintf( pass ? "OK\n" : "WRONG\n");
```

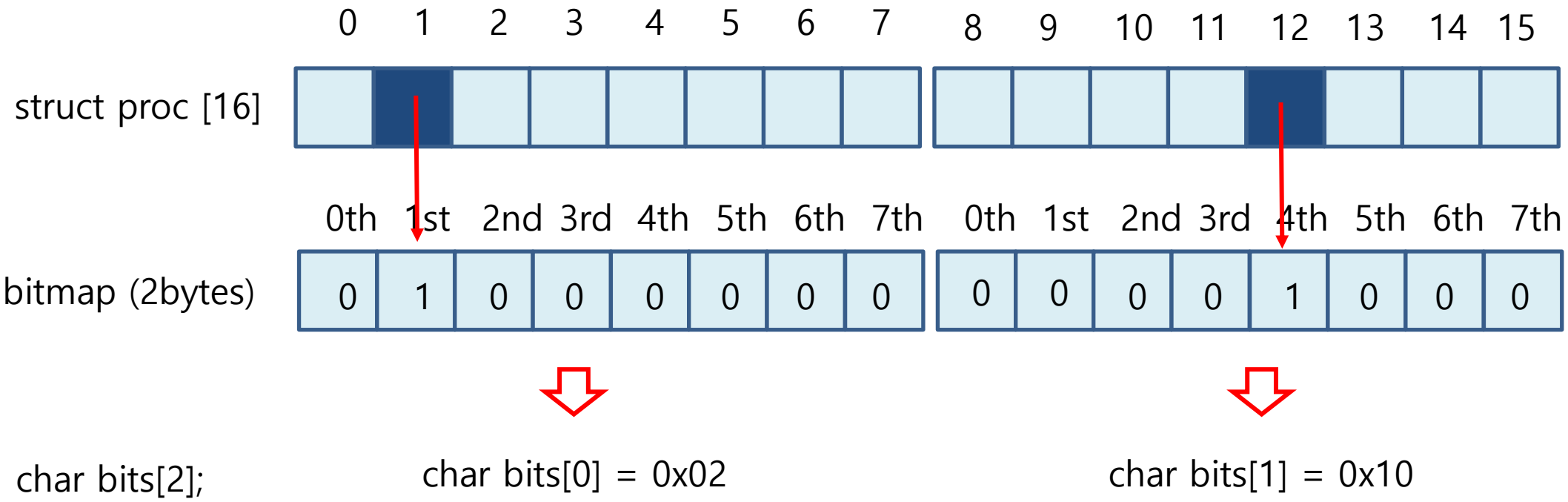
Note: bitmap



Distinguish which element is now allocating with the smallest data (i.e., a bit)

Note: In this project, use a single page (i.e., 4KB) as a bitmap for allocated slabs.
Then the bitmap can distinguish 4*1024*8 slabs.

Note: bitmap


0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
struct proc [16]
0th 1st 2nd 3rd 4th 5th 6th 7th 0th 1st 2nd 3rd 4th 5th 6th 7th
bitmap (2bytes)
0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0
char bits[2];
char bits[0] = 0x02
char bits[1] = 0x10

Note: Bit Manipulation

```
#include<stdio.h>
#include<stdbool.h>


bool get_bit(int num, int i) {
  return ((num & (1 << i)) != 0);
}


int set_bit(int num, int i) {
  return num | (1 << i);
}


int clear_bit(int num, int i) {
  int mask = ~(1 << i);
  return num & mask;
}
```

```
int main()
{
  int num = 0;


  num = set_bit(num, 1);
  printf("%x\n", num);


  if(get_bit(num, 1))
    num = clear_bit(num, 1);
  printf("%x\n", num);


}
```

Calculating Index

- Find a smallest power of 2 greater than or equal to n

C++ C Java Python3 C# PHP

```cpp
// C++ program to find
// smallest power of 2
// greater than or equal to n
#include <bits/stdc++.h>
using namespace std;

unsigned int nextPowerOf2(unsigned int n)
{
    unsigned count = 0;

    // First n in the below condition
    // is for the case where n is 0
    if (n && !(n & (n - 1)))
        return n;

    while( n != 0)
    {
        n >>= 1;
        count += 1;
    }

    return 1 << count;
}

// Driver Code
int main()
{
    unsigned int n = 0;
    cout << nextPowerOf2(n);
    return 0;
}

// This code is contributed by rathbhupendra
```

https://www.geeksforgeeks.org/smallest-power-of-2-greater-than-or-equal-to-n/

Project 3.

- Download template
  - https://github.com/KilhoLee/xv6-ssu.git (pull or clone)
  - `tar xvzf xv6_ssu_slab.tar.gz`

- Implement 3. slab allocator

Project 3.

- Compress your code (ID: 20201234)
  - `$tar cvzf xv6_ssu_slab_20201234.tar.gz xv6_ssu_slab`
  - Please command `$make clean` before compressing

- Submit your `tar.gz` file through class.ssu.ac.kr

- PLEASE DO NOT COPY !!

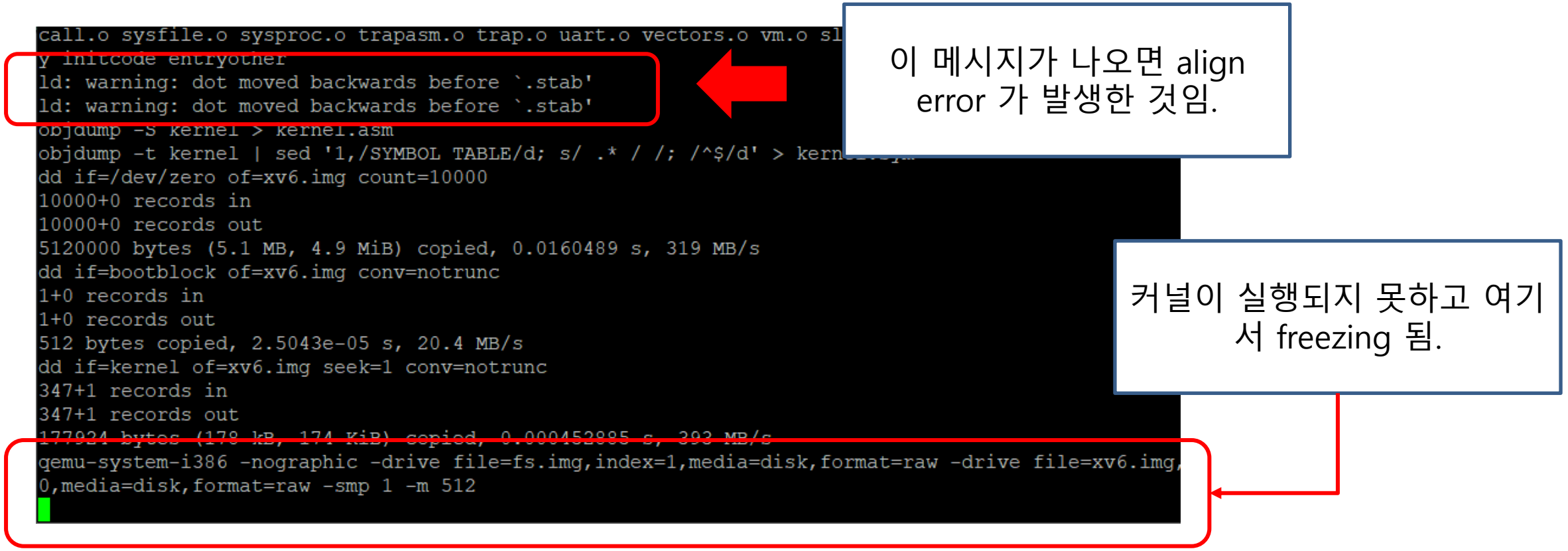
Supplements for P3

Other issues

- dynamic process allocation 구현 시 slabinit() 은 userinit() 보다 먼저 호출되어야 함.
- slabtest 수행 시 실제 xv6 가 보유한 물리 메모리보다 더 많은 양을 요구할 경우 out-of-memory 에러나면서 중단됨.
- 과제 평가 시 총 사용 공간이 5MB 이하인 경우로 테스트 할 예정임
  (함께 제공되는 slabtest.c 파일의 테스트 케이스들을 평가.)

Other issues

- 현재 xv6 kernel 에 memory linking 버그가 존재함. 특히, slabtest.c 파일에서 사용하는 string literal 에 따라, memory align이 제대로 되지 않아 커널이 실행되지 않는 경우가 발생.



Other issues

- 이 경우, slabtest.c 의 string literal 길이를 조절하여 .stab sectio의 alignment 를 강제조절.
  - # objdump –h kernel 명령어 입력시,“.stab” section의 VMA가 4의 배수가 되어야함.

```
root@7513cfb39170:/home/khlee/xv6-ssu/solution/xv6_ssu_slab# objdump -h kernel

kernel:     file format elf32-i386

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         00006efa  80100000  00100000  00001000  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  1 .rodata       00000b4b  80106f00  00106f00  00007f00  2**5
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  2 .stab         00000001  80107a4b  80107a4b  00008a4b  2**0
                  CONTENTS, ALLOC, LOAD, DATA
  3 .stabstr      00000001  80107a4c  80107a4c  00008a4c  2**0
                  CONTENTS, ALLOC, LOAD, DATA
  4 .data         00002516  80108000  80108000  00009000  2**12
```

여기가 4의 배수가 되어야함
예를 들면, 0x80107a4c.
현재 값에서 +1을 해주면 됨.

```
void slabtest(){
  int counter = 1;
  int start
  int pass;
  int* tmp;

  /* NOTE:
   * The cu
   * especi
   * More s
   * to locate the VMA (virtual memory address) of an mu
   * You can check the VMA of .stab by executing 'objdum
   * If the VMA is not ali d to the multiple of 4B, th
   * the literal string i e cprintf function
   */
  cprintf("=== SLAB TEST ====\n");
```

slabtest() 함수의 첫부분에 있는 이 string에 글자를 하나 더 추가 (+1) 한 후 컴파일하면, .stab VMA 위치가 0x80107a4c 로 변해 4의 배수가 되어 정상 작동함.