Systems Programming

Spring 2023

Q&A Session 진행 계획

- 6주차 (4/13 목)
  - 06-vm-concepts
  - 07-vm-systems
- 7주차 (4/20 목)
  - 08-allocation-basic
  - 09-allocation-advanced
- 8주차 (4/27 목)
  - 10-internet
  - 11-NAT-PAT
- 9주차 (5/4 목)
  - 12-network-programming
  - 13-webservices

- 중간고사2 (5/11 목)

중간고사2 ~ 기말고사 coverage plan

- 11주차
  - 14-concurrent-programming
  - 15-io-model
- 12주차
  - 16-sync-basic
  - 17-sync-advanced
  - 17-sync-supplement
- 13주차
  - 18-optimization
  - 19-optimization-cache
- 14주차
  - 20-parallelism

- 기말고사 (6/15 목)

공지사항

- 강의 중 제출된 숙제 LATE submission 받지 않습니다

- Lab Session Q&A
  - Lab 진행 관련한 대면 Q&A 진행
  - 일정은 추후 lab session 및 eTL 에 공지 예정

- 중간고사 1 점수 확인 및 claim 진행 예정
  - 4/18 (화) 수업시간 or 실습시간
  - eTL 통해서 시간/장소 추후 공지 예정

- 중간고사 2 시험범위
  - 6주차 ~ 9주차 Q&A Session coverage
  - Lab session

- 기말고사 시험범위
  - 11주차 ~ 14주차 Q&A Session coverage
  - Lab session

Q&A Session 가이드라인

- Use **common senses** !
  - 강의 중 명시적으로 공지된 내용에 대한 질문은 받지 않겠습니다
    - 숙제 등
  - 강의 중 제출된 숙제 풀이에 관한 질문은 받지 않겠습니다
  - 이전 session 에서 이미 다룬 내용과 동일한 질문은 받지 않겠습니다
  - 중복 질문이 많습니다.

    질문은 사전 공지된 구글 form 사용해주세요 (전일 오후 1시 마감)
  - 마감시간 ~ 수업시간 동안 Q&A form will be closed
  - 개인 질문 받지 않습니다
    - **No Questions → No Q&A Sessions**

4월 이번 학기 수업

- 자율학습 & Q&A Session 으로 진행

- 이론: 자율학습 (pdf 강의자료 + audio)
  - course materials will be uploaded

- 매주 목요일 Q&A Session 진행 (302동 105호)
  - Q&A form 으로 사전 접수된 질문이 없는 경우 Q&A session 미진행
  - 일정 변경 시 eTL 통해서 사전 공지 예정

평가

- ~~출석/태도 : 15%~~
- 숙제 : 15%
- 중간고사 : 15% + 15%
- 기말고사 : 20%
- 실습 : 20%

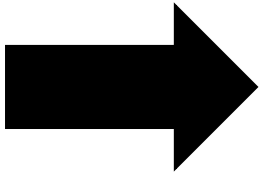
- ETL 에 접속해서 수업자료/숙제 확인할 것.

수업 진도

Submitted 3/11/23, 1:30 PM

안녕하세요, 혹시 예정 스케줄을 조정해주실 수 있을까 싶어 문의드립니다. 3/23에 5강까지의 세션이 예정되어있는데 개강한지 22일만에 1차 중간고사의 범위를 공부하려니 너무 급하고 머릿속에 잘 안들어오는 것 같습니다. 실제로 교수님께서 저번 주까지 1강을 공부하라고 첫 시간에 말씀하셨는데 2강까지가 진도표에 나와있었고, 다음주까지 2,3,4강을 다 보기에 부담이 큰 것 같습니다. 혹시 중간 전에 5강까지만 나갈 수 있게 진도를 조정해주실 수 있을지 여쭙고 싶습니다. 감사합니다.

- **1주차 (3/9 목)**
  - 01-linking
  - 02-relocation
- **2주차 (3/16 목)**
  - 03-exceptions
  - 04-signals
- **3주차 (3/23 목)**
  - 04-signals
  - 05-io
- **4주차 (3/30 목)**
  - 06-vm-concepts
  - 07-vm-systems



- ~~**1주차 (3/9 목)**~~
  - ~~01-linking~~
- **2주차 (3/16 목)**
  - 02-relocation
  - 03-exceptions
- **3주차 (3/23 목)**
  - 04-signals
- **4주차 (3/30 목)**
  - 05-io

수업 진도

Submitted 4/11/23, 6:13 PM

올려주신 6~14주차 계획을 봤는데, 원래 초기에 공지에는 중간고사가 VM, memory management까지고 기말고사가 Networking, concurrency +라고 되어있는데 원래대로라면 중간고사 범위는 09-allocation-advanced까지여야하는거 같은데 혹시 변경되었나요?
아니면 중간고사 범위는 그대로며 미리 기말범위까지 모든 QnA를 받기 위해 진도가 빨리 설정된 것인지 궁금합니다.
이번 강의를 듣는데 중간고사에 비해 일주일 당 강의량이 두배 가까이 증가된것 같아 여쭤봅니다.

수업 진도

- 기준
  - 1회 수업 75분 * 주 2회 = 150분

- 1주차 ~ 4주차

Week	1주차	2주차		3주차	4주차
Chapter	01-linking	02-relocation	03-exceptions	04-signals	05-io
Duration	106:30:00	39:07:00	74:26:00	99:15:00	84:20:00

- 6주차 ~ 9주차

Week	6주차	7주차	8주차	9주차
Chapter	06 / 07	08 / 09	10 / 11	12 / 13
Duration	**168:17:00** 80:18:00 87:59:00	**148:11:00** 62:58:00 85:13:00	**154:49:00** 100:02:00 54:47:00	**162:31:00** 76:59:00 85:32:00

6 주차

- Q&A
  - 06-vm-concepts
  - 07-vm-systems

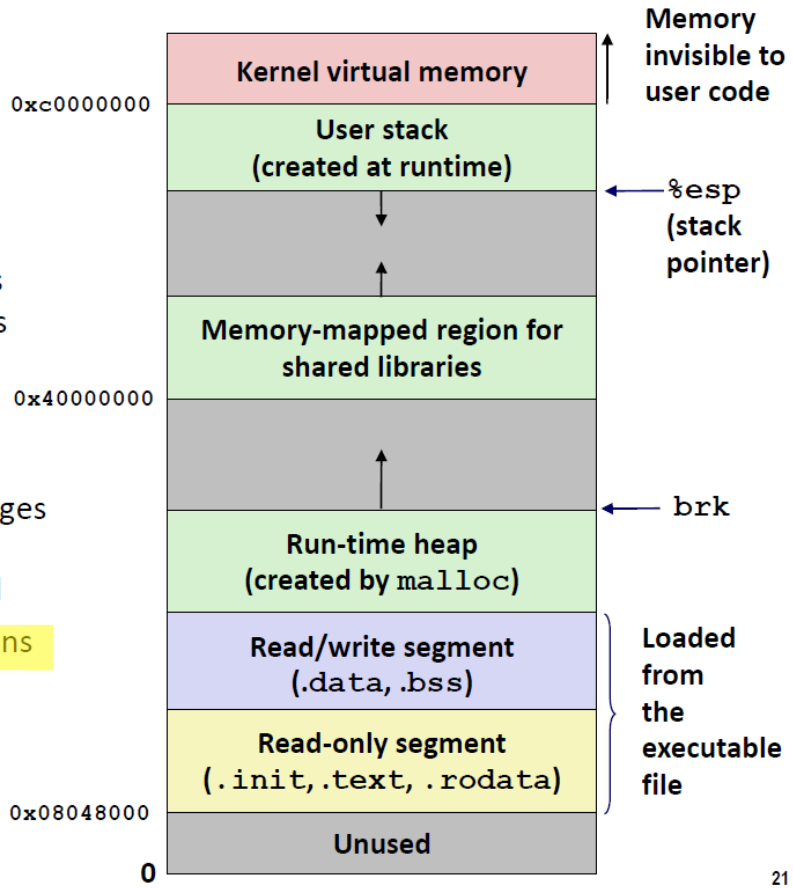
06-vm-concepts

Questions

1. 21p simplyfing linking and loading을 보고 중간고사 전범위 배웠던 내용이랑 설명이 달라져서 질문드립니다. 중간 전 loading과정 = memory에 data, text, symbol table 등등을 올리는 과정으로 보고있었으나 사실은 loading이 실행될 때 어떤 것도 memory에 올라가지 않고 오직 .text와 .data에 대한 page table만 만들고 끝이며, 이후에 실제로 특정 data나 text 등등이 필요할 때(on demand)가 되어서야 memory로 복사된다고 보는게 맞다고 이해하면 될까요?

Simplifying Linking and Loading

- **Linking**
  - Each program has similar virtual address space
  - Code, stack, and shared libraries always start at the same address
- **Loading**
  - `execve()` allocates virtual pages for .text and .data sections = creates PTEs marked as invalid
  - The `.text` and `.data` sections are copied, page by page, on demand by the virtual memory system



21

Questions

1. 6강 25p 교수님께서 map(a) = null와 같이 VA a가 피지컬 메모리에 존재하지 않을 때, 1) 그 부분을 사용하지 않을 수도 있고 2) 사용을 하지만 디스크에 저장하고 있을 수 있으니 구분이 필요하다고 교수님께서 말씀하셨는데 왜 이런 구분이 필요한지 궁금합니다. 사용하지 않는다는 게 어떤 뜻인가요?

- **Address Translation**
  - ***MAP:*** $V \rightarrow P \cup \{\varnothing\}$
  - For virtual address ***a***:
    - ***MAP(a)*** = ***a'*** if data at virtual address ***a*** is at physical address ***a'*** in ***P***
    - ***MAP(a)*** = $\varnothing$ if data at virtual address ***a*** is not in physical memory
      - Either invalid or stored on disk

- Invalid
- Stored on disk

https://stackoverflow.com/questions/26837356/what-is-meant-by-invalid-page-table-entry

Questions

2. 6강 37p Single level page table에서 2^64 addressable bytes/2^12 bytes per page = 2^52 page table entries라고 하셨는데, 이때 2^52가 의미하는 게 정확히 어떤 것인지 모르겠습니다. page가 2^52개가 있어야 하므로, 그를 나타내는 엔트리도 2^52개가 있어야 한다는 게 맞나요?

Questions

2. 6강 37p Single level page table에서 2^64 addressable bytes/2^12 bytes per page = 2^52 page table entries라고 하셨는데, 이때 2^52가 의미하는 게 정확히 어떤 것인지 모르겠습니다. page가 2^52개가 있어야 하므로, 그를 나타내는 엔트리도 2^52개가 있어야 한다는 게 맞나요?

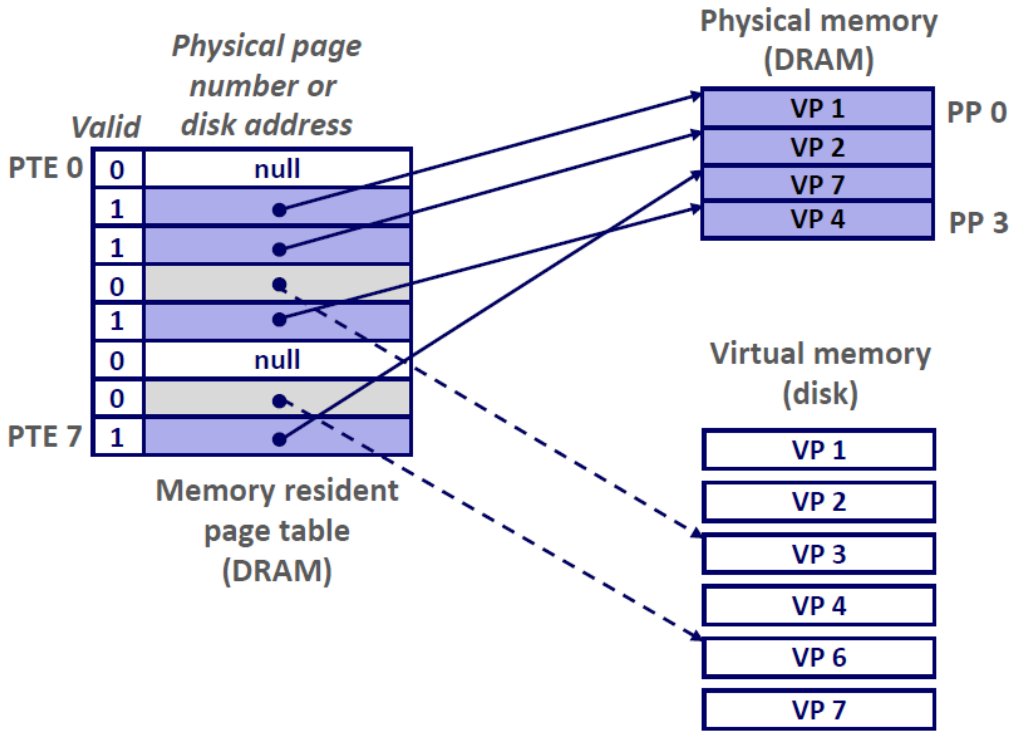
- Let's consider 샤 대학교 기숙사
  - 기숙사의 방이 총 2^64 호
  - 기숙사의 건물 하나에 방이 총 2^12 호 씩 존재
  - 그럼 기숙사 건물은 몇 동이 있어야 할까?

Questions

2. 6강 37p Single level page table에서 2^64 addressable bytes/2^12 bytes per page = 2^52 page table entries라고 하셨는데, 이때 2^52가 의미하는 게 정확히 어떤 것인지 모르겠습니다. page가 2^52개가 있어야 하므로, 그를 나타내는 엔트리도 2^52개가 있어야 한다는 게 맞나요?

Page Tables

- A *page table* is an array of page table entries (PTEs) that maps virtual pages to physical pages.
  - Per-process kernel data structure in DRAM



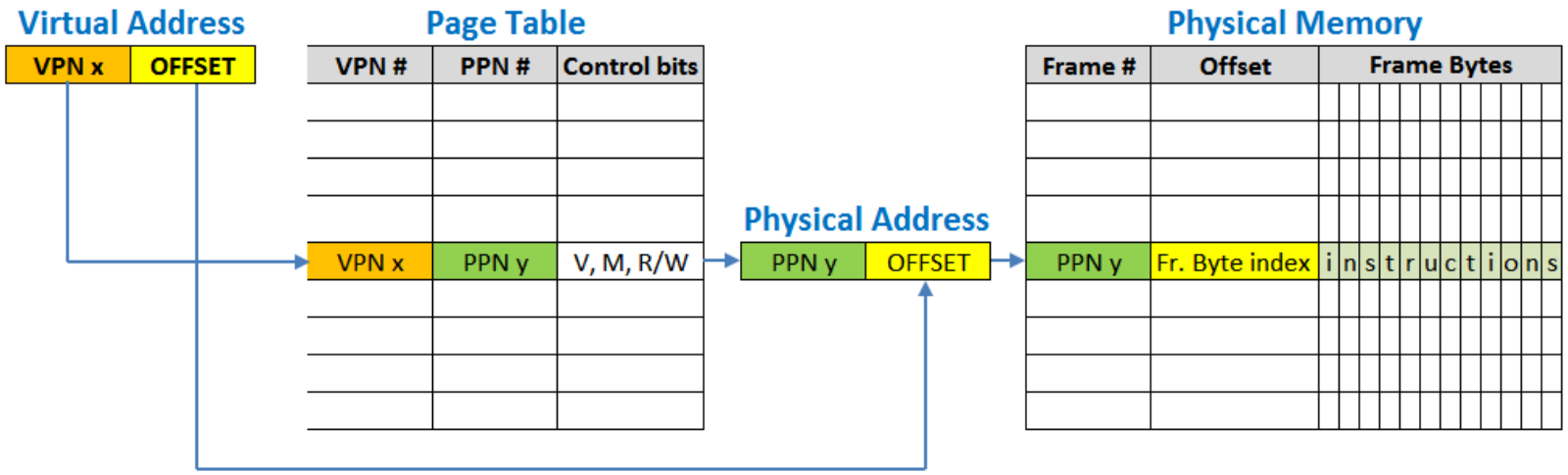
Questions

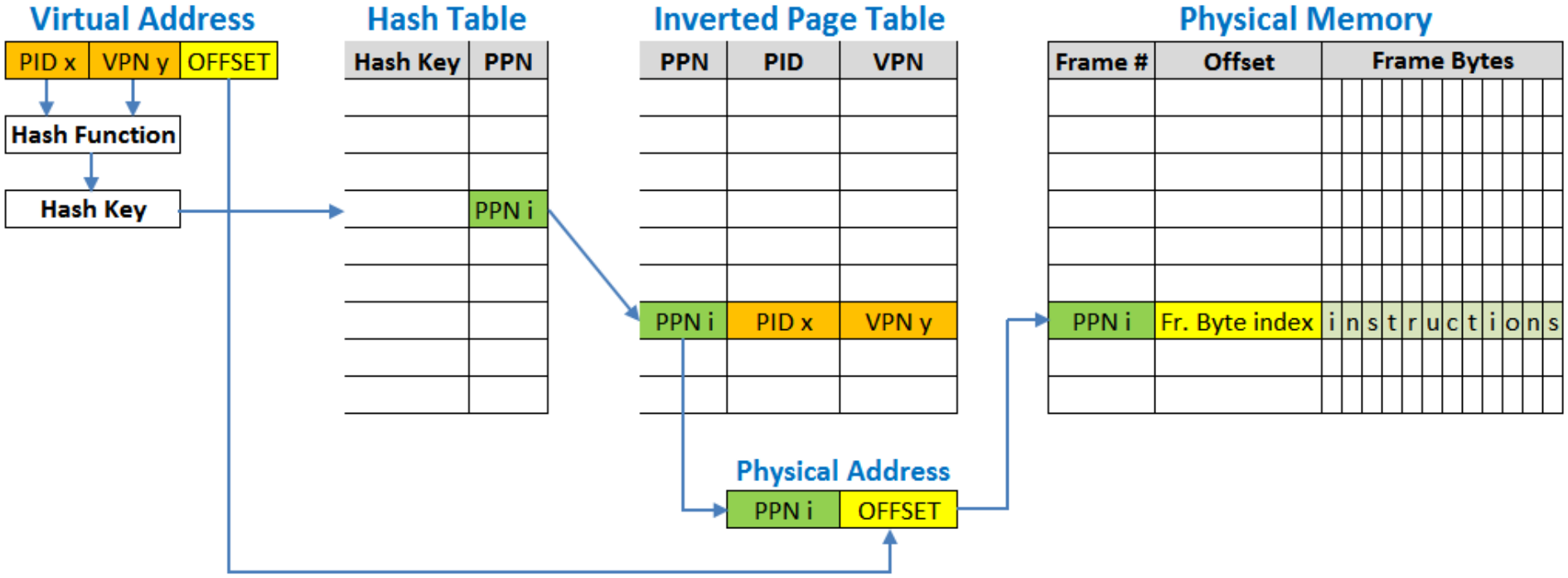
3. 6강 41p inverted table이 정확히 어떤 attribute를 key로 갖는 것인지 궁금합니다. 앞선 테이블과 달리 PFN에 따라 VPN을 정리한 것은 이해했는데, 결국 hashing 이후 검색에는 (VPN, pid)를 넣는 것이 맞나요? 추가적으로 설명해주시면 감사하겠습니다.

2. 46p쯤 있는 inverted page table using hash
원래 처음 소개할 때 inverted page table은 PPN을 index로 VPN & PID를 저장하는 방식이었기에 inverted라는 점이 이해가 갔는데, 이후 hashing을 이용할 때는 PPN이 아닌 VPN과 PID를 해싱해서 만들어진 결과를 기준으로 (VPN, PID, PPN, access info)를 저장해두는 것처럼 보이는데, 이러면 PPN으로 찾는 방식과는 사실상 관계가 없는것 아닌가요? 또 해싱을 할 때도 굳이 해시 결과값이 0~max(PPN)이 아니라 더 크거나 더 작게 해시 함수를 만들어도 상관이 없을 것 같은데. 주소를 찾는데 있어 PPN을 전혀 쓰지 않고 일반적인 page table과 비슷하게 VPN을 이용하는데 왜 inverted라고 불리는지 모르겠습니다.

Inverted Page Table이 어떤 방식으로 동작하는지 자세히 알고 싶습니다. (교재에 Inverted Page Table 관련 내용이 없는 거 같아 질문드립니다.)
또한, Inverted Page Table을 사용하는 경우, 한 page를 여러 프로세스가 share하는 경우는 어떤 방식으로 동작하는지 궁금합니다.

Hashed Inverted Page Table





https://answers.microsoft.com/en-us/windows/forum/all/physical-and-virtual-memory-in-windows-10/e36fb5bc-9ac8-49af-951c-e7d39b979938

Questions

1. 6p의 TLB를 보다 궁금해서 질문드립니다.
특정 태그가 원래 TLB에 있다가 빠지게 되는 경우는 다른 태그가 이를 밀어내고 들어오는 경우 뿐이라고 생각했는데, 이러면 ppt의 표에 나와있는 것처럼 태그는 있으나 vaild bit이 0인 경우가 TLB가 초기화된 경우 말고는 존재하면 안되는 것 아닌가요? 애초에 PM에 올라와 있지 않은 VPN은 TLB에 들어갈 수 없을텐데 저렇게 0인 경우가 많은 이유가 궁금합니다
(이는 7p의 캐시에서도 마찬가지로, 필요해서 가져온 것들만 저장해놓는 캐시에 태그는 있으면서 invaild한 값들이 존재할 이유가 없다고 생각했습니다)

Locality to the Rescue Again!

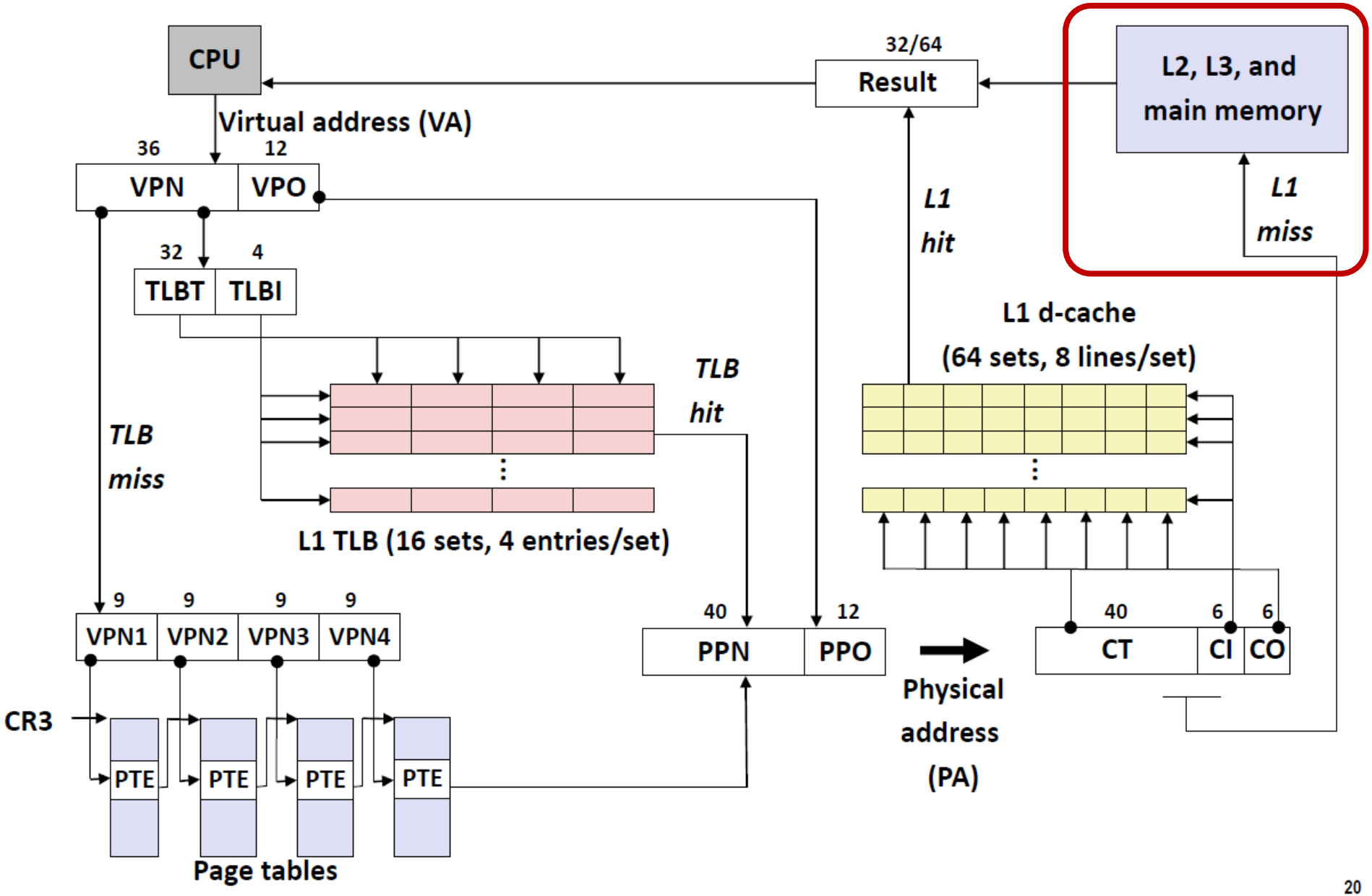
- **Virtual memory works because of locality**
- **At any point in time, programs tend to access a set of active virtual pages called the *working set***
  - Programs with better temporal locality will have smaller working sets
- **If (working set size < main memory size)**
  - Good performance for one process after compulsory misses
- **If (SUM(working set sizes) > main memory size)**
  - *Thrashing:* Performance meltdown where pages are swapped (copied) in and out continuously

More about Swap Memory
https://scbyun.com/984

Questions

2. 20p end to end core i7 address translation에서 18p에서는 소개한 L2 TLB는 그림에서 생략되었다고 보면 될까요? 생략되었다면 원래는 L1 TLB miss가 난 이후 바로 전체 page table을 보는게 아니라 L2 TLB hit여부를 먼저 확인한다고 이해하면 될지 여쭤보고 싶습니다.

End-to-end Core i7 Address Translation



Questions

3. 23, 24p cache offset에 대해
23p에 보면 'each cache line has 64 bytes'이기에 CO = 6bit임을 알 수 있다고 설명되어있는데 여기서 64byte가 어디서 나온 값인지 모르겠습니다. 주어진 64set은 CI의 값을 계산하는데 사용되었으며, 8line/set(심지어 그림에서는 line/set이고 설명에서는 entries/set인데 뭐가 맞는건지도 잘 모르겠습니다.)은 set마다 line수이지 line이 몇바이트인지는 알려주지 않고 있는데, 어떤 값을 보고 64byte라는 수치가 나왔는지 궁금합니다.
이후 생각해보았을때 내린 결론은 PPO가 12bit이고 CI가 6bit이므로 나머지 6bit가 CO일 것이다 정도의 추측이었는데, 24p의 예시들을 보면 PPO는 동일하게 12bit임에도 불구하고 CI의 bit수가 상황에 따라 바뀌는 것을 보아 꼭 PPO = CI + CO인것도 아닌거같아 잘 모르겠습니다.

- cache lines
  - Data is transferred between memory and cache in blocks of fixed size, called cache lines or cache blocks

- Core i7, cache lines hold 64 bytes

Questions

4. 33p anonymous file

빈 stack/ heap등등이 anonymous file을 이용한다는 것은 알겠는데, 이들이 처음 만들어진 이후 사용을 해야 될 경우 CPU에서 어떻게 부를 수 있는지 궁금합니다. swap file(area)에 한번이라도 갔다 오지 않은 page의 경우 PM상에서 만들어졌기에 물리적인 주소는 존재하나 가상 주소가 존재하지 않는 기형적인 특징을 가지고 있을 것 같은데, 그러면 CPU입장에서 이러한 page에 있는 데이터가 필요할 경우 어떻게 불러야 하는지 궁금합니다.

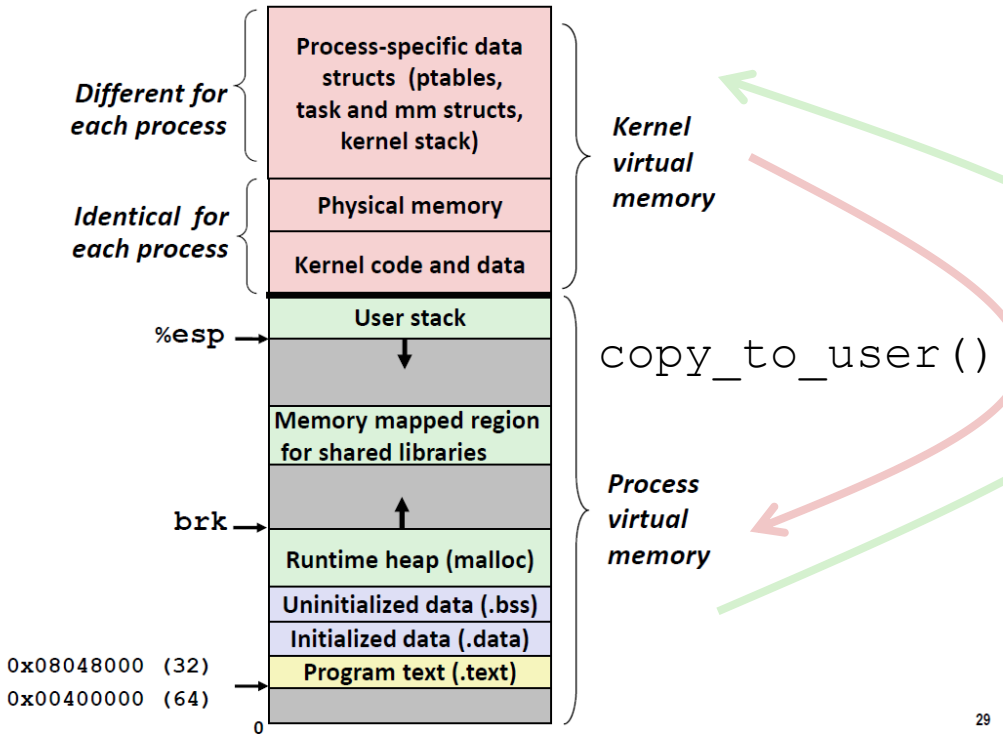
Questions

5. 43p using mmap to copy files
교수님께서 강의에서
"read이후 이를 write한다 할 때 먼저 kernel virtual memory로 데이터를 받아온 다음 이를 process virtual memory로 넘기고 다시 write를 위해 user virtual memory로 넘기는게 일반적이나 43p예시처럼 Mmap을 이용할 경우 이렇게 왔다갔다 하지 않고 바로 kernel에 있는 것을 출력할 수 있다"라고 말씀하셨는데, 앞에서 나온 내용들 중 read 또는 write를 할 때 kernel에 모두 저장되고 이후 user로 복사된다는 부분을 배웠던 기억이 없는 것 같아 질문드립니다.
+
bufp또한 user가 정의한 포인터임에도 불구하고 Write(1, bufp, size)를 통해 kernel virtual memory에 직접 접근해서 출력하도록 만들 수 있는게 왜 가능한지 모르겠습니다. 기본적으로 kernel의 공간은 유저가 접근이 불가능한 것 아니었나요?

Questions

5. 43p using mmap to copy files
교수님께서 강의에서
"read이후 이를 write한다 할 때 먼저 kernel virtual memory로 데이터를 받아온 다음 이를 process virtual memory로 넘기고 다시 write를 위해 user virtual memory로 넘기는게 일반적이나 43p예시처럼 Mmap을 이용할 경우 이렇게 왔다갔다 하지 않고 바로 kernel에 있는 것을 출력할 수 있다"라고 말씀하셨는데, 앞에서 나온 내용들 중 read 또는 write를 할 때 kernel에 모두 저장되고 이후 user로 복사된다는 부분을 배웠던 기억이 없는 것 같아 질문드립니다.

Virtual Memory of a Linux Process



- protection
- checking accessibility

Questions

+

bufp또한 user가 정의한 포인터임에도 불구하고 Write(1, bufp, size)를 통해 kernel virtual memory에 직접 접근해서 출력하도록 만들 수 있는게 왜 가능한지 모르겠습니다. 기본적으로 kernel의 공간은 유저가 접근이 불가능한 것 아니었나요?

mmap(2) — Linux manual page

NAME | SYNOPSIS | DESCRIPTION | RETURN VALUE | ERRORS | ATTRIBUTES | CONFORMING TO | NOTES | BUGS | EXAMPLES | SEE ALSO | COLOPHON

Search online pages

MMAP(2) Linux Programmer's Manual MMAP(2)

NAME top

mmap, munmap - map or unmap files or devices into memory

SYNOPSIS top

```
#include <sys/mman.h>

void *mmap(void *addr, size_t length, int prot, int flags,
           int fd, off_t offset);
int munmap(void *addr, size_t length);
```

See NOTES for information on feature test macro requirements.

DESCRIPTION top

mmap() creates a new mapping in the virtual address space of the calling process. The starting address for the new mapping is specified in *addr*. The *length* argument specifies the length of the mapping (which must be greater than 0).

If *addr* is NULL, then the kernel chooses the (page-aligned) address at which to create the mapping; this is the most portable method of creating a new mapping. If *addr* is not NULL, then the kernel takes it as a hint about where to place the mapping; on Linux, the kernel will pick a nearby page boundary (but always above or equal to the value specified by */proc/sys/vm/mmap_min_addr*) and attempt to create the mapping there. If another mapping already exists there, the kernel picks a new address that may or may not depend on the hint. The address of the new mapping is returned as the result of the call.

https://stackoverflow.com/questions/258091/when-should-i-use-mmap-for-file-access

https://blog.minhazav.dev/memory-sharing-in-linux/

https://devraphy.tistory.com/428