**Pintos Project 4: Virtual Memory**

담당 교수 : 김영재 교수님

이름 / 학번 :최재원 / 20171700

개발 기간 : 2021.11.05~2021.11.27

1. **개발 목표**

현재 Pintos 상에서 돌아가는 프로그램의 크기는 main memory 크기에 제약을 받는다. 즉, 프로그램을 실행하려면 실행에 필요한 데이터가 모두 Physical memory에 올라가야 하며, 현재 user stack 은 한개의 page만이 할당되어 있다. 따라서 현재 페이지 테이블에서 필요한 부분을 덧붙여 구현하고(supplemental page table) 현재 main memory에 데이터가 없어서 발생하는 page fault의 경우 이전과 다르게 page fault handler를 통해 disk로 부터 해당 페이지 메모리를 가져오는 swap disk를 구현한다(이 때 LRU 알고리즘을 사용 할 수 있다.) 그리고 스택의 영역이 모자라서 발생하는 Page fault를 개선하기 위해 하나의 page 이상의 데이터를 저장할 수 있도록 Stack growth를 구현한다.

1. **개발 범위 및 내용**
   1. **개발 범위**
   2. Page Table & Page Fault Handler

기존 page fault() 함수에서는 Page fault가 발생하면 무조건 프로그램을 종료시켰지만,

Fault\_addr, not\_present, write, user 등의 변수를 통하여 Page fault의 원인에 따라 다른 조치를 취하도록 수정하도록 한다. 이를 통해 현재 물리 메모리에 데이터가 올라와 있지 않은 경우 등에 대해서 프로그램을 종료시키는 것이 아닌 다른 수행을 할 수 있도록 한다, 즉 유효한 메모리 주소일 경우 디스크에 있는 물리 페이지 프레임을 가져오며 스택공간이 부족한 경우는 페이지 추가 할당을 해줌으로써 이전의 Limitation을 극복할 수 있다.. 그리고 기존 Page table에 필요한 정보를 추가함으로써 supplemental page table을 구현 한다.

* 1. Disk Swap

현재 pintos는 Physical memory에 프로그램의 data가 올라와 있지 않으면 page fault를 발생시키고 프로그램을 종료시킨다. 이 때문에 프로그램들은 그 size에 큰 제약이 생기며, 실제 프로그램들의 코드 용량이 매우 크기 때문에 모든 data를 물리 메모리에 적재할 수 없다. 따라서 프로그램의 일부만을 물리 메모리에 적재 시키고, 필요할 때 disk로 부터 Swap 해오는 ( LRU, LFU 등이 사용 가능하다. ) Lazy Load방식을 구현함으로써 기존의 제약조건을 없애고, 더 효율적인 메모리 운용을 가능하게 한다.

* 1. Stack Growth

Project2에서 구현한 stack은 고정된 size의 Page로 구현되었기 때문에, program들은 해당 영역만큼의 virtual address space를 사용할 수 밖에 없었다. 하지만 필요에 따라 stack의 크기를 증가시켜 줌으로써, 즉 페이지를 추가 할당해 줌으로써 더 많은 가상 메모리를 사용할 수 있도록 한다. 아래와 같이 가상메모리 3GB에 대해서 pintos에서 정의된 PGSIZE 4096B를 기준으로 현재 할당되어 있는 스택 메모리 주소 범주 안에서 벗어날 경우 추가로 할당해주게 된다. 이 때 pintos에서는 최대 8MB까지 그 크기를 제한하고 있다.

텍스트이(가) 표시된 사진

자동 생성된 설명

* 1. **개발 내용**

1. Page fault가 발생하는 이유와 이를 handling하는 전반적인 과정을 서술

Page fault가 발생하는 이유는 현재 physical memory에 해당되는 주소의 data가 없는 경우( disk에 있는 경우 ), 현재 할당된 Page의 범위 이외의 주소를 reference 하는 경우, 그리고 잘못된 주소를 reference 하려는 경우(Kernel 영역 및 max 또는 min으로 정해 놓은 범위 외의 주소를 참조하려는 경우) 등이 있다. 맨 마지막의 경우는 threads/vaddr.h에 존재하는 is\_user\_code() 함수와 앞선 1번에서 언급한 bool 변수들을 사용하여 예외처리를 할 수 있으며, 이 때는 error situation이기 때문에 Exit() 함수를 호출하여 Thread를 종료시킨다. 이 외에 앞의 두가지 경우는 각각 swap disk 및 Stack growth 부분을 구현함으로써 disk로부터 필요한 data를 mapping하거나, 필요한 만큼의 page를 추가적으로 할당 해줄 수 있다.

1. Disk swap 발생 시 사용한 page replacement algorithm에 대해 서술

**텍스트, 시계이(가) 표시된 사진

자동 생성된 설명**LRU, LFU 등의 page replacement policy가 있고, min algorithm, random algorithm, fifo algorithm등이 있지만, 이번 프로젝트에선 LRU policy를 따르는 LRU 기반의 알고리즘을 사용한다. 가장 오랫동안 참조되지 않은 page를 교체하는 수행을 원칙으로 하며, locality에 기반을 둔 교체 기법이다. Page 참조시 마다 시간을 기록해야 하는 Overhead가 있지만, 간소화된 정보 수집으로 해소 가능하다. 즉, approximating LRU를 하는 것인데, 그 중 second chance algorithm을 사용한다. 주기적인 초기화 없이 reference bit을 사용한다. Page frame들을 순차적으로 가리키는 pointer를 사용하여 교체될 Page를 결정한다. 현재 가리키고 있는 page의 reference bit을 확인하여 그 값이 0인 경우, update bit까지 0인지 확인하고 0이면 교체 page로 결정하고, update bit의 값이 1인 경우는 write back list에 추가 한 후에 update bit를 0으로 바꿔준다. reference bit의 값이 1인 경우엔 0으로 초기화 후에 pointer를 다음 page로 이동시킨다. 먼저 적재된 page가 교체될 가능성이 높으며, 이는 fifo와 유사하다. 그리고 reference bit을 사용하여 locality를 활용해 교체 페이지를 결정한다는 점에서 LRU에 유사하게 작동한다고 할 수 있다.

1. Stack growth 구현 시 stack 확장 여부를 판단할 수 있는 방법에 대해 서술

Pintos에서 정의된 PHYS\_BASE는 Virtual memory에서 Kernel 영역과 User 영역의 경계점이자, stack 주소의 base를 저장하고있다. 메뉴얼에서 stack의 size를 최대 8MB로 규정했으므로, userprog/exception.c의 page\_fault() 함수에서 cr2 register로부터 page fault가 발생한 주소를 fault\_addr 변수에 저장하고 해당 주소가 PHYS\_BASE – 8\*2^20보다 크고 PHYS\_BASE보다 작은지 확인함으로써 stack 확장 여부를 판단할 수 있다. 해당 범위에 포함되고, not\_present변수값이 true여서 Page가 할당되지 않은 주소에 대한 reference인 경우에는 page를 새로 할당해주며, 이 때 1개의 Page를 할당하여도 여전히 Page fault가 발생하는 범위에 있으면 추가적으로 필요한 만큼 Page를 할당해준다. 추가적으로, fault\_addr가 Stack 영역에 대한 접근인지 확인하기 위해 esp를 사용해줄 수 있다. Esp – 32보다 하위 주소이면 stack을 더 이상 증가시킬 수 없는 상태인 것을 알 수 있다. 한편 새로운 page를 할당시켜 줄 때는 새로운 frame을 할당받고, 해당 frame을 page에 연결한다.

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

|  |  |
| --- | --- |
| Duration | Contents |
| 11.05 – 11.12 | Studying project background |
| 11.13 – 11.17 | Supplemental page table |
| 11.18 – 11.22 | Paging & swap disk |
| 11.23 – 11.25 | Stack growth |
| 11.26 – 11.27 | Writing the report |

* 1. **개발 방법**
* II. B.의 개발 내용을 구현하기 위해 각각에 대해 다음 사항들을 포함하여 설명
  + 수정해야하는 소스코드
  + 수정하거나 추가해야 하는 자료구조
  + 수정하거나 추가해야 하는 함수
  1. Page Table & Page Fault Handler

Page table을 관리하기 위해서 supplemental page table이 필요하며, page fault handler를 위해서는 userprog/exception.c의 page\_fault()함수를 수정하는 것이 필요하다. 우선, supplemental page table은 userprog의 process.c의 setup\_stack()함수와 load\_segment()함수를 수정하여 page를 생성하고 할당하는 과정을 page.h 존재하는 page 구조체를 이용하여 추가 정보를 저장하도록 한다. Page에 저장되는 page table entry는 lib 디렉토리의 hash 자료구조를 사용한다. 이외에 저장되는 meta data로는 virtual address, 어떤 정보를 저장하고 있는지에 대한 status 변수, 그리고 write 가능한 page인지를 나타내는 writable 변수 등이 있다. Program에서 임의의 가상 주소에 접근하려고 할 때, thread는 해당 주소가 포함된 page의 시작 주소( = pg\_round\_down() 함수 사용)를 찾고, 이에 mapping되는 physical address가 있는지 Page table을 찾게 된다. 해당 page에 대응되는 frame에 대한 정보를 저장할 구조체를 vm 디렉토리의 frame.h에서 구현한게 되고, 해당 구조체에 속해 있는 변수로 physical address 등을 저장할 변수들을 선언한다. 다시 page table로 넘어가서 만약 대응되는 frame에 대한 정보가 없다면 page fault를 발생시키게 되고, 경우에 따라서 앞에서 언급한 바와 같이 stack growth 또는 disk swap을 수행하도록 한다.

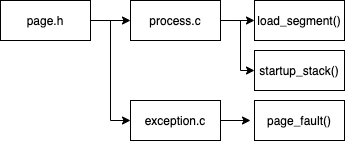
* 1. Disk Swap

Disk 와 frame(물리 메모리)간의 swap을 구현하기 위해서, 앞 서 frame table 에 대한 구조체를 frame.h에 선언한 바와 같이 swap.h에 Swap table에 대한 구조체를 선언한다. 위의 page fault() 함수에서 프로그램의 메모리 범위 안에 포함되지만, 대응되는 Frame table 정보가 없을시에 disk로 부터 해당 메모리 데이터를 가져오기 위해서 swap table을 사용하게 되고, 해당 page replacement algorithm을 swap.c에 구현하게 된다. 해당 알고리즘에 따라서 교체할 frame을 선택하게 되며, frame이 아직 swap되지 않았는지, page가 유효한지 등을 고려하게 된다. 그리고 frame.c, swap.c등에서 해당 동작을 수행할 때 lock이 필요한 경우 사용해 주었으며, deadlock 발생을 최소화 시키기 위해서 block\_lock, frame\_lock등으로 나누어 사용하였다.

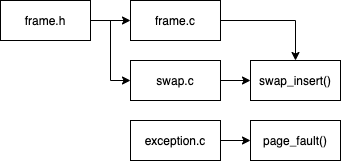
* 1. Stack Growth

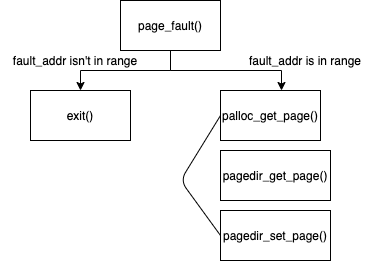
기본적으로 page fault가 발생한 주소를 사용하여 해당 주소가 정상적인 주소를 참조하는지 판단한 후, stack growth를 진행해주게 된다. Userprog 디렉토리의 Exception.c의 page\_fault() 함수에서 구현해주며, 앞 서 언급했던 writer, user, not\_present 등의 변수를 사용하여 stack growth를 진행해주면 안되는 예외의 상황에 대한 예외 처리를 해주게 되며, 스택의 범위에 해당하는지는 8MB라는 범주 안에 있는지와 struct intr\_frame 의 esp를 사용하여 esp – 32보다 크거나 같은 주소인지를 판별하게 된다. 이 후 정상적으로 stack growth를 진행해도 되는 경우 새로운 페이지를 할당받아서 페이지를 추가해주게 되고, 해당 정보들에 대해서 마찬가지로 업데이트 해주게 된다.

1. **연구 결과**
   1. **Flow Chart**
      1. Page Table



* + 1. Disk Swap



* + 1. Stack Growth
  1. **제작 내용**
     1. 텍스트이(가) 표시된 사진

        자동 생성된 설명Page Table

왼쪽은 page.h에 정의된 page 구조체이다. Hash\_elem 은 Hash table element를 의미하며, hash.h로부터 hash table 구조체를 사용하게 된다. Addr은 virtual address를 저장하고 있고, 이밖에 status는 ALL\_ZERO, IN\_FRAME\_TABLE 등의 총 4가지 상태를 나타내며, 현재 페이지가 저장하고 있는 메모리 내용에 대해서 정수값으로 저장하고 있다. Writable은 Write 가능한 page인지를 나타내며, read\_bytes는 file로부터 읽어온 바이트 수를 저장하고, block\_sector는 swap table 안에 존재하는 sector를 의미하며, 이는 frame\_entry인 frame에 대응된다. 마지막으로 해당 페이지에 대한 page directory 정보가 저장되어있는 pagedir 변수도 선언해주었다.

* + 1. Disk Swap

Swap disk를 구현하기 위해서 pintos-mkdisk swap.dsk –swap-size=2 로, 2MB 크기의 디스크를 생성해 주었고, 이밖에 Swap table에 대해서 Device 디렉토리에 존재하는 block.c의 block\_read() 및 block\_write()을 활용하였다. 다만, swap에 있어서 제대로 구현하지 못하였고, 결과적으로 최종 Test에서 5개에 대해서 fail이 나오게 되었다.

* + 1. Stack Growth

**텍스트이(가) 표시된 사진

자동 생성된 설명**

위 코드는 userprog 디렉토리의 exception.c의 page\_fault() 함수에 대한 코드이다. Stack growth에 대한 구현을 하기 위해서 thread 디렉토리의 thread.h에 int32\_t 타입의 current\_stack 변수를 선언해 주었다. 해당 변수에 현재 할당받은 스택의 최소 주소 (마이너스로 증가하므로 )를 저장하게 된다. 우선 not\_present 변수와 is\_user\_vaddr() 함수를 사용하여 page fault처리를 해야 하는 경우에 대한 예외처리를 해주고, 이 후 해당 fault\_addr 변수가 저장하고 있는 변수를 사용하여 허용된 범위에 있는지를 if문을 사용하여 검사한다. PHYS\_BASE 변수는 kernel 영역과 user 영역의 경계 주소를 나타내고 있고, PGSIZE는 page 하나의 크기인 4KB를 나타낸다. 따라서 PHYS\_BASE – PGSIZE \* 20은 user 영역의 메모리 영역 최대 8MB에 포함되는지를 의미한다. 다음으로 f->esp로부터 -32인 주소보다 크거나 같은지 검사 함으로써, stack 영역의 코드 데이터인지를 검사한다. 조건문에 포함되지 않는 경우에는 exit(-1)로 해당 thread를 종료시켜 주고, 두가지 조건문에 해당할 경우 while문을 사용하여 새로운 페이지를 할당해주게 된다. Palloc\_get\_page 함수를 사용하여 새로운 페이지를 할당받고, pagedir\_get\_page 함수를 사용하여 page directory에 대해 할당 받은 후, pagedir\_set\_page 함수를 사용하여 해당 페이지와 디렉토리를 mapping 함으로써 stack growth를 구현하였다. 초기에 Page 변수는 pg\_round\_down() 함수를 사용하여 현재 주소를 Page 단위의 주소로 내림한 정보를 저장하고 있다. While문을 사용하여 구현해 주었기 때문에 페이지를 할당해주고, 여전히 해당 주소가 할당된 페이지 주소 범위에 포함되지 않는 경우 해당 주소까지 페이지를 여러개 할당해주도록 한다. 마지막으로 thread\_current()를 나타내는 cur 변수에 대해 current\_stack 변수의 값을 update해주고 return 해주게 된다.

* 1. 텍스트이(가) 표시된 사진

     자동 생성된 설명**시험 및 평가 내용**



총 16개의 test중에 11개를 통과하였다.