설계과제 4

컴퓨터학부 20180740 김희엽

개요

페이지 교체 알고리즘을 알아보고 간단한 방식으로 구현해보고 페이지 폴트 비율을 비교해보는 프로그램. 참조 스트림을 500 ~ 10500개 생성하도록 제한하였습니다. 프로그램의 이름은 ui로 설정 하였습니다. ./u로 실행할 수 있습니다.

OPTIMAL algorithm

알고리즘의 동작방식

가장 오랫동안 사용하지 않을 페이지를 선택하여 교체하는 알고리즘입니다. 페이지 테이블 내의 각각의 페이지 숫자가 프로세스에서 향후에 참조할 페이지 숫자까지의 거리를 계산하여 가장 먼 미래에 사용될 페이지를 선택하여 페이지 테이블 내에서 교체하는 방식입니다.

[Operating System Concepts 10th edition, [Abraham Silberschatz](http://www.yes24.com/Product/Search?domain=ALL&query=Abraham%20Silberschatz&authorNo=299985&author=Abraham%20Silberschatz), [Peter Baer Galvin](http://www.yes24.com/Product/Search?domain=ALL&query=Peter%20Baer%20Galvin&authorNo=299986&author=Peter%20Baer%20Galvin), [Greg Gagne](http://www.yes24.com/Product/Search?domain=ALL&query=Greg%20Gagne&authorNo=299987&author=Greg%20Gagne) ] p.406, 407

Pseudo code

OPTIMAL(page\_table, streams)

{

if PAGE\_FAULT

{

victim = get\_max\_diff\_page\_table\_entry(page\_table, streams);

first\_page\_table\_entry = victim->next;

get\_last\_page\_table\_entry(page\_table)->next = victim;

return first\_page\_table\_entry;

}

}

get\_next\_stream\_index(streams, stream)

{

loop:

if cur\_stream ==stream

{

return stream->index;

}

next\_stream;

if next\_stream != NULL

{

goto loop;

}

return NULL;

}

get\_diff(streams, index)

{

next\_index = get\_next\_stream\_index(streams, stream);

return next\_index – index;

}

get\_max\_diff\_page\_table\_entry(page\_table, streams)

{

max\_diff\_page\_table\_entry = first\_page\_table;

loop:

if get\_diff(streams, index, max\_diff\_page\_table\_entry) < get\_diff(streams, index, cur\_page\_table\_entry)

{

max\_diff\_page\_table = cur\_page\_table\_entry;

}

next\_page\_table\_entry;

if next\_page\_table\_entry != NULL

goto loop;

}

get\_last\_page\_table\_entry(page\_table)

{

loop:

if page\_table\_entry -> next != NULL

{

next\_page\_table

goto loop;

}

if page\_table->next == NULL

{

return page\_table;

}

}

FIFO algorithm

알고리즘의 동작방식

FIFO 교체 알고리즘은 오래된 페이지를 선택하여 교체하는 알고리즘입니다. 오래된 페이지는 연결리스트의 앞쪽에 위치하게 하고 새로운 페이지는 마지막에 위치하도록 구현하였습니다. 모든 페이지를 메모리에 보관하기 위해 FIFO 대기열을 만들 수 있습니다. 대기열의 맨 위 페이지를 교체합니다. 페이지를 메모리로 가져오면 대기열 끝에 삽입합니다.   
제 프로그램에서 맨 앞 페이지를 교체하여 맨 뒤로 보내는 방식을 채택하여 구현하였습니다.

[Operating System Concepts 10th edition, [Abraham Silberschatz](http://www.yes24.com/Product/Search?domain=ALL&query=Abraham%20Silberschatz&authorNo=299985&author=Abraham%20Silberschatz), [Peter Baer Galvin](http://www.yes24.com/Product/Search?domain=ALL&query=Peter%20Baer%20Galvin&authorNo=299986&author=Peter%20Baer%20Galvin), [Greg Gagne](http://www.yes24.com/Product/Search?domain=ALL&query=Greg%20Gagne&authorNo=299987&author=Greg%20Gagne) ] p.404, 405

Pseudo code

FIFO(page\_table, streams)

{

if PAGE\_FAULT

{

victim = first\_page\_table\_entry;

first\_page\_table\_entry = first\_page\_table\_entry->next;

page\_replacement(victim, stream);

get\_last\_page\_table\_entry(page\_table)->next = victim;

return first\_page\_table\_entry;

}

}

get\_last\_page\_table\_entry(page\_table)

{

loop:

if page\_table\_entry -> next != NULL

{

next\_page\_table

goto loop;

}

if page\_table->next == NULL

{

return page\_table;

}

}

LIFO algorithm

알고리즘의 동작방식

스택은 항목을 추가하고 제거하기 위해 LIFO(후입선출) 원칙을 사용하는 순차적으로 정렬된 데이터 구조입니다. 즉, 스택에 배치된 마지막 항목이 제거된 첫 번째 항목입니다. 스택에서 항목을 삽입하고 제거하는 작업을 각각 푸시 및 팝이라고 합니다. 운영 체제는 종종 함수 호출을 호출할 때 스택을 사용합니다. 매개 변수, 지역 변수 및 반환 주소는 함수가 호출될 때 스택에 푸시됩니다. 함수 호출에서 반환하면 해당 항목이 스택에서 꺼집니다.

최근에 교체한 페이지를 리스트의 마지막 부분에 위치시켜 해당 페이지만 교체되도록 구현하였습니다.

[Operating System Concepts 10th edition, [Abraham Silberschatz](http://www.yes24.com/Product/Search?domain=ALL&query=Abraham%20Silberschatz&authorNo=299985&author=Abraham%20Silberschatz), [Peter Baer Galvin](http://www.yes24.com/Product/Search?domain=ALL&query=Peter%20Baer%20Galvin&authorNo=299986&author=Peter%20Baer%20Galvin), [Greg Gagne](http://www.yes24.com/Product/Search?domain=ALL&query=Greg%20Gagne&authorNo=299987&author=Greg%20Gagne)]p.37, 38

Pseudo code

LIFO(page\_table, streams)

{

if PAGE\_FAULT

{

victim = get\_last\_page\_table\_entry(page\_table);

page\_replacement(victim, stream);

}

return first\_page\_table\_entry;

}

get\_last\_page\_table\_entry(page\_table)

{

loop:

if page\_table\_entry -> next != NULL

{

next\_page\_table

goto loop;

}

if page\_table->next == NULL

{

return page\_table;

}

}

LRU algorithm

알고리즘의 동작방식

페이지를 교체해야 할 때 LRU는 가장 오랫동안 사용하지 않은 페이지를 선택합니다. 가장 오랫동안 사용되지 않은 페이지는 맨 뒤로 보내어 오래된 페이지가 리스트의 앞으로 오도록 하여 교체할 것을 선택할 때 오랫동안 교체되지 않은 페이지를 선택할 수 있도록 하였습니다. 해당 알고리즘은 좋은 알고리즘이라고 하는데 구현하는데에 어려움이 있습니다. 해당 구현을 위해선 두가지 방법을 제시합니다.

카운터를 두어 해당 페이지들에 대해 언제 참조했는지 기록할 수 있는 추가적인 공간을 사용합니다. 페이지를 교체해야할 때 이 값을 토대로 가장 오랫동안 사용되지 않은 페이지를 선택할 수 있습니다.

스택을 이용합니다. 오래된 페이지는 스택에서 위쪽으로 올라가게 만들어 시간 순서를 유지할 수 있도록 할 수 있습니다.

제 프로그램에선 스택을 이용하여 오래된 페이지를 앞으로 오도록 하고 교체되면 뒤로 보내 계속해서 페이지 테이블의 모양이 변합니다.

[Operating System Concepts 10th edition, [Abraham Silberschatz](http://www.yes24.com/Product/Search?domain=ALL&query=Abraham%20Silberschatz&authorNo=299985&author=Abraham%20Silberschatz), [Peter Baer Galvin](http://www.yes24.com/Product/Search?domain=ALL&query=Peter%20Baer%20Galvin&authorNo=299986&author=Peter%20Baer%20Galvin), [Greg Gagne](http://www.yes24.com/Product/Search?domain=ALL&query=Greg%20Gagne&authorNo=299987&author=Greg%20Gagne) ] p.407, 408

Pseudo code

LRU(page\_table, streams)

{

if PAGE\_FAULT

{

victim = get\_last\_page\_table\_entry(page\_table);

page\_replace(victim, stream);

victim->next = first\_page\_table\_entry;

}

if NOT PAGE\_FAULT

{

victim = get\_referred\_page\_table\_entry(page\_table, stream);

victim->next = first\_page\_table\_entry;

}

get\_last\_page\_table\_entry(page\_table)

{

loop:

if page\_table\_entry -> next != NULL

{

next\_page\_table

goto loop;

}

if page\_table->next == NULL

{

return page\_table;

}

}

get\_referred\_page\_table\_entry(page\_table, stream)

{

loop:

if page\_table\_entry == stream

{

return page\_table\_entry;

}

next\_page\_table\_entry

if next\_page\_table != NULL

{

goto loop;

}

return NULL;

}

LFU algorithm

알고리즘의 동작방식

해당 알고리즘은 스트림으로부터 페이지 테이블에 참조한 횟수를 카운터를 두어 기록합니다. 많이 참조하면 참조할수록 카운터의 값이 커지고 이는 교체될 확률이 낮아집니다. LFU페이지 교체 알고리즘은 카운트가 가장 작은 페이지를 교체합니다. 그러나 프로세스의 초기 단계에서 페이지가 많이 사용되었다가 다시는 사용되지 않는 경우 문제가 발생합니다. 많이 사용했기 때문에 카운트가 많고 더 이상 필요하지 않아도 메모리에 남아 있습니다. 한 가지 해결책은 일정한 간격으로 카운트를 오른쪽으로 1비트 이동하여 기하급수적으로 감소하는 평균 사용 카운트를 형성하는 것입니다.

제 프로그램에선 일정 주기동안 카운터를 쉬프트 연산을 수행하지 않고 계속해서 중첩시키기만 하였습니다.

[Operating System Concepts 10th edition, [Abraham Silberschatz](http://www.yes24.com/Product/Search?domain=ALL&query=Abraham%20Silberschatz&authorNo=299985&author=Abraham%20Silberschatz), [Peter Baer Galvin](http://www.yes24.com/Product/Search?domain=ALL&query=Peter%20Baer%20Galvin&authorNo=299986&author=Peter%20Baer%20Galvin), [Greg Gagne](http://www.yes24.com/Product/Search?domain=ALL&query=Greg%20Gagne&authorNo=299987&author=Greg%20Gagne) ] p.411, 412

Pseudo code

LFU(page\_table, streams)

{

if PAGE\_FAULT

{

victim = get\_least\_used\_page\_table\_entry(page\_table);

page\_replacement(victim, stream);

get\_last\_page\_table\_entry(page\_table)->next = victim;

return new\_first\_page\_table\_entry;

}

if NOT PAGE\_FAULT

{

get\_page\_table\_entry(page\_table, stream)->count++;

return first\_page\_table\_entry;

}

get\_least\_used\_page\_table\_entry(page\_table)

{

victim = first\_page\_table\_entry;

loop:

if victim->count > cur\_page\_table\_entry

{

victim = cur\_page\_table\_entry;

}

next\_page\_table\_entry

if next\_page\_entry != NULL

goto loop

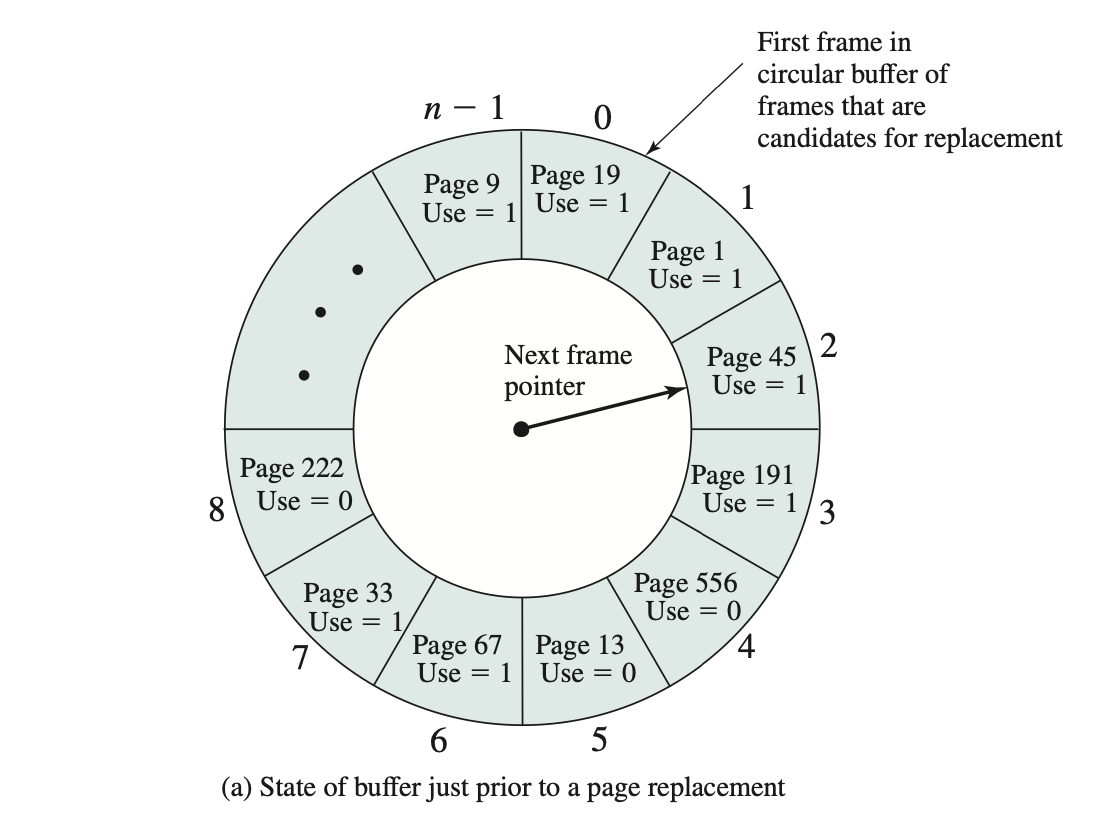
return victim

}

SC(Second Chance/ Simple Clock) algorithm

알고리즘의 동작방식

Second-chance replacement의 기본 알고리즘은 FIFO replacement algorithm이다. 포인터가 가리키는 페이지의 참조비트 값이 0이면 이 페이지를 교체합니다. 그러나 참조 비트가 1로 설정되면 페이지에 참조비트를 0으로 바꾸고 다음 페이지테이블엔트리로 넘어가 확인합니다. 따라서 두 번째 기회가 주어진 페이지는 다른 모든 페이지가 교체될 때까지(또는 두 번째 기회가 주어질 때까지) 교체되지 않습니다. 또한 페이지가 참조 비트가 1인 상태를 유지할 정도로 자주 사용되는 페이지는 절대 교체되지 않습니다. SC알고리즘(시계 알고리즘이라고도 함)을 구현하는 한 가지 방법은 순환 대기열입니다. 포인터(즉, 시계 바늘)는 다음에 교체할 페이지를 나타냅니다. 프레임이 필요할 때 포인터는 참조 비트가 0인 페이지를 찾을 때까지 진행합니다. 진행하면서 참조 비트를 0으로 변환합니다. 참조비트가 0인페이지가 발견되면 해당 페이지가 교체되고 새 페이지가 순환 대기열의 해당 위치에 삽입됩니다. 최악의 경우 모든 비트가 1이라면 포인터가 전체 대기열을 순환하기때문에 FIFO알고리즘과 같은 효과를 갖게 됩니다.

텍스트, 시계이(가) 표시된 사진

자동 생성된 설명

그림을 통해 SC알고리즘 작동 과정을 알 수 있습니다. 페이지 교체를 위해 n 메인 메모리 프레임의 순환 버퍼를 사용할 수 있습니다. 페이지 727로 버퍼의 페이지를 교체하기 직전에 다음 프레임 포인터는 페이지 45를 포함하는 프레임 2를 가리킵니다. 프레임 2의 페이지 45에 대한 사용 비트가 1이기 때문에 이 페이지는 교체되지 않습니다. 대신 사용 비트가 0으로 설정되고 포인터가 전진합니다. 마찬가지로 프레임 3의 191페이지는 교체되지 않습니다. 사용 비트는 0으로 설정되고 포인터가 진행됩니다. 다음 프레임인 프레임 4에서 사용 비트는 0으로 설정되어있습니다. 따라서 페이지 556은 페이지 727로 대체됩니다. 이 프레임에 대해 사용 비트는 1로 설정되고 포인터는 프레임 5로 이동하여 페이지 교체 절차를 완료합니다.

[[Operating Systems: Internals and Design Principles (8th Edition), stallings] p.394, p.395](https://www.amazon.com/Operating-Systems-Internals-Design-Principles/dp/0133805913)

Pseudo code

SC(page\_table, streams, clock) {

if PAGE\_FAULT

{

traverse\_page\_table:

if page\_table\_entry->reference\_bit == 0

{

page\_replacement();

return victim->next;

}

else

{

page\_table\_entry->reference\_bit = 0;

next\_page\_table\_entry

goto traverse\_page\_table;

}

}

if NOT PAGE\_TABLE

{

page\_table\_entry->reference\_bit = 1;

return clock;

}

}

ESC(Enhanced Second Chance/ Two handed clock/ Not Recently Used) algorithm

알고리즘의 동작방식

요약하면 페이지 교체 알고리즘은 모든 페이지를 순환합니다. 가져온 이후 수정되지 않았으며 최근에 액세스되지 않은 버퍼를 찾습니다. 이러한 페이지는 교체하기에 좋은 방법이며 수정되지 않았기 때문에 보조 메모리에 다시 쓸 필요가 없다는 이점이 있습니다. 첫 번째 스윕에서 후보 페이지가 발견되지 않으면 알고리즘은 버퍼를 다시 순환하여 최근에 액세스하지 않은 수정된 페이지를 찾습니다. 이러한 페이지는 교체를 위해 작성해야 하지만 지역성의 원칙 때문에 조만간 다시 필요하지 않을 수 있습니다. 이 두 번째 패스가 실패하면 버퍼의 모든 프레임이 최근에 액세스되지 않은 것으로 표시되고 세 번째 스윕이 수행됩니다.

1. 최근에 액세스하지 않았으며 수정하지 않음(u = 0; m = 0)

2. 수정되지 않은 최근 액세스(u = 1; m = 0)

3. 최근 액세스되지 않음, 수정됨(u = 0; m = 1)

4. 최근에 액세스, 수정됨(u = 1; m = 1)

1. 포인터의 현재 위치에서 시작하여 프레임 버퍼를 스캔합니다. 이 스캔 중에는 사용 비트를 변경하지 마십시오. (u = 0; m = 0)과 만나는 첫 번째 프레임이 교체를 위해 선택됩니다.

2. 1단계가 실패하면 다시 스캔하여 (u = 0; m = 1)인 프레임을 찾습니다. 발견된 첫 번째 프레임이 교체를 위해 선택됩니다. 이 스캔 중에 바이패스되는 각 프레임에서 사용 비트를 0으로 설정합니다.

3. 2단계가 실패하면 포인터는 원래 위치로 돌아가야 하며 세트의 모든 프레임은 사용 비트가 0이 됩니다. 1단계를 반복하고 필요한 경우 2단계를 반복합니다. 이번에는 프레임이 교체를 위해 선택될 것이다.

저의 경우 페이지가 교체되면 참조비트는 무조건 1, 수정 비트는 입력받거나 랜덤으로 생성된 수로 지정하는 방식을 채택하여 구현하였습니다. 또한 참조비트는 페이지의 참조하는 행위로 0으로 바꾸거나 1로 바꿀 수 있기 때문에 수정 비트만 무작위적으로 생성하거나 입력 받았습니다.

[Operating Systems: Internals and Design Principles (8th Edition), stallings] p.395 ~ 398

Pseudo code

ESC(page\_table, streams) {

if PAGE\_FAULT

loop:

find\_(0, 0)

if EXIST (0, 0)

page\_replacement();

return victim->next;

else

find\_(0, 1)

if EXIST (0, 1)

page\_replacement();

return victim->next;

else

reference\_bit = 0;

GOTO loop

else

page\_table\_entry->reference\_bit = 1;

if modify\_bit == 1

page\_table\_entry->modify\_bit = 1;

}

성능분석 비교

페이지 개수 5 스트림 개수 500

텍스트이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 1회 | 2회 | 3회 | 4회 | 5회 | 평균 |
| OPTIMAL | 58.20 | 60.60 | 60.20 | 61.20 | 61.20 | 60.28 |
| FIFO | 83.40 | 84.00 | 84.80 | 87.40 | 87.40 | 85.4 |
| LIFO | 86.00 | 82.80 | 83.80 | 84.60 | 84.60 | 84.36 |
| LRU | 82.40 | 83.00 | 84.80 | 87.40 | 87.40 | 85 |
| LFU | 83.40 | 85.60 | 85.20 | 86.00 | 86.00 | 85.24 |
| SC | 83.40 | 83.80 | 85.00 | 87.60 | 87.60 | 85.48 |
| ESC | 82.40 | 83.80 | 85.40 | 85.40 | 85.40 | 84.48 |

페이지 개수 7 스트림 개수 500

텍스트이(가) 표시된 사진

자동 생성된 설명 텍스트이(가) 표시된 사진

자동 생성된 설명 텍스트이(가) 표시된 사진

자동 생성된 설명 텍스트이(가) 표시된 사진

자동 생성된 설명 텍스트이(가) 표시된 사진

자동 생성된 설명

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 1회 | 2회 | 3회 | 4회 | 5회 | 평균 |
| OPTIMAL | 50.20 | 49.40 | 51.20 | 48.80 | 50.80 | 50.08 |
| FIFO | 78.20 | 77.40 | 79.00 | 76.00 | 79.40 | 78 |
| LIFO | 78.20 | 73.00 | 79.80 | 75.40 | 73.00 | 75.88 |
| LRU | 75.80 | 78.80 | 78.80 | 76.00 | 78.80 | 77.64 |
| LFU | 75.80 | 75.80 | 79.80 | 76.80 | 76.60 | 76.96 |
| SC | 77.20 | 78.40 | 79.00 | 77.00 | 79.60 | 78.24 |
| ESC | 77.60 | 78.60 | 78.00 | 76.80 | 80.60 | 78.32 |

페이지 개수 9 스트림 개수 500

텍스트이(가) 표시된 사진

자동 생성된 설명 텍스트이(가) 표시된 사진

자동 생성된 설명 텍스트이(가) 표시된 사진

자동 생성된 설명 텍스트이(가) 표시된 사진

자동 생성된 설명 텍스트이(가) 표시된 사진

자동 생성된 설명

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 1회 | 2회 | 3회 | 4회 | 5회 | 평균 |
| OPTIMAL | 44.40 | 40.20 | 41.60 | 42.20 | 40.60 | 41.8 |
| FIFO | 74.20 | 67.60 | 69.80 | 69.60 | 67.40 | 69.72 |
| LIFO | 69.60 | 64.60 | 69.20 | 70.20 | 68.20 | 68.36 |
| LRU | 74.40 | 68.20 | 70.40 | 71.20 | 66.60 | 70.16 |
| LFU | 71.80 | 72.00 | 67.00 | 70.60 | 69.00 | 70.08 |
| SC | 75.00 | 69.40 | 70.40 | 70.40 | 68.20 | 70.68 |
| ESC | 75.40 | 69.60 | 70.60 | 69.60 | 68.80 | 70.8 |

페이지 개수 5 스트림 개수 5000

텍스트이(가) 표시된 사진

자동 생성된 설명 텍스트이(가) 표시된 사진

자동 생성된 설명 텍스트이(가) 표시된 사진

자동 생성된 설명 텍스트이(가) 표시된 사진

자동 생성된 설명 텍스트이(가) 표시된 사진

자동 생성된 설명

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 1회 | 2회 | 3회 | 4회 | 5회 | 평균 |
| OPTIMAL | 58.80 | 58.22 | 58.58 | 58.00 | 58.68 | 58.456 |
| FIFO | 83.16 | 82.94 | 83.32 | 83.20 | 82.70 | 83.064 |
| LIFO | 83.44 | 83.06 | 83.54 | 82.98 | 83.24 | 83.252 |
| LRU | 83.20 | 92.74 | 83.60 | 83.16 | 83.06 | 85.152 |
| LFU | 83.86 | 83.54 | 83.70 | 83.14 | 83.32 | 83.512 |
| SC | 83.30 | 82.88 | 83.44 | 83.24 | 83.00 | 83.172 |
| ESC | 83.76 | 83.26 | 83.60 | 83.62 | 83.14 | 83.476 |

페이지 개수 7 스트림 개수 5000

텍스트이(가) 표시된 사진

자동 생성된 설명 텍스트이(가) 표시된 사진

자동 생성된 설명 텍스트이(가) 표시된 사진

자동 생성된 설명 텍스트이(가) 표시된 사진

자동 생성된 설명 텍스트이(가) 표시된 사진

자동 생성된 설명

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 1회 | 2회 | 3회 | 4회 | 5회 | 평균 |
| OPTIMAL | 49.14 | 48.68 | 48.92 | 49.02 | 48.52 | 48.856 |
| FIFO | 77.38 | 77.88 | 77.06 | 76.48 | 76.66 | 77.092 |
| LIFO | 77.56 | 77.32 | 76.48 | 75.44 | 76.92 | 76.744 |
| LRU | 77.22 | 77.76 | 76.96 | 76.32 | 76.72 | 76.996 |
| LFU | 76.14 | 78.18 | 77.92 | 75.98 | 77.34 | 77.112 |
| SC | 77.26 | 77.78 | 77.22 | 76.30 | 76.52 | 77.016 |
| ESC | 77.44 | 77.56 | 77.10 | 76.52 | 76.98 | 77.12 |

페이지 개수 9 스트림 개수 5000

 텍스트이(가) 표시된 사진

자동 생성된 설명 텍스트이(가) 표시된 사진

자동 생성된 설명 텍스트이(가) 표시된 사진

자동 생성된 설명 텍스트이(가) 표시된 사진

자동 생성된 설명

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 1회 | 2회 | 3회 | 4회 | 5회 | 평균 |
| OPTIMAL | 40.46 | 40.32 | 40.40 | 40.56 | 41.12 | 40.572 |
| FIFO | 69.84 | 69.44 | 68.66 | 69.68 | 69.86 | 69.496 |
| LIFO | 69.92 | 70.10 | 70.12 | 70.72 | 70.42 | 70.256 |
| LRU | 69.82 | 69.56 | 69.30 | 70.08 | 70.34 | 69.82 |
| LFU | 70.96 | 70.30 | 69.68 | 68.68 | 70.44 | 70.012 |
| SC | 69.82 | 69.16 | 69.50 | 69.50 | 70.12 | 69.62 |
| ESC | 69.96 | 68.86 | 69.22 | 69.84 | 70.22 | 69.62 |

페이지 프레임의 개수가 많을수록, 참조 스트림이 많을수록 페이지 폴트가 일어날 확률은 줄어든다. optimal 알고리즘을 제외하고 나머지 알고리즘은 복잡함에 상관없이 다 비슷한 비율을 보인다.

소스코드

page\_replacement.h

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

struct pte {

int page\_num;

int count;

int reference\_bit;

int modify\_bit;

struct pte\* next;

};

struct stream {

int index;

int page\_num;

int modify\_bit;

struct stream\* next;

};

typedef struct pte pte;

typedef struct stream stream;

// stream list의 첫 노드를 생성(-1로 초기화 하여 비어잇음을 나타내고자 함)

void init\_stream(stream\*\* list) {

stream\* new\_stream = (stream\*) malloc(sizeof(stream));

new\_stream->index = -1;

new\_stream->page\_num = -1;

new\_stream->next = NULL;

\*list = new\_stream;

}

// stream list에 노드 추가

void add\_stream(stream\* list, int page\_num) {

// 리스트를 할당만 하고 비어있는 상태일때 처리

if (list->index == -1) {

list->index = 0;

list->page\_num = page\_num;

list->next = NULL;

return ;

}

// 다른 노드가 존재하는 경우 새로 할당받아 채운다.

stream\* new\_stream = (stream\*) malloc(sizeof(stream));

stream\* cur = list;

// 가장 마지막 노드를 선택한다.

while (cur->next != NULL) {

cur = cur->next;

}

새로 할당받은 구조체를 값을 채워 가장 마지막 노드뒤에 연결한다.

new\_stream->index = cur->index + 1;

new\_stream->page\_num = page\_num;

new\_stream->next = NULL;

cur->next = new\_stream;

}

// stream list 출력

void print\_streamlist(stream\* list) {

stream\* cur = list;

printf("stream list : \n");

// 각 노드를 순회하며 스트림의 인덱스와 페이지 넘버를 읽는다.

while (cur != NULL) {

printf("index : %d page\_num : %d\n", cur->index, cur->page\_num);

cur = cur->next;

}

printf("\n");

}

// page table을 num만큼 생성하고 0으로 초기화

pte\* init\_pagetable(int num) {

// 첫번째 노드 생성

pte\* head = (pte\*)malloc(sizeof(pte));

// num – 1번 순회하며 생성하여 추가해준다.

for (int i = 0; i < num - 1; i++) {

pte\* cur = head;

while(cur->next != NULL) {

cur = cur->next;

}

cur->next = (pte\*)malloc(sizeof(pte));

}

// 첫번째 노드의 주소를 반환

return head;

}

// page fault여부 확인하여 페이지폴트이면 1, 아니면 0 리턴

int is\_pagefault(pte\* head, int stream\_num) {

pte\* cur = head;

// stream\_num must not be 0

// 페이지 테이블에 스트림의 페이지 넘버와 같은 것이 있는지 확인

while (cur != NULL) {

if(cur->page\_num == stream\_num) {

//printf("page is in page table\n");

// 동일한 숫자 존재시 0 반환

return 0;

}

cur = cur->next;

}

// 없으면 1 반환

//printf("page fault\n");

return 1;

}

// page table 출력

void print\_pagetable(pte\* head, FILE\* fp) {

pte\* cur = head;

while (cur != NULL) {

if (cur->page\_num != 0) {

printf("%2d ", cur->page\_num);

fprintf(fp, "%2d ", cur->page\_num);

}

cur = cur->next;

}

}

// fifo 방식으로 가장 앞 페이지 테이블의 노드를 변경하여 가장 뒤로 이동한 page table의 시작주소를 리턴

pte\* replace\_fifo(pte\* head, int stream\_num) {

pte\* cur = head;

// if pagetable is not full

// 순회하며 0인 곳을 교체한다.

while (cur != NULL) {

if (cur->page\_num == 0) {

cur->page\_num = stream\_num;

//printf("page\_num : 0 -> stream\_num : %d\n", stream\_num);

return head;

}

cur = cur->next;

}

// if pagetable is full

// 맨 앞 페이지 테이블 엔트리를 교체한다.

pte\* replace = head;

//printf("page\_num : %d -> stream\_num : %d\n", replace->page\_num, stream\_num);

replace->page\_num = stream\_num;

pte\* new\_head = head->next;

cur = new\_head;

// 마지막 페이지 테이블 엔트리를 찾는다.

while (cur->next != NULL) {

cur = cur->next;

}

// 마지막 페이지 테이블 엔트리의 뒤에 교체한 페이지 테이블 엔트리 연결

cur->next = replace;

//printf("replace : %d\n", cur->next->page\_num);

cur->next->next = NULL;

// 새로운 첫번째 페이지 테이블 엔트리의 주소 리턴

return new\_head;

}

// lifo 방식으로 가장 뒷 페이지 테이블의 노드를 변경하고 새로운 page table 시작주소 리턴

pte\* replace\_lifo(pte\* head, int stream\_num) {

pte\* cur = head;

// if pagetable is not full

// 빈 페이지 테이블 엔트리가 있을때 우선적으로 채운다.

while (cur != NULL) {

if (cur->page\_num == 0) {

cur->page\_num = stream\_num;

return head;

}

cur = cur->next;

}

cur = head;

// if pagetable is full

// 마지막 테이블 엔트리를 찾는다.

while (cur->next != NULL) {

cur = cur->next;

}

pte\* replace = cur;

replace->page\_num = stream\_num;

// 마지막 페이지 테이블을 수정함으로 페이지테이블 엔트리를 교체하는 효과를 구현한다.

return head;

}

// min과 max사이의 무작위 수를 생성 ,최소는 min, 최대는 max – 1 + min이다.

int get\_random\_number(int min, int max) {

int result;

result = rand() % max + min;

return result;

}

// stream list 에서 해당 index의 노드를 리턴

stream\* get\_stream(stream\* list, int index) {

stream\* cur = list;

// 비어있는 경우

if (cur->index == -1)

return NULL;

// 인덱스만큼 스트림 이동

for(int i = 0; i < index; i++) {

cur = cur->next;

}

// 이동시킨 스트림 반환

return cur;

}

// stream list에서 현재 인덱스(curIndex)에서 특정 stream list의 페이지 넘버(cur\_pageNum)까지 떨어진 거리 리턴

int get\_index\_gap(stream\* list, int curIndex, int cur\_pageNum) {

// 다음 스트림을 찾는다.

stream\* cur = get\_stream(list, curIndex + 1);

// 현재 스트림의 페이지 넘버와 그 이후의 스트림의 페이지 넘버가 같아질 때까지 순회한다.

while (cur != NULL && cur->page\_num != cur\_pageNum) {

cur = cur->next;

}

// 현재 인덱스 이후 페이지 넘버가 등장하지 않으면 999 리턴

if (cur == NULL)

return 999; // MAX\_INT

// 존재한다면 가장 먼저 발견된 인덱스에서 현재 인덱스를 빼서 리턴

return cur->index - curIndex;

}

// stream list에서 curIndex로부터 page table에 저장된 page\_num과의 거리를 구해 가장 큰 거리의 page table entry를 리턴

pte\* get\_change\_pte(pte\* head, stream\* list, int curIndex) {

pte\* cur = head;

pte\* max = head;

// 페이지 테이블을 순회하며 페이지 테이블에 있는 페이지 넘버들이 얼마의 스트림 뒤에 다시 등장하는지 측정한다. 현재 max로 지정된 것보다 큰 차이를 얻으면 해당 페이지 테이블 엔트리로 교체한다.

while (cur->next != NULL) {

if (get\_index\_gap(list, curIndex, max->page\_num) < get\_index\_gap(list, curIndex, cur->next->page\_num)) {

max = cur->next;

}

cur = cur->next;

}

return max;

}

// page table에서 change\_pte의 바로 이전 노드를 리턴

pte\* get\_prev\_pte(pte\* head, pte\* change\_pte) {

pte\* cur = head;

// 가장 앞의 페이지 테이블 엔트리는 널값 리턴

if (cur == change\_pte)

return NULL;

// 다음 페이지 테이블 엔트리가 교체할 페이지 테이블 엔트리가 될 때까지 순회한다.

while (cur->next != change\_pte && cur->next != NULL) {

cur = cur->next;

}

// 결과를 리턴

return cur;

}

// stream list의 curIndex에서 page table 에 stream\_num로 교체.

pte\* replace\_optimal(pte\* head, stream\* list, int stream\_num, int curIndex) {

pte\* cur = head;

// if pagetable is not full

// 비어있는 페이지 테이블 엔트리를 채운다.

while (cur != NULL) {

if (cur->page\_num == 0) {

cur->page\_num = stream\_num;

return head;

}

cur = cur->next;

}

// 교체할 페이지 테이블 엔트리를 선택한다.(가장 거리가 먼 페이지 테이블 엔트리)

cur = head;

// if pagetable is full

// pte\* change\_pte = get\_change\_pte(head, list, curIndex);

pte\* change\_pte = get\_change\_pte(head, list, curIndex);

pte\* prev\_pte;

// 교체될 페이지 테이블 엔트리가 첫번째 페이지 테이블 엔트리일때

if (change\_pte == head) {

// 두번째 페이지 테이블을 새로운 첫번째 페이지 테이블로 설정하고 값들을 변경한 뒤 가장 뒤에 연결

pte\* new\_head = change\_pte->next;

while (cur->next != NULL) {

cur = cur->next;

}

change\_pte->page\_num = stream\_num;

change\_pte->next = NULL;

cur->next = change\_pte;

return new\_head;

}

// 교체될 페이지 테이블 엔트리가 첫번째가 아니면 이전 페이지 테이블 엔트리를 구해 이전 페이지 테이블 엔트리와 교체될 엔트리의 다음 엔트리와 연결하고 값을 수정한뒤 가장 뒤에 연결한다.

prev\_pte = get\_prev\_pte(head, change\_pte);

prev\_pte->next = change\_pte->next;

change\_pte->page\_num = stream\_num;

change\_pte->next = NULL;

while (cur->next != NULL) {

cur = cur->next;

}

cur->next = change\_pte;

return head;

}

// 참조된 페이지테이블엔트리 리턴

pte\* get\_referred\_pte(pte\* head, int stream\_num) {

pte\* cur = head;

while (cur != NULL) {

if(cur->page\_num == stream\_num) {

return cur;

}

cur = cur->next;

}

return NULL;

}

pte\* replace\_lru(pte\* head, int stream\_num, int\* number\_of\_page\_fault, FILE\* fp) {

if (is\_pagefault(head, stream\_num)) {

\*number\_of\_page\_fault = \*number\_of\_page\_fault + 1;

pte\* cur = head;

while(cur->next != NULL) {

cur = cur->next;

}

pte\* tail = get\_prev\_pte(head, cur);

cur->page\_num = stream\_num;

cur->next = head;

//head = cur;

tail->next = NULL;

printf(" F |");

fprintf(fp, " F |");

return cur;

}

pte\* change = get\_referred\_pte(head, stream\_num);

if (change == NULL) {

printf("get\_referred\_pte가 잘못되었습니다.\n");

return NULL;

}

if (change == head) {

change->page\_num = stream\_num;

printf(" |");

fprintf(fp, " |");

return change;

}

pte\* prev = get\_prev\_pte(head, change);

if (prev == NULL) {

printf("get\_prev\_pte가 잘못되었습니다.\n");

return NULL;

}

prev->next = change->next;

change->page\_num = stream\_num;

change->next = head;

printf(" |");

fprintf(fp, " |");

return change;

}

pte\* get\_least\_used\_pte(pte\* head) {

pte\* min = head;

pte\* cur = head;

while (cur->next != NULL) {

if (min->count > cur->next->count) {

min = cur->next;

}

cur = cur->next;

}

return min;

}

void print\_pagetable\_count(pte\* head) {

pte\* cur = head;

while (cur != NULL) {

printf("%d ", cur->count);

cur = cur->next;

}

printf("\n");

}

void increase\_pagetable\_count(pte\* head, int page\_num) {

pte\* cur = head;

while (cur != NULL) {

if (cur->page\_num == page\_num) {

cur->count = cur->count + 1;

}

cur = cur->next;

}

}

pte\* replace\_lfu(pte\* head, int stream\_num, int\* number\_of\_page\_fault, FILE\* fp) {

if (is\_pagefault(head, stream\_num)) {

\*number\_of\_page\_fault = \*number\_of\_page\_fault + 1;

pte\* min = get\_least\_used\_pte(head);

if (min == head) {

pte\* new\_head = min->next;

min->page\_num = stream\_num;

min->next = NULL;

pte\* cur = new\_head;

while (cur->next != NULL) {

cur = cur->next;

}

cur->next = min;

printf(" F |");

fprintf(fp, " F |");

return new\_head;

}

pte\* prev = get\_prev\_pte(head, min);

prev->next = min->next;

min->page\_num = stream\_num;

min->next = NULL;

pte\* cur = prev;

while (cur->next != NULL) {

cur = cur->next;

}

cur->next = min;

printf(" F |");

fprintf(fp, " F |");

return head;

}

increase\_pagetable\_count(head, stream\_num);

printf(" |");

fprintf(fp, " |");

return head;

}

// sc를 위한 서큘라 페이지 테이블 생성

pte\* init\_circular\_table(int num) {

pte\* head = (pte\*)malloc(sizeof(pte));

head->next = head;

for (int i = 0; i < num - 1; i++) {

pte\* cur = head;

// 다음 페이지 테이블이 첫번째 페이지 테이블 엔트리를 가리킬때까지 순회한다.

while(cur->next != head) {

cur = cur->next;

}

// 첫번째 테이블 엔트리의 이전에 새로운 노드를 넣는다.(마지막과 첫 엔트리는 연결되어 있음)

cur->next = (pte\*)malloc(sizeof(pte));

cur->next->next = head;

}

return head;

}

// 서큘라 페이지 테이블 출력

void print\_circular\_table(pte\* head, FILE\* fp) {

pte\* cur = head;

// 다음 페이지 테이블 엔트리가 첫번째 페이지 테이블 엔트리가 될 때까지 순회하며 출력한다.

while (cur->next != head) {

if (cur->page\_num != 0) {

printf("%2d", cur->page\_num);

fprintf(fp, "%2d", cur->page\_num);

if (cur->reference\_bit == 1) {

// 참조비트가 0이 아니면 \* 표시한다.

printf("\* ");

fprintf(fp, "\* ");

}else {

printf(" ");

fprintf(fp, " ");

}

}

cur = cur->next;

}

// 빈페이지는 출력하지 않도록 처리한다.

if (cur->page\_num != 0) {

printf("%2d", cur->page\_num); //통일성을 위해 나중에 \n은 뺀다.

fprintf(fp, "%2d", cur->page\_num);

if (cur->reference\_bit == 1) {

printf("\* ");

fprintf(fp, "\* ");

}else {

printf(" ");

fprintf(fp, " ");

}

}

}

// esc 알고리즘 사용시 참조비트와 수정비트를 표현할 수 있는 함수

void print\_esc\_table(pte\* head, FILE\* fp) {

pte\* cur = head;

//다음 페이지 테이블 엔트리가 첫 페이지 테이블 엔트리가 될 때까지 순회하며 출력

while (cur->next != head) {

if (cur->page\_num != 0) {

// 빈페이지테이블엔트리가 아니면 페이지 넘버, 참조비트, 수정 비트를 출력하도록 한다.

printf("%2d(%d, %d) ", cur->page\_num, cur->reference\_bit, cur->modify\_bit);

fprintf(fp, "%2d(%d, %d) ", cur->page\_num, cur->reference\_bit, cur->modify\_bit);

}

cur = cur->next;

}

// 가장 마지막 페이지 테이블엔트리를 출력한디/

if (cur->page\_num != 0) {

printf("%2d(%d, %d) ", cur->page\_num, cur->reference\_bit, cur->modify\_bit);

fprintf(fp, "%2d(%d, %d) ", cur->page\_num, cur->reference\_bit, cur->modify\_bit);

}

}

// 서큘러 페이지 테이블을 사용할 때 페이지 폴트를 확인한다.

int is\_circular\_fault(pte\* head, int stream\_num) {

pte\* cur = head;

// stream\_num must not be 0

// 다음 페이지 테이블 엔트리가 첫 페이지 테이블 엔트리가 될 때까지 비교한다.

while (cur->next != head) {

if (cur->page\_num == stream\_num) {

return 0;

}

cur = cur->next;

}

// 마지막 페이지 테이블 엔트리를 확인하는 과정

// last pte

if (cur->page\_num == stream\_num) {

//printf("last pte : %d\n", cur->page\_num);

return 0;

}

// 스트림의 페이지 넘버가 페이지 테이블에 존재하지 않을 때 1 리턴

return 1;

}

// 서큘라 페이지 테이블에서 페이지 폴트가 아닐 때 인자로 넘겨받은 페이지 넘버와 일치하는 페이지 테이블 엔트리를 리턴

pte\* get\_referring\_pte(pte\* head, int stream\_num) {

pte\* cur = head;

// 페이지 순회하며 스트림 페이지 넘버가 페이지 테이블에 포함되어있는지 확인

while (cur->next != head) {

if (cur->page\_num == stream\_num) {

// 존재시 해당 페이지테이블 엔트리 리턴

return cur;

}

cur = cur->next;

}

// 마지막 페이지 테이블 엔트리 확인

if (cur->page\_num == stream\_num) {

return cur;

}

//없으면 널. 리턴

return NULL;

}

// sc 알고리즘

pte\* replace\_sc(pte\* head, pte\* clock, int stream\_num, int\* number\_of\_page\_fault, FILE\* fp) {

if (is\_circular\_fault(head, stream\_num)) {

\*number\_of\_page\_fault = \*number\_of\_page\_fault + 1;

pte\* cur = clock;

while (cur->reference\_bit != 0) {

cur->reference\_bit = 0;

cur = cur->next;

}

cur->page\_num = stream\_num;

cur->reference\_bit = 1;

printf(" F |");

fprintf(fp, " F |");

return cur->next;

}

get\_referring\_pte(head, stream\_num)->reference\_bit = 1;

printf(" |");

fprintf(fp, " |");

return clock;

}

// ESC 알고리즘을 시행할 때 스트림 리스트를 만드는 기능

void add\_esc\_stream(stream\* list, int page\_num, int modify\_bit) {

if (list->index == -1) {

list->index = 0;

list->page\_num = page\_num;

list->next = NULL;

list->modify\_bit = modify\_bit;

return ;

}

stream\* new\_stream = (stream\*) malloc(sizeof(stream));

stream\* cur = list;

while (cur->next != NULL) {

cur = cur->next;

}

new\_stream->index = cur->index + 1;

new\_stream->page\_num = page\_num;

new\_stream->next = NULL;

new\_stream->modify\_bit = modify\_bit;

cur->next = new\_stream;

}

// 스트림 리스트의 수정 비트를 출력하는 기능

void print\_stream\_modify\_bits(stream\* list) {

stream\* cur = list;

printf("modify\_bits : ");

while (cur != NULL) {

printf("%d ", cur->modify\_bit);

cur = cur->next;

}

printf("\n");

}

// 스트림 리스트의 참조 비트를 출력하는 기능

void print\_pagetable\_reference\_bits(pte\* head) {

pte\* cur = head;

while (cur != NULL) {

printf("%d ", cur->reference\_bit);

cur = cur->next;

}

printf("\n");

}

// 서큘러 페이지 테이블의 참조비트 출력

void print\_circular\_table\_reference\_bits(pte\* head) {

pte\* cur = head;

printf("reference\_bits : ");

while (cur->next != head) {

printf("%2d ", cur->reference\_bit);

cur = cur->next;

}

printf("%2d\n", cur->reference\_bit);

}

// 서큘러 페이지 테이블 수정 비트 출력

void print\_circular\_table\_modify\_bits(pte\* head) {

pte\* cur = head;

printf("modify\_bits : ");

while (cur->next != head) {

printf("%2d ", cur->modify\_bit);

cur = cur->next;

}

printf("%2d\n", cur->modify\_bit);

}

// 페이지 교체시 값 변경해주는 함수

void set\_esc\_pte(pte\* change\_pte, int stream\_num, int modify\_bit) {

change\_pte->page\_num = stream\_num;

change\_pte->reference\_bit = 1;

change\_pte->modify\_bit = modify\_bit;

}

//esc 알고리즘

pte\* replace\_esc(pte\* head, pte\* clock, int stream\_num, int modify\_bit, int\* number\_of\_page\_fault, FILE\* fp) {

if (is\_circular\_fault(head, stream\_num)) {

//페이지 폴트 숫자 늘린다.

\*number\_of\_page\_fault = \*number\_of\_page\_fault + 1;

pte\* cur = clock;

while (1) {

// find (0, 0)

while (cur->next != clock) {

if (cur->reference\_bit == 0 && cur->modify\_bit == 0) {

// (0, 0)발견시 페이지 테이블 엔트리 값 변경

set\_esc\_pte(cur, stream\_num, modify\_bit);

printf(" F |");

fprintf(fp, " F |");

return cur->next;

}

cur = cur->next;

}

// 마지막 페이지 테이블 엔트리에 대한 확인

if (cur->reference\_bit == 0 && cur->modify\_bit == 0) {

set\_esc\_pte(cur, stream\_num, modify\_bit);

printf(" F |");

fprintf(fp, " F |");

return cur->next;

}

// or cur = cur->next;

cur = clock;

// find (0, 1)

while (cur->next != clock) {

if (cur->reference\_bit == 0 && cur->modify\_bit == 1) {

// (0, 1)을 찾았을 때 해당 페이지 테이블의 값을 변경한다.

set\_esc\_pte(cur, stream\_num, modify\_bit);

printf(" F |");

fprintf(fp, " F |");

return cur->next;

}

cur->reference\_bit = 0;

cur = cur->next;

}

// 마지막 페이지 테이블 엔트리에 대한 확인

if (cur->reference\_bit == 0 && cur->modify\_bit == 1) {

set\_esc\_pte(cur, stream\_num, modify\_bit);

printf(" F |");

fprintf(fp, " F |");

return cur->next;

}

// 이 단계에서는 참조비트를 1에서 0으로 변경한다.

cur->reference\_bit = 0;

cur = clock; // or cur = cur->next;

}

}

// 페이지 폴트가 아닐 때 페이지 테이블 엔트리의 참조비트를 1로 바꾼다.

get\_referring\_pte(head, stream\_num)->reference\_bit = 1;

// 스트림으로 들어온 수정 비트가 1일때 페이지 테이블 엔트리의 수정 비트도 1로 변경한다.

if (modify\_bit == 1) {

get\_referring\_pte(head, stream\_num)->modify\_bit = 1;

}

printf(" |");

fprintf(fp, " |");

return clock;

}

// 선형 페이지 테이블 free하는 함수

void free\_pagetable(pte\* head) {

pte\* cur = head;

while(cur != NULL) {

pte\* del = cur;

cur = del->next;

free(del);

}

}

// 선형 스트림 free하는 함수

void free\_streams(stream\* streams) {

stream\* cur = streams;

while (cur != NULL) {

stream\* del = cur;

cur = del->next;

free(del);

}

}

ui.c

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>

#include <unistd.h>

#include "page\_replacement.h"

int tokenize(char\* input, char\* argv[]) {

char\* ptr = NULL;

int argc = 0;

ptr = strtok(input, " ");

while (ptr != NULL) {

argv[argc++] = ptr;

ptr = strtok(NULL, " ");

}

return argc;

}

int main() {

char input[1024];

char\* selected\_algorithm[3];

int number\_of\_selected;

int algorithms[3];

int number\_of\_page\_frame;

int input\_way;

int argc;

int number\_of\_streams;

char input\_file\_name[1024];

char output\_full\_name[1024];

char\* output\_file\_name = "output";

char\* ext = "txt";

int same\_file\_count = 0;

sprintf(output\_full\_name, "%s.%s", output\_file\_name, ext);

while (access(output\_full\_name, F\_OK) != -1) {

same\_file\_count++;

sprintf(output\_full\_name, "%s(%d).%s",output\_file\_name, same\_file\_count, ext);

}

srand(time(NULL));

printf("A. Page Replacement 알고리즘 시뮬레이터를 선택하시오 (최대 3개)\n");

printf(" (1) Optimal (2) FIFO (3) LIFO (4) LRU (5) LFU (6) SC (7) ESC (8) ALL\n");

fgets(input, sizeof(input), stdin);

input[strlen(input) - 1] = '\0';

number\_of\_selected = tokenize(input, selected\_algorithm);

if (number\_of\_selected > 3 || number\_of\_selected < 1) {

printf("최대 3가지 알고리즘을 고를 수 있습니다.\n");

exit(-1);

}

argc = 0;

for (int i = 0; i < number\_of\_selected; i++) {

algorithms[i] = atoi(selected\_algorithm[i]);

if (algorithms[i] > 8 || algorithms[i] < 1) {

printf("1 ~ 8 사이의 값을 입력할 수 있습니다.\n");

exit(-1);

}

argc++;

}

if (argc > 1) {

for (int i = 0; i < argc; i++) {

if (algorithms[i] == 8) {

printf("8번은 단독으로 선택하세요.\n");

exit(-1);

}

}

}

printf("B. 페이지 프레임의 개수를 입력하시오.(3~10)\n");

fgets(input, sizeof(input), stdin);

input[strlen(input) - 1] = '\0';

number\_of\_page\_frame = atoi(input);

if (number\_of\_page\_frame < 3 || number\_of\_page\_frame > 10) {

printf("페이지 프레임의 수는 3과 10 사이의 정수만 가능합니다.\n");

exit(-1);

}

printf("C. 데이터의 입력 방식을 선택하시오. (1,2)\n");

printf(" (1) 랜덤하게 생성 (2) 사용자 생성 파일 오픈\n");

fgets(input, sizeof(input), stdin);

input[strlen(input) - 1] = '\0';

input\_way = atoi(input);

FILE\* fp;

int length = 0;

int offset = 0;

int sep = 0;

int i = 0;

char buf[1024];

char input\_stream[10];

char string\_page\_number[10];

char string\_modify\_bit[10];

stream\* streams;

init\_stream(&streams);

//number\_of\_streams = (rand() % 10000) + 500;

number\_of\_streams = 20; // 변경해야함

if (input\_way == 1) {

for (int i = 0; i < number\_of\_streams; i++) {

add\_esc\_stream(streams, get\_random\_number(1, 30), get\_random\_number(0, 2));

}

}else if (input\_way == 2) {

printf("파일 이름을 입력해 주세요.\n");

fgets(input\_file\_name, sizeof(input\_file\_name), stdin);

input\_file\_name[strlen(input\_file\_name) - 1] = '\0';

if (access(input\_file\_name, F\_OK) == -1) {

printf("존재하지 않는 파일입니다.\n");

return -1;

}

if((fp = fopen(input\_file\_name, "r")) == NULL) {

printf("ERROR: fail to open\n");

exit(-1);

}

while ((length = fread(buf,1, 1024, fp)) > 0) {

buf[length] = '\0';

i = 0;

sep = 0;

int string\_input\_length = 0;

for (i = 0; i < length; i++) {

if (buf[i] == ' ')

continue;

if (buf[i] == '|') {

offset = offset + i + 1;

fseek(fp, offset, SEEK\_SET);

break;

}

input\_stream[string\_input\_length] = buf[i];

string\_input\_length++;

}

input\_stream[i] = '\0';

for (i = 0; i < 10; i++) {

if (input\_stream[i] == ',') {

sep = i + 1;

break;

}

string\_page\_number[i] = input\_stream[i];

}

string\_page\_number[sep - 1] = '\0';

int modify\_bit\_length = 0;

for (i = sep; i < 10; i++) {

if (input\_stream[i] == '\0')

break;

string\_modify\_bit[modify\_bit\_length] = input\_stream[i];

modify\_bit\_length++;

}

string\_modify\_bit[modify\_bit\_length] = '\0';

int a = atoi(string\_page\_number);

int b = atoi(string\_modify\_bit);

if (a != 0) {

add\_esc\_stream(streams, a, b);

}

}

fclose(fp);

stream\* last = streams;

while (last->next != NULL) {

last = last->next;

}

number\_of\_streams = last->index + 1;

}else {

printf("1이나 2를 입력 하세요.\n");

exit(-1);

}

// 실제 알고리즘 수행

fp = fopen(output\_full\_name, "a");

pte\* pt1 = init\_pagetable(number\_of\_page\_frame);

int optimal\_page\_fault\_counter = 0;

printf(" OPTIMAL\n\n");

printf(" page fault | stream | page table\n");

fprintf(fp, " OPTIMAL\n\n");

for (int i = 0; i < number\_of\_streams; i++) {

if(is\_pagefault(pt1, get\_stream(streams, i)->page\_num)) {

pt1 = replace\_optimal(pt1, streams, get\_stream(streams, i)->page\_num, i);

printf(" F |");

fprintf(fp, " F |");

optimal\_page\_fault\_counter++;

}else {

printf(" |");

fprintf(fp, " |");

}

printf(" %2d | ", get\_stream(streams, i)->page\_num);

fprintf(fp, " %2d | ", get\_stream(streams, i)->page\_num);

print\_pagetable(pt1, fp);

printf("\n");

fprintf(fp, "\n");

}

printf("Page Fault : %d\nRate : %.2f\n\n\n", optimal\_page\_fault\_counter, (float)optimal\_page\_fault\_counter / number\_of\_streams \* 100);

fprintf(fp, "Page Fault : %d\nRate : %.2f\n\n\n", optimal\_page\_fault\_counter, (float)optimal\_page\_fault\_counter / number\_of\_streams \* 100);

free\_pagetable(pt1);

for (int i = 0; i < argc; i++) {

if (algorithms[i] == 1) {

;

}else if (algorithms[i] == 2) {

pte\* pt2 = init\_pagetable(number\_of\_page\_frame);

int number\_of\_page\_fault = 0;

printf(" FIFO\n\n page fault | stream | page table\n");

fprintf(fp, " FIFO\n\n page fault | stream | page table\n");

for (int i = 0; i < number\_of\_streams; i++) {

if (is\_pagefault(pt2, get\_stream(streams, i)->page\_num)) {

pt2 = replace\_fifo(pt2, get\_stream(streams, i)->page\_num);

printf(" F |");

fprintf(fp, " F |");

number\_of\_page\_fault++;

}else {

printf(" |");

fprintf(fp, " |");

}

printf(" %2d | ", get\_stream(streams, i)->page\_num);

fprintf(fp, " %2d | ", get\_stream(streams, i)->page\_num);

print\_pagetable(pt2, fp);

printf("\n");

fprintf(fp, "\n");

}

printf("Page Fault : %d\nRate : %.2f\n\n\n", number\_of\_page\_fault, (float)number\_of\_page\_fault / number\_of\_streams \* 100);

fprintf(fp, "Page Fault : %d\nRate : %.2f\n\n\n", number\_of\_page\_fault, (float)number\_of\_page\_fault / number\_of\_streams \* 100);

free\_pagetable(pt2);

}else if (algorithms[i] == 3) {

pte\* pt3 = init\_pagetable(number\_of\_page\_frame);

int number\_of\_page\_fault = 0;

printf(" LIFO\n\n page fault | stream | page table\n");

fprintf(fp, " LIFO\n\n page fault | stream | page table\n");

for (int i = 0; i < number\_of\_streams; i++) {

if (is\_pagefault(pt3, get\_stream(streams, i)->page\_num)) {

pt3 = replace\_lifo(pt3, get\_stream(streams, i)->page\_num);

printf(" F |");

fprintf(fp, " F |");

number\_of\_page\_fault++;

}else {

printf(" |");

fprintf(fp, " |");

}

printf(" %2d | ", get\_stream(streams, i)->page\_num);

fprintf(fp, " %2d | ", get\_stream(streams, i)->page\_num);

print\_pagetable(pt3, fp);

printf("\n");

fprintf(fp, "\n");

}

printf("Page Fault : %d\nRate : %.2f\n\n\n", number\_of\_page\_fault, (float)number\_of\_page\_fault / number\_of\_streams \* 100);

fprintf(fp, "Page Fault : %d\nRate : %.2f\n\n\n", number\_of\_page\_fault, (float)number\_of\_page\_fault / number\_of\_streams \* 100);

free\_pagetable(pt3);

}else if (algorithms[i] == 4) {

pte\* pt4 = init\_pagetable(number\_of\_page\_frame);

int number\_of\_page\_fault = 0;

printf(" LRU\n\n page fault | stream | page table\n");

fprintf(fp, " LRU\n\n page fault | stream | page table\n");

for (int i = 0; i < number\_of\_streams; i++) {

pt4 = replace\_lru(pt4, get\_stream(streams, i)->page\_num, &number\_of\_page\_fault, fp);

printf(" %2d | ", get\_stream(streams, i)->page\_num);

fprintf(fp, " %2d | ", get\_stream(streams, i)->page\_num);

print\_pagetable(pt4, fp);

printf("\n");

fprintf(fp, "\n");

}

printf("Page Fault : %d\nRate : %.2f\n\n\n", number\_of\_page\_fault, (float)number\_of\_page\_fault / number\_of\_streams \* 100);

fprintf(fp, "Page Fault : %d\nRate : %.2f\n\n\n", number\_of\_page\_fault, (float)number\_of\_page\_fault / number\_of\_streams \* 100);

free\_pagetable(pt4);

}else if (algorithms[i] == 5) {

pte\* pt5 = init\_pagetable(number\_of\_page\_frame);

int number\_of\_page\_fault = 0;

printf(" LFU\n\n page fault | stream | page table\n");

fprintf(fp, " LFU\n\n page fault | stream | page table\n");

for (int i = 0; i < number\_of\_streams; i++) {

pt5 = replace\_lfu(pt5, get\_stream(streams, i)->page\_num, &number\_of\_page\_fault, fp);

printf(" %2d | ", get\_stream(streams, i)->page\_num);

fprintf(fp, " %2d | ", get\_stream(streams, i)->page\_num);

print\_pagetable(pt5, fp);

printf("\n");

fprintf(fp, "\n");

}

printf("Page Fault : %d\nRate : %.2f\n\n\n", number\_of\_page\_fault, (float)number\_of\_page\_fault / number\_of\_streams \* 100);

fprintf(fp, "Page Fault : %d\nRate : %.2f\n\n\n", number\_of\_page\_fault, (float)number\_of\_page\_fault / number\_of\_streams \* 100);

free\_pagetable(pt5);

}else if (algorithms[i] == 6) {

pte\* pt6 = init\_circular\_table(number\_of\_page\_frame);

pte\* clock1 = pt6;

int number\_of\_page\_fault = 0;

printf(" SC\n\n page fault | stream | page table\n");

fprintf(fp, " SC\n\n page fault | stream | page table\n");

for (int i = 0; i < number\_of\_streams; i++) {

clock1 = replace\_sc(pt6, clock1, get\_stream(streams, i)->page\_num, &number\_of\_page\_fault, fp);

printf(" %2d | ", get\_stream(streams, i)->page\_num);

fprintf(fp, " %2d | ", get\_stream(streams, i)->page\_num);

print\_circular\_table(pt6, fp);

printf("\n");

fprintf(fp, "\n");

}

printf("Page Fault : %d\nRate : %.2f\n\n\n", number\_of\_page\_fault, (float)number\_of\_page\_fault / number\_of\_streams \* 100);

fprintf(fp, "Page Fault : %d\nRate : %.2f\n\n\n", number\_of\_page\_fault, (float)number\_of\_page\_fault / number\_of\_streams \* 100);

free\_pagetable(pt6);

}else if (algorithms[i] == 7) {

pte\* pt7 = init\_circular\_table(number\_of\_page\_frame);

pte\* clock2 = pt7;

int number\_of\_page\_fault = 0;

printf(" ESC\n\n page fault | stream | modify bit | page table\n");

fprintf(fp, " ESC\n\n page fault | stream | modify bit | page table\n");

for (int i = 0; i < number\_of\_streams; i++) {

clock2 = replace\_esc(pt7, clock2, get\_stream(streams, i)->page\_num, get\_stream(streams, i)->modify\_bit, &number\_of\_page\_fault, fp);

printf(" %2d | %d | ", get\_stream(streams, i)->page\_num, get\_stream(streams, i)->modify\_bit);

fprintf(fp, " %2d | %d | ", get\_stream(streams, i)->page\_num, get\_stream(streams, i)->modify\_bit);

print\_esc\_table(pt7, fp);

printf("\n");

fprintf(fp, "\n");

}

printf("Page Fault : %d\nRate : %.2f\n\n\n", number\_of\_page\_fault, (float)number\_of\_page\_fault / number\_of\_streams \* 100);

fprintf(fp, "Page Fault : %d\nRate : %.2f\n\n\n", number\_of\_page\_fault, (float)number\_of\_page\_fault / number\_of\_streams \* 100);

free\_pagetable(pt7);

}else if (algorithms[i] == 8) {

pte\* pt2 = init\_pagetable(number\_of\_page\_frame);

int number\_of\_page\_fault = 0;

printf(" FIFO\n\n page fault | stream | page table\n");

fprintf(fp, " FIFO\n\n page fault | stream | page table\n");

for (int i = 0; i < number\_of\_streams; i++) {

if (is\_pagefault(pt2, get\_stream(streams, i)->page\_num)) {

pt2 = replace\_fifo(pt2, get\_stream(streams, i)->page\_num);

printf(" F |");

fprintf(fp, " F |");

number\_of\_page\_fault++;

}else {

printf(" |");

fprintf(fp, " |");

}

printf(" %2d | ", get\_stream(streams, i)->page\_num);

fprintf(fp, " %2d | ", get\_stream(streams, i)->page\_num);

print\_pagetable(pt2, fp);

printf("\n");

fprintf(fp, "\n");

}

printf("Page Fault : %d\nRate : %.2f\n\n\n", number\_of\_page\_fault, (float)number\_of\_page\_fault / number\_of\_streams \* 100);

fprintf(fp, "Page Fault : %d\nRate : %.2f\n\n\n", number\_of\_page\_fault, (float)number\_of\_page\_fault / number\_of\_streams \* 100);

free\_pagetable(pt2);

pte\* pt3 = init\_pagetable(number\_of\_page\_frame);

number\_of\_page\_fault = 0;

printf(" LIFO\n\n page fault | stream | page table\n");

fprintf(fp, " LIFO\n\n page fault | stream | page table\n");

for (int i = 0; i < number\_of\_streams; i++) {

if (is\_pagefault(pt3, get\_stream(streams, i)->page\_num)) {

pt3 = replace\_lifo(pt3, get\_stream(streams, i)->page\_num);

printf(" F |");

fprintf(fp, " F |");

number\_of\_page\_fault++;

}else {

printf(" |");

fprintf(fp, " |");

}

printf(" %2d | ", get\_stream(streams, i)->page\_num);

fprintf(fp, " %2d | ", get\_stream(streams, i)->page\_num);

print\_pagetable(pt3, fp);

printf("\n");

fprintf(fp, "\n");

}

printf("Page Fault : %d\nRate : %.2f\n\n\n", number\_of\_page\_fault, (float)number\_of\_page\_fault / number\_of\_streams \* 100);

fprintf(fp, "Page Fault : %d\nRate : %.2f\n\n\n", number\_of\_page\_fault, (float)number\_of\_page\_fault / number\_of\_streams \* 100);

free\_pagetable(pt3);

pte\* pt4 = init\_pagetable(number\_of\_page\_frame);

number\_of\_page\_fault = 0;

printf(" LRU\n\n page fault | stream | page table\n");

fprintf(fp, " LRU\n\n page fault | stream | page table\n");

for (int i = 0; i < number\_of\_streams; i++) {

pt4 = replace\_lru(pt4, get\_stream(streams, i)->page\_num, &number\_of\_page\_fault, fp);

printf(" %2d | ", get\_stream(streams, i)->page\_num);

fprintf(fp, " %2d | ", get\_stream(streams, i)->page\_num);

print\_pagetable(pt4, fp);

printf("\n");

fprintf(fp, "\n");

}

printf("Page Fault : %d\nRate : %.2f\n\n\n", number\_of\_page\_fault, (float)number\_of\_page\_fault / number\_of\_streams \* 100);

fprintf(fp, "Page Fault : %d\nRate : %.2f\n\n\n", number\_of\_page\_fault, (float)number\_of\_page\_fault / number\_of\_streams \* 100);

free\_pagetable(pt4);

pte\* pt5 = init\_pagetable(number\_of\_page\_frame);

number\_of\_page\_fault = 0;

printf(" LFU\n\n page fault | stream | page table\n");

fprintf(fp, " LFU\n\n page fault | stream | page table\n");

for (int i = 0; i < number\_of\_streams; i++) {

pt5 = replace\_lfu(pt5, get\_stream(streams, i)->page\_num, &number\_of\_page\_fault, fp);

printf(" %2d | ", get\_stream(streams, i)->page\_num);

fprintf(fp, " %2d | ", get\_stream(streams, i)->page\_num);

print\_pagetable(pt5, fp);

printf("\n");

fprintf(fp, "\n");

}

printf("Page Fault : %d\nRate : %.2f\n\n\n", number\_of\_page\_fault, (float)number\_of\_page\_fault / number\_of\_streams \* 100);

fprintf(fp, "Page Fault : %d\nRate : %.2f\n\n\n", number\_of\_page\_fault, (float)number\_of\_page\_fault / number\_of\_streams \* 100);

free\_pagetable(pt5);

pte\* pt6 = init\_circular\_table(number\_of\_page\_frame);

pte\* clock1 = pt6;

number\_of\_page\_fault = 0;

printf(" SC\n\n page fault | stream | page table\n");

fprintf(fp, " SC\n\n page fault | stream | page table\n");

for (int i = 0; i < number\_of\_streams; i++) {

clock1 = replace\_sc(pt6, clock1, get\_stream(streams, i)->page\_num, &number\_of\_page\_fault, fp);

printf(" %2d | ", get\_stream(streams, i)->page\_num);

fprintf(fp, " %2d | ", get\_stream(streams, i)->page\_num);

print\_circular\_table(pt6, fp);

printf("\n");

fprintf(fp, "\n");

}

printf("Page Fault : %d\nRate : %.2f\n\n\n", number\_of\_page\_fault, (float)number\_of\_page\_fault / number\_of\_streams \* 100);

fprintf(fp, "Page Fault : %d\nRate : %.2f\n\n\n", number\_of\_page\_fault, (float)number\_of\_page\_fault / number\_of\_streams \* 100);

// free\_pagetable(pt6);

pte\* pt7 = init\_circular\_table(number\_of\_page\_frame);

pte\* clock2 = pt7;

number\_of\_page\_fault = 0;

printf(" ESC\n\n page fault | stream | modify bit | page table\n");

fprintf(fp, " ESC\n\n page fault | stream | modify bit | page table\n");

for (int i = 0; i < number\_of\_streams; i++) {

clock2 = replace\_esc(pt7, clock2, get\_stream(streams, i)->page\_num, get\_stream(streams, i)->modify\_bit, &number\_of\_page\_fault, fp);

printf(" %2d | %d | ", get\_stream(streams, i)->page\_num, get\_stream(streams, i)->modify\_bit);

fprintf(fp, " %2d | %d | ", get\_stream(streams, i)->page\_num, get\_stream(streams, i)->modify\_bit);

print\_esc\_table(pt7, fp);

printf("\n");

fprintf(fp, "\n");

}

printf("Page Fault : %d\nRate : %.2f\n\n\n", number\_of\_page\_fault, (float)number\_of\_page\_fault / number\_of\_streams \* 100);

fprintf(fp, "Page Fault : %d\nRate : %.2f\n\n\n", number\_of\_page\_fault, (float)number\_of\_page\_fault / number\_of\_streams \* 100);

// free\_pagetable(pt7);

}else {

printf("1번 문항 문제\n");

exit(-1);

}

}

fclose(fp);

free\_streams(streams);

printf("D. 종료\n");

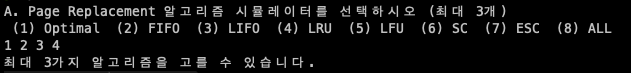
}

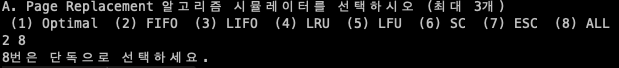
사용자 매뉴얼

텍스트이(가) 표시된 사진

자동 생성된 설명

A에서 최대 3개의 알고리즘을 고르거나 8번과 다른 번호를 같이 고를 수 없도록 하였습니다.





B에서 페이지 테이블 엔트리의 개수를 정할 수 있도록 하였으며 3보다 작거나 10보다 큰 경우 오류 메세지를 표시하도록 하였습니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

C에서 스트림을 랜덤으로 생성할 것인지 파일 형태로 입력할 것인지 결정합니다.

stallings의 책에 나온 예제에 임의의 m비트만 추가하여 나타낸 input.txt를 입력 파일로 선택하여 실행하였습니다.

해당 인풋은 , 를 통해 스트림 페이지 넘버와 수정비트를 구분하였습니다. |를 통해 한 스트림과 다음 스트림을 구분하였습니다.

텍스트, 스크린샷, 전자기기, 모니터이(가) 표시된 사진

자동 생성된 설명

해당 결과

텍스트이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명테이블이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명테이블이(가) 표시된 사진

자동 생성된 설명

테이블이(가) 표시된 사진

자동 생성된 설명

해당 결과는 터미널에 출력되며 또한 output(n).txt의 이름으로 저장됩니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

FIFO 알고리즘은 새로 들어온 스트림이 가장 앞부분이 오도록 구현이 되어 있어 페이지 테이블을 출력하였을 때 오래된 엔트리가 오른쪽(뒤쪽)으로 밀려서 표현됩니다.

LRU 알고리즘은 사용된 엔트리를 왼쪽(앞쪽)으로 가져오도록 표현됩니다.

LFU 알고리즘은 많이 사용될수록 왼쪽(앞쪽)으로 가져오도록 표현됩니다.