Report : Operating System

Designing a Virtual Memory Manager

서형중 - 2019년 6월 20일

1. 목적

Operating System 에서 Memory Management 시에 사용되는 page replacement Algorithm들 중 LRU 와 LRU Approximation을 직접 구현함으로써 알고리즘에 대한 이해를 높이고 어떤 알고리즘이 더 효율적인지 비교해 봄.

1. 설계 방식

OS : macOS Mojave

Using built-in specs.

COLLECT\_GCC=g++

Target: x86\_64-apple-darwin18.5.0

Configured with: ../gcc-8.3.0/configure --enable-checking=release --enable-languages=c,c++,fortran

Thread model: posix

gcc version 8.3.0 (GCC)

Frame table 과 TLB 는 각 각 pair<int, int> 와 tuple<int, int, int> 혹은 pair<int, int> 와 tuple<int, int, unsigned int> 의 배열로 구현하였으며, page number 의 범위가 0 부터 255 까지 이고 frame number 도 0 부터 255 까지 이므로 frame table이 overflow 되지 않기 때문에 frame table 에서 찾고자 하는 page 가 할당이 되지 않았으면 비어있는 frame(무조건 존재함) 을 찾아서 할당해 주는 방식으로 함.

Address 가 10진수 input으로 들어오면 해당 input 을 int형 배열을 이용하여 binary 로 바꾼 다음

최상위 8자리는 page number, 최하위 8자리는 offset 으로 설정해 준 다음 Address Translator 에 넣어서 physical address로 변환해줌.

Address Translator 를 LRU 방식과 LRUA 방식으로 두 가지를 구현하여 입력이 들어올 때 동시에 LRU 와 LRUA 가 동시에 수행되도록 설계함.

LRU 를 이용한 주소변환은 counter 방식으로 설계 하였으며, TLB HIT 가 발생하면 해당 page 의 counter 를 0 으로 초기화 해주고, TLB miss 가 되면 replacement 를 해준 다음 새로 할당되는 page 의 counter 를 0으로 초기화 해주고 이외 page 의 counter 를 하나 증가 시킴.

LRUA 를 이용한 주소변환은 Additional reference bit 방식으로 설계하였으며, additional bit를 32bit로 저장하기 위해 자료형을 unsigned int 로 함.

1. 구현 상세

10진수에서 2진수(16 bit)로 바꿔주는 함수는 10진수 주소가 들어오면 나머지가 0이 될 때 까지 2에 대한 나머지를 vector 에 저장하는 방식으로 구현함. -> 처음 10진수 주소가 주어졌을 때 이진수로 바꿔줄 때 사용함.

2진수에서 10진수로 바꿔주는 함수는 page number 와 offset 을 int 형으로 data를 넘겨주고 저장하기 위해 사용함.

LRUA 를 구현할 때 additional bit를 10진수로 저장하기 위해 자료형을 unsigned int 로 설정하였으며, TLB 의 크기가 32이기 때문에 additional bit의 size 역시 32로 구현함.

*Address translation LRU pseudocode*

*logical address -> physical address*

*logical address <page#, offset>*

*1. TLB table* 에찾고자하는 *page#* 가있는지확인

*1.1 EXIST ->* 해당 *page* 의 *counter 0* 으로초기화

*1.1.1 return frame# \* 256 + offset*

*1.2 NOT EXIST -> frame table* 에서해당 *page* 탐색

*1.2.1 EXIST*

*1.2.1.1 return pair<frame#, page#>*

*1.2.2 NOT EXIST*

*1.2.2.1 empty frame* 에해당 *page* 할당

*1.2.2.2 return pair<frame#, page#>*

*1.2.3 TLB IS FULL?*

*1.2.3.1 TRUE*

*1.2.3.1.1 TLB* 에서 *counter* 가가장큰 *tuple* 을선택후추방

*1.2.3.2* 빈공간에 *frame table* 에서반환된 *pair* 할당및해당 *tuple* 의 *counter 0* 으로세팅

*1.2.3.3* 나머지 *tuple* 의 *counter* 를하나올려줌

*2. return frame\_number \* 256 + offset*

*Address translation LRUA pseudocode*

*logical address -> physical address*

*logical address <page#, offset>*

*1. TLB table* 에찾고자하는 *page#* 가있는지확인

*1.1 page# == TLB tuple* 의 *page -> reference bit*를 *shift* 해주고최상위 *bit*를 *1*로초기화

*1.2 page# != TLB tuple* 의 *page -> reference bit*를 *shift* 해주고최상위 *bit*를 *0*로초기화

*2. EXIST -> return frame# \* 256 + offset*

*3. NOT EXIST -> frame table* 에서해당 *page* 탐색

*3.1 EXIST*

*3.1.1 return pair<frame#, page#>*

*3.2 NOT EXIST*

*3.2.1 empty frame* 에해당 *page* 할당

*3.2.2 return pair<frame#, page#>*

*4. TLB IS FULL?*

*4.1 TRUE*

*4.1.1 TLB* 에서 *counter* 가가장큰 *tuple* 을선택후추방

*5.* 빈공간에 *frame table* 에서반환된 *pair* 할당및해당 *tuple* 의 *reference bit*의최상위 *bit*를 *1*로세팅

*6. return frame\_number \* 256 + offset*

1. 결론

LRU 와 LRUA 를 구현하여 수행해 본 결과 LRU 를 이용하였을 때는 TLB HIT 가 1000개의 input 에 대해 121회 발생하였고, LRUA 를 이용하였을 때는 TLB HIT 가 117회 발생하였다.

LRU 와 LRUA 의 성능을 비교해보았을 때 LRUA 에 비해 LRU 가 성능이 좋게 나왔다.