7장 (메모리 관리)

- Linker는 모듈을 함께 묶어서 실행파일을 생성하는 역할이고 loader는 프로그램을 적재한다. 이 두 기능이 재배치 가능주소를 절대주소로 바인딩 시켜준다.

- 주소 바인딩이란 명령어/데이터 주소를 물리적 주소로 바인딩한다는 것을 의미한다

- 주소 바인딩 시점은 1. 컴파일시 2. 적재시 3. 실행시

- 외부 단편화란 남아있는 공간들을 다 합쳤을 때 충분한 공간이 되지만 그것들이 작은 조각들로 분산되어 있을 때를 의미하고 내부 단편화는 할당이 정수배로 이루어지기 때문에 할당 받았지만 사용하지 않는 부분을 뜻한다.

- 라이브러리를 부르는 곳마다 스텁이 생긴다. 이 스텁은 메모리에 라이브러리가 있는지 검사하고 없다면 디스크에서 꺼내온다. 스텁은 자신을 그 라이브러리 루틴의 번지로 대체한다. 다음번에 다시한번 그 부분이 호출되면 동적연결을 할 필요없이 직접 그곳의 라이브러리 루틴을 수행하면 된다

연속적 메모리 할당:

- 서로 다른 크기의 세그먼트들에 대해 메모리를 올리고 내리는 작업을 반복하면 외부 단편화가 발생한다. 내부단편화는 발생하지 않는다.
- 세그먼트를 통해 프로세스 사이의 코드 공유가 가능하다 .
- 외부 단편화가 발생한다. 주소공간이 연속적으로 되어 있어서 내부 단편화는 발생하지 않는다. 또한 프로세스의 가상 메모리 때문에 프로세스 사이의 코드 공유가 불가능하다.
- 순수 페이징 기법(가상메모리를 크기가 같은 블록으로 분할): 연속된 공간을 할당하지 않아도 된다.
- 외부 단편화는 발생하지 않는 대신 내부 단편화가 발생한다.
- page 를 통해서 프로세스 사이의 코드 공유가 가능하다 .
- 순수 세그먼테이션(가상메모리를 서로 크기가 다른 논리적 단위인 세그먼트로 분할, 각각의 세그먼트들은 연속적인 공간에 저장 되어있다 .
- 메모리 보호를 위해 page table entry에 각각 보호비트와 유효비트를 추가하였다 .

- 물리 메모리는 프레임이라 불리는 같은 크기 블록으로 나뉘어 지고 논리 메모리는 페이지라고 불리는 같은 크기의 블록으로 나뉘어 진다. 프로세스가 실행될 때 그 프로세스의 페이지는 파일 시스템 또는 예비 저장장치의 주 메모리 프레임으로 적재된다.

- 기술의 발전으로 저장공간이 커짐에 따라 페이지 테이블도 커지게 되어 관리가 어려워졌다 따라서 계층적 페이징 방법. 즉 페이지테이블을 다시 여러 개 페이지 테이블로 나누는 기법을 사용하여 개선하였다. 단점은 각 단계가 높아지면 높아질수록 각 논리주소를 사상하기 위해 너무 많은 메모리 접근을 한다는 것이 단점이다. 이로 인해 해시 페이지 테이블방식이 제안되었다.

- CPU 하드웨어에서 사용자공간에서 발생한 주소를 base register와 limit register를 사용하여 유

효한지를 검사한다.

- 세그먼테이션 논리주소는 < segment number , offset> 으로 구성된다. 하지만 이러한 2차원 주소는 메모리가 1차원 구조이기 때문에 사상 되어야 한다. 이 사상은 세그먼트테이블이 관리하며 세그먼트 base 와 limit을 가지고 있다

8장 (가상 메모리)

- 가상메모리란 실제 물리 메모리 개념과 논리 메모리 개념을 분리한 것이다. 장점은 작은 메모리를 가지고도 큰 가상 주소 공간을 제공할 수 있다는 것이다.

- 두 개의 프로세스들이 같은 프로그램의 값에 접근할 때 읽기용으로 물리적 페이지들을 두 개의 가상 메모리 공간으로 맵핑하여 준다. 실제적으로 쓰기가 일어나게 되면 쓰기 된 영역을 따로 할당하여 독립적으로 맵핑하여 준다. 이를 통해 라이브러리나 프로그램을 사용하는 프로세스들이 공통된 영역을 공유함으로서 메모리 공간을 작고 효율적으로 사용할 수 있다. 읽기용으로 열린 페이지들을 하드웨어적으로 검사하여 페이지 테이블에 표시한다. 그런 읽기용 페이지에 쓰기를 할 경우 trap을 일으키고 그 trap은 운영체제가 잡아 새로운 Copy-on-Write되는 영역을 잡아줌으로써 처리한다.

- 하나의 프로세스가 페이지 부재에 의해서 봉쇄형이 되어버리면 다른 스레드도 봉쇄가 되어 그 페이지가 메모리에 적재될 때까지 기다려야 한다.

부가적 참조 비트 알고리즘

- 처음에 모든 참조비트는 os에 의해 0으로 채워지고 프로세스가 실행되면서 참조되는 페이지 비트는 하드웨어가 1로 세트한다. 일정한 시간 간격마다 참조비트를 오른쪽으로 shift 시킨다. 따라서 가장 작은 수를 갖는 페이지가 교체된다.

- Thrashing은 프로세스가 페이지 프레임을 할당 받지 못하기 때문에, 발생한다. 페이지 프레임을 추가 할당하면 해결된다.

- Open() read() 등 시스템 호출을 이용하여 디스크에 있는 파일을 순차적을 읽을 때 파일이 매번 액세스 될 때마다 디스크를 접근해야 한다. 이와 같이 하는 대신 디스크 입출력을 메모리 참조 방식으로 할 수 있다. 이것을 메모리 사상이라고 한다