**Pintos Project 4: Virtual Memory**

담당 교수 : 김영재 교수님

이름 / 학번 : 김동건 / 20211507

개발 기간 : 2024.12.01 ~ 2024.12.22

1. **개발 목표**

* 해당 프로젝트에서 구현할 내용을 간략히 서술  
  기존에는 virtual memory없이 main memory에만 저장을 했는데, virtual memory를 구현하여 이를 해결한다. page\_fault 발생 시 기존에 kill 방식으로 구현되어 있는 부분을 수정한다. page가 없을 경우 강의 시간에 배운 Second Chance Algorithm을 사용하여 swap을 하면서 메모리를 관리할 수 있는 운영체제를 만드는 것을 목표로 한다. 또한, 스택 영역을 2MB로 고정하는 것이 아닌 8MB까지 필요할 때 늘어날 수 있게 구현한다.

1. **개발 범위 및 내용**
   1. **개발 범위**

* 아래 각 항목 개발의 필요성 또는 개발 시 기대되는 결과를 간략히 서술
  1. Page Table & Page Fault Handler  
       
     프로그램에 필요한 메모리를 한 번에 저장하고 있는데, 필요할 때에만 Load할 수 있게 된다. 그로 인하여 page fault가 발생할 시 프로그램이 종료되지 않고, 디스크에서 Load하고 다시 실행될 수 있게 한다.
  2. Disk Swap  
       
     물리 메모리에서 공간이 부족할 경우를 처리하기 위해 disk swap을 구현한다. 물리 메모리가 부족한 경우에도 비정상적으로 구동되지 않고, Disk로 swap을 진행하여 프로그램이 정상적으로 구현되게 된다.
  3. Stack Growth  
       
     현재 핀토스에서는 기존에 제공된 스택 영역인 4KB를 벗어나게 되는 경우 올바르게 작동하지 않는데, 이번 프로젝트를 통해 스택 영역을 확장시켜 적정 범위 내에서는 스택을 성장시켜 프로그램이 더 넓은 스택 영역을 가지고 구동될 수 있게 구현한다.
  4. **개발 내용**
* 아래 항목의 내용만 서술
  1. Page fault가 발생하는 이유와 이를 handling하는 전반적인 과정을 서술  
       
     현재 page table에 존재하지 않는 페이지에 접근하려 하면 page fault가 발생한다. 다시 말해 물리 메모리에 존재하지 않는 영역에 접근하는 경우 page fault가 발생한다. 또, 할당된 영역 밖을 접근할 때에도 page fault가 발생한다. 첫 번째 경우에는 해당 entry를 찾아 disk에서 메모리를 로드하고, 다시 찾으면 fault가 발생하지 않게 된다. 두 번째 경우에는 exit(-1)을 통해 코드를 종료시킨다.
  2. Disk swap 발생 시 사용한 page replacement algorithm에 대해 서술  
       
     Disk Swap 시에 page replacement algorithm은 second chance algorithm을 사용하려 하였다. 이는 disk swap이 필요로 하는 상황이 발생 시 victim pointer가 가르키고 있는 것을 확인 후, access bit가 0이라면 그것을 evict한다. 만약 1이라면 0으로 바꾼 후, 계속 순회한다. 즉, evict되기 전 한 번의 기회를 부여받아 살아남는다. 이 알고리즘은 LRU알고리즘과 비슷하게 돌아가는 LRU Approximation 알고리즘이다.
  3. Stack growth 구현 시 stack 확장 여부를 판단할 수 있는 방법에 대해 서술  
       
     우선, 확장이 되더라도 8MB이상 확장은 시키지 않을 것이다. 따라서, 스택포인터의 영역을 체크한다. 즉, PHYS\_BASE를 기준으로 8MB영역 안에 fault\_address가 있는지 확인한다. 만약 이를 만족한다면 현재 esp에서 32를 뺀 값이 fault\_address보다 작은지도 확인한다. 이를 하는 이유는 메뉴얼에도 나와있듯 PUSHA 명령어를 사용하면 32byte가 push되기 때문이다. 따라서 이를 참고하여 esp-32역시 fault\_address보다 작은지 확인하여 주었다. 여기까지의 조건을 모두 만족하면 stack을 확장할 수 있는 경우이다.

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

* II. A. 개발 범위를 포함하여 구현 내용에 대한 일정 작성
* 과제 명세서, 메뉴얼 읽고 방식 생각하기: 12/01 ~ 12/05
* page, frame, swap 구현 : 12/06 ~ 12/10
* expand stack 구현 및 init.c thread.c syscall.c, process.c 로직 추가: 12/11 ~ 12/21
* 보고서 작성 및 제출 : 12/22
  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 Fault

우선, page\_fault 관리를 위해 vm 폴더 밑에 page.c, page.h를 구현한다. 여기서 vm\_entry를 관리하기 위한 구조체를 추가하고, get\_vme, insert\_vme 등 vme를 관리하기 위한 함수들을 구현하고, file을 load하는 load\_file을 구현한다. 다음으로, thread.h에 hash vm을 추가하여 관리한다. init.c에서는 선언한 자료구조들을 초기화 해주는 함수를 추가한다. 마지막으로 exception.c에서 page fault가 난 경우 vm\_entry를 찾아 page fault를 해결하는 로직을 추가한다.

2. Disk Swap

Disk Swap을 위한 swap.c swap.h를 vm 폴더 밑에 구현한다. disk swap in/out의 로직을 여기에 구현하고, 핀토스에서 제공하는 bitmap등의 자료구조를 사용하여 구현한다. page.c page.h에서는 page 정보를 관리하는 구조체를 추가하여 관리한다. frame.c frame.h도 vm 폴더 밑에 구현하는데, second chance algorithm의 로직이 frame 파일에 담긴다.

3. Stack Growth

exception.c에서 vme에 없는 경우 주소의 유효성을 검사한 뒤 스택을 늘리는 로직을 구현한다.

마지막으로 vm 폴더 밑에 있는 파일들을 위해 Makefile도 적절히 수정하여 준다.

1. **연구 결과**
   1. **Flow Chart**

* II. B. 개발 내용의 각 항목에 대하여 Flow Chart 작성

1. Page Fault

도표, 스케치, 패턴이(가) 표시된 사진

자동 생성된 설명

2. Disk Swap

도표, 기술 도면, 평면도, 텍스트이(가) 표시된 사진

자동 생성된 설명

3. Stack Growth

도표, 기술 도면, 평면도, 라인이(가) 표시된 사진

자동 생성된 설명

* 1. **제작 내용**
* II. B. 개발 내용의 각 항목에 대하여 실질적으로 구현한 코드의 관점에서 작성 (구현 내용, 알고리즘 등을 명확히 서술할 것)
  + 구현에 있어 Pintos에 내장된 라이브러리나 자체 제작한 함수를 사용한 경우 이에 대해서도 설명
* 개발 중 발생한 문제나 이슈가 있으면 이를 간략히 설명하고 해결한 방식에 대해 설명

각 파일에 대한 전체적인 작동 로직은 |||. B개발방법에 작성되어 있고, 이번 문항에서는 각 코드의 구체적인 동작을 살펴본다.

1. page fault

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

위와 같이 구조체를 생성하였다. 이 구조체에는 가상 메모리에 해당하는 물리 메모리 정보가 담겨있다. load시 기존에는 바로 물리 메모리에 페이지를 할당하였는데, 프로젝트4 부터는 이 구조체를 활용하여 여기에 우선 저장을 한다. thread에는 텍스트, 폰트, 스크린샷, 디자인이(가) 표시된 사진

자동 생성된 설명과 같이 hash 자료구조를 선언하였고, vm entry들을 관리하는 데 사용하였다.

이제 page\_fault가 발생되었을 때를 살펴보자.

텍스트, 소프트웨어, 스크린샷이(가) 표시된 사진

자동 생성된 설명

vme에 있는 경우 명세서에서와 같이 mm\_fault를 handling하는 코드를 작성하였다. 남은 프레임이 있는 경우 swap을 진행하는 것을 확인할 수 있다. 만약 그렇지 않은 경우는 바로 아래에 stack growth 로직을 작성하였는데 이는 3번 문항에서 다룰 예정이다.

2. Disk Swap

메모리가 꽉 찼는지 확인하고, 그렇다면 disk 영역을 사용하여 가상메모리처럼 사용하여야 한다. 메모리가 꽉 찼는지 여부는 아래 구조체를 통해 관리하였다.

텍스트, 폰트, 스크린샷, 번호이(가) 표시된 사진

자동 생성된 설명

이것을 pintos에서 제공하는 리스트 자료구조로 관리하였다. 이제, page replacement 정책에 대해 살펴보자. 해당 부분은 vm/frame.c에 구현되어 있다. 가장 핵심이 되는 try\_to\_free\_pages를 살펴보자.

텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명

victim pointer를 옮겨가면서 e에 해당 element를 저장한다. 그 element를 now에 담고, 만약 accessed bit가 0이라면 evict하는 것을 구현하였다. 해당 부분은 do while내의 첫 if문 내에서 확인할 수 있다. 이때, dirtybit가 켜진 경우에는 디스크에 수정 작업이 필요하기 때문ㅇ if문으로 관리하는 모습을 확인할 수 있다.

만약 accessed bit가 0이 아닌 경우에는 accessed bit를 0으로 변경 후, e를 옮긴다. 이는 기회를 한 번 더 준다는 의미이다.

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

second chance 알고리즘을 구현하기 위한 사용 함수들인데, 예외처리할 부분에 유의하여야 하였다.

이제, swap\_in/out에 대한 부분을 살펴보자.  
텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

핀토스에서 제공하는 bitmap 자료구조를 활용하였다. 비트맵을 통해 해당 영역이 비어있는지를 확인할 수 있다. in, out 모두 블록 상태의 변경이 있으므로 filp을 해주는 작업이 있는 것을 확인할 수 있다.

3. Stack Growth

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

exception.c에 있는 page\_fault 부분이다. 과제 명세서에 따르면 valid한 reference가 아닌 경우 즉, 할당된 PTE가 아닌 경우에는 Stack이 growable한지 확인 후, 가능하다면 expand하게 해야 한다고 나타나있다.

따라서 사진 코드 바로 위쪽에서는 1에서 설명한 page fault로직이 작성되어 있다. 만약 valid하지 않은 reference라면 stack growth 영역까지 오게 되고 위 코드가 실행된다. 첫 3개의 if문은 stack grow가 가능한 Region인지 판단하는 부분이다. 첫 두 if문은 확장 가능한 영역 내에 있는지 확인하는 코드이다. 8MB까지 확장가능하므로 8 \* 1024 \* 1024 (=8MB)를 이용하여 계산하였다. 마지막 if문은 핀토스 메뉴얼에 명시되어 있는 조건으로, 핀토스에서 한 번에 32byte를 push하는데, 그 때에 올바른 접근이 되는지 확인하는 코드이다. 여기까지 문제가 없는 경우 fault\_addr의 시작 주소에 접근하여 get\_page, set\_page를 호출하여 스택 영역을 확장한다.

* 1. **시험 및 평가 내용**
* (채점 대상 테스트 케이스에 해당하는) make check 수행 결과를 캡처하여 첨부  
  전체 16개의 테스트 케이스 중 page-merge-seq, page-merege-par, page-merge-stk, page-merge-mm을 제외한 12개의 테스트 케이스가 통과한다.
* 텍스트, 폰트, 스크린샷, 흑백이(가) 표시된 사진

  자동 생성된 설명