Project #4: Virtual Memory

Operating Systems (CSE4070) Project

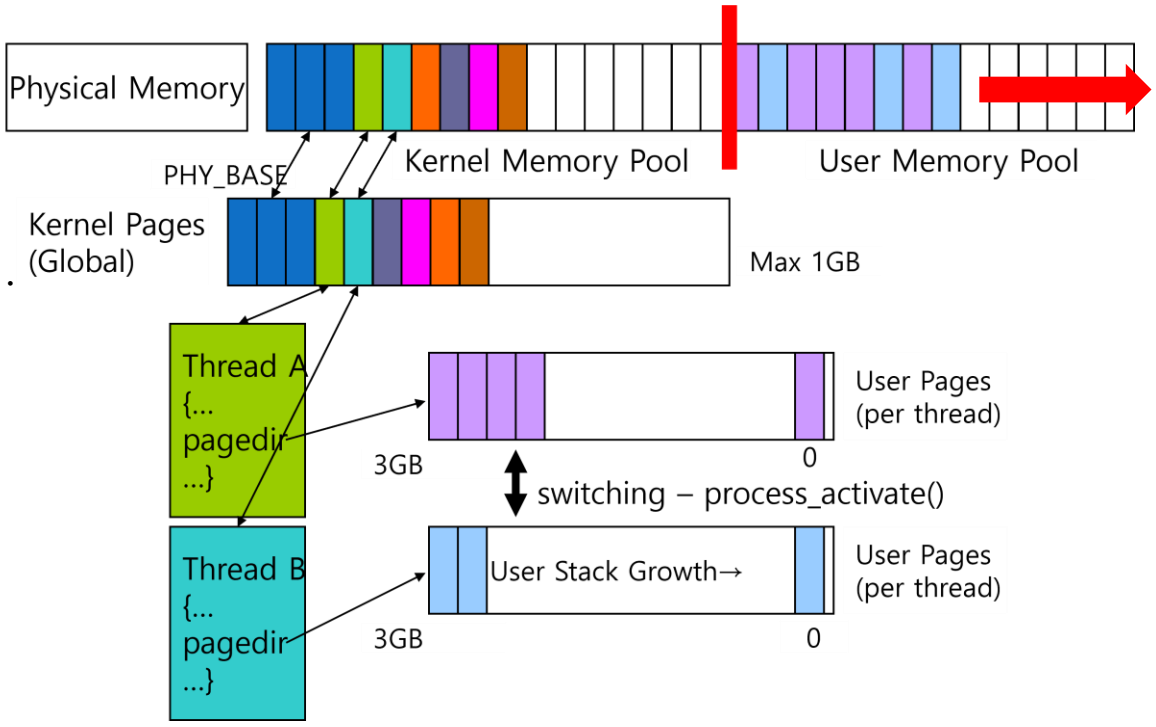
Fall 2024

Prof. Youngjae Kim

TA: Junhyeok Park, Junghyun Ryu,
Sungjin Byeon, Seoyeong Lee

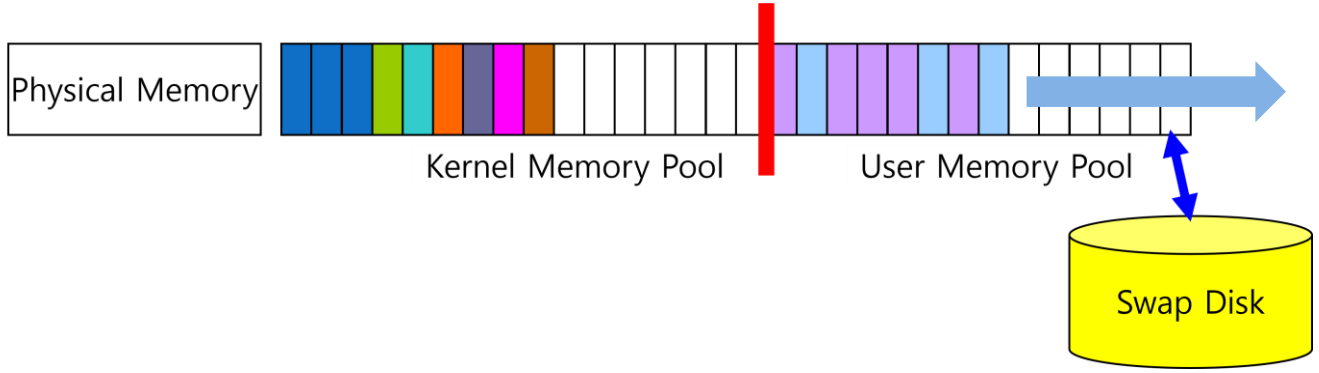
Problem Definition

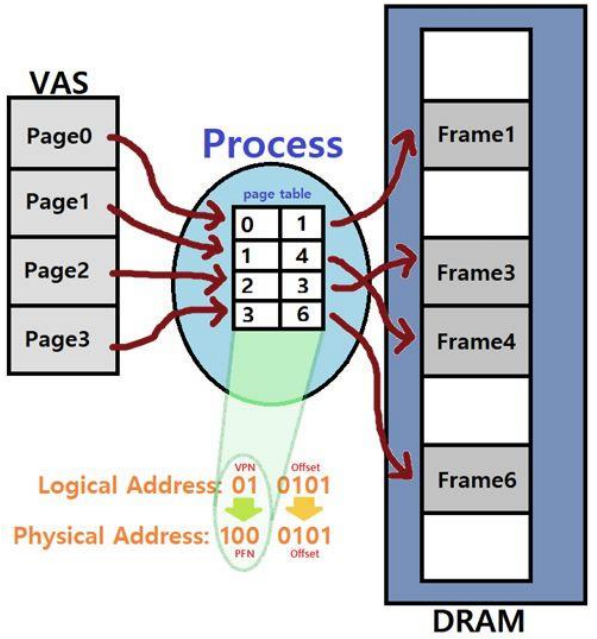
- **Current State:** Your pintOS can properly handle multiple threads of execution with proper synchronization, and can load multiple user programs at once.
- **Problem:** However, the number and size of programs that can run is limited by the machine's main memory size.
  - 현재 pintOS는 Program을 Memory에 올릴 때 Virtual Page 단위로 자르긴 하지만, Page들을 Physical Memory 상에 별다른 작업 없이 Physical Page (Frame)를 할당 받아 바로 올림.
    - ➔ pintOS가 Load할 수 있는 Program의 크기와 개수가 Physical Memory의 사이즈에 의해 제한됨.



What We Need

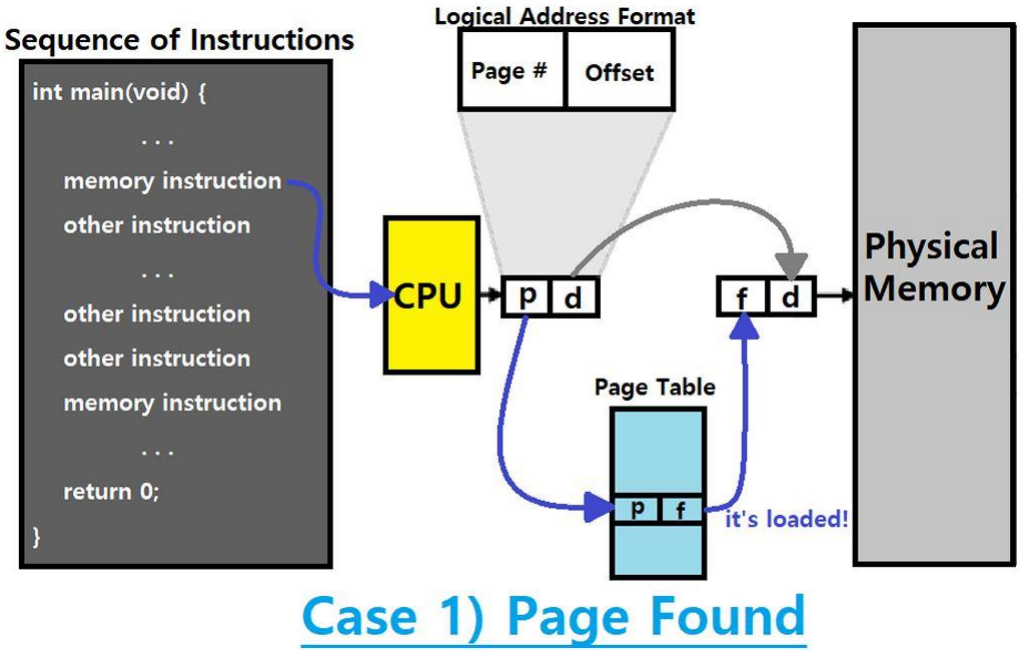
- **What We Need:** Virtual Memory (VM) Paging
  - 현재 pintOS에서는 User Memory Pool (User Memory를 위한 Frame Pool)을 잡아놓고 내장 형태로 Global Page Table을 제공해 Program을 위한 Frame을 할당하고 있음.
    → 여기에 몇 가지 '보조' 정보들을 추가해 VM을 위한 프로세스별 **Supplemental Page Table**을 구현!
  - Supplemental Page Table을 이용해 Program이 매 순간 필요한 Page만을 Frame Pool로부터 할당 받아 **Demand Paging (Lazy Loading)**을 수행하고, Frame Pool에 Frame이 부족할 때 **Disk Swapping**을 통해 Physical Memory Size 제약을 타파한다.



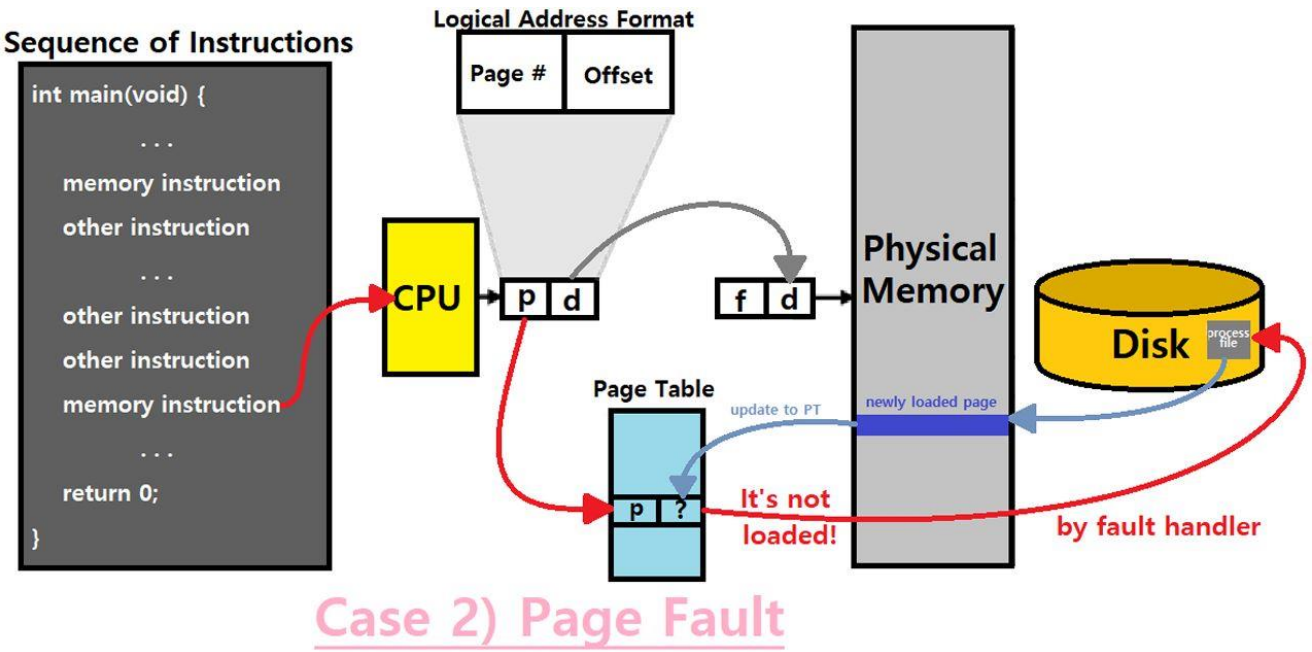


What We Need

- **What We Need:** Virtual Memory (VM) Paging
  - 이를 위해 Page Fault Handler의 수정이 불가피 → 이전 Phase까지의 pintOS는 Page Fault 발생 시 바로 해당 Program을 종료했으나, 본 Phase에선 이 Handler가 실질적인 **Page Fault Handling**, 즉, Fault Exception을 야기한 Page에 대해 Frame을 할당 받고, System이 해당 Faulting Instruction을 다시 실행시켜 Hit를 야기하는 절차를 밟을 수 있도록 만들어야 함.



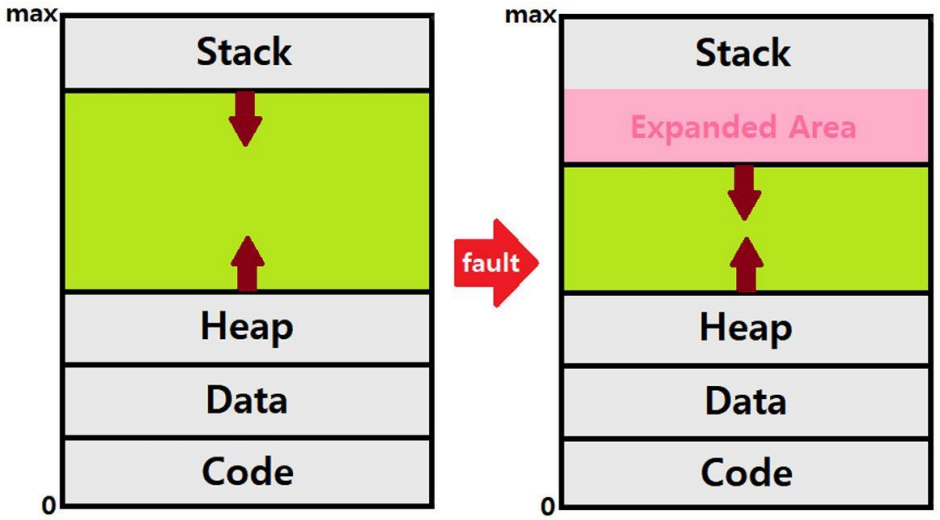
Case 1) Page Found

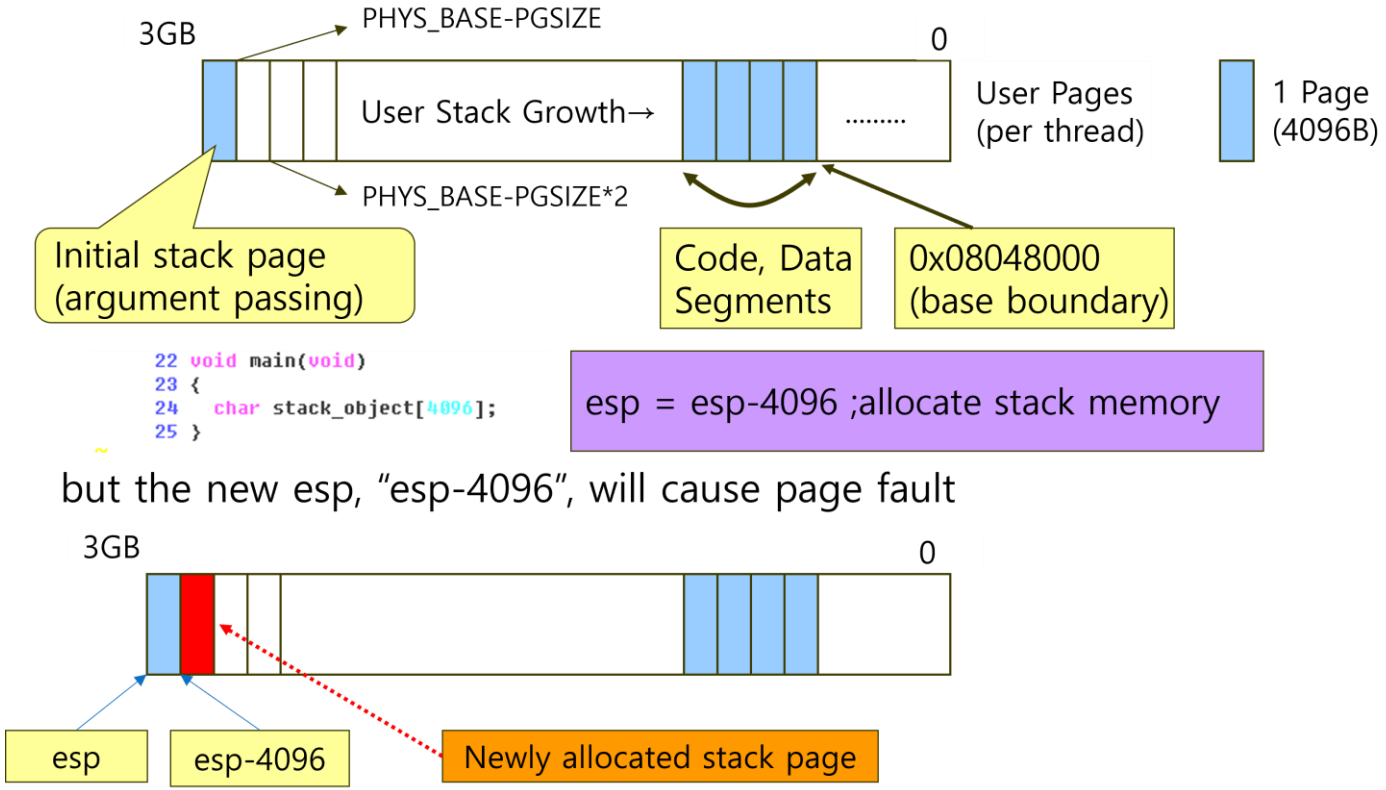


Case 2) Page Fault

What We Need

- **What We Need:** Virtual Memory (VM) Paging
  - Faulting Address가 Un-mapped Page가 아닌, Un-grown Stack Segment에서 기인한 경우, Stack을 이전 할당 사이즈에서 확장시키는 작업이 필요 → 역시나 마찬가지로 Page 단위로 **Stack Growth**를 수행해야 함.



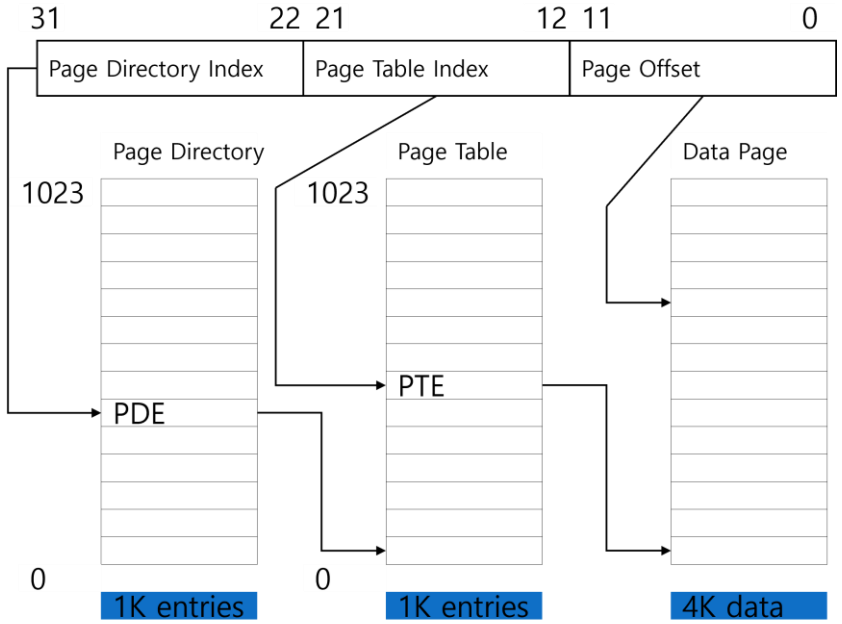


Project Requirement

- Page Table Management
  - Supplemental page table and page fault handling
- Paging to and from (swap) disk
  - Implement pseudo-LRU policies (second chance)
- Stack Growth

Supplemental Page Table

- 기존 Page Table에 보조 정보를 추가해 (Supplemental) Page Table Entry (PTE) 생성
  - Since the given page table in pintos has limitations, we need to supplement the page table with additional data about each page
  - You can exploit the functions in userprog/pagedir.c to implement supplemental page table



단순히 'VPN to PFN Translation 정보' 뿐만 아니라 Memory Load 여부, 연결된 File, 그 File에서 어디까지 읽었는지 정보, 해당 Page가 Page Replacement의 대상이 될 경우 필요한 자료구조 등을 추가해야 함.

➔ PTE를 관리하기 위한 자료구조는 Hash or List 등 가능

➔ Process마다 할당되고 해제되어야 함.

➔ Program Binary File을 Page 단위로 쪼개고 각각 PTE 생성

Page Fault Handler

- Memory Access 시엔 넘겨 받은 Virtual Address의 Format에서 VPN (Virtual Page Number)을 추출해 해당 Value를 Index로 하여 Page Table을 탐색함.
  - Index가 가리키는 Slot에 'VPN to PFN Translation 정보'가 있는지 확인함.
- 변환 정보가 있다면, PFN (Page Frame Number)을 그대로 추출하여 Physical Address를 도출해 동작을 수행한다.
- 변환 정보가 없다면? 접근하고자 하는 Page에 대해 Mapping된 Physical Frame이 없음 → Page Fault 상황
  - Process VAS 상의 특정 Address가 포함된 Virtual Page에 대해, Page Table을 탐색했는데, Mapping된 Physical Frame이 없는 상황
- 이때, MMU(Memory Management Unit)는 Exception을 일으키고, OS는 이 Signal을 받아 Page Fault Handler를 수행함.
  - In pintOS: userprog/exception.c의 page_fault()

Page Fault Handler

- Page Fault가 발생하면, OS 커널의 Page Fault Handler는 다음 순서로 동작해야 함.

(1) 특정 Register (주로, CR2 Register)에 저장되어 있는 Faulting Address를 추출함.

(2) 해당 Address가 포함된 Virtual Page가 Valid한지 확인함 → 해당 Page에 대한 PTE (Page Table Entry)가 있는지 확인한다.

A. 만약, PTE가 있다면, 이는 단순히 해당 Page에 대한 Physical Frame이 현재 부재한 상황, 즉, 일반적인 Page Fault 상황임.

➔ Disk에서 해당하는 Data를 Memory로 올리고 (Read from filesystem or swap in), 새롭게 만들어진 Frame을 해당 Page에 다시 Mapping해 Page Table을 업데이트한 후, 다시 명령을 실행하면 됨.

B. 만약, PTE가 없다면? 이땐 Faulting Address가 Stack Segment가 확장했을 때 Cover할 수 있는 영역 내에 존재하는지 확인한다. Growable Region에 해당하는지를 보는 것임.

➔ 해당한다면? 후술할 Stack Growth를 수행하면 된다.

➔ Growable Region에 포함되지 않는다면? 이는 Segmentation Fault에 해당한다. 즉, Process를 죽이고 Free해야 한다.

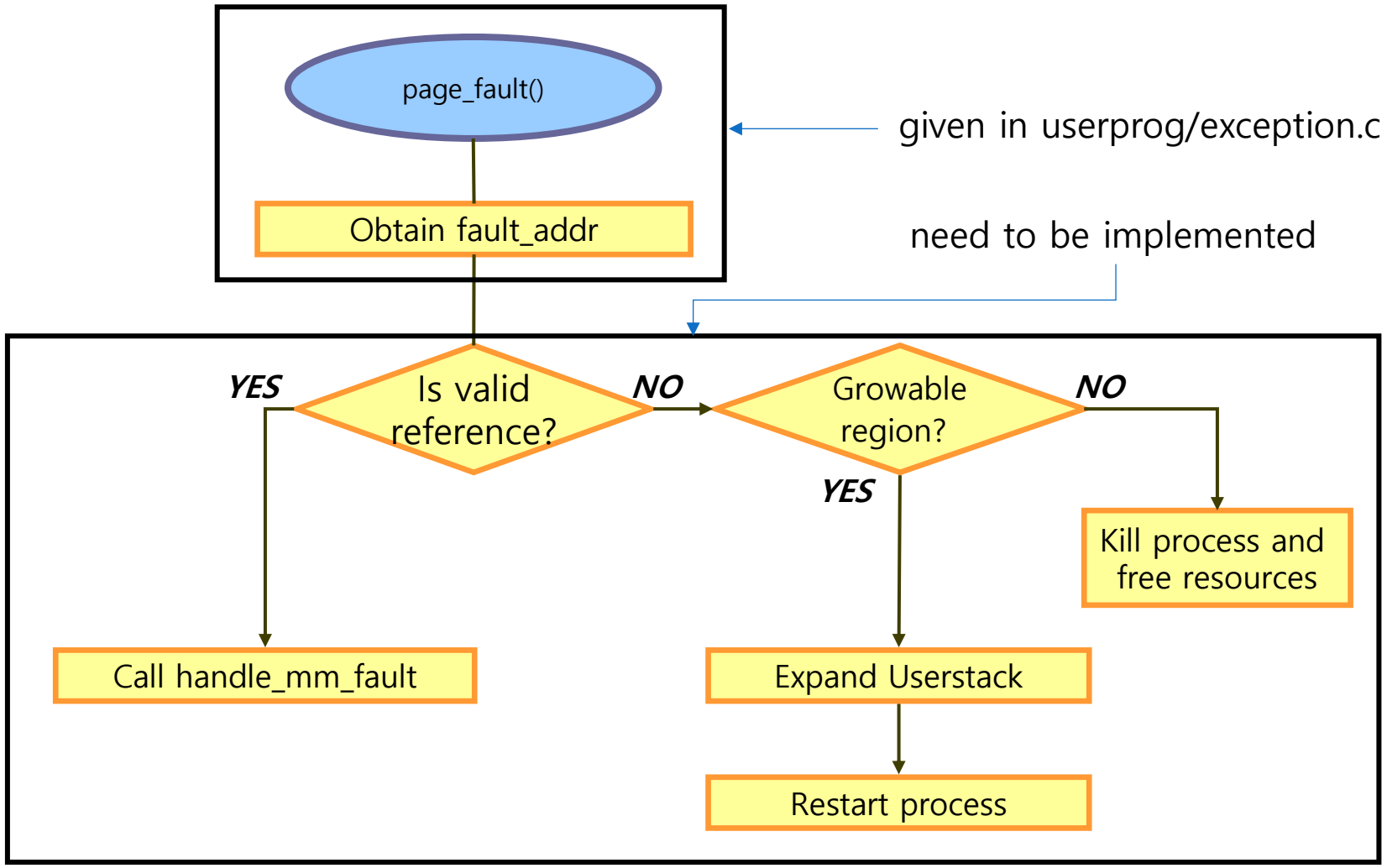
Page Fault Handler

- Some boolean variables in page_fault() will help you
  - bool not_present; // not present in memory or rights violation
  - bool write; // write or read fault
  - bool user; // fault from user or kernel space

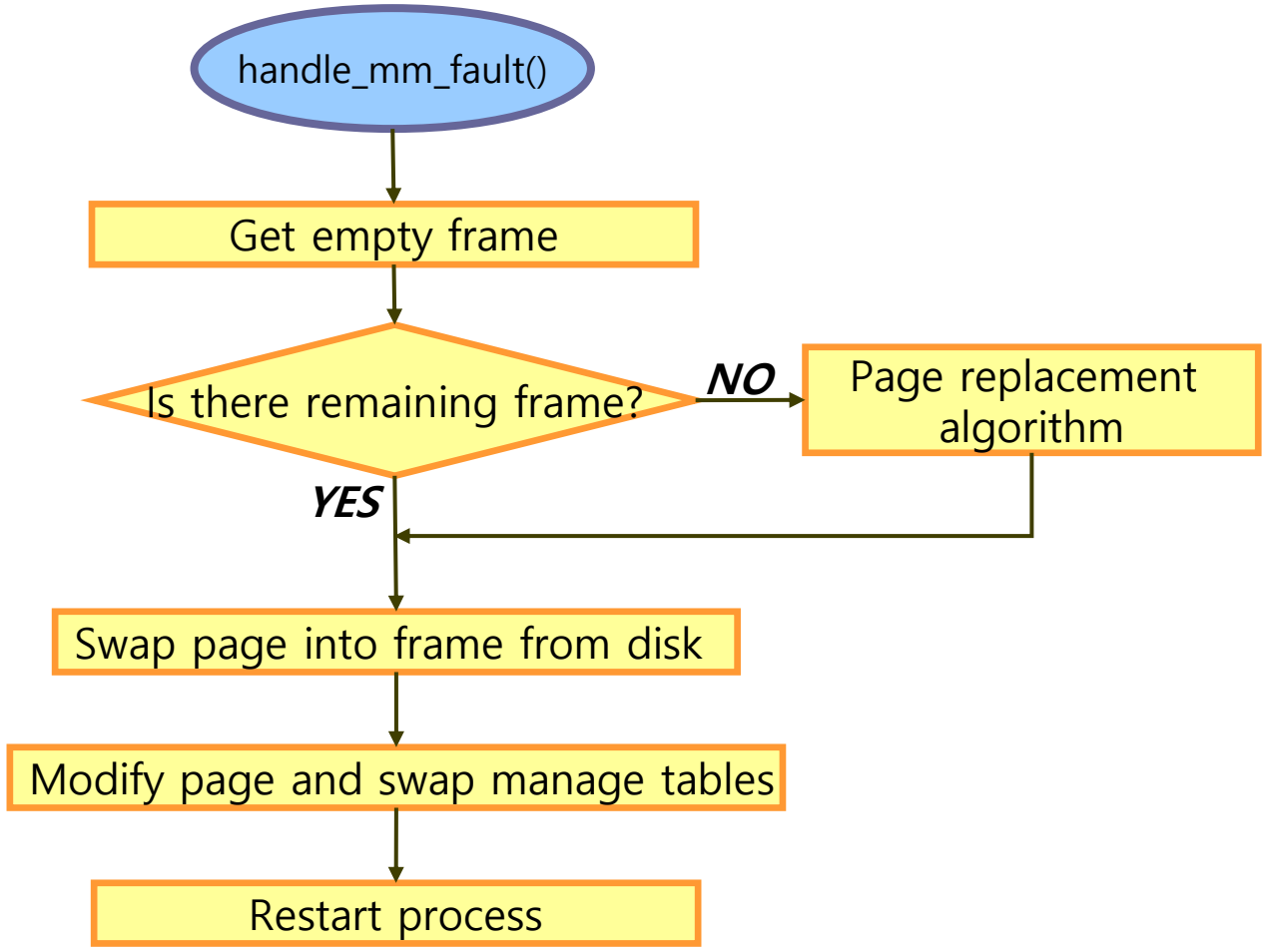
```
static void
page_fault (struct intr_frame *f)
{
  bool not_present;  /* True: not-present page, false: writing r/o page. */
  bool write;        /* True: access was write, false: access was read. */
  bool user;         /* True: access by user, false: access by kernel. */
  void *fault_addr;  /* Fault address. */

  asm ("movl %%cr2, %0" : "=r" (fault_addr));
```

Page Fault Handler: page_fault()



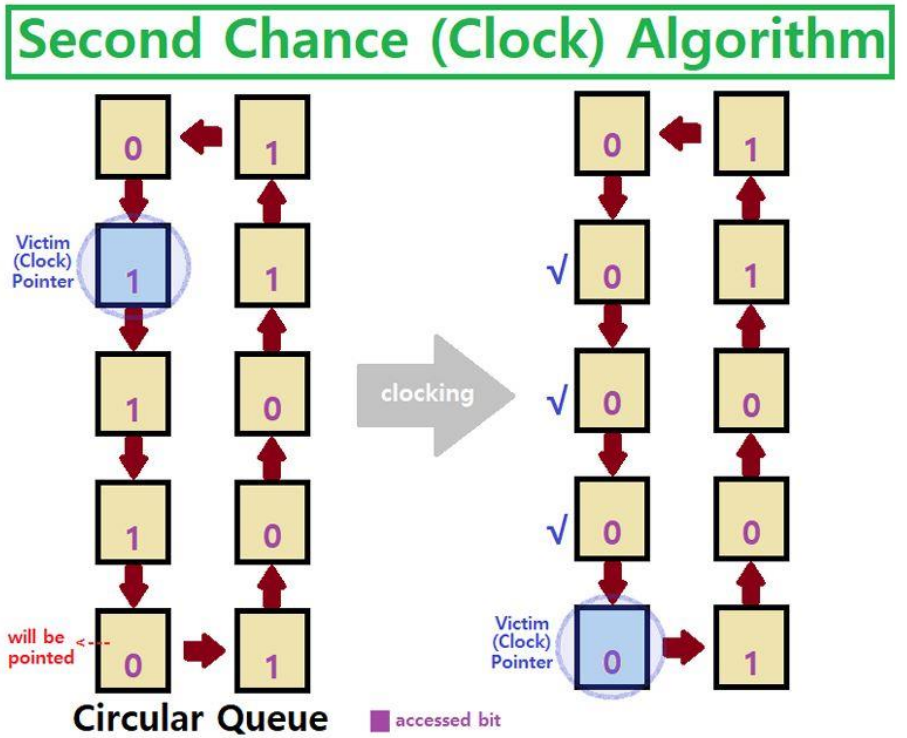
Page Fault Handler: handle_mm_fault()



Paging to and from (swap) disk

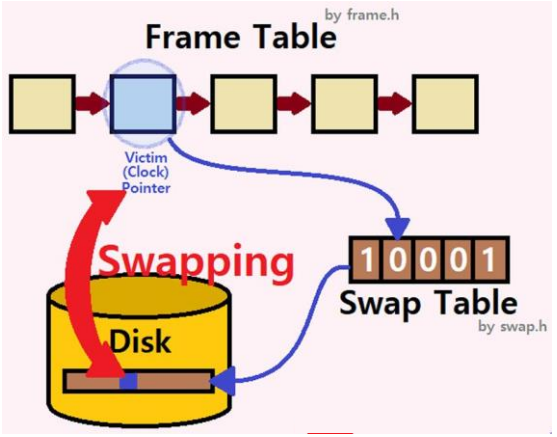
- Swap disk
  - Process에 할당해 줄 Physical Memory가 부족할 때 (Frame Pool에 Free Frame이 없을 때) Disk Swapping (Disk로의 Swap-Out)이 일어남
  - Swap (Evict)할 Page의 결정은 Page Replacement Algorithm 사용
    → Approximate LRU (Least Recently Used)인 'Second Chance (Clock) Algorithm 권장

"한참 전에 Reference되고 나서 이후로 Reference되지 않고 있는데 여전히 Referenced Bit가 Set되어 있는 Page가 발생하는 것을 방지하기 위해 Frame들을 하나의 Queue에 넣어두고, 이를 쭉 훑으면서 Referenced Bit가 Unset인 Victim Page를 찾되, 한 번 Search할 때마다 Referenced Bit가 Set된 Page들을 다시 Unset하고 넘어가는 (즉, 재-기회를 부여하는) 알고리즘"



Paging to and from (swap) disk

- Frame Pool을 Queue 형태로 관리하기 위한 자료 구조가 필요 → 여기서 Clock Algorithm 구동
- Disk 내 Swap Space를 OS에서 가상화하기 위한 Swap Table 구축 필요
  - Swap Disk가 현재 사용하고 있는 슬롯과 빈 슬롯 관리 목적
  - Frame Table과 Swap Table 간의 유기적 동작을 구현해야 함.
  - *These are global data structures …*



- devices/block.h의 block_read() / block_write()을 활용
- You may use the **BLOCK_SWAP block device** for swapping, obtaining the **struct block** that represents it by calling **block_get_role()**
- BLOCK_SWAP에 대해서는 devices/partition.c, thread/init.c 참조
- Swap Disk 생성
  - vm/build에서
    - pintos-mkdisk swap.dsk --swap-size=*n* --> swap.dsk 라는 이름으로 *n* MB swap disk 생성
    - swap.dsk는 pintos의 실행 시 자동으로 hdb1에 attach됨
  - Pintos 실행 시 argument로 '--swap-size=*n*'을 추가해도 *n* MB swap disk가 생성됨
    - swap.dsk는 pintos의 실행 시 자동으로 hda4에 attach됨

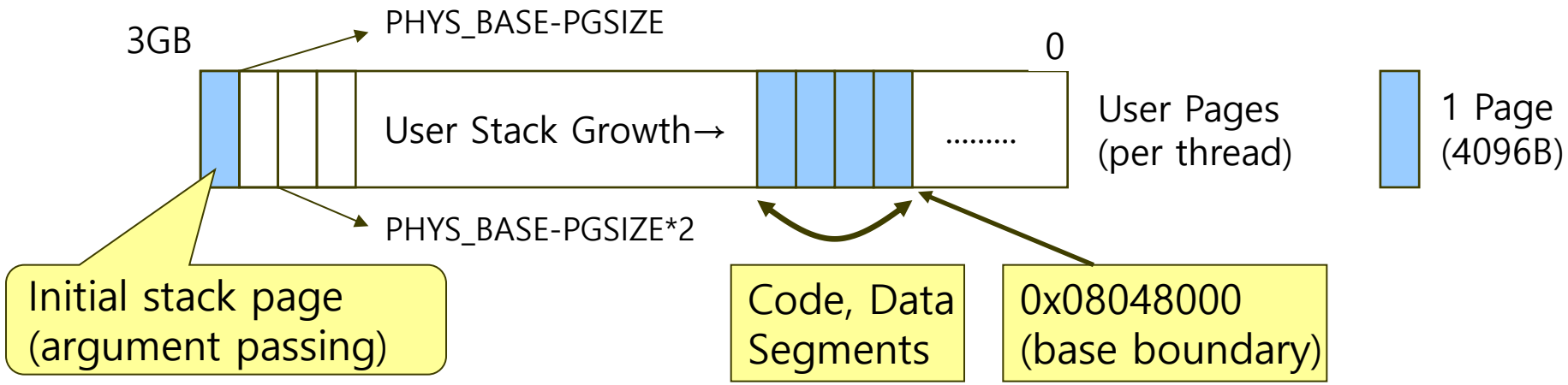
Stack Growth

- Page Fault를 야기한 Faulting Address가 포함된 Virtual Page에 대해 PTE가 존재하지 않는 상황 → 즉, Invalid Reference 상황: 해당 Process를 최초에 Load할 때 Program의 Binary Image에 포함되어 있지 않은 Data임.
- 그렇다면, 우선, Faulting Address가 Valid한지 확인
  - Faulting Address가 VAS에서 User가 접근할 수 있는 범위 안에 들어오는가?
  - Faulting Address가 Stack의 Start Address로부터 8MB 안에 들어오는가? (pintOS가 권장하는 Stack Size의 Limit)
  - pintOS의 기반인 80x86 System의 Assembly PUSHA 연산의 ESP Pointer 점검 조건을 충족하는가? (PUSHA 연산은 ESP Pointer에서 32Bytes 보다 더 멀리 떨어진 Virtual Address에 대해선 Page Fault를 일으키지 않는다. 이 경우엔 그냥 오류 상황이 됨)
  - 이러한 조건을 충족하는지를 확인한다.

- 이 조건들을 모두 충족할 경우, 이는 Fault를 야기한 Virtual Address가 Process의 Stack Segment로부터 온 것임을 의미 (Demand Paging에선 공간 효율을 위해 Stack을 초기에 크게 할당하지 않음)하고, 따라서 Stack을 위한 Page가 추가적으로 더 마련되어야 함.

➔ Stack을 위한 Page를 더 마련하고, 그에 대해 Physical Frame들을 할당하면 된다.

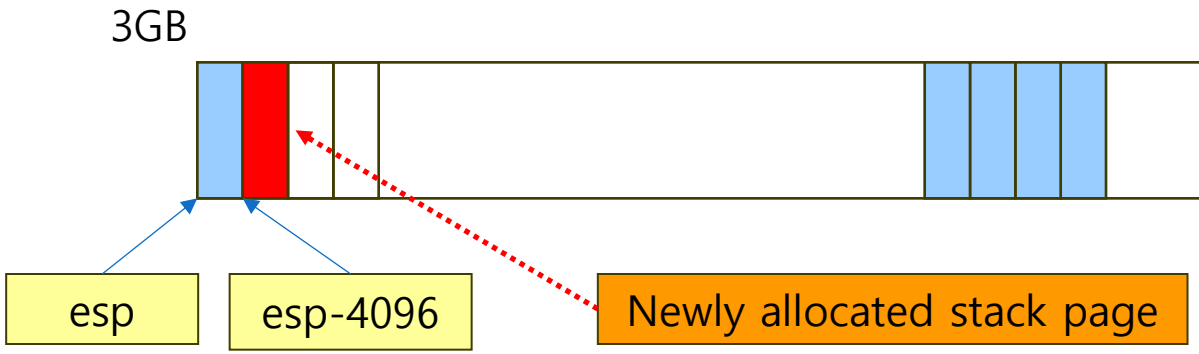
Stack Growth (Example)




```
22 void main(void)
23 {
24   char stack_object[4096];
25 }
```

esp = esp-4096 ;allocate stack memory

but the new esp, "esp-4096", will cause page fault



Another Action Item

- 일부 Test Case (ex. page-merge-mm)는 mmap( ), munmap( ) System Call을 위시한 Memory Mapped File의 구현도 요구함.
  - 각 Process는 Memory Mapped File을 가질 시 해당 New File에 대해 Mapping된 Memory Page 정보를 별도로 Tracking할 수 있어야 한다.
  - 이들은 앞선 VM 루틴과 Compatible하게 설계되어야 한다.
  - 자세한 내용은 PintOS Manual 참고

Tips

- PTE 구조 안에 기록될 Page Type을 잘 정의하자
  - Binary Image File Load 시에 생성된 PTE인지, Swap-In되어야 하는 Page인지, Memory Mapping 시 만들어진 Page인지 구분
    - Stack Growth 시의 Page들, Memory Mapped Page들은 Page Replacement를 어떻게 수행할지 고민 !
- Page Table – Frame Pool – Swap Table – Memory Mapped Pages 간의 관계를 명확히 이해하고 적절한 자료구조를 도입
- System-Wide한 데이터 접근 시 Synchronization을 보장하자
  - 어떤 데이터, 자료구조가 System-Wide할지 고민 !
- 각 Entry 할당 및 해제, Process 생성 및 종료 시 메모리 관리 유의

Reference

- **Reading pintos document is highly recommended especially for this project (pp. 39-55)**
- Basically, the 'vm' directory contains only 'Makefile's
- All code you write will be in new files or in files introduced in earlier projects
- **Use the code implemented in 'User Programs' project.**
- Refer 2nd paragraph in pg.39.

> You will build this assignment on top of the last one. Test programs from project 2 should also work with project 3. You should take care to fix any bugs in your project 2 submission before you start work on project 3, because those bugs will most likely cause the same problems in project 3.

Reference

- You can add new files to src/vm/ if it is required.
  - **Newly added files should be written to src/Makefile.build**

```
Makefile.build       |   4
devices/timer.c      |  42 ++
threads/init.c       |   5
threads/interrupt.c  |   2
threads/thread.c     |  31 +
threads/thread.h     |  37 +-
userprog/exception.c |  12
userprog/pagedir.c   |  10
userprog/process.c   | 319 +++++++++++++++++-----
userprog/syscall.c   | 545 ++++++++++++++++++++++++++++++++-
userprog/syscall.h   |   1
vm/frame.c           | 162 +++++++++
vm/frame.h           |  23 +
vm/page.c            | 297 ++++++++++++++++
vm/page.h            |  50 ++
vm/swap.c            |  85 ++++
vm/swap.h            |  11
17 files changed, 1532 insertions(+), 104 deletions(-)
```

You can follow this approach if you want

Evaluation

- 16 tests will be graded.

  (Refer to the test case list in the next slide)

- Total score is 100 which consists of 80 for test cases and 20 for documentation.

- Refer to

  src/tests/vm/Grading

  src/tests/vm/Rubric.functionality

  src/tests/vm/Rubric.robustness

  to check the points of each test.

Evaluation

Functionality		
No.	Test Case	Point
1	pt-grow-stack	3
2	pt-grow-stk-sc	3
3	pt-big-stk-obj	3
4	pt-grow-pusha	3
5	page-linear	3
6	page-parallel	3
7	page-shuffle	3
8	page-merge-seq	4
9	page-merge-par	4
10	page-merge-mm	4
11	page-merge-stk	4
Total		37

Robustness		
No.	Test Case	Point
1	pt-bad-addr	2
2	pt-bad-read	3
3	pt-write-code	2
4	pt-write-code2	3
5	pt-grow-bad	4
Total		14

Documentation

- Use the document file uploaded on e-class
- Documentation accounts for 20% of total score.
  (Development 80%, Documentation 20%)

Submission

- Make 'ID' directory and copy **'src' directory** in the pintos directory and the document file (**[ID].docx**).
- Compress 'ID' directory to '**os_prj4_[ID].tar.gz**'.
- We provide the script 'submit.sh' to make tar.gz file which contains 'src' directory and document file.
  학생들의 편의를 위해 pintos 디렉토리 내 submit.sh 스크립트를 제공합니다.
  이 스크립트는 src 디렉토리와 document file을 포함한 tar.gz 파일을 생성합니다.
- **Disclaimer**
  - **Any result produced from the 'submit.sh' script is at your own risk.**
  - **You must check the contents of the tar.gz file before submission.**
  - **'submit.sh' 스크립트로 생성된 결과의 모든 책임은 사용자에게 귀속됩니다.**
  - **제출하기 전, tar.gz 파일의 내용물을 반드시 다시 한 번 체크하기 바랍니다.**

Submission

- **Notice – ‘submit.sh’**
  - The ‘submit.sh’ script should be executed on a directory where ‘src’ folder is located.
    submit.sh 스크립트는 src 폴더가 위치한 디렉토리에서 실행되어야 합니다.
  - ‘ID’ folder should not be in the directory.
    해당 디렉토리에 ‘학번’ 폴더가 없어야 합니다.
  - ‘ID.docx’ file should be located in the directory.
    Also, report file with extensions other than ‘docx’ will not be compressed.
    해당 디렉토리에 ‘학번.docx’ 파일이 있어야 함께 압축됩니다.
    또한 ‘docx’ 이외의 확장자를 가진 보고서 파일은 압축되지 않습니다.
  - Be sure to backup your code in case of an unexpected situation.
    만일의 경우를 대비해 반드시 코드를 백업하여 주세요.

Submission

- It is a **personal project**.
- Due date : **2024. 12. 22 23:59**
- Submission
  - The form of submission file is as follows:

Name of compressed file	Example (ID: 20241234)
os_prj4_[ID].tar.gz	os_prj4_20241234.tar.gz

  - No hardcopy.
  - **Copy will get a penalty (1st time: 0 Point and downgrading, 2nd time: F grade)**

Submission

- **Contents**
  ① Pintos source codes (Only **'src' directory** in pintos directory)
  최소한의 용량을 위해 **src 디렉토리만** 압축파일에 포함합니다.
  ② Document: [**ID].docx** (e.g. 20241234.docx; Other format is not allowed such as .hwp)
- **How to submit**
  1) Make tar.gz file.
     - Copy the document file ([ID].docx) to pintos directory.
     - **Execute submit.sh script in the pintos directory and follow the instructions of the script.**
       pintos 디렉토리 내의 submit.sh 스크립트를 실행하고 스크립트의 지시를 따르십시오.
     - Check that **os_prj4_[ID].tar.gz** is created.
     - Decompress **os_prj4_[ID].tar.gz** and check the contents in it. ($ **tar -zxf os_prj4_[ID].tar.gz**)
       (Only **[ID].docx** and **src** directory should be contained in the tar.gz file.)
     - For example, if your ID is 20241234, os_prj4_20241234.tar.gz should be created.
       To decompress the tar.gz file, execute **tar -zxf os_prj4_20241234.tar.gz**
     - **Please check the contents of tar.gz file after creating it.**
  2) Upload the **os_prj4_[ID].tar.gz** file to e-class.