**Pintos Project 4: Virtual Memory**

담당 교수 : 박성용 교수님

이름 / 학번 : 20181625 남현준

개발 기간 : 23.11.20 ~ 23.12.10

1. **개발 목표**

* 해당 프로젝트에서 구현할 내용을 간략히 서술

현재의 pintos에서는 page fault가 일어났을 시에 제대로 대응하지 못하는 상태이다. Pintos project #4에서는 page fault가 일어났을 때에 page fault handler를 통해 page가 memory에 load되어 있지 않은 문제를 해결하고 해당 instruction을 재실행하여 정상적으로 실행될 수 있도록 구현할 것이다.

이 과정에서 physical memory에 page를 할당하기에 충분한 공간이 존재하지 않는다면 victim을 선정하여 해당 page의 load를 해제하고 swap disk 상에 존재하는 구동에 필요한 page가 load될 수 있도록 구현될 것이다.

또한, pintos의 경우 target page가 stack 영역에서 존재하지 않아 page fault가 발생하는 경우 stack expansion을 통해 최대 8Mb 까지의 확장이 가능하고 이를 통해 stack 영역이 여러 개의 page에 걸치게 되는 경우, page의 할당이 가능한지 확인하게 될 것이다.

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

기존에는 page fault가 일어나는 경우 process를 종료하게 된다. Skeleton code로 주어진 pintos에서는 paging 등과 같은 memory 관리 방식이 적용되어 있지 않고 fault가 발생한 page를 찾을 logic이 구현되어 있지 않기 때문에 다음과 같은 결과가 발생하게 되는 것이다.

따라서, page fault handler를 구현하게 된다면 현재 physical memory에 load되어 있지 않은 page를 handler의 logic에 따라 찾은 다음 memory에 load하고 instruction을 재실행하여 fault를 적절하게 처리하는 방식으로 프로그램이 구동될 것이다.

Page table의 경우, 기존에 page table에 page fault 발생 시 fault를 처리할 수 있도록 추가적인 정보를 저장할 필요가 있다. 어떤 virtual page에서 fault가 발생했는지 확인하고 해당 page에 저장되어 있는 physical page 정보를 탐색 및 physical memory에 load하여 해당 정보를 관리하게 될 것이다.

* 1. Disk Swap

Page의 정보를 physical memory (=frame)에 load를 계속하다 보면 어느 순간 새로운 page를 할당할 공간이 부족해질 시기가 찾아오게 된다. 이 경우, 새 virtual page에 대응되는 정보를 physical memory에 load하기 위해서는 기존에 load되어 있는 frame 중 하나의 할당을 해제하여 공간을 만든 다음 load를 진행하게 된다.

이 과정에서 운영체제는 어떤 frame을 replace할 지 replacement policy를 통해 결정하게 된다. 이후, 해당 frame을 disk로 swap out하고 fault를 해결하기 위해 필요한 page를 특정 frame이 있었던 자리에 swap in을 통해 사용가능한 상태로 만들고 fault를 해결하게 된다. 이러한 방식을 통해 demand paging의 구현이 가능해지게 된다.

* 1. Stack Growth

Virtual memory의 구현이 되지 않은 상태의 pintos는 virtual memory 상에서 stack 영역의 일부분만 사용하고 있었다. 초기에 지정한 stack의 범위를 넘어가며 분할되는 page에 stack 영역의 정보가 저장되는 경우 page fault가 발생하게 되고 process의 실행이 중지되었다. Stack growth 부분을 개발하게 된다면 stack의 크기가 증가함에 따라 page를 추가적으로 할당하게 될 것이고 추가된 페이지에 대해서도 fault handling이 가능해질 것이다.

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

프로그램이 진행되며 virtual page를 읽었을 때, 해당 address에 대응되는 physical frame의 정보가 page table에 load되어 있지 않을 경우 page fault가 발생하게 된다. 이 경우, page fault handler를 통해 fault가 발생한 fault\_addr을 검사한다.

검사를 진행할 때 invalid reference, 즉 해당 page에 대한 정보를 찾을 수 없다면 stack의 grow를 통해 해결될 수 있는지 문제인지 검사한다. 해당 case라면 user stack의 크기를 확장하여 page의 정보를 table에 update 후 process의 실행을 재개한다.

Stack의 growth를 통해 해결되는 fault가 아니라면 page table에 단순히 현재 사용할 page의 정보가 load 되어 있지 않을 것일 수 있기 때문에 fault\_addr의 virtual page 정보를 탐색한다. 그 후, 해당 정보를 이용해서 disk에서의 위치를 특정하고 physical memory frame으로 load한 후, fault가 발생했던 부분부터 instruction을 restart한다.

이 문제들에 해당되지 않는 경우 virtual page가 존재하지 않는 것이므로 오류가 있다는 의미이고 process의 종료 후 사용하던 resource를 반환한다.

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

Page fault 발생 시 사용할 page의 정보를 table에 load할 수 없는, 즉 page table이나 physical memory에 공간이 모자란 경우 victim을 선정해서 evict한 다음 OS는 해당 위치에 정보를 load하게 된다. 이 때, 주로 사용되는 알고리즘은 Least Recently Used (LRU) 알고리즘으로 frame 중 eviction 알고리즘이 실행되는 시점에서 가장 적게 사용된 frame 중 하나를 골라 evict하는 방식이다.

하지만 해당 구현의 경우 overhead가 상당하고 각 frame에 추가적인 data를 통해 몇 번 사용되었는지 기록할 필요가 있기 때문에 번거롭다. 그렇기에 간단한 second chance 알고리즘의 사용을 고려하였다.

evict하게 될 frame을 나타내는 victim pointer가 처음에는 첫 번째로 load된 frame을 가리키게 될 것이다. 이 때 page의 정보는 reference bit가 1로 설정되어 있을 것이다. 이후, page fault가 생겨 replace를 진행해야 하는 경우 load된 page의 정보를 저장하는 자료구조 (통상적으로 circular queue)를 순회하며 reference bit가 0인 page의 정보를 탐색한다. 순회하며 reference bit가 1인 경우, bit를 0으로 set하고 두 번째 기회를 주게 된다. 모든 페이지의 reference bit가 1이었던 경우 자료구조에 저장되어 있는 첫 번째 원소가 victim으로 선정되게 된다.

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

기존 project #3까지의 code에서는 fault가 발생하게 되면 곧바로 process를 종료시켰다. Project #4에서는 fault\_addr에 대해 기존에 설정된 4 Mb의 크기를 넘어서는 주소에 접근하게 되는 경우 grow limit를 확장하여 8 Mb까지의 범위에서 stack의 시작 주소에서 8Mb를 뺀 주소까지의 범위에서 stack이 확장되었을 때 유효한지 검사하는 것으로 확정 여부를 판단할 수 있다. 해당 범위에 포함되지 않는다면 stack의 확장은 불가능하거나 비유효한 접근으로 판단되어 segmentation fault를 발생시키게 될 것이다. 이 때, fault가 일어난 주소가 PHYS\_BASE보다 크거나 growth를 통해 확장될 수 있는 최대 크기인 8Mb를 넘는 위치에 있거나 PHYS\_BASE에서 32 뺀 값이 fault\_addr보다 크다면 비유효한 접근이며 exit(-1)을 통해 종료될 수 있도록 한다.

1. **추진 일정 및 개발 방법**
   1. **추진 일정**

* II. A. 개발 범위를 포함하여 구현 내용에 대한 일정 작성

23.11.20 ~ 23.12.05 virtual memory, frame, swap disk 구현 시도

23.11.20 ~ 23.12.09 stack growth 구현

23.12.05 ~ 23.12.10 보고서 작성

* 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

Project #3까지의 구현이 완료된 pintos를 기준으로 vm 디렉토리에 있는 Makefile을 실행 및 functionality check를 진행하게 되면 오류가 발생한다. 해당 오류는 scheduling를 구현하며 사용한 thread\_prior\_aging에 관련된 구현과 sema\_up() 함수에서의 thread\_yield()에서 발생하는 것인데 이를 주석 처리/삭제하는 것으로 에러를 해결할 수 있다.

vm 폴더에 필요한 파일들을 추가적으로 생성하고 컴파일을 진행하는 경우, 새로 생성한 파일들에서 정의한 함수가 선언되지 않았다는 에러 메시지를 출력하게 된다. 이 문제를 해결하려면 생성한 파일들을 Makefile.build의 dependency에 추가해주어야 한다.

기존 exception.c에서는 fault\_addr에 대해 page가 존재하지 않거나 user 영역의 주소가 아닌 경우 곧바로 process를 종료시켰다. Project #4에서는 fault\_addr에 대해 page가 존재하지 않는 case였을 경우, fault를 handle하여 page 정보를 추가하고, stack 영역의 확장이 필요한 경우 확장을 진행하는 코드를 추가한다.

1. **연구 결과**
   1. **Flow Chart**

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

텍스트, 도표, 평면도, 기술 도면이(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
  + - Device/timer.c

기존에 aging과 관련되어 있던 thread\_prior\_aging과 관련된 부분들이 에러를 유발하므로 해당 부분을 삭제했다.

텍스트, 스크린샷, 소프트웨어, 디스플레이이(가) 표시된 사진

자동 생성된 설명

* + - Threads/synch.c

Sema\_up 함수에 있는 thread\_yield() 부분이 에러를 유발하게 되므로 해당 부분을 주석처리 했다.

텍스트, 스크린샷, 폰트, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

* + - Makefile.build

해당 파일에 새로 생성한 파일들의 dependency를 추가했다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

* + - Userprog/exception.c

기존의 코드에서는 not present가 true일 때 exit(-1)을 실행하여 process를 종료시키는 방식이었다. Project #4에서는 종료시키는 대신 fault\_addr의 위치를 체크하는 로직을 추가했다. 이 과정에서 stack의 크기를 8Mb까지 늘린 범위에서 fault\_addr이 유효한 접근인지 확인을 하게 되는데 유효하지 않을 경우 이전과 마찬가지로 exit(-1)을 호출한다. 유효한 접근일 경우 stack growth를 통해 stack을 확장하고 새로운 page의 매핑을 추가한다.

텍스트, 스크린샷, 디스플레이, 폰트이(가) 표시된 사진

자동 생성된 설명

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

텍스트, 스크린샷, 폰트, 타이포그래피이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

Pt-grow-stk-sc의 경우 출력의 밀림으로 인해 test가 fail하였습니다.

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

Page parallel의 경우 출력의 밀림으로 인해 test가 fail하였습니다.