SWE3004 Operating Systems, Spring 2024

Project 4. Page Replacement

TA)
Gwanjong Park
Younghoon Jun
Chanu Yu
Shinhyun Park

Project plan

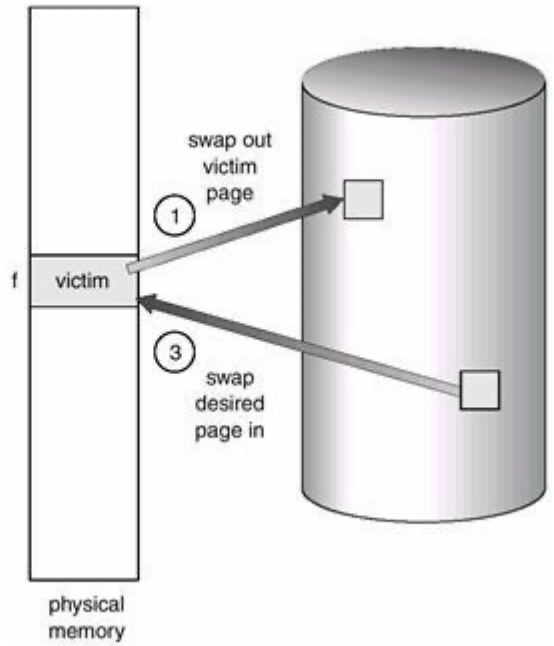
- Total 6 projects
  - ~~0) Booting xv6 operating system~~
  - ~~1) System call~~
  - ~~2) CPU scheduling~~
  - ~~3) Virtual memory~~
  - 4) Page replacement
  - 5) File systems (optional)

Project Objective

- Implement page-level swapping
  - Swap-in: move the victim page from backing store to main memory
  - Swap-out: move the victim page from main memory to backing store
- Manage swappable pages with LRU list
  - Page replacement policy: clock algorithm
- Codes you need to create or modify in xv6
  - Swap-in, swap-out operation
  - LRU list management
  - Some extras

What is Swapping?

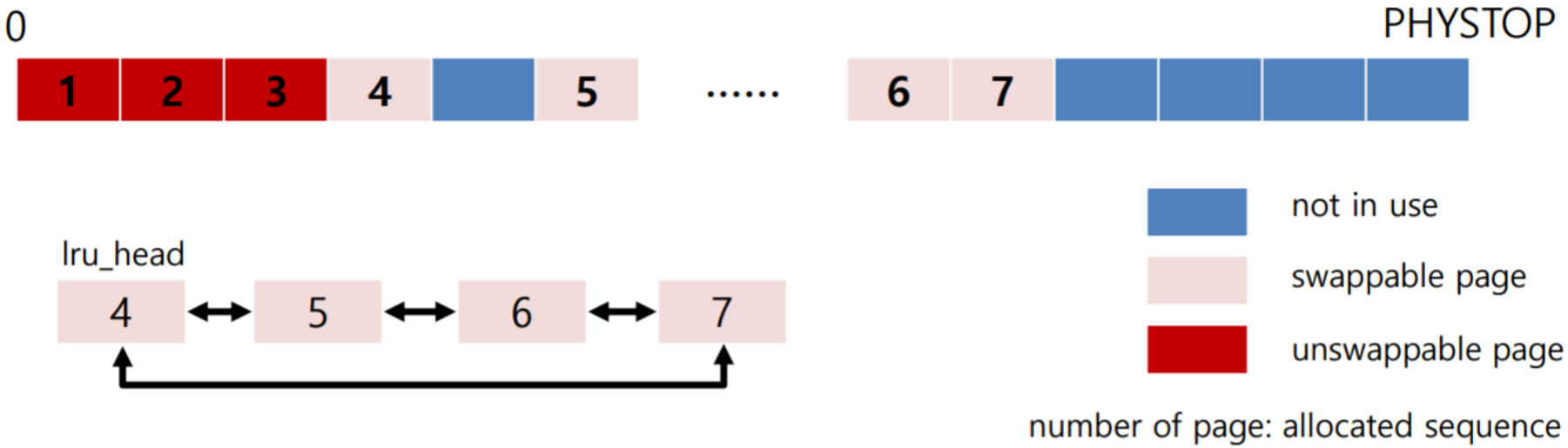
- Support processes when not enough physical memory
  - Can be swapped temporarily out of memory to a backing store
    - Swap pages out of memory to a backing store (swap-out)
    - Swap pages into memory from the backing store (swap-in)



⇨But, not implemented in xv6!

Swappable Pages in xv6 (1)

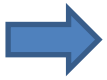
- Only user pages are swappable
  - Some of physical pages should not be swapped out
    - E.g., page table pages
  - So, manage swappable pages with LRU list (circular doubly linked list)
    - When init/alloc/dealloc/copy user virtual memories



Swappable Pages in xv6 (2)

- Page replacement algorithm: clock algorithm
  - Use A (Accessed) bit in each PTE (PTE_A : `0x20`)
  - From `lru_head`, select a victim page following `next` pointer
    - If `PTE_A==1`, clear it and send the page to the tail of LRU list
    - If `PTE_A==0`, evict the page (victim page)
  - QEMU automatically sets `PTE_A` bit when accessed
- If free page is not obtained through the **kalloc**() function, swap-out the victim page

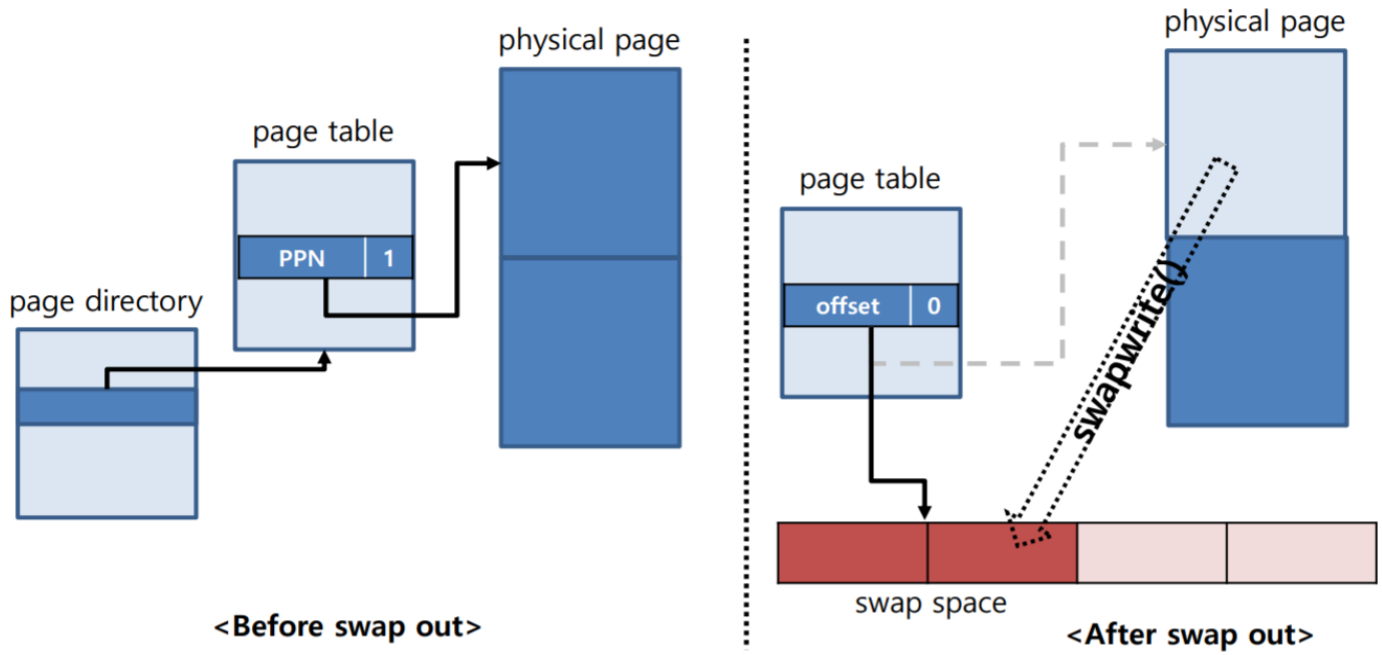
```
char*
kalloc(void)
{
    struct run *r;

    if(kmem.use_lock)
        acquire(&kmem.lock);
    r = kmem.freelist;
    if(r)
        kmem.freelist = r->next;
    if(kmem.use_lock)
        release(&kmem.lock);
    return (char*)r;
}
```



When freelist is empty: if(!r)
Do page reclaim

Swap-out Operation in xv6

1. Use **swapwrite**() function, write the victim page in swap space
   - **swapwrite**() will be provided in skeleton code
2. Victim page's PTE will be set as swap space offset
3. `PTE_P` will be cleared



Swap-in Operation in xv6

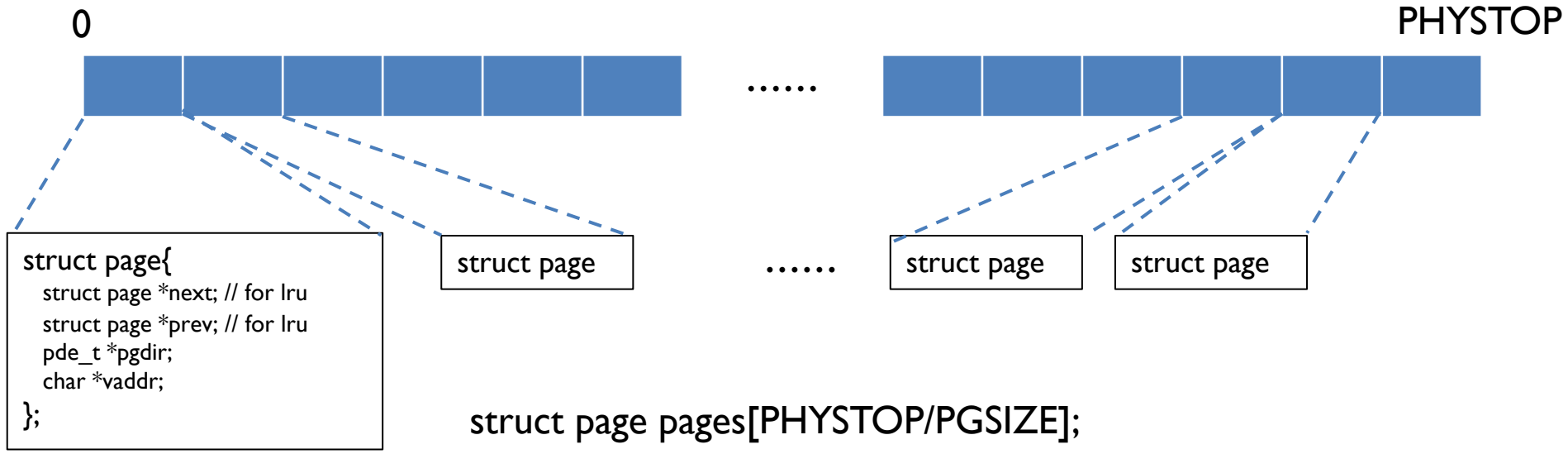
- When accessing a page that has been swapped out
  1. Get new physical page
  2. Using **swapread**() function, read from swap space to physical page
     - **swapread**() will be provided in skeleton code
  3. Change PTE value with physical address & set `PTE_P`
     - Tip: do not need to call **mappages**(), because page table had alre ady been allocated

Several Considerations and Assumptions

- Use 1 physical page for bitmap to track swap space
  - Bit in bitmap is set when page swapped out to swap space
  - Bit in bitmap is cleared when page swapped in
- When user virtual memory is copied
  - Present pages should be copied
  - Swapped-out pages should also be copied
- When user virtual memory is deallocated
  - Present pages should be freed, set PTE bits to 0 and remove them from LRU list
  - Swapped-out pages should be cleared in bitmap and set PTE bits to 0

Several Considerations and Assumptions

- When swap-out should be occurred and there is no page in LRU list, OOM(Out of memory) error should occur
  - Inside the kalloc function, just cprintf error message
  - kalloc should return 0 when OOM occurs
- Lock should be considered with shared resource for synchronization
- All pages are managed in a **struct page**
  - Already implemented in skeleton code (mmu.h)



Skeleton Code

- Skeleton code will provide three functions
  - Functions to read/write from/to swap space are provided
    - void **swapread**(char* **ptr**, int **blkno**)
    - void **swapwrite**(char* **ptr**, int **blkno**)
  - Function for measuring swap space accesses
    - void **swapstat**(int* **nr_sectors_read**, int* **nr_sectors_write**)

- Kernel bootblock has been expanded to use as swap space

`Makefile`

```
xv6.img: bootblock kernel
    dd if=/dev/zero of=xv6.img count=100000
```

`param.h`

```
#define SWAPBASE 500
#define SWAPMAX (100000 - SWAPBASE)
```

How to Test? (1)

- main() of main.c

```
main(void)
{
  kinit1(end, P2V(4*1024*1024)); // phys p
  kvmalloc();      // kernel page table
  mpinit();        // detect other process
  lapicinit();     // interrupt controller
  seginit();       // segment descriptors
  picinit();       // disable pic
  ioapicinit();    // another interrupt co
  consoleinit();   // console hardware
  uartinit();      // serial port
  pinit();         // process table
  tvinit();        // trap vectors
  binit();         // buffer cache
  fileinit();      // file table
  ideinit();       // disk
  startothers();   // start other processo
  kinit2(P2V(4*1024*1024), P2V(PHYSTOP));
  userinit();      // first user process
  mpmain();        // finish this processo
```

```
void
kinit1(void *vstart, void *vend)
{
  initlock(&kmem.lock, "kmem");
  kmem.use_lock = 0;
  freerange(vstart, vend);
}

void
kinit2(void *vstart, void *vend)
{
  freerange(vstart, vend);
  kmem.use_lock = 1;
}

void
freerange(void *vstart, void *vend)
{
  char *p;
  p = (char*)PGROUNDUP((uint)vstart);
  for(; p + PGSIZE <= (char*)vend; p += PGSIZE)
    kfree(p);
}
```

- When xv6 is started, the kinit1 and kinit2 functions are called
  - The call to kinit1 sets up for lockless allocation in the first 4 megabytes
  - And the call to kinit2 enables locking and arranges for more memory to be allocatable

How to Test? (2)

There are too many free pages in the beginning, you need to reduce the `PHYSTOP` to reduce free pages

1. Choose one of the followings to consume memory
    1. Create many processes (**fork()**)
    2. User command **ls**
    3. **sbrk()** system call
2. Choose one of the followings to monitor swap
    1. **swapstat()**
    2. Monitor LRU list length & swap in/out operations

Using above methods, test your own code

Submission

- Begin with skeleton code (not with the prev. project)
  - $ cp ~swe3004/pa4_sklt_xv6.tar.gz ~/
  - $ tar -xzf ~/pa4_sklt_xv6.tar.gz

- Use the ***submit*** & ***check-submission*** binary file in Ui Server
  - $ ~swe3004/bin/submit pa4 xv6-public
  - You can submit several times, and the submission history can be checked through check-submission
    - Only the last submission will be graded

Submission

- PLEASE DO NOT COPY
  - We will run inspection program on all the submissions
  - Any unannounced penalty can be given to **both students**
    - 0 points / negative points / F grade …

- Due date: 5/29(Wed.), 23:59:59 PM
  - **-25%** per day for delayed submission

Questions

- If you have questions, please ask on i-campus
  - Please use the discussion board
  - We don't reply i-campus messages

- You can also visit Corporate Collaboration Center #85533
  - Please e-mail TA before visiting

- Reading xv6 commentary will help you a lot
  - http://csl.skku.edu/uploads/SSE3044S20/book-rev11.pdf