OS Team Project
중간 보고서

7팀
20192613 이동현(팀장)
20192634 정민교
20192612 이건우
20180759 박천명

목차

1.개요

1.1 프로젝트 개요 및 목표

miniOS를 사용해 팀별로 자유롭게 OS관련 개념이 들어간 주제를 선정하여 그에 맞는 시스템을 구현하고 분석한다.

1.2 주제 선정

메모리는 모든 프로그램과 데이터가 실행되고 저장되는 장소로, 이를 효율적으로 관리하는 것은 시스템의 성능과 안정성을 보장하는 데 필수적이다. 운영체제 강의를 수강하면서 대표적인 메모리 관리 방법으로 페이징 기법을 배웠다. 이는 메모리 관리를 체계적으로 구현하는 방법 중 하나로, 이를 우리 팀만의 방식으로 miniOS에서 구현해본다면 프로젝트의 목표와도 잘맞고 운영체제와 특히 메모리를 좀더 깊게 이해할 수 있다고 생각하여 이를 프로젝트 주제로 선정하게 되었다.

1.3 수행 계획

팀원	박천명	이건우	이동현	정민교
역할분담	- 프로세스 생성 - 물리 메모리 주소 할당	- 프로세스 생성 - 프로세스별 가상 메모리 주소 할당	- 프로세스 생성 - page table 생성	- 프로세스 생성 - 메인 메모리 생성
모임일자	발표내용			
2024-04-18	- 공부 및 과제 내용 복습 - Minios 관련 주제 토의 - 회의 일정 조율			

2024-04-23	- 7주차 과제 내용 복습 - Minios 관련 주제 구체화 - 리눅스 관련 명령어 구체적으로 공부
2024-04-30	- 프로젝트 주제 선정 Pixel 기반 경계 검출 프로그램
2024-05-09	- 프로젝트 주제에 대한 재토의 - 프로젝트 주제 최종 후보 선정 및 구체화
2024-05-13/14	- 프로젝트 주제 최종 선정 및 구체화 Minios 에서 Paging 기법 구현
2024-05-18/21	- Paging 구현 Main Memory, Virtual Memory, Page Table 구현

2. 수행과정

우리는 앞서 선정한 페이징 기법을 miniOS에서 구현하기 위해 팀 내부적으로 다음과 같은 순서로 process를 진행하기로 하였다.

1. 메인 메모리(Main Memory) 생성 :

   프로젝트에서 사용할 메인 메모리를 정의한다.

2. 프로세스 생성 및 프로세스 별 가상 메모리 주소(Virtual Address)할당 :

   사용할 프로세스를 생성하고, Paging 기법을 통해 각 Page에 Virtual Address를 부여한다.

3. 할당 된 Virtual Address 기반 프로세스 별 Page Table 생성 :

   Virtual Address를 Physical Address로 변환하기 위해 각 주소마다 Main Memory에 Frame Number를 부여하여 Page Table를 생성한다.

4. Page Table 에 의거하여 물리 메모리 주소(Physical Address) 할당 :

   Page Table의 형식에 맞춰서 각 Page를 Frame Number를 통해 실제 메모리 주소를 부여한다.

5. 변한된 물리 주소(Physical Address)를 통해 Main Memory 접근 :

   Physical Address를 통해 Main Memory로 접근하여 해당 주소에 write한다.

6. Main Memory 로 부터 저장된 주소 및 값 load :

   위에서 저장된 주소와 그에 해당하는 값을 Main Memory로부터 불러온다.

위와 같은 process를 통해 각 순서 마다의 구현 결과를 분석하고, 이를 모아서 하나의 시스템으로 개발할 수 있도록 하였다.

2.1. Main Memory

우선 2048MB의 크기로 Main Memory를 할당하여 진행하였다. 프레임의 크기는 256MB로 설정하여 Physical Memory를 8개의 Frame으로 나누었다. 프로젝트를 보다 수월하게 시작하고 실행이 잘 되는지를 먼저 확인하기 위해 256MB의 크기로 Frame을 설정하였다.

프로젝트가 어느 정도 진행이 되면 Frame의 크기를 4KB로 줄여 프로젝트를 진행할 계획이다. Frame의 크기를 4KB로 하고자하는 이유는 대부분의 운영체제에서 Page 크기로 4KB를 사용하기 때문이다.

2.2. Virtual Memory

각 Process별 Virtual Memory는 Main(Physical) Memory보다 작은 1024MB의 크기로 할당하였다. 원래는 CPU에 저장되어야 하지만, 구현의 어려움으로 인해 메모리에 임의로 할당했다. 페이지의 크기는 프레임의 크기와 동일하게 256MB로 설정하여 Virtual Memory를 4개의 Page로 나누었다.

Paging을 통해 하나의 Process의 Virtual Memory를 1byte 단위로 나누어 Virtual Address를 부여했다. 하나의 Page이자 Frame의 크기가 256MB(2^{28})이므로, 현재의 Offset은 28비트가 된다. 또한 Frame이 8개 있으므로 3비트의 Tag를 갖는다. 현재는 이렇게 31비트로 Virtual Address를 부여했지만, 위에서 언급한 대로 Frame과 Page 크기를 변경할 예정이며, 이에 따라 Virtual Address도 32비트에 맞추어 수정할 것이다.

Frame과 Page 크기를 4KB로 줄이면, 더 세밀한 메모리 관리를 시뮬레이션할 수 있으며, 이는 실제 운영체제의 메모리 관리 방식과 더 유사하게 구현될 것이다.

2.3. Page table

각각의 Virtual memory는 1024byte, Page Size는 256byte이다. 따라서 Page의 개수는 4개이다.

Process가 할당되면, Page의 개수(=4)만큼 Page Table의 Index를 통해 PageTableEntry의 Size만큼 동적메모리를 할당한다. 그 후 각각의 Page Table의 Index마다의 Valid =0 과 Frame_number = -1로 초기화를 진행해 준다. Valid는 아직 해당 페이지가 Physical Memory에 적재되지 않았기 때문에 0으로 초기화하고, Frame_number 또한 마찬가지로 Physical Memory에 적재되어 있지 않기 때문에 번호를 할당하지 않고 -1로 초기화한다.

프레임 할당(Frame allocated)

1. 사용 가능한 프레임 목록 작성
   frame_bitmap을 순회하면서 사용 가능한 프레임을 Search한다.
   frame_bitmap은 Physical Memory의 각 프레임이 사용 중인지 여부를 추적하는 배열이다. 이를 통해 값이 0인 Index를 찾아 0일 경우 free_frame배열에 추가한다.

2. 사용 가능한 프레임 검증
   free_count가 0인 경우 return -1을 한다.

3. 랜덤 프레임 선택
   rand()를 통해 Frame을 할당할 때 free_frame중 Random하게 Index를 선택한다.

4. 프레임 할당 및 반환
   선택된 프레임 번호를 frame_bitmap 배열에서 1로 설정하여 사용중임을 표시한다.

다음으로 메모리의 할당을 진행하는 경우이다.

1. Logical 페이지 ID 검증
   logical_page_id가 0부터 NUM_PAGE -1 사이에 있는 지를 확인한다. 만약 이를 벗어나는 경우 'Out of bound'라는 오류 메시지를 출력한다.

2. 프레임 할당
   frame_number를 allocate_frame()을 호출하여 사용 가능한 Frame을 찾는다. 만약 사용가능한 Frame이 없는 경우 오류를 출력한다.

3. 페이지 생성 및 초기화
   Page를 Page구조체의 크기만큼 malloc함수를 통해 동적 할당을 진행해준다. 그 후 page_id를 파라미터로 받은 logical_page_id를 부여받은 뒤, content변수에 파라미터로 받은 Content와 Page_size를 Copy한다.

4. 물리 메모리에 페이지 저장
   할당된 Frame 번호에 해당되는 Physical Memory 위치에 Page를 저장한다.

5. 가상 메모리와 Page table 업데이트
   프로세스의 가상 메모리와 페이지 테이블을 업데이트한다. 앞서 valid=0 을 1로 변경하고, frame_number = -1 을 frame_number로 업데이트한다.

2.4. 실행 결과

```
[MiniOS SSU] Hello, World!
커맨드를 입력하세요(종료:exit) : process
Available frames: 0 1 2 3 4 5 6 7
Process 1: Allocated Page 0 to Frame 5
Process 1(Page ID 0): Virtual Address [0x0000 - 0x00ff]
Available frames: 0 1 2 3 4 6 7
Process 1: Allocated Page 1 to Frame 2
Process 1(Page ID 1): Virtual Address [0x0100 - 0x01ff]

Process 1 Page Table:
Logical Page 0 -> Frame 5
Logical Page 1 -> Frame 2

Available frames: 0 1 3 4 6 7
Process 2: Allocated Page 0 to Frame 7
Process 2(Page ID 0): Virtual Address [0x0000 - 0x00ff]
Available frames: 0 1 3 4 6
Process 2: Allocated Page 1 to Frame 1
Process 2(Page ID 1): Virtual Address [0x0100 - 0x01ff]

Process 2 Page Table:
Logical Page 0 -> Frame 7
Logical Page 1 -> Frame 1
```

[그림 1] 실행 결과

그림 **1**은 지금까지 구현한 Paging 결과를 보여준다. 이 그림에서 Frame이 8개로 나뉜 것을 확인할 수 있다. Page 크기에 맞게 Virtual Address가 부여되며, Page Table을 통해 Virtual Memory의 Page가 Frame으로 Mapping 되어 Physical Memory에 저장되는 것을 볼 수 있다.

현재까지 구현한 결과에 따르면, Frame이 랜덤으로 할당되며 하나의 Frame이 할당되면 나머지 Frame들은 사용 가능한 Frame(Available Frame)이 되며, 이들 중 하나가 다시 Page와 Mapping 된다. 이러한 방식으로 Page Table을 통해 Virtual Memory와 Physical Memory 간의 Mapping이 이루어진다.

3. 앞으로의 진행 계획

1. Page Table을 사용해서 얻은 각 Page의 Virtual Address마다 Frame Number를 실제 메모리 주소로 변환하는 작업

2. 지금까지 간단한 프로세스를 통해 검증을 했지만, 프로세스의 내용을 팀 의논을 통해 채워서 다시 재구성하는 작업

3. Paging 기법을 통해 얻은 실제 주소값을 이전에 정의했던 메인 메모리와 연결하는 작업

4. 메인 메모리에서 데이터를 로드하고, 데이터를 메인 메모리에 쓰는 기능을 가진 함수를 구현하는 작업

5. 지금까지 진행했던 전체 시스템을 연결시켜서 한개의 kernel(kernel.c)에서 실행 및 결과 분석하는 작업