고급운영체제 Progress Report

2019-20797 최윤서

2020-26170 이정은

2020-27346 황인휘

**1. Proposal 보완**

*1)PTE의 valid bit 등을 조절하여 강제로 page fault가 나게 하는 방법 대비 TLB를 이용하는 방법의 이점이 무엇인지 궁금합니다.*

🡺 TLB 이용하는 방법을 쓰지 않는 것으로 결정 - Proposal에서는 그런 방법도 있을 수 있다는 생각에 제안했습니다. 그러나 성능과 구현 난이도의 측면에서 봤을 때 이점이 없다고 판단되어 이 방법은 쓰지 않기로 했습니다. Valid bit을 조절하는 방법으로 페이지 폴트를 발생시키는 방법을 사용하기로 결정했습니다.

*2)어떤 분석을 위해 page 단위 메모리 트레이싱이 필요한지 구체적인 응용 예와 활용 방안 등을 보완바랍니다.*

🡺 현대 computing workload는 거대한 working set과 low locality의 특성을 가지고 있다. 그리고 메모리의 성장률은 CPU와 storage 같은 다른 디바이스의 성장률보다 뒤떨어져서 메모리에서 병목현상이 발생한다. 이를 극복하기 위해서 메모리에 데이터 할당과 자주 사용되는 페이지를 찾는 것이 중요하고 각 페이지들의 메모리 접근 양상을 파악하는 것이 필요하다. 본 연구를 통해 자주 사용되는 페이지와 자주 사용되지 않는 페이지를 구분하고 이 데이터를 기반으로 메모리 관리를 한다면 메모리의 효율을 높일 수 있다.

**2. 진행상황**

현재 진행상황은 다음과 같습니다.

1) System call을 사용하여 커맨드 이름을 가지고 특정 커맨드(e.g. ls)의 트레이싱을 enable 시킴

시스템 콜로 커맨드 이름을 커널에 넘기면 커널에서는 그 값을 전역 변수로 저장. exec 될 때 지금 시작하려는 커맨드 이름과 저장된 커맨드를 비교하여 같을 경우 enable. enable 변수는 mm\_struct 구조체에 정의되어 있음.

2) 페이지 폴트 상황 : enable 된 커맨드가 실행되는 동안 page fault가 일어나면 트레이싱. 페이지 주소, R/W, Instruction/Data 등의 정보를 printk()로 출력.

handle\_pte\_fault에서 트레이싱. 필요한 정보는 vm\_fault 구조체에서 획득. 주소 정보는 vm\_fault의address 변수를, R/W 또는 Instruction/Data 정보는 vm\_fault의 flags 변수에서 찾을 수 있음.

3) 페이지 폴트가 일어나지 않는 메모리 접근을 트레이싱하는 방법 구현

일반적인 page fault가 일어날 때 그 pte를 mm\_struct에 변수를 두어 기록하고 다음 번 page fault 시에 기록해둔 pte를 invalidate하는 식으로 구현함. 프로세스의 접근 후에 처음으로 하는 메모리의 접근은 항상 페이지 폴트가 나고 현재 처리하고 있는 PTE를 제외한 이전의 모든 PTE가 invalidate 되기 때문에 모든 페이지 접근에 대하여 페이지 폴트가 발생함.

* 이전 페이지 폴트에서 처리한 PTE를 invalidate하는 이유 : page fault 처리가 끝나면 page fault를 유발한 유저 코드가 다시 실행됨. 일반적인 경우 이미 page fault가 처리되었기 때문에 이 코드에 예외가 발생하지 않고 잘 동작함. Page fault 처리가 끝난 이후에는 이미 유저 context로 돌아가기 때문에 커널 영역에서 PTE를 invalidate 시킬 수 없음. 그러나 page fault 시에 바로 그 pte를 invalidate할 경우, page fault 처리가 끝나서 다시 유저의 코드로 돌아갈 때 다시 page fault가 발생. page fault가 끝나지 않고 무한 루프를 돌게 됨. 따라서 다음 번 Page fault 시에 그 전 page fault에서 처리되었던 PTE를 invalidate 시키는 방식으로 해결.

4) PTE의 Flag를 바꾸어 PTE를 invalidate하는 방법 구현.

PTE invalidate는 단순히 PTE의 flag 중 하나인 present bit을 0으로 바꾸거나 프로그래머가 이용가능한 비트인 9번째 비트(이하 fake bit)를 바꾸어도 페이지 폴트 발생함. 지금은 fake bit을 1로 바꾸는 방식으로 invalidate함. 페이지 폴트 발생 시에는 fake 비트를 보고 1일 경우에는 이것을 fake 페이지 폴트로 보고, fake 비트를 0으로 바꾸고 바로 리턴한다. 이렇게 할 경우 별다른 처리 없이 그 페이지에 대한 페이지 폴트 처리가 완료됨. 그러나 이 방법을 사용하면 exit할 때 에러 발생. 커널의 다른 자료구조에서 기록한 PTE의 값과 실제 PTE 값이 다르기 때문에 발생하는 것으로 추측됨. 이를 해결하기 위해 exit 함수에서 fake bit이 1인 pte들을 unset하는 루틴이 필요할 것을 보임.

**3. To do**

* 4)번에서 서술한 exit시에 발생하는 에러를 처리해야함. exit시에 해당 프로세스에 관련된 모든 PTE의 fake 값을 0으로 바꾸면 될 것으로 생각되나 오버헤드가 클 것으로 예상되어 더 좋은 방법을 구상 중.
* 현재 방법으로 구현했을 경우에 다른 문제가 발생하지는 않는지 확인해야함. 예를 들어, fake bit set한 PTE를 swap하게 되는 경우 등.
* (1) 실제로 모든 페이지에 대해서 페이지 폴트가 발생하고 그것을 트레이싱 할 수 있는지, (2) 이 방법을 썼을 때 바닐라 프로그램 대비 어느 정도의 오버헤드가 있는지, (3) 다른 메모리 트레이싱 툴과의 성능 비교 등의 실험을 수행.

추가적으로, 4k page만이 아니라, Huge page 등에도 적용(시간이 되는 경우)