**Pintos Project 4: Virtual Memory**

담당 교수 : 김영재 교수님

이름 / 학번 : 박정원 / 20200183

개발 기간 : 2023-11-27 ~ 2023-12-03

1. **개발 목표**

* 해당 프로젝트에서 구현할 내용을 간략히 서술

핀토스 프로젝트는 현재 VM 구현이 되어 있지 않은데, 이를 구현하여 Page Fault Handling을 가능하게 하고, stack growth 및 disk swap 등 메모리를 효율적으로 관리하는 메커니즘 등을 구현해야 한다.

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

현재 Page Fault 가 일어나게 되면, 프로그램이 종료되도록 설계되어 있지만, 새로운 Page Fault Handler는 valid reference 인지 체크하여 상황에 따라 다른 조치를 취하도록 한다.

* 1. Disk Swap

가상 메모리에 더 이상 할당 가능한 page가 없다면, disk swap을 통해서 현재 메모리에 있는 page를 disk로 내보내고, 남은 공간에 새로운 페이지를 할당한다. Disk swap을 통해 내보낼 페이지를 선정하는 알고리즘은 LRU 페이지를 예측할 수 있는 Second Chance 알고리즘을 통하여 페이지를 evict 할 수 있도록 한다.

* 1. Stack Growth

만약 Page Fault가 일어났는데, 접근하려던 주소가 Stack 영역이라면, 페이지를 더 할당하여 Stack 영역을 증가시킨다.

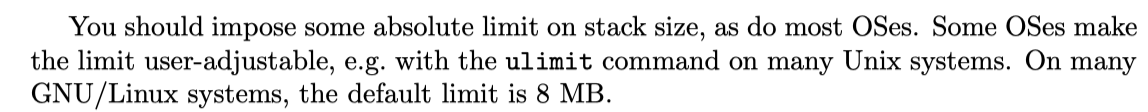
* 1. **개발 내용**
* 아래 항목의 내용만 서술
  1. Page fault가 발생하는 이유와 이를 handling하는 전반적인 과정을 서술

Page Fault 란 현재 메모리에 존재하지 않는 Page에 접근하려고 할 때 일어나는 Fault 입니다. 현재 접근하려고 하는 페이지가 disk에 저장되어 있을 수도 있는 경우, 현재 메모리에 있는 공간을 비우고, 요구된 페이지를 disk로부터 가져와 다시 메모리 접근을 시도하게 됩니다. 또한, 만약 접근하려는 페이지의 영역이 확장 가능한 영역, 즉 힙이나 스택이라면, 페이지를 추가적으로 할당하여 영역을 넓힌다. 핀토스에서는 힙 영역이 따로 존재하지 않기에 스택 영역만을 가정한다.

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

Disk swap 발생 시에 사용할 page replacement algorithm 은 second chance algorithm으로 불리우는 알고리즘이다. 이 알고리즘은 페이지를 둘러보면서, reference bit가 1인지 0인지 체크를 하는데, 이 값이 1이라면, 0으로 바꾼 후에 다음 페이지로 넘어간다. 만약 현재 가리키고 있는 페이지의 reference bit가 0 이면, 해당 페이지를 즉시 evict 하고, 해당 자리에 disk에 존재하던 페이지를 가져오고 reference bit를 1로 만들어준다. 현재 메모리의 페이지를 순회하고, 만약 모든 페이지들의 reference bit가 1인 경우에 다시 처음 시작한 페이지로 돌아올 수 있기 때문에 대게 circular queue를 이용하여 구현한다. Evict pointer 라는 것을 두어 페이지를 순회하며, 내쫓을 페이지를 찾는다.

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

****

스택의 default 크기가 8MB라고 pintos 매뉴얼에 나와있으므로, 페이지 폴트가 일어난 주소가 PHYS\_BASE – 8MB보다 크거나 같고, PHYS\_BASE보다는 작거나 같은 곳이면, 확장 가능한 Stack 영역이라는 것을 알 수 있다.

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

* II. A. 개발 범위를 포함하여 구현 내용에 대한 일정 작성

핀토스 매뉴얼 Virtual Memory 파트 정독 및 프로젝트 구상하기 11/27 ~ 11/29

Page Fault Handler 구현 11/30 ~ 12/1

Disk Swap 구현 12/2 (실패)

보고서 작성 12/3

* 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. **연구 결과**
   1. **Flow Chart**

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

스케치, 텍스트, 그림, 도표이(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

**텍스트, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명**

먼저, page fault handler를 구현하였다. 함수의 primitive는 pintos/src/userprog 의 exception.h에 선언해두었고, 구현은 exception.c에 하였다. 핀토스 매뉴얼과 PPT 슬라이드를 참고하여, 구현 사항에 맞추어 구현했다.

**텍스트, 스크린샷, 도표, 라인이(가) 표시된 사진

자동 생성된 설명**

먼저, 페이지 폴트가 발생하면, 우선적으로 확인해야 하는 것은 해당 참조가 valid reference 인지에 대한 것이다.

**텍스트, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명**

PPT를 보면, Access가 invalid한 두 가지 경우를 제시하고 있는데, 첫번째는 page가 unmapped인 경우, 즉 페이지가 참조되는 곳에 아무런 데이터가 존재하지 않는 경우와 두번째로 읽기 전용 페이지에 쓰기를 수행하려고 하는 접근일 때이다.

먼저, 첫번째 page가 unmapped되어 있는지 판단하기 위해서 pagedir\_get\_page 함수를 써서 fault\_addr가 아무런 데이터도 존재하지 않는 곳을 참조하고 있는지 확인한다.

만약 valid reference가 아니라면, 접근하려는 페이지가 확장 가능한 스택 영역인지 확인하는 절차가 필요하다.

텍스트, 스크린샷, 폰트, 문서이(가) 표시된 사진

자동 생성된 설명

핀토스 매뉴얼을 읽어보면, page\_fault의 인자로 넘어오는 intr\_frame의 esp 포인터의 주소값보다 32바이트 아래에서 페이지 폴트가 일어날 수 있다고 나와 있다. 따라서, 해당 페이지 폴트가 스택 영역에서의 페이지 폴트인지 판단할 때, fault\_addr >= f->esp – 32 체크를 해주어야 한다. 또한, 핀토스 역시 여타 OS과 같이 기본적인 스택 영역의 제한 크기는 8MB이다. 그렇기 때문에, PHYSBASE에서 8 \* 1024 \* 1024만큼 뺀 곳보다 fault\_addr가 크거나 같은지 PHYBASE보다 작거나 같은지 체크 해주어야 한다. 만약 스택 영역이라는 것이 확인된다면, 유저 스택 영역을 늘려주어야 한다. 이 때, 필요한 것이 pagedir\_set\_page 함수이다. 현재 page fault가 일어난 fault\_addr의 VPN의 주소를 pg\_round\_down 함수를 통해서 받아온다. 이 때, page offset는 0으로, VPN의 시작 주소를 받아온다고 생각하면 된다. 이제 현재 스레드의 페이지 테이블에 fault\_addr의 페이지를 등록시킬 것인데, pagedir\_set\_page 함수에 차례대로 현재 스레드의 pagedir, 아까 받아온 fault\_addr의 vpn 주소, 그리고 palloc\_get\_page(PAL\_USER), 1을 인자로 넘겨주게 되면, 페이지 폴트가 난 해당 fault\_addr에 대한 매핑이 생기게 된다.

텍스트, 스크린샷, 폰트, 라인이(가) 표시된 사진

자동 생성된 설명

palloc\_get\_page 함수에 PAL\_USER를 넘겨주게 되면, user pool의 빈 공간에 페이지를 할당하도록 도와준다. 이렇게 받아온 페이지를 fault\_addr가 요청한 페이지에 매핑시켜주도록 하는 것이 pagedir\_set\_page인 것이다.

이렇게 Stack growth가 끝나면, exit을 하는 것이 아닌 원래 실행 흐름대로 돌아가도록 return을 해주면 된다. 원래 같았으면 모든 페이지 폴트는 exit(-1)을 하면서 종료되었는데, 페이지 폴트 핸들링을 통해서 상황에 따라서만 exit(-1)을 하고 종료하도록 한다.

아까 주소 체크를 했을 때, 확장 가능한 스택 영역이 아닌 경우 exit(-1)을 통해 종료시키면 된다.

이렇게 page fault handling stack growth을 구현하였다.

Page fault handling의 handle\_mm\_fault 부분과 disk swap은 구현하지 못하여 보고서에 첨부하지 못했습니다.

* 1. **시험 및 평가 내용**
* (채점 대상 테스트 케이스에 해당하는) make check 수행 결과를 캡처하여 첨부

텍스트, 스크린샷, 폰트, 흑백이(가) 표시된 사진

자동 생성된 설명