+---------------------------+

| CS 330 |

| PROJECT 3: VIRTUAL MEMORY |

| DESIGN DOCUMENT |

+---------------------------+

---- GROUP ----

>> Fill in the names and email addresses of your group members.

김건호(Kunho Kim) <kaist984@kaist.ac.kr>

김철환(Cheolhwan Kim) <maxwell98@kaist.ac.kr>

---- PRELIMINARIES ----

>> If you have any preliminary comments on your submission, notes for the

>> TAs, or extra credit, please give them here.

토큰 3개 사용하겠습니다.

>> Please cite any offline or online sources you consulted while

>> preparing your submission, other than the Pintos documentation, course

>> text, lecture notes, and course staff.

PAGE TABLE MANAGEMENT

=====================

---- DATA STRUCTURES ----

>> A1: Copy here the declaration of each new or changed `struct' or

>> `struct' member, global or static variable, `typedef', or

>> enumeration. Identify the purpose of each in 25 words or less.

In vm/page.h

#define SWAP\_DISK 0

#define EXEC\_FILE 1

#define MMAP\_FILE 2

#define MEMORY 3

struct spte

{

uint8\_t type; /\* Indicate page status such as

SWAP\_DISK, EXEC\_FILE and MEMORY \*/

void \*vaddr; /\* Virtual address that process may

access \*/

bool writable; /\* If writable is true, the user

process may modify the page. Otherwise, it is read-only. \*/

bool is\_loaded; /\* If this struct in memory, true,

otherwise false \*/

/\* for lazy loading \*/

struct file \*file; /\* File that opened from block \*/

size\_t offset; /\* Location of current file

pointer \*/

size\_t read\_bytes; /\* Read bytes that this spte read

from file \*/

size\_t zero\_bytes; /\* zero\_bytes = PGSIZE -

page\_read\_bytes \*/

struct hash\_elem elem; /\* Find spte by hash. \*/

struct list\_elem map\_elem; /\* Element of map\_file's list. \*/

size\_t swap\_location; /\* Swap location. \*/

};

In vm/frame.h

struct fte

{

void \*kaddr; /\* Kernel address \*/

struct spte \*spte; /\* spte that connected with this frame \*/

struct thread \*thread; /\* Thread that contains this frame \*/

struct list\_elem elem; /\* List element. \*/

};

In vm/frame.c

static struct list frame\_table;

struct lock frame\_lock;

---- ALGORITHMS ----

>> A2: In a few paragraphs, describe your code for locating the frame,

>> if any, that contains the data of a given page.

우리는 이 질문이 page fault가 발생되는 시점, 즉 주어진 페이지에 대한 데이터를 포함하는 프레임이 메모리에 상주하지 않고 요청을 생성하는 프로세스에 대해 매핑되는 시점부터 시작된다고 가정한다.

page\_fault\_handler에서 시작하여 일단 page fault가 발생한 fault address에 대해 유효한 supplement page table entry가 있는지 검사한다. 이후 not\_present 플래그가 설정된 경우, fault address에 대해 유효한 supplement page table entry가 존재하지 않는다면 stack의 condition을 체크해준 후, stack growth()를 실행해준다. fault address에 대해 유효한 supplement page table entry가 존재한다면, handle\_pf()를 호출한다. 이 함수는 다시 spte의 타입에 따라 load\_file() 혹은 swap\_out()/swap\_in()을 해준다.

load\_file()은 해당 spte의 read\_bytes가 0이라면, PG\_SIZE의 크기를 모두 0으로 채우고 true를 반환한다. read\_bytes가 0이 아니라면 해당 spte와 프레임의 정보로 파일을 읽어 물리메모리에 파일의 실제 내용을 적재한 후 true를 반환한다. load\_file() 함수는 page\_fault\_handler에 성공을 반환하며, 이후 프로세스는 다시 재개될 수 있다.

>> A3: How does your code coordinate accessed and dirty bits between

>> kernel and user virtual addresses that alias a single frame, or

>> alternatively how do you avoid the issue?

우리는 항상 user virtual addresses를 사용하여 데이터를 처리함으로써 이 문제를 피한다. page fault 처리의 마지막 과정으로 install\_page()에서 thread의 pagedir를 통해 spte의 vaddr와 frame의 kaddr를 연결시켜준다. 이후 pagedir의 관리를 받기 때문에 process가 vaddr에 접근, 수정하려 할때 pagedir를 거쳐 자동적으로 연결된 kaddr의 정보를 읽고 수정하게 되고, pagedir를 거치기 때문에 accessed 및 dirty bits 역시 자동적으로 설정된다.

---- SYNCHRONIZATION ----

>> A4: When two user processes both need a new frame at the same time,

>> how are races avoided?

frame table의 순서를 바꿔주는 모든 경우에 대해 (ex. list\_push\_back(), list\_remove()) 앞 뒤로 frame\_lock을 걸어주었기 때문에 2개 이상의 프로세스가 race condition을 일으키는 경우는 없다.

---- RATIONALE ----

>> A5: Why did you choose the data structure(s) that you did for

>> representing virtual-to-physical mappings?

우리는 spte를 관리하기 위해 hash map을 사용했는데, hash map은 각 프로세스의 매핑된 페이지를 관리하기 위한 O(1)의 공간 효율적인 메커니즘이 가능하기 때문이다.

프레임을 통해 spte를 관리하는 것은 kernel address와 spte를 thread와 pagedir을 함께 포함함으로써 효율적으로 관리해주기 위함이다.

핵심적인 기능은 vaddr와 kaddr를 연결해주는 것이다. 하지만 효율적인 관리를 위해 vaddr를 포함할, kaddr를 포함할 새로운 구조체를 만든 것이다.

1. spte -> 메모리에 적재할 data의 원본 소유한다. swap disk등 memory에만 존재 할 수 없기 때문에 file, read\_bytes, offset 등 upload할 실제 data의 원본을 담고 있다. 이런 spte를 관리하기 위한 hash table인 spt가 있고, 따라서 hash\_elem를 구조체 안에 가지고 있다.

2. fte -> 메모리와 대응되는 kaddr와 이에 대한 spte, 연결된 pagedir의 정보를 알기 위한 thread를 담고 있다. 이런 fte를 관리하기 위한 frame\_table이 list로 존재하고, 따라서 elem을 구조체 안에 가지고 있다.

PAGING TO AND FROM DISK

=======================

---- DATA STRUCTURES ----

>> B1: Copy here the declaration of each new or changed `struct' or

>> `struct' member, global or static variable, `typedef', or

>> enumeration. Identify the purpose of each in 25 words or less.

struct spte

{

...

size\_t swap\_location; /\* Swap location. \*/

};

swap disk를 관리하기 위해 8\*1024 size bitmap을 만들었고, swap 시 disk의 어느 부분에 spte의 정보가 담겨있는지 알기 위해 swap\_location을 추가했다.

---- ALGORITHMS ----

>> B2: When a frame is required but none is free, some frame must be

>> evicted. Describe your code for choosing a frame to evict.

희생될 frame을 찾기 위해 FIFO 방식을 채택했다. alloc\_fte() 함수를 통해 만들어진 frame은 list\_push\_back을 통해 frame들의 list frame\_table의 뒤로 들어가고, find\_victim() 함수로 frame\_table의 가장 앞 list\_elem을 추출한다. 이후에 mmap을 적용했을때 mmap 으로 생성된 spte와 그 frame은 victim 선정에서 제외했다.

>> B3: When a process P obtains a frame that was previously used by a

>> process Q, how do you adjust the page table (and any other data

>> structures) to reflect the frame Q no longer has?

우리 code는 특정 frame의 위치를 사용하기 위해서 우선적으로 해당 위치의 frame이 free되었어야 한다. (victim\_frame 선정 후 free\_frame) 따라서 문제에서 물어보는 상황(Q 프로세스가 사용하던 frame을 P 프로세스가 사용)은 애초에 발생하지 않는다.

>> B4: Explain your heuristic for deciding whether a page fault for an

>> invalid virtual address should cause the stack to be extended into

>> the page that faulted.

먼저 fault\_addr에 해당하는 vaddr를 갖는 spte가 있는지 찾아준다.

struct spte \*spte = check\_valid\_addr (fault\_addr);

해당 fault\_addr로 spte를 찾아내지 못한다면 아래의 조건문으로 fault\_addr가 stack영역에 해당하는지 판별한다.

if (f->esp - fault\_addr > 32 || 0xC0000000UL - (uint32\_t)fault\_addr > 8 \* 1024 \* 1024)

이 조건문도 통과한다면 stack영역의 주소이므로 stack 확장을 시도한다.

---- SYNCHRONIZATION ----

>> B5: Explain the basics of your VM synchronization design. In

>> particular, explain how it prevents deadlock. (Refer to the

>> textbook for an explanation of the necessary conditions for

>> deadlock.)

dead lock 현상을 막기 위해 최소한의 lock만 사용했다. file을 보호하기 위한 file\_lock, frame\_table을 보호하기 위한 frame\_lock 이 두가지이다.(swap disk를 참조하는 swap in/out에만 swap lock을 사용했다.) 또 이 두가지 lock끼리 얽혀서 dead lock이 발동되지 않도록 한 lock을 release 하기전에 다른 lock을 acquire하는 과정을 피했다.

>> B6: A page fault in process P can cause another process Q's frame

>> to be evicted. How do you ensure that Q cannot access or modify

>> the page during the eviction process? How do you avoid a race

>> between P evicting Q's frame and Q faulting the page back in?

frame table의 순서를 바꿔주는 모든 경우에 대해 (ex. list\_push\_back(), list\_remove()) 앞 뒤로 lock을 걸어주었기 때문에 2개 이상의 프로세스가 race condition을 일으키는 경우는 없다. 따라서 Q의 프레임을 찾기 위한 모든 과정이 차단되기 때문에 P가 완료될 때(frame을 제거할 때)까지 Q의 프레임은 참조 될 수없다.

Q가 해당 프레임에 대해 faulting을 발동했다는 것 자체로 이미 P에서 Q의 프레임을 evicting했다는 것이 된다. 이후 faulting에서 Q는 새로운 frame을 찾아 allocate 할것이고, frame을 할당할 공간이 없다면 FIFO 방식으로 새로운 victim frame을 찾아낼 것이다. 이것은 P가 Q의 frame을 evicting 한 것과는 다른 문제이므로 race condition은 일어나지 않는다.

>> B7: Suppose a page fault in process P causes a page to be read from

>> the file system or swap. How do you ensure that a second process Q

>> cannot interfere by e.g. attempting to evict the frame while it is

>> still being read in?

read를 진행중이었다면 file\_lock이 걸려있는 상황이었을 것이다. 이때 victim 선정하는 victim\_frame() 함수 역시 file\_lock을 얻어야 해당 과정을 진행하기 때문에 문제의 상황은 발생하지 않는다.

>> B8: Explain how you handle access to paged-out pages that occur

>> during system calls. Do you use page faults to bring in pages (as

>> in user programs), or do you have a mechanism for "locking" frames

>> into physical memory, or do you use some other design? How do you

>> gracefully handle attempted accesses to invalid virtual addresses?

system call 진행중에 page out된 page를 읽어야 한다면 page fault가 발생한다. 이후 해당 spte에 관한 pf\_handle()이 일어나며 정상적으로 물리 메모리에 load 될 것이다.

virtual address라면 해당 addr에 맞는 spte가 있을 것이다. 우리는 그 점을 이용해 spte를 찾아주고, spte가 없으면 그것을 invalid virtual address로 간주한다.

---- RATIONALE ----

>> B9: A single lock for the whole VM system would make

>> synchronization easy, but limit parallelism. On the other hand,

>> using many locks complicates synchronization and raises the

>> possibility for deadlock but allows for high parallelism. Explain

>> where your design falls along this continuum and why you chose to

>> design it this way.

기존에 userprog를 진행하면서 synch문제를 해결하기 위해 만든 file\_lock, frame\_table이라는 공유 자원을 사용할 때 일어나는 문제를 방지하기 위해 frame\_lock, swap in/out 시 swap disk에 여러 process가 접근하는 문제를 방지하기 위해 swap\_lock 총 3개의 lock을 사용했고, 만들어진 lock은 최대한 해당 구조체를 건들때만 사용되도록 code를 작성했다.

MEMORY MAPPED FILES

===================

---- DATA STRUCTURES ----

>> C1: Copy here the declaration of each new or changed `struct' or

>> `struct' member, global or static variable, `typedef', or

>> enumeration. Identify the purpose of each in 25 words or less.

struct map\_file

{

int map\_id;

struct file \*file;

struct list\_elem elem;

struct list spte\_list;

};

물리 메모리에 적재될 때 spte의 단위에 맞춰 4096byte로 쪼개져야 하는 file을 file단위로 묶어주기 위해 만든 구조체이다. map\_id는 thread가 여러개 소유할 map\_file들을 관리할 id이고, file은 map\_file의 핵심 변수이다. elem은 thread의 mmap\_list에 들어갈 list\_elem, spte\_list는 쪼개진 file의 spte들을 관리할 list이다.

struct spte

{

...

struct list\_elem map\_elem; /\* Element of map\_file's list. \*/

...

}

map\_file의 spte\_list에 들어가기 위한 list\_elem map\_elem이다.

struct spte

{

...

int map\_id; /\* ID for mmap. \*/

struct list mmap\_list; /\* List of memory mapping

files. \*/

...

}

thread마다 여러개의 mmap\_file이 생길 수 있으므로 그것을 관리하기 위해 list를 추가했다.

---- ALGORITHMS ----

>> C2: Describe how memory mapped files integrate into your virtual

>> memory subsystem. Explain how the page fault and eviction

>> processes differ between swap pages and other pages.

먼저 system call에 의해 mmap이 발동되면 map\_file 구조체를 할당한다. 이후 process.c의 load\_segment()의 형식과 비슷하게 while문을 순회하며 file의 모든 내용을 PG\_SIZE 마다 끊어서 spte를 만들고 thread의 hash table인 spt에 넣어준다. 추가적으로 map\_file구조체의 spte\_list에도 spte들을 삽입하고, 마지막으로 만들어진 map\_file 구조체를 thread의 mf\_list에 넣어준다.

여기서 중요한 점은 만들어주는 spte의 type을 MMAP\_FILE로 두어 victim frame의 선정을 피해준다는 것이다.

>> C3: Explain how you determine whether a new file mapping overlaps

>> any existing segment.

spte들은 spte의 vaddr를 기준으로 current thread가 소유하고 있는 hash\_table, supplemental page table에 삽입된다. 즉, mmap하고싶은 virtual address에 이미 다른 spte가 존재한다면 supplemental page table에 삽입이 실패하게 된다. 그러므로, 기존에 메모리에 존재하고 있던 다른 파일들과 겹치는 일은 발생하지 않는다.

---- RATIONALE ----

>> C4: Mappings created with "mmap" have similar semantics to those of

>> data demand-paged from executables, except that "mmap" mappings are

>> written back to their original files, not to swap. This implies

>> that much of their implementation can be shared. Explain why your

>> implementation either does or does not share much of the code for

>> the two situations.

우리 code에서 mmap과 load\_segment 는 spte->type == MMAP\_FILE(or EXEC\_FILE), read, zero bytes와 offset 설정 부분을 제외하고 전부 동일하다. 문제의 질문대로 요구 페이징과 성격이 비슷하기 때문이다. 다만 차이점은 mmap으로 생성된 spte는 swap되면 안되는 것이었고, 이것 때문에 spte의 type에 MMAP\_FILE을 추가했다.

SURVEY QUESTIONS

================

Answering these questions is optional, but it will help us improve the

course in future quarters. Feel free to tell us anything you

want--these questions are just to spur your thoughts. You may also

choose to respond anonymously in the course evaluations at the end of

the quarter.

>> In your opinion, was this assignment, or any one of the three problems

>> in it, too easy or too hard? Did it take too long or too little time?

demand paging(lazy loading)을 구현할 때, 크게 눈에 띄는 변화가 없어서 디버깅을 하기 힘들었다.

특히 여러 file을 거쳐서 코드가 진행됐기 때문에 printf를 찍어서 확인할 때도 상당히 복잡했다.

평일에는 매일 12시부터 3~4시까지, 주말과 연휴에는 거의 하루종일 zoom 화상회의를 통해 직접 얘기를 주고받으며 아이디어 design과 코딩을 했다. 또 디자인 과제2 해나로호와 due date가 겹쳐 체감적인 로드가 project 1, 2에 비해 더 컸다.

추가적으로 이 document의 문제들이 까다로워서 이것을 작성하는 것만 해도 하루정도의 시간이 소요되었다.

>> Did you find that working on a particular part of the assignment gave

>> you greater insight into some aspect of OS design?

VM의 개념이 몹시 복잡하다고 들어서 걱정이 많았지만 이번 구현을 통해서 주소값을 읽어주고 page directory를 통해 virtual address와 물리메모리의 address를 변환하고 적재하는 과정에 대해 이해하면서 크게 한걸음 성장한 것 같습니다. 직접 구현해보고 printf 로 주솟값 하나하나 찍어 디버깅하면서 구체적인 과정들을 이해할 수 있었고 복잡한 시스템의 디자인, 구현에 자신감이 생겼습니다.

I/O에 대한 개념이 거의 없다시피 한 상황이었지만 3-2 mmap file을 진행하면서 어느정도 이해할 수 있었고, project 4에 대한 자신감을 키울 수 있었습니다.

>> Is there some particular fact or hint we should give students in

>> future quarters to help them solve the problems? Conversely, did you

>> find any of our guidance to be misleading?

없습니다. 조교님들이 물심양면으로 도와주시니 항상 감사할 따름입니다.

>> Do you have any suggestions for the TAs to more effectively assist

>> students, either for future quarters or the remaining projects?

공식 document를 읽어도 알 수 없어 구글링한 부분들이 있습니다. mmap file시 고려해야하는 detail들이 많았는데 이런 것들이 좀 더 자세히 안내되었으면 좋겠습니다.

>> Any other comments?

감사합니다!