Operating Systems

Lab. Class

Project #3

Project Plan

- 5 projects
  - Install xv6
  - System call
  - Scheduling
  - Memory
  - Concurrency
  - File system

- Individual projects

Xv6 Memory Allocator

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

```
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 }
```

```
15 struct run {
16   struct run *next;
17 };
18
19 struct {
20   struct spinlock lock;
21   int use_lock;
22   struct run *freelist;
23 } kmem;
```

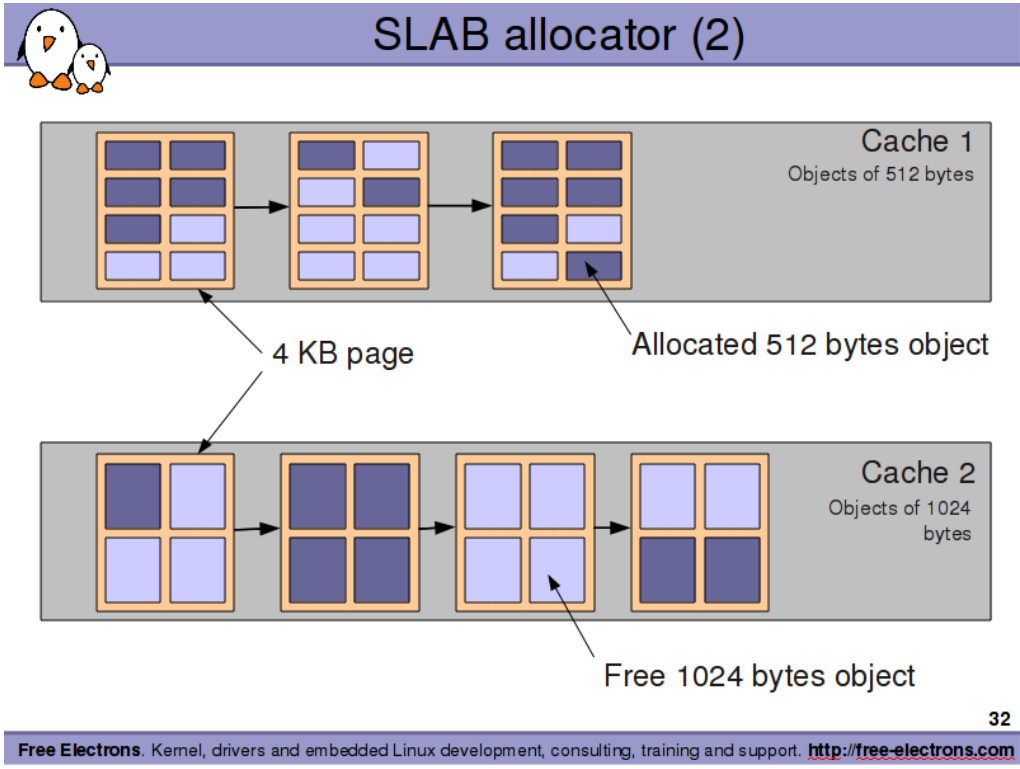
Xv6 Memory Allocator (Cont'd)

- `kfree()` in kalloc.c

```
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
  - Dentry, inode, page, task_struct, file, etc.
- Pre-defined fixed size cache is called general cache
  - 32, 128, 512, … bytes



PA #3 - Byte-level Memory Allocator

- Implement a byte-level memory allocator on xv6
  - Individual allocator per fixed size of bytes
    - 8, 16, 32, 64, 128, 256, 512, 1024, and 2048 bytes
  - New files
    - slab.c, slab.h, sysslab.c, slabtest.c

PA #3 - Implementation Details (1)

- 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
  - page: page array for a slab allocator

```
1  #define NSLAB 9 // {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 };
```

DO NOT CHANGE!!

PA #3 - Implementation Details (2)

- slab.c
  - stable: pre-defined slab allocators & lock
  - `void slabinit();`
    - Initialization function
  - `char *kmalloc(int size);`
    - Byte-level allocation function
    - $0 < size \leq 2048$
  - `void kmfree(char *addr);`
    - Free function
    - Memory should be allocated by kmalloc() first
  - `void slabdump();`
    - Dump function for slab allocator

PA #3 - Implementation Details (3)

- slabtest system call
  - slabtest.c, sysslab.c
  - We will test the project via various `slabtest()` functions

- **DO NOT PRINT ANY DEBUGGING MESSAGE!**

Project Assignment #3 Template Code

- `git clone https://github.com/jinsoox/xv6-skku.git -b pa3`
- Modifications
  - slab.c, slab.h, sysslab.c, slabtest.c
  - slabtest system call
  - Dynamically allocated process structure

Project Assignment #3 Template Code (Cont'd)

- Dynamically allocated process structure (proc.c, proc.h)

```
51 // Per-process state
52 struct proc {
53   uint sz;
54   pde_t* pgdir;
65   char name[16];
66   struct proc *prev;
67   struct proc *next;
68 };
```

```
10 struct {
11     struct spinlock lock;
12     struct proc dummy;
13     struct proc *proc;
14 } ptable;
```

```
40 static struct proc*
41 allocproc(void)
42 {
43     struct proc *p;
44     char *sp;
45
46     acquire(&ptable.lock);
47
48     p = (struct proc *)kmalloc(sizeof(struct proc));
49     memset(p, 0, sizeof(struct proc));
```

```
245 int
246 wait(void)
247 {
269                 p->prev->next = p->next;
270                 p->next->prev = p->prev;
271                 p->state = UNUSED;
272                 kmfree((char *)p);
273                 release(&ptable.lock);
274                 return pid;
```

Project Submission Procedure

- http://sys.skku.edu
  - Submit a tarball file made from "**make tarball**"

- Due date
  - 2017-04-30 23:59

- Since 6nd submission, -5% penalty of the project score
  - Up to 5 submissions are free to accept
- Every one day delay, -25% penalty of the project score
  - You can use up to 5 *slip* days