HW6. XV6 Slab Allocator 구현하기

학번: 20170404

이름: 한종수

제출일: 2021. 5. 11

1. 과제 개요

본 과제에서는 Slab Allocator를 구현하고 test 코드를 통해 올바른 동작을 확인한다.

2. 소스코드

(수정한 코드를 캡처하고 간단히 설명함)

1. Main.c

```
int
main(void)
{
  kinit1(end, P2V(4*1024*1024)); // phys page allocator
  kvmalloc();      // kernel page table
  mpinit();        // detect other processors
  lapicinit();     // interrupt controller
  seginit();       // segment descriptors
  picinit();       // disable pic
  ioapicinit();    // another interrupt controller
  consoleinit();   // console hardware
  uartinit();      // serial port
  pinit();         // process table
  tvinit();        // trap vectors
  binit();         // buffer cache
  fileinit();      // file table
  ideinit();       // disk
  startothers();   // start other processors
  kinit2(P2V(4*1024*1024), P2V(PHYSTOP)); // must come after startothers()
  slabinit();      // slab allocator
  userinit();      // first user process
  mpmain();        // finish this processor's setup
}
```

A. Kinit2함수 아래 slabinit함수를 넣어 slaballocator를 초기화한다.

2. Slabintit()

A.

```
void slabinit()
{
    /* fill in the blank */
    acquire(&stable.lock);
    struct slab *slab_ptr;

    for (int i = 0; i < NSLAB; i++)
    {
        slab_ptr = &stable.slab[i];

        // set slab size
        slab_ptr->size = 0x8 << i;

        // first slab page allocation and initialization
        slab_ptr->page[0] = kalloc();
        memset(slab_ptr->page[0], 1, PGSIZE);
        slab_ptr->num_pages = 1;
        slab_ptr->num_objects_per_page = (int)PGSIZE / slab_ptr->size;
        slab_ptr->num_free_objects = slab_ptr->num_objects_per_page;
        slab_ptr->num_used_objects = 0;

        // bitmap allocation and initialization
        slab_ptr->bitmap = kalloc();
        memset(slab_ptr->bitmap, 0, PGSIZE);
    }
    release(&stable.lock);
}
```

B. Stable의 lock변수의 권한을 취득하고 슬랩 할당자의 슬랩캐시 개수만큼 루프를 돌면서 첫 페이지 할당, 그리고 그 페이지의 초기화, 페이지 개수, 페이지당 오브젝트 수, 현재 사용가능한 오브젝트 수, 사용중인 오브젝트 수, 등 slab구조체의 값을 초기화한다. 비트맵을 kalloc으로 할당받고 비트맵을 모두 0으로 초기화한다.

3. Kmalloc()

A.

```
char *kmalloc(int size)
{
    /* fill in the blank */
    // calculate which slab to put
    int where_to_put = 0;
    where_to_put = nextPowerOf2Idx((unsigned int)size) - 3;
    struct slab *slab_ptr;
    slab_ptr = &stable.slab[where_to_put];

    acquire(&stable.lock);
    // check if slab available
    if (slab_ptr->num_free_objects == 0) // check free object exists
    {
        // no free objects
        if (slab_ptr->num_pages <= MAX_PAGES_PER_SLAB) // check if new page allocation ava
        {
            // available
            slab_ptr->page[slab_ptr->num_pages] = kalloc();
            slab_ptr->num_pages++;
            slab_ptr->num_free_objects += slab_ptr->num_objects_per_page;
        }
        else
        {
            //not available
            cprintf("page allocation limit\n");

            return 0;
        }
    }

    // search bitmap for free space
    int free_slab_idx = 0;                              // free slab index from bitmap
    char *bitmap_ptr = slab_ptr->bitmap; // get bitmap start address
    char *free_space_addr;                              // return value;
    unsigned char temp = 0;                             // temporary space for bitmap cal

    for (int i = 0; i * 8 < slab_ptr->num_pages * slab_ptr->num_objects_per_page; i++, bit
    {
        if (~((unsigned char)*bitmap_ptr) & 0xFF) // if not FF, there's empty space
        {
            temp = ((unsigned char)*bitmap_ptr);
            while ((temp && 0x01) != 0) // search from front
            {
                temp >>= 1;
                free_slab_idx++;
            }

            // check bitmap to 1 and return allocated space
            *bitmap_ptr = ((unsigned char)*bitmap_ptr) | (0x01 << free_slab_idx);
            free_space_addr = slab_ptr->page[(i * 8 + free_slab_idx) / slab_ptr->num_objec
            free_space_addr += (((i * 8 + free_slab_idx) % (slab_ptr->num_objects_per_page
            slab_ptr->num_free_objects--;
            slab_ptr->num_used_objects++;
            release(&stable.lock);

            return free_space_addr;
        }
    }

    release(&stable.lock);

    // something wrong
    cprintf("something wrong\n");

    return 0;
}
```

B. kmalloc에서 size가 들어오면 적절한 슬랩크기를 찾아서 할당해준다. 맨 처음 인자로 들어온 크기보다 큰 가장작은 2진수를 얻는다. 이후 공유변수의 접근 권한을 취득하고 해당 슬랩캐시의 빈 공간이 있는지 확인하여 없는 경우 할당가능한 최대 페이지와 현재 할당된 페이지 수를 비교하여 현재 새페이지 할당가능한 경우 새 페이지를 할당한다.

C. 새롭게 오브젝트를 할당할 수 있는 조건이 확인 된 후에 가장 처음 비어있는 자리에 넣기위해서 비트맵을 탐색한다. 가장 먼저 0이 발견되는 자리를 찾아 그 자리의 페이지, 페이지 내에서의 인덱스를 통해 주소를 계산하고 그 주소를 리턴한다. 종료전 공유변수의 권환을 놔준다.

4. Kmfree()

A.

```
void kmfree(char *addr, int size)
{
    /* fill in the blank */
    // calculate which slab to free
    int where_to_free = 0;
    where_to_free = nextPowerOf2Idx((unsigned int)size) - 3;
    struct slab *slab_ptr;
    slab_ptr = &stable.slab[where_to_free];

    acquire(&stable.lock);
    // calculate index from addr
    char *ptr;
    char *bitmap_ptr = slab_ptr->bitmap; // get bitmap start address

    for (int i = 0; i * 8 < slab_ptr->num_pages * slab_ptr->num_objects_per_page; i++)
    {
        for (int j = 0; j < 8; j++)
        {
            ptr = slab_ptr->page[(i * 8 + j) / slab_ptr->num_objects_per_page];
            ptr += (((i * 8 + j) % (slab_ptr->num_objects_per_page) * slab_ptr->size));

            if (ptr == addr)
            {
                bitmap_ptr += i;
                *bitmap_ptr = (((unsigned char)*bitmap_ptr) & ~(1 << j));
                slab_ptr->num_free_objects++;
                slab_ptr->num_used_objects--;
                memset(addr, 1, slab_ptr->size);
                release(&stable.lock);

                return;
            }
        }
    }

    release(&stable.lock);

    // something wrong
    cprintf("something wrong\n");
    cprintf("input addr : %p\n", addr);
}
```

B. kmfree에서도 kmalloc과 마찬가지로 size가 들어오면 적절한 슬랩크기를 찾는다. 이후 공유 변수를 접근하기 때문에 권한을 취득한다. 그리고 인자로 들어온 주소를 찾기 위해 해당 슬랩 캐시의 페이지들을 순차적으로 탐색하며 페이지 내의 인덱스들의 주소와 일치하는 주소를 찾는다. 일치하는 주소를 찾은 경우에 해당 비트맵의 비트 0으로 초기화, 사용중인 오브젝트 수감소, 사용가능한 오브젝트 수 증가, 반납된 메모리 영역 1로 초기화 등의 작업을 수행한 후, 함수가 종료된다. 종료전 공유변수의 권환을 놔준다.

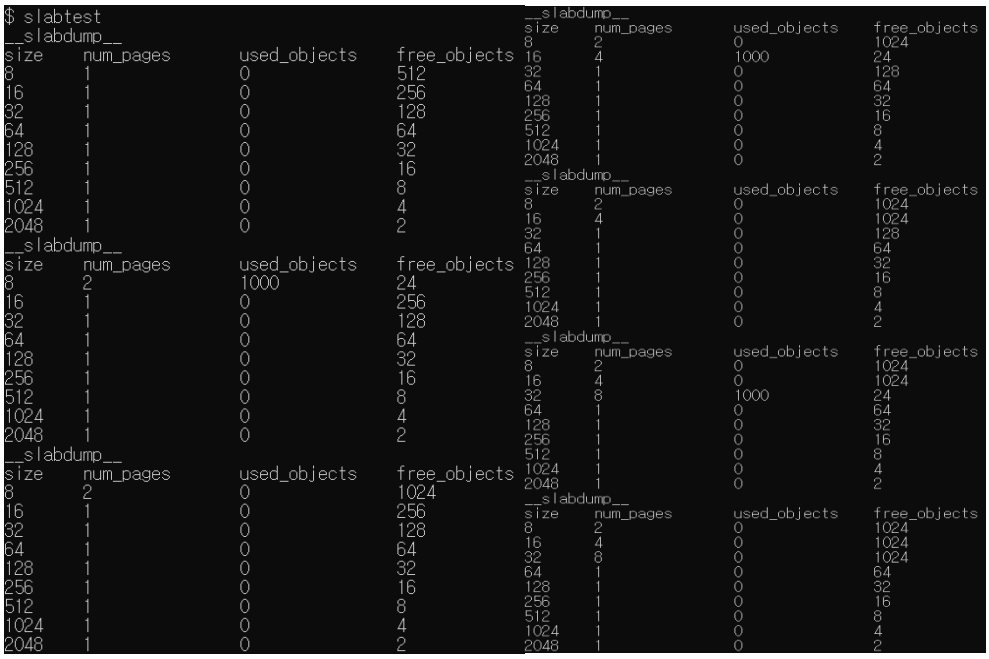
5. nextPowerOf2Idx()

A.
```
unsigned int nextPowerOf2Idx(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 0;

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

    return count;
}
```

B. 입력받은 변수보다 큰 2의 제곱수 중에 가장 작은 수의 지수를 리턴한다.

3. 결과

(테스트 실행결과를 캡처)



```
$ slabtest
__slabdump__
size    num_pages       used_objects    free_objects
8       1               0               512
16      1               0               256
32      1               0               128
64      1               0               64
128     1               0               32
256     1               0               16
512     1               0               8
1024    1               0               4
2048    1               0               2
__slabdump__
size    num_pages       used_objects    free_objects
8       2               1000            24
16      1               0               256
32      1               0               128
64      1               0               64
128     1               0               32
256     1               0               16
512     1               0               8
1024    1               0               4
2048    1               0               2
__slabdump__
size    num_pages       used_objects    free_objects
8       2               0               1024
16      1               0               256
32      1               0               128
64      1               0               64
128     1               0               32
256     1               0               16
512     1               0               8
1024    1               0               4
2048    1               0               2
__slabdump__
size    num_pages       used_objects    free_objects
8       2               0               1024
16      4               1000            24
32      1               0               128
64      1               0               64
128     1               0               32
256     1               0               16
512     1               0               8
1024    1               0               4
2048    1               0               2
__slabdump__
size    num_pages       used_objects    free_objects
8       2               0               1024
16      4               0               1024
32      1               0               128
64      1               0               64
128     1               0               32
256     1               0               16
512     1               0               8
1024    1               0               4
2048    1               0               2
__slabdump__
size    num_pages       used_objects    free_objects
8       2               0               1024
16      4               0               1024
32      8               1000            24
64      1               0               64
128     1               0               32
256     1               0               16
512     1               0               8
1024    1               0               4
2048    1               0               2
__slabdump__
size    num_pages       used_objects    free_objects
8       2               0               1024
16      4               0               1024
32      8               0               1024
64      1               0               64
128     1               0               32
256     1               0               16
512     1               0               8
1024    1               0               4
2048    1               0               2
```