SWE3004 Operating Systems, Fall 2025

Project 4. Page Replacement

TA)
Gwanjong Park
Yunseong Shin

Project plan

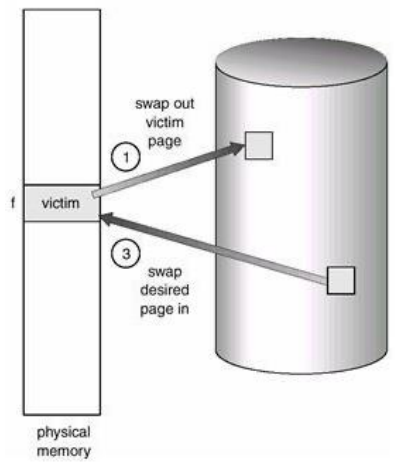
- Total 6 projects
  - ~~0) Booting xv6 operating system~~
  - ~~1) System call~~
  - ~~2) CPU scheduling~~
  - ~~3) Virtual memory~~
  - 4) Page replacement
  - 5) File systems

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?

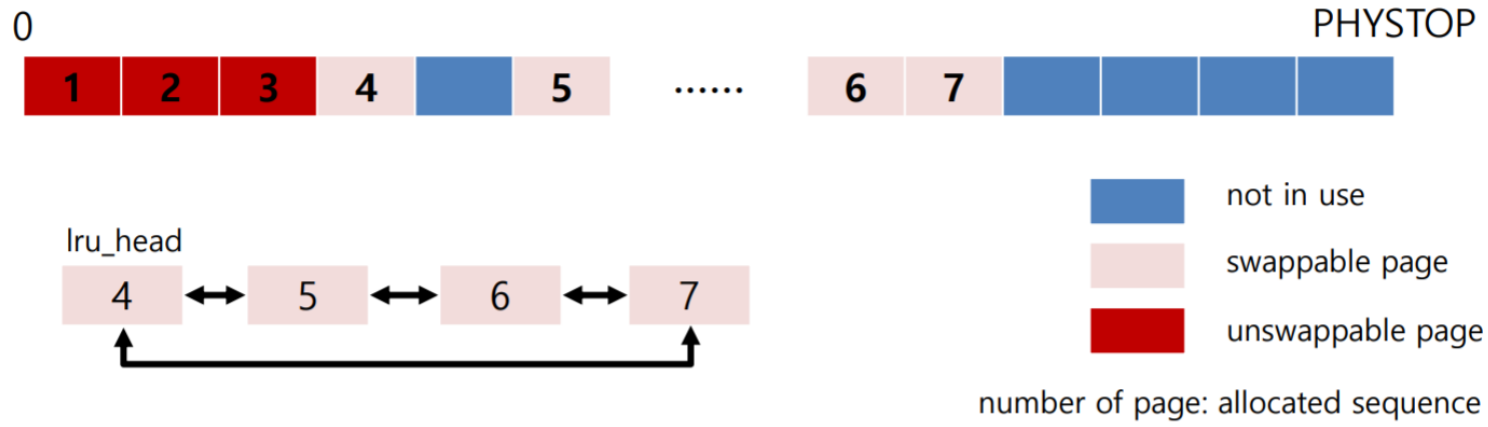
- Swapping is moving a page of memory between main memory and disk to make room for another page when physical memory is full
- Swapping allows processes to continue running even when physical memory is insufficient
    - Swap pages out of memory to a backing store (swap-out)
    - Swap pages into memory 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 a user virtual memory page is mapped to or unmapped from physical memory, the corresponding physical page has to be added to or removed from the LRU list



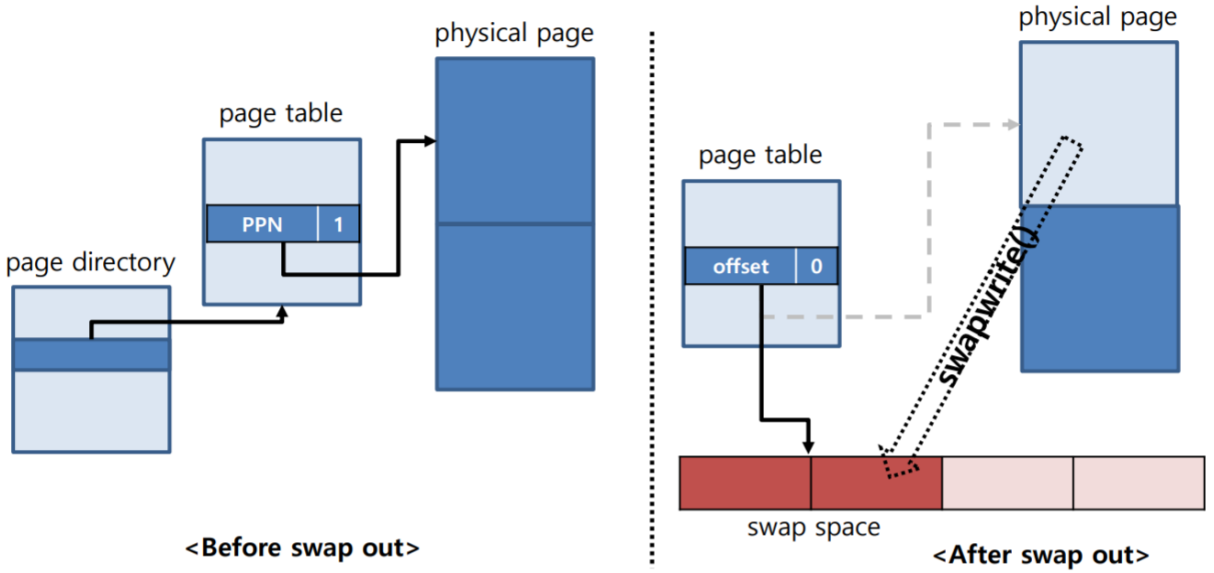
Swappable Pages in xv6 (2)

- Page replacement algorithm: clock algorithm
  - Use PTE_A (0x40) in each PTE
  - 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
  - Define `#define PTE_A(1L << 6)` in riscv.h

Swap-out Operation in xv6

- Use the swap-out operation when a free page is not obtained by the **kalloc**() function

1. Use the **swapwrite**() function to write the victim page in swap space
   - **swapwrite**() will be provided in skeleton code
2. Set the Physical Page Number(PPN) field of the victim page's PTE to the offset in the swap space
3. Clear PTE_V (Valid bit)



Swap-in Operation in xv6

- Use the swap-in operation when accessing a page that has been swapped out

1. Get a new physical page
2. Use the **swapread**() function to load from the swap space into the physical page
   - **swapread**() will be provided in skeleton code
3. Update the PPN value to the physical address of the physical page
   - Tip: do not need to call the **mappages**() function, because the page table had already been allocated

Several Considerations and Assumptions

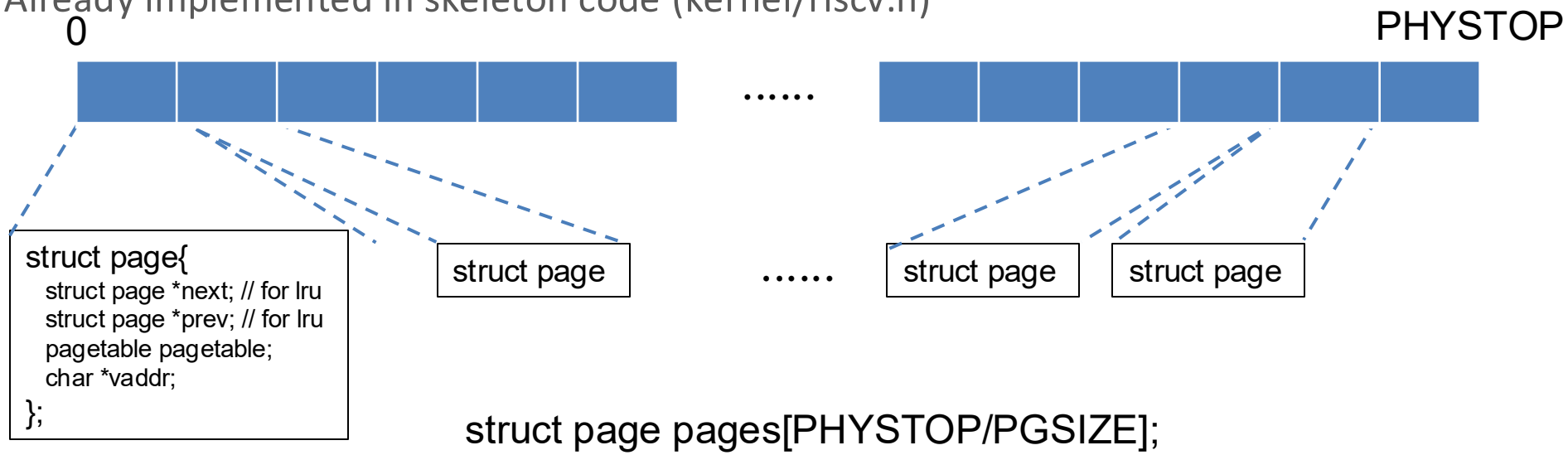
- Use 1 physical page for the bitmap to track the swap space
  - Set a bit in the bitmap when a page is swapped out to to the swap space
  - Clear a bit in the bitmap when a page is swapped in from the swap space

- When a user virtual memory page is copied,
  - Valid pages should be copied
  - Swapped-out pages should also be copied
    - Swap in the pages and then copy them

- When a user virtual memory page is deallocated,
  - Valid pages should be freed
    - set PTE bits to 0 and remove them from LRU list
  - Swapped-out pages should also be cleared
    - clear the corresponding bits in the bitmap and set PTE bits to 0

Several Considerations and Assumptions

- When swap-out needs to occur and there is no page in LRU list, an out-of-memory (OOM) error should occur
  - just printf error message inside the kalloc() function
  - kalloc() function return 0 when the OOM error occurs
- Lock should be considered when you do fork
- All pages are managed using the **page structure**
  - Already implemented in skeleton code (kernel/riscv.h)



Skeleton Code

- Skeleton code will provide three functions
  - Functions to read/write from/to the swap space are provided
    - void **swapread**(uint64 **ptr**, int **blkno**)
    - void **swapwrite**(uint64 **ptr**, int **blkno**)
  - Function to measure the number of accesses to the swap space
    - void **swapstat**(int* **nr_sectors_read**, int* **nr_sectors_write**)
- File system size has been expanded to use as swap space

```
// Disk layout:
// [ boot block | sb block | log | inode blocks | free bit map | data blocks | swap blocks ]
```

- Disk layout(mkfs/mkfs.c)

```
#define FSSIZE       30000  // size of file system in blocks
```

- FSSize(kernel/param.h)

```
#define SWAPBASE     2000
#define SWAPMAX     (30000 - SWAPBASE)
```

- Swap Space(kernel/param.h)

How to Test? (1)

- main() of main.c

```
// start() jumps here in supervisor mode on all CPUs.
void
main()
{
  if(cpuid() == 0){
    consoleinit();
    printfinit();
    printf("\n");
    printf("xv6 kernel is booting\n");
    printf("\n");
    kinit();         // physical page allocator
    kvminit();       // create kernel page table
    kvminithart();   // turn on paging
    procinit();      // process table
    trapinit();      // trap vectors
    trapinithart();  // install kernel trap vector
    plicinit();      // set up interrupt controller
    plicinithart();  // ask PLIC for device interrupts
    binit();         // buffer cache
    iinit();         // inode table
    fileinit();      // file table
    virtio_disk_init(); // emulated hard disk
    userinit();      // first user process
    __sync_synchronize();
    started = 1;
  } else {
    while(started == 0)
      ;
    __sync_synchronize();
    printf("hart %d starting\n", cpuid());
    kvminithart();    // turn on paging
    trapinithart();   // install kernel trap vector
    plicinithart();   // ask PLIC for device interrupts
  }

  scheduler();
}
```

```
void
kinit()
{
  initlock(&kmem.lock, "kmem");
  freerange(end, (void*)PHYSTOP);
}
```

- When the xv6 is started, the kinit() function is called
  - The kinit() function makes more memory available for allocation

How to Test? (2)

- There are too many free pages in the beginning, you need to reduce the `PHY STOP` value 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 Ye 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: 11/25(Tue.), 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
  - Discussion board preferred over messages

- You can also visit Corporate Collaboration Center #85533
  - Please iCampus message TA before visiting

- Reading xv6 commentary will help you a lot

https://pdos.csail.mit.edu/6.828/2023/xv6/book-riscv-rev3.pdf