**Pintos Project 4: Virtual Memory**

담당 교수 : 김영재 교수님

이름 / 학번 : 20201614 이민석

개발 기간 : 2024.12.02~12.20

1. **개발 목표**

* 이번 PintOS 4번째 프로젝트 Virtual Memory에서는 기존의 PintOS 시스템에 가상 메모리 기능을 추가하고, 효율적인 메모리 관리를 구현하는 것이 목표이다. 이를 통해 프로세스마다 Virtual Address 영역을 확장하고, 메모리를 효율적으로 사용하게 된다. 우선 기존의 Page Table을 보완하는 Supplemental Page Table 및 Supplemental Page Table Entry를 구현하여 Virtual Page와 Physical Frame간의 매핑에 필요한 추가 정보를 효율적으로 관리해야 한다. 또한 Page Fault 발생 시, Page Fault Handler가 동작하여 Disk에서 필요한 데이터를 Physical Memory에 load하거나 새 Page를 할당하게 된다. 또한 Physical Memory에 저장 공간이 부족할 경우 page를 disk로 swap-out하거나, 다시 swap-in 하는 기능을 구현한다. 이 때 알맞은 page 교체 알고리즘을 택해야 한다. 마지막으로 스택 크기를 동적으로 확장하여 프로세스의 메모리 접근이 가능한 범위를 유연하게 확장하는 기능을 구현해야 한다. 이 또한 Page Fault handler에서 적절히 새로운 스택 page를 할당해야 한다.

1. **개발 범위 및 내용**
   1. **개발 범위**
   2. Page Table & Page Fault Handler
      * 기존 PintOS는 page fault가 발생할 경우, 가상 주소를 통해 찾는 physical frame이 존재하지 않는다면, 그 즉시 프로세스를 종료하였다. 그러나 virtual memory 시스템을 구현한다면, page fault 발생 시, 유효한 접근임을 확인한 후, 만약 데이터가 memory에 존재하지 않는다면(Stack growth 예외), 해당 데이터를 disk에서 swap-in을 하고 memory에 탑재해야 한다. 이를 통해 해당 페이지가 적절히 할당되어 page table에 추가된다. 이를 위해 Supplemental page table entry를 구현하여, 필요한 자료구조들을 바탕으로 page를 삽입하거나 삭제, page table을 관리하는 함수를 구현해야 한다.
   3. Disk Swap
      * Physical Memory의 용량이 가득 차 더 이상 frame를 추가할 수 없는 경우, Disk의 swap 공간을 이용하여 메모리를 관리해야 한다. 이 때 LRU 기반의 page replacement algorithm(이번 프로젝트에선 Second-Chance algorithm을 구현)을 통해 적절히 victim frame를 선택하여 swap space에 swap-out하고, 필요한 data의 page를 memory로 swap-in 할 수 있게 된다.
   4. Stack Growth
      * 기존 PintOS는 스택 size가 고정되어 확장된 스택 영역에 접근 시, page fault가 발생하였다. 그러나 실제 시스템에선 stack이 동적으로 확장되어야 하며, 이번 프로젝트에선 이를 구현해야 한다. PintOS문서에 따르면 최대 8MB까지 확장할 수 있도록 하며, 현재 stack범위를 벗어난 접근을 할 때, 유효한 스택 접근인지 확인한 후, 만약 유효하다면 해당 영역에 frame을 할당한다. 만약 아니라면, Segmentation fault로 처리하게 된다.
   5. **개발 내용**
2. Page fault가 발생하는 이유와 이를 handling하는 전반적인 과정을 서술
   * + 기존 PintOS는 process나 thread가 할당되지 않은 주소 공간을 접근하거나, 쓰기 권한이 없는 page에 write 작업을 할 때 page fault가 발생한다. 또한 기존 Virtual memory시스템은 모든 page를 physical memory에 load하지 않기 때문에, 만약 필요한 page가 메모리에 없다면 page fault가 발생한다. 따라서 이번 프로젝트에선, page fault handler를 직접 구현하여, 위와 같은 상황을 적절하게 처리해야 한다.
     + 이를 위해 supplemental page table entry가 필요하며, 이를 위한 구조체를 선언하여, virtual page의 상태를 기록하고 저장할 수 있도록 한다. 구조체의 필드로는 swap index, file에 관한 내용 등이 있을 수 있다. 이를 통해 page fault가 발생한 virtual page가 메모리에 없는 경우(swap space에 존재 또는 파일에서 읽어야 하는 경우) 해당 page를 메모리에 load해야 한다. 만약 stack 영역에서 page fault가 발생하였다면 stack을 확장해야 하며, 필요하다면 새로운 page를 할당(allocation)해야 한다. 이 과정에서, physical memory에 page를 load할 frame을 확보해야 한다. 앞에서 말한 두 가지 경우에 해당하지 않은, 유효하지 않은 page에 대해선 프로세스를 종료한다.
3. Disk swap 발생 시 사용한 page replacement algorithm에 대해 서술
   * + 앞서 1번에서 언급한 physical memory에 새 page를 load할 때, 사용 가능한(load 가능한) frame이 존재하는 경우, 그 위치에 page를 load할 수 있다. 그러나 사용 가능한 frame이 없는 경우, frame table entry 정보를 바탕으로 page replacement algorithm에 따라 메모리에서 evict할 frame을 선정하고, 이를 내보낸다. 이후 page를 해당 위치에 load한다.
     + 이번 프로젝트에선 page replacement algorithm으로 second-chance 알고리즘을 사용하였다. LRU 알고리즘의 근사적 형태이며, 구현이 비교적 간단하며 성능이 괜찮은 알고리즘으로 알려져 있기에 이를 택하였다. 우선 모든 frame은 circular buffer 형태로 관리되며, clock 포인터를 설정하여 이를 순회한다. 각 frame마다 accessed bit을 통해 최근에 참조 되었는지의 여부를 확인한다. 만약 페이지가 참조 된다면, 해당 bit은 1로 설정되며, page fault 발생 시, 해당 page를 clock 포인터가 확인한다면 1->0으로 bit을 전환한다. 순회하며, accessed bit = 0인 경우 이 frame을 replacement하는 대상 frame으로 선정한다.
4. Stack growth 구현 시 stack 확장 여부를 판단할 수 있는 방법에 대해 서술
   * + 우선 page fault가 발생한 address가 user valid address space인지 확인해야 한다. 또한 address가 스택의 최대 크기를 벗어나지 않는지 확인해야 한다. 마지막으로 stack pointer인 esp의 위치를 기준으로, page fault가 발생한 address가 esp보다 32byte 안쪽으로 벗어난 경우이면 스택 확장을 진행한다. 이 때 새 frame을 할당한 후, 해당 frame을 page table에 mapping한다. 또한 해당 page의 정보를 supplemental page entry에 저장한다.
5. **추진 일정 및 개발 방법**
   1. **추진 일정**
6. 12.02 ~ 12.06: Virtual memory 프로젝트의 전반적 이해
7. 12.08 ~ 12.11: Supplemental page entry 및 관련 함수 구현, page fault handler 구현
8. 12.19 ~ 12.20: frame table entry와 관련 함수 구현, "swap.h/c”, stack growth 구현.
9. 12.21: 최적화 및 보고서 작성.
   1. **개발 방법**
10. Page fault handling 및 Supplemental page entry
    * + “vm/page.h”, “vm/page.c”

우선 supplemental page entry역할을 하는 vm\_entry 구조체를 “page.h”에 선언한다. 해당 구조체의 필드로는 page의 type을 나타내는 변수, virtual page의 시작 주소를 저장하는 변수 등등 추가적으로 필요한 정보들을 저장하는 변수가 선언되어 있다 (이후 제작 내용에서 상세히 설명). 또한 page type을 나타내는 매크로를 설정한다. 또한 virtual memory table에 vm\_entry를 추가하거나 삭제, 검색하는 함수들을 “page.c”에 구현한다. virtual memory table을 제거하는 함수도 구현한다.

* + - “userprog/process.c”

“load\_segment()” 함수에서 실행 파일의 각 segment를 처리하면서, 해당 segment의 정보를 새로 추가한 supplemental page entry(vm\_entry) 구조체에 저장하여, “insert\_vme()”함수를 통해 virtual page table에 추가한다. “load\_menu()”함수는 page fault가 발생한 vm\_entry에 대해서 type을 바탕으로 “load\_file()” 또는 “load\_swap()”을 호출한다. “load\_file()”은 vm\_entry의 정보를 바탕으로 file에서 page를 읽어와 memory에 load한다. 마찬가지로 “load\_swap()”은 swap space에서 page를 일거와 memory에 load한다. 추가로 “process\_exit()”에서 프로세스가 종료될 시, “vm\_destroy()”함수를 호출하여 virtual table을 해제한다.

* + - “userprog/syscall.c”

모든 syscall에 대해서 해당 함수에 넘어가는 인자들의 주소들이 유효한지 확인해야 한다. 이 때 string 또는 buffer의 각 글자에 대해서 valid한 virtual address인지 확인해야 한다.

* + - “userprog/exception.c”

page fault handler의 함수를 재구성한다. 페이지가 존재하지 않는 page fault인지 확인한 후, “find\_vme()”함수를 사용하여 page fault가 발생한 주소에 해당하는 vm\_entry를 찾는다. 만약 찾은 경우, “load\_menu()”함수를 호출하여 file이나 swap space에서 page 데이터를 load한다. 또는 “grow\_stack()”함수를 호출하여 스택을 확장한다. 만약 위 경우에 대해서 실패한다면, process를 종료한다.

* + - “threads/thread.h”, “threads/thread.c”,

thread 구조체에 각 thread마다 virtual page table을 관리할 수 있는 hash table을 선언하고 이를 “init\_thread()”함수에서 초기화한다.

1. Disk swap 및 page replacement algorithm
   * + “vm/frame.h”, “vm/frame.c”

새로 frame\_entry 구조체를 “frame.h”에 선언한다. 구조체 필드로는 physical frame의 주소를 저장하는 변수, 해당 frame에 mapping 된 virtual page의 정보를 담고있는 vm\_entry 구조체의 포인터 등이 존재한다. “frame.c”에 구현된 함수로는 frame table을 초기화하는 “frame\_table\_init()”, frame table에서 특정 frame을 제거하고 해제하는 “free\_frame\_from\_table()”함수, 교체할 frame을 찾고 swap\_out 또는 file에 쓰는 “find\_evict\_frame()”함수, 새 frame을 할당하는 “frame\_alloc()”함수가 존재한다.

* + - “vm/swap.h”, “vm/swap.c”

swap space의 사용 여부를 bitmap 형태로 관리하는 변수와 swap space에 대하여 동시 접근을 방지하기 위한 lock변수를 추가해야 한다. 또한 함수로는 swap space를 초기화 하는 “swap\_init()”, 주어진 frame을 swap space에 저장하고, swap slot index를 반환하는 “swap\_out()”, swap space에 주어진 index에 해당하는 data를 frame으로 load하는 “swap\_in()”을 구현한다.

* + - “threads/init.c”

disk swap과 frame에 관련된 변수들 및 자료구조를 초기화 해야한다.

* + - “userprog/process.c”

“setup\_stack()”함수에서 stack 영역을 설정할 시, 초기화 시점에 stack에 사용할 frame을 “frame\_alloc()”함수를 이용하여 할당한다. 또한 “load\_file()”, “load\_swap()”함수에서 새 physical frame을 할당 시, “frame\_alloc()”함수를 이용하도록 한다.

1. Stack growth
   * + “userprog/process.c”

“setup\_stack()”함수는 프로그램 시작 시, stack에 사용할 한 page를 할당하고 이를 page table에 mapping한다. 이후 스택 영역에서 page fault가 발생한다면, “grow\_stack()”함수를 호출하여 해당 스택 확장이 적절한지 판단 후 확장한다. 새로운 physical frame을 할당하고, 이를 virtual address space에 mapping한다.

1. **연구 결과**
   1. **Flow Chart**
2. 텍스트, 도표, 그림, 스케치이(가) 표시된 사진

   자동 생성된 설명Page fault handling
3. 텍스트, 도표, 폰트, 평면도이(가) 표시된 사진

   자동 생성된 설명Disk swap
4. 텍스트, 스크린샷, 도표, 폰트이(가) 표시된 사진

   자동 생성된 설명stack growth

* 1. **제작 내용**

1. Page fault handling 및 Supplemental page entry
   * + “vm/page.h”



텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명supplemental page entry의 구조체 필드이다. “type” 변수는 virtual page의 타입을 나타내며, 각 type인 “VM\_BIN”, “VM\_FILE”, “VM\_ANON”은 “page.h”에 매크로로 값이 설정되어 있다. ‘VM\_BIN”은 실행 파일(binary file)에서 load된 page를 의미하며, “VM\_FILE”은 Memory-mapped file에 mapping된 page를 의미한다. “VM\_ANON”은 swap space에서 load되거나, 0으로 채워진 anonymous page를 의미한다. “vaddr는 vm\_entry가 관리하는 virtual page의 주소를 저장한다. 이는 hash table에서 hash 값을 추출하는데 사용된다. “writable”은 해당 page에 write 작업이 허용되는지 여부를 나타내고, “is\_loaded”는 해당 virtual page가 physical memory에 load되어 있는지를 나타낸다. “file”, “offset”은 memory mapped file에 해당하는 page에 대하여 해당 page를 가리키는 포인터와 파일 내에서 시작하는 offset을 저장한다. “bytes\_read”는 page를 load할 때, 파일에서 읽어야 할 data의 byte수를 저장하고, “bytes\_zero”는 0으로 채워야 할 data의 양을 저장한다. “elem”은 해당 vm\_entry를 hash table에 저장하기 위한 hash\_table 요소이다. “swap\_idx”는 swap space에 저장된 page의 위치를 저장한다.

* + - “vm/page.c”

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명page.c에 구현된 함수들이며 함수 구현 코드는 생략하였다. “vm\_hash\_func()”은 vm\_entry를 현재 thread의 hash\_table에 저장할 때 사용할 hash함수이다. “vm\_init()”함수는 vm\_entry들을 저장할 hash table을 초기화하는 함수이며, “threads/init.c”에서 호출된다. “hash\_init()”함수를 이용하여 hash table을 초기화하며, “vm\_hash\_func()”, “vm\_less\_func()”를 hash함수와 비교함수로 지정한다. “insert\_vme()”함수는 vm\_entry 구조체를 hash table에 삽입하며, hash\_insert()함수를 사용하여 hash\_table인 “vm”에 삽입한다. “delete\_vme()”함수는 hash table에서 특정 vm\_entry구조체를 삭제한다. “find\_vme()”함수는 인자로 넘어오는 vaddr에 해당하는 vm\_entry 구조체를 hash table에서 찾는다. 우선 vaddr를 page 단위로 정렬하기 위해 pg\_round\_down() 함수를 사용하며, 이를 바탕으로 새로운 vm\_entry를 생성하고 정렬된 가상 주소를 저장한다. 이를 현재 thread의 hash table에 저장한다. 만약 이를 성공하지 못한다면 NULL을 반환한다. “vm\_destroy()”는 vm\_entry들을 저장하는 hash table을 제거한다. “vm\_destructor()”함수를 이용하며 관련 resource들을 해제한다.

* + - “userprog/process.c”

기존 “load\_segment()” 함수에서 새로운 vm\_entry 구조체를 동적으로 할당하는 부분을 추가해야 한다.vm\_entry의 type을 “VM\_BIN”으로 설정하고 다른 구조체 필드 값도 초기화한다. vme->is\_loaded는 false로 설정하여 page loading을 지연한다. “insert\_vme()”함수를 사용하여 새로 할당한 vm\_entry를 현재 thread의 virtual page table에 삽입한다. 추가로 “setup\_stack()”함수의 기존 코드에서 “frame\_alloc()”을 이용하여 새 frame을 할당한다. 마찬가지로 vm\_entry를 동적으로 할당하고, type은 VM\_ANON으로 설정한다. 이후 “insert\_vme()”함수를 사용하여 vm\_entry를 현재 thread의 virtual page table에 삽입한다.

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명이텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명 함수는 “vm\_entry”의 type에 따라 적절한 page load함수(“load\_file()”, “load\_swap()”)을 호출한다. 이는 “page\_fault()”함수에서 page fault가 일어난 address의 mapping된 page가 존재한다면 호출되게 된다.

이 함수는 ELF file 또는 memory-mapped file에서 data를 읽어와 physical memory에 load한다. vm\_entry 구조체의 정보를 기반으로 어떤 file의 offset에 따라 얼마나 많은 data를 읽어야하는지 결정한다. “file\_read\_at()”함수를 사용하여 vme->file에서 data를 읽어온다. 만약 성공적으로 읽어왔다면, 해당 vme->is\_loaded를 true로 설정하여 해당 page가 memory에 load되었음을 표시하고, install\_page()함수를 호출하여 virtual address와 physical address를 page table에 mapping한다. 만약 실패한다면 할당받은 frame을 “free\_frame\_from\_table()”을 호출하여 해제한다.

이텍스트, 스크린샷, 디스플레이, 소프트웨어이(가) 표시된 사진

자동 생성된 설명 함수는 swap space에서 data를 읽어와 physical memory에 load한다. vm\_entry 구조체의 swap\_idx 필드를 통해 swap space에서 swap frame의 위치를 토대로 “swap\_in()”함수를 호출하여 data를 frame\_ptr->frame에 읽어온다. “load\_file()”과 마찬가지로 “install\_page()”함수를 호출하여 가상 주소와 physical address를 page table에 mapping하고, 성공여부에 따라 vme->is\_loaded 변수값을 설정한다.

* + - 텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

      자동 생성된 설명“userprog/syscall.c”

위 3개의 함수들은 syscall function의 인자로 넘어가는 변수들의 address를 확인하는 함수이다. “check\_valid\_addr()”함수는 주어진 가상 주소(addr)가 유효한 user space address인지 확인한다. 그리고 이에 해당하는 vm\_entry를 반환한다. 만약 해당 주소에 대한 vm\_entry가 없다면 stack 확장을 시도한다. “check\_valid\_buffer()”함수는 주어진 buf가 유효한 user space address인지 확인한다. 각 byte에 대해 유효성 검사를 수행하며, vm\_entry의 write가 가능한지도 확인한다. “check\_valid\_string()”함수도 주어진 str이 유효한 user space address인지 확인한다. str의 각 byte에 대해 검사를 수행하며 “\x00”을 만날때까지 반복한다. 앞서 말한 두 함수 모두 내부적으로 “check\_valid\_addr()”를 호출하여 주소를 검사하게 된다.

* + - “userprog/exception.c”

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

“page fault()”함수의 일부이다. 페이지가 존재하지 않는 page fault인지 확인한 후, “find\_vme()”함수를 사용하여 page fault가 발생한 주소에 해당하는 vm\_entry를 찾는다. 만약 찾은 경우, “load\_menu()”함수를 호출하여 file이나 swap space에서 page 데이터를 load한다. 또는 “grow\_stack()”함수를 호출하여 스택을 확장한다. 만약 위 경우에 대해서 실패한다면, exit(-1)을 통해 process를 종료한다.

* + - “threads/thread.h”, “threads/thread.c”,

thread 구조체에 각 thread마다 virtual page table을 관리할 수 있는 hash table을 선언하고 이를 “init\_thread()”함수에서 초기화한다.

1. Disk swap 및 page replacement algorithm
   * + 텍스트, 폰트, 스크린샷이(가) 표시된 사진

       자동 생성된 설명“vm/frame.h”

frame\_entry 구조체의 필드이다. “frame”은 physical frame의 시작 주소를 저장하고, “vme”는 해당 frame에 매핑된 supplemental virtual page 정보를 담고있는 vm\_entry구조체의 포인터이다. “elem”은 frame table list인 “ft\_list”에서 사용할 list요소이며, “thread”는 해당 frame을 사용하고 있는 thread의 포인터이다.

* + - “vm/frame.c”

폰트, 텍스트, 스크린샷, 타이포그래피이(가) 표시된 사진

자동 생성된 설명“frame\_table\_lock”변수는 frame table에 대한 동시 접근을 방지하기 위한 lock변수이다. “ft\_list”는 frame\_table을 list 형태로 관리하는 변수이며, “ft\_clock”은 second-chance algorithm을 위한 pointer이다.

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명위에텍스트, 폰트, 스크린샷, 번호이(가) 표시된 사진

자동 생성된 설명 선언된 frame과 관련된 변수들을 초기화하는 작업을 한다.

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명새로운 physical frame을 할당하고, 해당 frame에 대하여 frame\_entry를 생성한다. 이때 “malloc()”, “memset()”을 사용하여 frame\_entry 구조체를 동적으로 할당 후, 0으로 초기화를 한다. “palloc\_get\_page()”함수를 사용하여 실제 physical frame을 할당받는데, 만약 할당에 성공하면, frame\_ptr에 해당 정보를 저장하고, 만약 실패한다면 while문을 통해 “palloc\_get\_page()”가 성공할 때까지, “find\_evict\_frame()”함수를 호출하여 교체할 frame을 찾고 교체작업을 수행한다.

인자로 넘어오는 frame에 대하여, 해당 frame을 frame table에서 제거하고 해제하는 작업을 한다. ft\_list를 순회하며, 인자로 넘어온 frame과 일치하는 frame\_entry->frame을 찾게되면, “pagedir\_clear\_page()”를 통해 해당 page의 table 매핑을 제거하고, “palloc\_free\_page()”를 통해 physical frame도 해제한다. 이 때, ft\_clock 포인터가 현재 list\_elem을 가리키고 있다면 다음을 가리킬 수 있도록 해야 한다.



함수의 코드가 길어 코드 부분 사진은 생략하였다. Clock 알고리즘을 사용하여 교체할 frame을 선택하고, 해당 frame을 swap\_out하거나 file에 쓰는 함수이다. 우선 while 루프를 통해 교체할 frame을 찾는다. “pagedir\_is\_accessed()”함수를 사용하여 해당 page의 “accessed” bit이 0인 프레임을 찾아 교체대상으로 선택한다. “pagedir\_is\_dirty()"함수를 사용하여 해당 page의 “dirty” bit을 확인하는데 이와 더불어 “vme->type”을 확인하여 각 type에 맞게 ‘swap\_out’하거나, frame의 내용을 file에 쓴다. 마지막으로 해당 frame\_ptr->vme->is\_loaded 변수를 false로 하여 memory에서 제거되었음을 시하고, frame을 해제하는 과정을 거친다.

* + - “vm/swap.h”, “vm/swap.c”

텍스트, 폰트, 스크린샷, 타이포그래피이(가) 표시된 사진

자동 생성된 설명“swap\_map”은 swap space의 사용 여부를 bitmap으로 관리하는 변수이다. 각 bit자리는 swap slot에 해당되며, 0은 사용가능, 1은 사용 중을 나타낸다. “swap\_lock”은 swap space에 대한 동시 접근을 방지하기 위한 lock 변수이다. “swap\_block”은 swap partition을 나타내는 block 구조체이다.

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명“bitmap\_create()”를 사용하여 swap bitmap을 생성한다. 만약 bitmap 생성이 실패하면 함수를 종료한다. 또한 “bitmap\_set\_all()”을 사용하여 swap bitmap의 모든 bit를 0으로 설정한다. “swap\_lock”도 초기화한다.

텍스트, 스크린샷, 폰트, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

swap space에서 인자로 넘어오는 지정된 index에 해당하는 page data를 physical frame으로 load한다. 우선 used\_index가 0보다 큰 유효한 인덱스인지 확인후, for loop을 통해 swap slot의 모든 sector에 대해서 data를 읽어와 frame에 저장한다.

인자로 주어진텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명 frame의 data를 swap space에 저장하고, swap slot의 index를 반환한다. “bitmap\_scan\_and\_flip()”를 사용하여 swap bitmap에서 사용가능한 slot을 찾고 해당 slot을 1로 변경한다. “block\_write()”함수를 사용하여 frame의 data를 swap partition에 저장한다.

* + - “threads/init.c”

폰트, 텍스트, 스크린샷, 디자인이(가) 표시된 사진

자동 생성된 설명

“main()”함수의 일부이며, disk swap과 frame에 관련된 변수들 및 자료구조를 초기화하는 함수를 호출한다.

1. Stack growth
   * + “userprog/process.c”

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명“grow\_stack()”함수의 일부이다. 인자로 받은 “esp”와 “addr”를 통해 스택 확장을 해도되는지에 대한 여부를 판단한다. 이후 “frame\_alloc()”함수를 통해 새 frame을 할당하고, “install\_page()”함수를 통해 vaddr와 new\_frame->frame의 매핑을 추가한다. vm\_entry를 새로 동적으로 할당 후, 정보를 초기화 한다. frame할당에 실패하거나 vm\_entry할당에 실패한다면, 할당된 frame을 “free\_frame\_from\_table()”함수를 통해 해제한다. 성공적으로 stack을 확장했다면 true를 반환한다.

* 1. **시험 및 평가 내용**

텍스트, 스크린샷, 폰트, 흑백이(가) 표시된 사진

자동 생성된 설명