◆ 자율진도표

| 단계 | 완료여부 |
|---|---|
| 각 프로세스별 하나의 page table 개념 및 구조체 학습 | 0 |
| 페이지와 프레임의 개념 학습 | 0 |
| page fault가 언제 일어나는지 및 처리 과정 학습 | 0 |
| page table의 크기와 물리메모리의 크기 및 개수 학습 | 0 |
| 3-1 과제 해결 | 0 |
| hierarchical table에서 각 level별 page table의 역할 학습 | 0 |
| 각 level별 page table의 페이지가 어떤 물리메모리를 어떻게 갖는지 학습 | 0 |
| 각 level 별 어떠한 형태로 pagetable이 만들어지는지 학습(동적할당 또는 배열 사용) | 0 |
| 3-2과제 수행 | 0 |

◆ 각 주요 부분 설명

[3-1]

-load_process ():

프로세스 정보를 로드하여 프로세스 구조체에 저장한다.

pid, references(프로세스가 참조하는 페이지 번호 배열), ref_len(페이지 번호 배열의 길이)가 프로세스 구조체에 저장되고 각 프로세스의 페이지 테이블에는 초기에 모든 페이지가 무효 상태로 설정된다.

-print_page_table ():

각 프로세스의 페이지 테이블을 순회하면서 프로세스의 페이지 테이블 정보(할당된 프레임 수, 페이지 폴트 수, 참조 횟수)를 출력한다.

또한, 각 페이지 테이블의 엔트리에 대한 정보를 출력한다.

-page_fault_handler():

page table이 invalid한 경우 page fault가 발생된다. page table의 정보를 업데이트하고, 물리 메모리의 frame에 page를 매핑한다.

-execute_processes() :

각 프로세스가 참조할 페이지를 순회하면서 page table이 valid한지 invalid한지 확인하면서 invalid하면 page_fault_handler를 발생시키고 valid하면 참조 횟수를 증가시키면서 page table을 완성한다.

[3-2]

-page_fault_handler ():

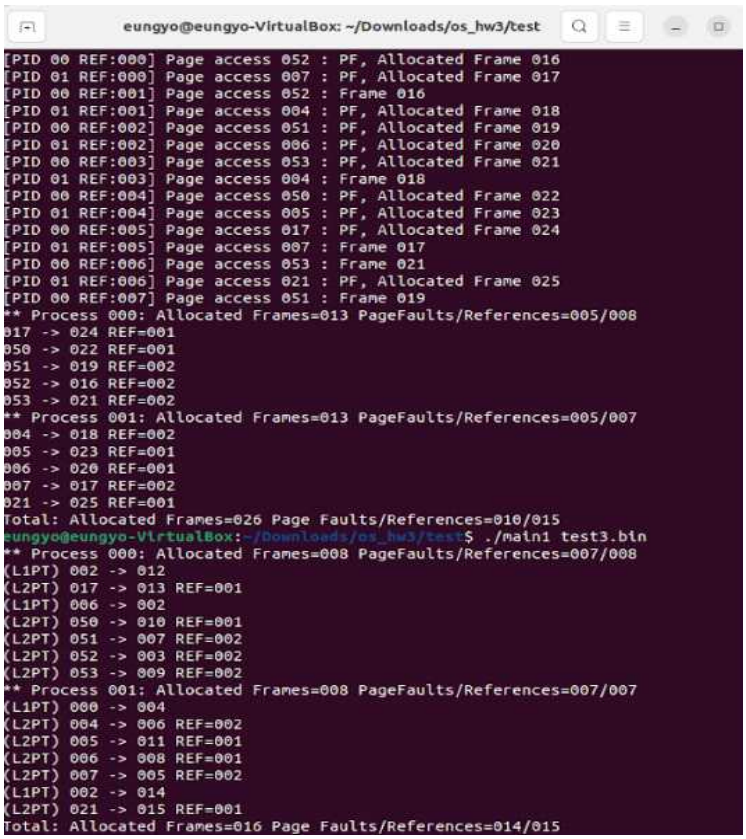
먼저 level1 page table에서 invalid한 경우에 level1 page table entry의 프레임을 할당해주고 level2 page table이 저장된다. level2 page table의 프레임을 할당해주고 level1 page

table entry에 level2 page table의 frame을 매핑한다. 여기서는 page fault가 총 2번 일어난다.
level1 page table은 valid한데 level2 page table이 invalid한 경우, level2 page table의 frame을 할당해주고  level 1 page table entry에 매핑해준다. page fault가 한 번 일어난다.

-execute_processes ():
각 프로세스의 페이지 참조를 시뮬레이션하고 페이지 폴트를 처리한다.
각 프로세스의 참조 배열에서 페이지 번호를 가져와 해당 page에 해당하는 page table 1에 접근하고 page table2에서 해당 data에 접근한다.
이 과정에서 page table1과 page table2에서의 상태(valid, invalid)를 확인하고 페이지 폴트가 발생하면 page_fault_handler 함수를 호출하여 처리한다.

-read_processes ():
파일에서 프로세스 정보를 읽어들여 프로세스 구조체에 저장한다.
파일에서 pid, 참조 배열의 길이 및 참조 배열을 읽어들여 프로세스를 생성하고 저장한다.

◆ test3.bin 3-1,3-2 수행결과



◆ JOTA 제출결과

# Submission of 운영체제 과제 3-1 by os202019161

View source
Resubmit

## Execution Results

✔ ✔ ✔ ✔ ✔

- **Test case #1:** **AC** [0.003s,1.02 MB] (2/2)
- **Test case #2:** **AC** [0.003s,1.02 MB] (2/2)
- **Test case #3:** **AC** [0.005s,1.02 MB] (2/2)
- **Test case #4:** **AC** [0.003s,1.02 MB] (2/2)
- **Test case #5:** **AC** [0.003s,1.02 MB] (2/2)

**Resources:** 0.018s, 1.02 MB
**Maximum single-case runtime:** 0.005s
**Final score:** 10/10 (10.0/10 points)

# Submission of 운영체제 과제 3-2 by os202019161

View source
Resubmit

## Execution Results

✔ ✔ ✔ ✔ ✔

**Test case #1:** **AC** [0.003s,1.02 MB] (2/2)
**Test case #2:** **AC** [0.003s,1.02 MB] (2/2)
**Test case #3:** **AC** [0.003s,1.02 MB] (2/2)
**Test case #4:** **AC** [0.003s,1.02 MB] (2/2)
**Test case #5:** **AC** [0.003s,1.02 MB] (2/2)

**Resources:** 0.015s, 1.02 MB
**Maximum single-case runtime:** 0.003s
**Final score:** 10/10 (19.0/19 points)

◆ 과제수행에서 어려움을 겪었던 부분 / 해결 방안

[3-1]

-문제 상황:

1. 두 개의 프로세스가 Demand Paging을 사용하는 상황에서, 16번 프레임부터 할당되는 이유를 이해하는 데 어려움이 있었다. 1번부터 15번 프레임이 비어있음에도 불구하고, 52번 페이지가 접근할 때 1번 프레임이 아닌 16번 프레임에 할당되는 이유에 대해 의문을 가졌다.
문제의 조건에서 각 프로세스마다 하나의 페이지 테이블을 할당하기 위해 8개의 프레임이 필요하다는 것을 이해했다. 이처럼 각각의 프로세스마다 8개의 프레임이 필요한 이유를 이해하지 못하여, 처음에는 여러 프로세스가 로드될 때도 각각의 페이지 테이블이 아닌 하나의 페이지 테이블 구조체만 사용하여 여러 프로세스에 재사용하는 방식을 사용하여 어려움을 겪게 했다.
2. 처음에는 여러 개의 프로세스가 로드되더라도 하나의 페이지 테이블 구조체를 사용하였다. 각 페이지 엔트리에 추가 변수를 할당하여 어떤 프로세스가 사용 중인지 해당 변수를 통해 나타내는 방식으로 구현했다.
그러나 여러 프로세스가 동일한 페이지를 사용하고 해당 페이지가 유효(Valid) 상태라면, 단일 프로세스 번호만 저장할 수 있는 변수의 특성상 이를 제대로 나타내지 못한다는 문제점이 있었다.
배열로 여러 프로세스 번호를 저장하더라도, 페이지 엔트리에 저장된 다른 정보들도 배열로 각각의 프로세스에 대해 나타내야 하기 때문에 코드가 복잡해지고 비효율적이었다.

-해결 방법:

각 프로세스마다 페이지 테이블 구조체를 별도로 할당하는 방식으로 문제를 해결하였다. 이로써 각 프로세스의 페이지 테이블을 독립적으로 관리할 수 있게 되어, 여러 프로세스가 동일한 페이지를 사용할 때 발생하는 문제를 해결할 수 있었다.
또한, 각 프로세스마다 독립적인 페이지 테이블을 할당함으로써 코드의 가독성과 유지보수성을 향상시킬 수 있었다.

[3-2]

-문제 상황:

1. Level 2 페이지 테이블에 접근하기 위해서는 먼저 Level 1 페이지 테이블에 접근해야 한다. 그러나 유효성(Valid/Invalid) 비트 정보는 Level 2 페이지 테이블에만 들어 있어, 해당 페이지에 접근하려면 Level 2 페이지 테이블에 먼저 접근한 후 다시 Level 1 페이지 테이블로 돌아와야 하는지에 대한 의문이 들었다. 하지만 이러한 방식은 코드가 매우 복잡해질 것 같아  어려움이 있었다.
2. level1의 페이지 테이블을 구현하는 데 어려움을 겪었다. 64개의 연속된 페이지 테이블 엔트리가 필요했고, 각 엔트리에는 valid 또는 invalid 여부를 나타내는 정보도 포함되어야 했다. 처음에는 여러 정보를 담을 수 있는 구조체를 사용하여 해결하려고 생각했지만, 이러한 방식은 접근 시간이 느려질 수 있고 코드가 길어지고 가독성이 떨어질 것 같아 다른 방법을 생각했다.

-해결 방법:

1. 교수님께 질문을 드렸고, Level 1 페이지 테이블에도 해당 페이지가 유효(Valid)한지 무효(Invalid)한지를 알 수 있는 정보가 있다는 답변을 받았다.
답변을 받고 어떻게 Level 1 페이지 테이블에서 Valid/Invalid 정보를 나타낼지 고민했다.
이를 해결하기 위해 Level 1 페이지 테이블 배열을 모두 -1로 초기화하여 무효(Invalid) 상태를 나타내기로 했다. 이는 Level 2 페이지 테이블의 인덱스가 절대로 -1이 될 리가 없기 때문에 적합하다고 생각했다.
이 과정을 통해, Level 1 페이지 테이블만으로도 페이지의 유효성 여부를 확인할 수 있게 되었고, 코드의 복잡성을 줄일 수 있었다.
2. 각 페이지에 바로 접근할 수 있도록 index를 사용하여 배열에 페이지 테이블을 구현하기로 결정했다. 이 방법을 통해 특정 페이지에 직접 접근할 수 있으므로 접근 시간을 줄였다. 또한, 각 페이지의 valid 또는 invalid 여부를 나타내기 위해 배열을 -1로 초기화하는 방법을 사용했다. 이렇게 함으로써 valid한 페이지는 실제 프레임 번호로 초기화되고, invalid한 페이지는 -1로 남아 있게 된다.
이 방법을 통해 페이지 테이블을 효율적으로 구현할 수 있었고, 프로세스의 페이지에 빠르게 접근할 수 있게 되었으며 코드의 가독성도 향상되었다.