Xilinx Zynq FPGA, TI DSP, MCU 기반의 프로그래밍 및 회로 설계 전문가 과정

#35

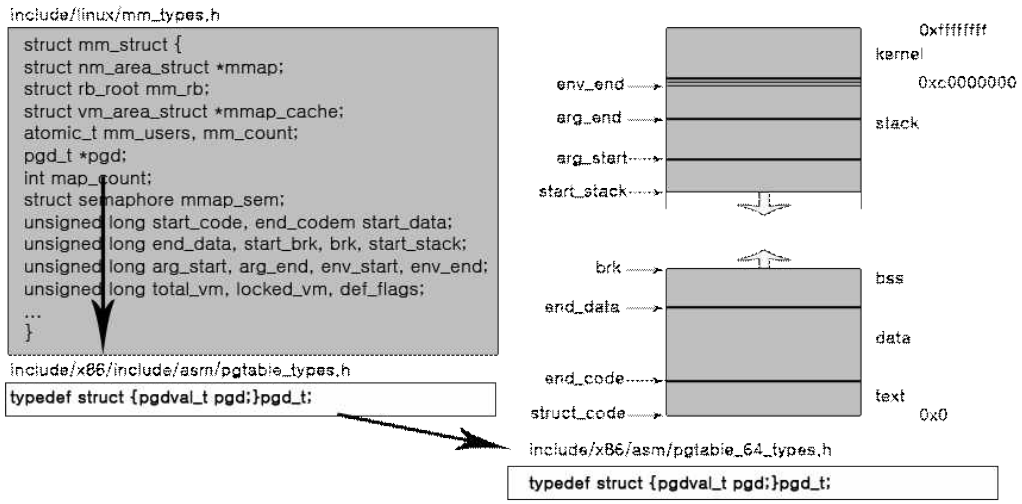
강사 : Innova Lee(이 상훈)

학생 : 김 시윤

1.배운내용 복습.

가상메모리 관리기법

태스크는 자신의 고유한 가상 메모리를 갖는다.
커널은 가상 메모리가 어디에 존재하는지 관리를 해야한다.
-> task_struct 밑에 mm_struct에 있음.



위 그림은 mm_struct의 내용을 나타낸다.

mm_struct자료구조가 관리하는 정보들은 크게 3부분으로 구분한다.

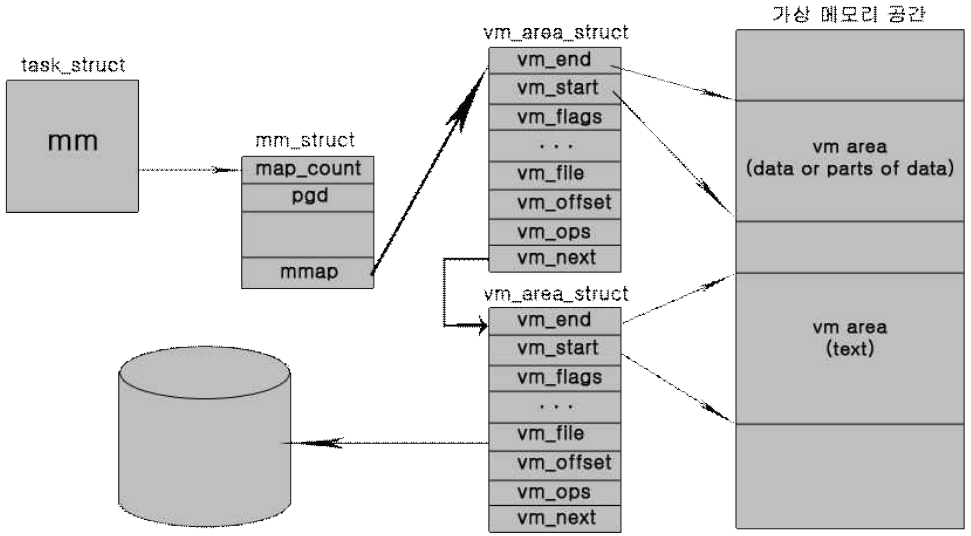
1. 태스크를 구성하고 있는 vm_area_struct구좇체들이다. 리눅스 커널은 가상 메모리 공간 중 같은 속성을 가지며 연속인 영역을 segment(메모리추상화)라 부른다. 즉 vm_area_struct는 segment를 관리한다.

vm_area_struct들은 효율적인 관리를 위해 레드-블랙 트리로 연결되어있다. 그 트리를 가르키는 변수는 위 그림에서 mm_rb이다.

mmap_cache는 최근에 접근한 vm_area_struct를 가리킨다.

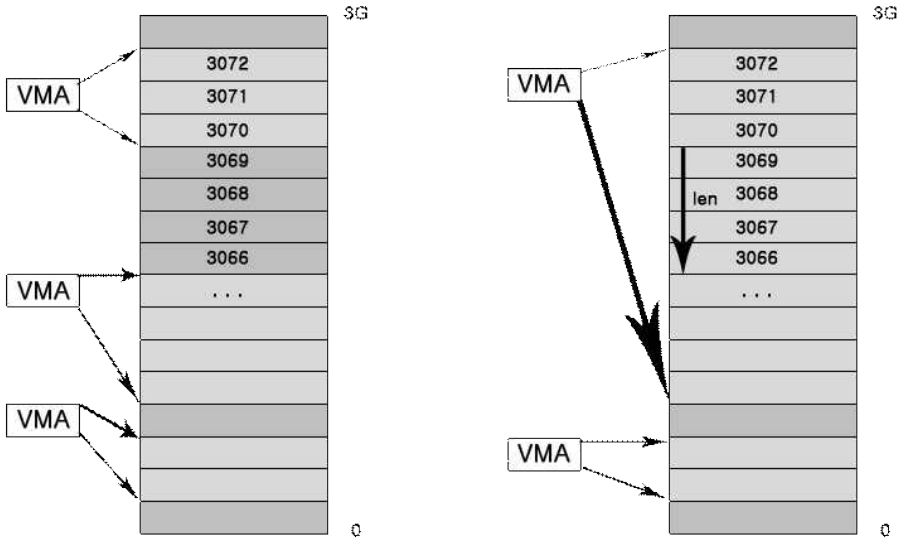
2. page_dir의 시작점 주소를 pgd라는 이름의 변수에 유지한다.

3. 가상 메모리의 구조에 대한 변수들을 갖는다.
-> start_code, start_data, start_stack 등



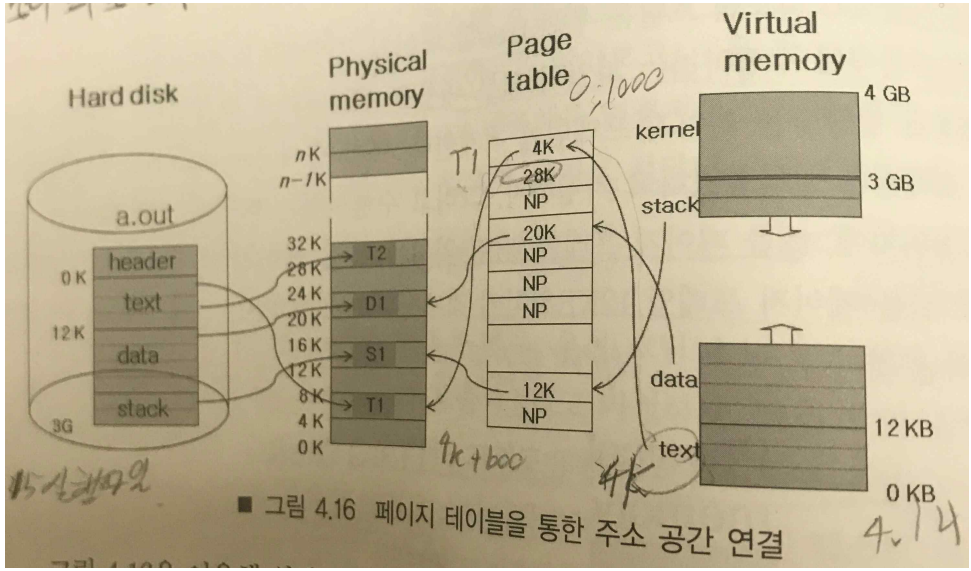
vm_area_struct를 구체적으로 나타내면 위와 같다.
세그먼트의 시작주소와 끝주소(vm_start,vm_end), 접근제어를 기록(vm_flags) 파일이 어느 위치에 있는지를 vm_file,vm_offset변수로 관리한다. 이변수들은 page fault가 났을 때 어느부분을 읽어야하는지 알려준다.

vm_area_sturct 메모리 할당 해제



네 개의 페이지 공간을 사용하려 할 때 우선 사용 가능한 가상 주소 공간을 찾아야 한다. 원하는 size를 인자로 주고 사용 가능한 연속한 가상 주소 공간을 찾는 함수는 arch_get_unmapped_area()함수다.(가상주소공간을 찾음)
위의 그림의 경우에는 3066~3069번의 페이지가 할당된다고 가정한다.
새로 할당받은 가상주소공간을 관리하는 vm_area_struct가 새로 만들어져야하는데 인접된 vm_area_struct와 속성이 갖고 연속된다면 하나로 통합되어 관리될 수 있다.
do_mmap_pgoff에서 다음에 올 VMA를 가르킨다.

가상 메모리와 물리 메모리의 연결 및 변환



페이지 테이블을 통한 주소 공간 연결

CPU가 가상주소 1000번지를 접근하려 한다고 가정했을 때 커널은 이 가상주소를 가상 메모리의 최소 단위인 페이지로 나눈다. 몫을 페이지 테이블의 index로 사용하고, 나머지는 페이지 프레임 내에서 offset으로 사용한다.
가상주소가 1000이면 몫은 0 이고 나머지는 1000이므로 page table 0번의 4KB를 가리키고 물리메모리 4KB에 접근하여 1000byte를 올라가면 우리가 찾는 데이터가 들어있다 즉 물리메모리 5096byte위치에 데이터가 들어있다.
이런식으로 페이지 테이블을 통한 주소 공간이 연결된다.

파일 시스템

파일시스템은 보조 기억 장치라고 불리는 저장장치를 관리하는 소프트웨어이다.

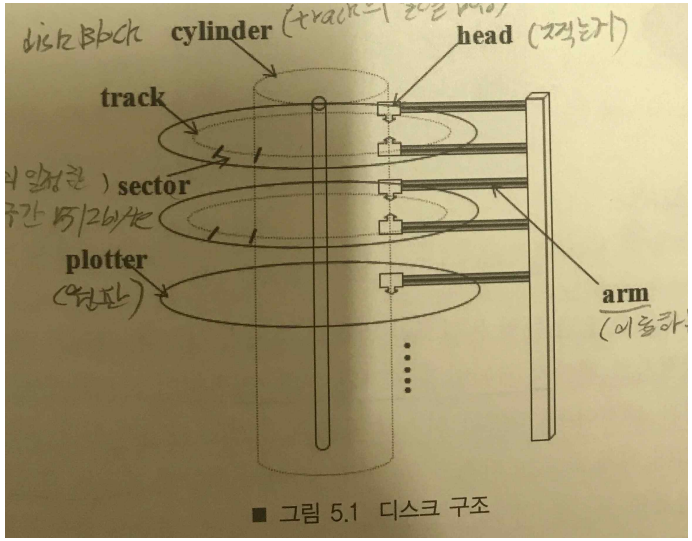
메모리 관리 기법과 파일시스템간의 차이점은 이름이라는 특성과 가상파일시스템 두 개가 추가되었다는 차이점이 있다.

즉 파일시스템은 (메모리 관리 기법 + 이름 + 가상파일시스템)이다.

하드디스크에 저장하는 정보는 크게 메타데이터와 사용자데이터 로 나뉜다.
메타데이터는 사용자 속상 정보나 데이터 블록 인덱스 정보등이 해당되며,
사용자가 실제 기록했던 데이터가 사용자 데이터를 뜻한다.

"12월 9일 토요일 오후 3시"라는 문자열로 구성된 약속 시간을 저장하려 한다고 가정해보면 파일시스템은 디스크블록을 하나 할당받아서 기록한다.
나중에 사용자가 약속이 언제였는지 확인하기위해 이름이라는 특성이 없으면 저장해둔 디스크 블록 번호를 사용자가 외우고 있어야한다.
하지만 약속.txt라는 이름을 통해 추상적인 자원인 파일을 접근하면 사용자가 접근하기 쉬울 것이다.

디스크 구조와 블록 관리 기법



■ 그림 5.1 디스크 구조

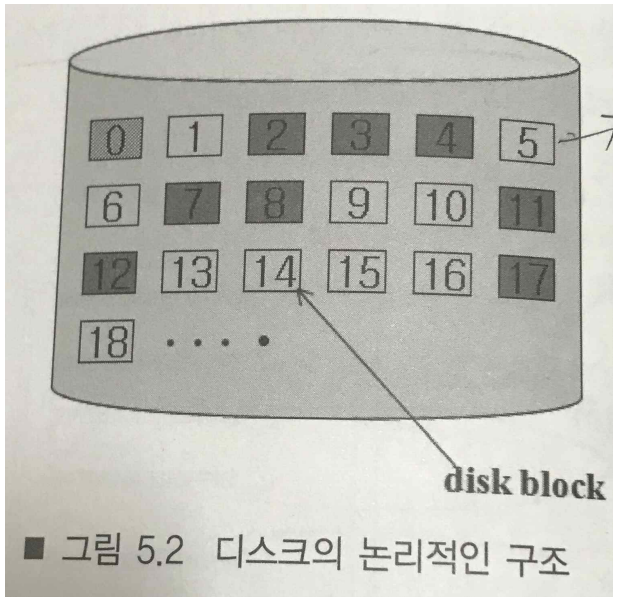
디스크 구조

디스크는 원판(plotter) , 팔(arm), 헤드(head) 으로 구성되어있으며 원판에 원 모양의 트랙(track)들이 존재한고, 트랙들의 집합을 cylinder라 한다. 트랙은 몇 개의 섹터(sector)로 구성되며 일반적으로 섹터의 크기는 512byte이다.

디스크에서 데이터를 접근하는 데 걸리는 시간을 탐색시간(seek time), 회전지연시간(rotational latency) 그리고 데이터 전송 시간(transmission time)이라는 세가지로 구성된다. 탐색시간이란 헤드를 요청한 데이터가 존재하는 트랙 위치까지 이동하는데 걸리는시간, 회전시간은 요청한 섹터가 헤드 아래로 위치될 때까지 디스크 원판을 회전시키는 데 걸리는 시간, 마지막으로 데이터 전송 시간은 헤드가 섹터의 내용을 읽거나 또는 기록하는데 걸리는 시간이다.

여기서 탐색시간이 제일 느리다

하지만 파일시스템은 디스크를 물리적인 구조로 보지 않고 디스크를 논리적인 디스크 블록(disk block)들의 집합으로 본다.



■ 그림 5.2 디스크의 논리적인 구조

disk block의 크기는 4KB이다.(하지만 최근 개발되는 시스템은 속도향상을 이유로 4K x 2^n을 기본단위로 갖는 경우도 있다 , 대표적으로 군시설 장비)
디시크 블록의 크기가 4KB이고 섹터의 크기가 512byte라면, 하나의 디스크 블록에 8개의 섹터 내용이 읽혀지게 된다.
위의 그림을 보고 예를 들면 새깔이 칠해진 디스크 블록들은 현재 사용 중이며, 색깔이 칠해지지 않은 블록들은 현재 사용 중이 아닌 상태라고 가정한다.

이와 같은 환경에서 파일 시스템이 14KB크기의 파일 생성 요청을 받았다고 하면 데이터를 할당하는 방법은 두가지다.

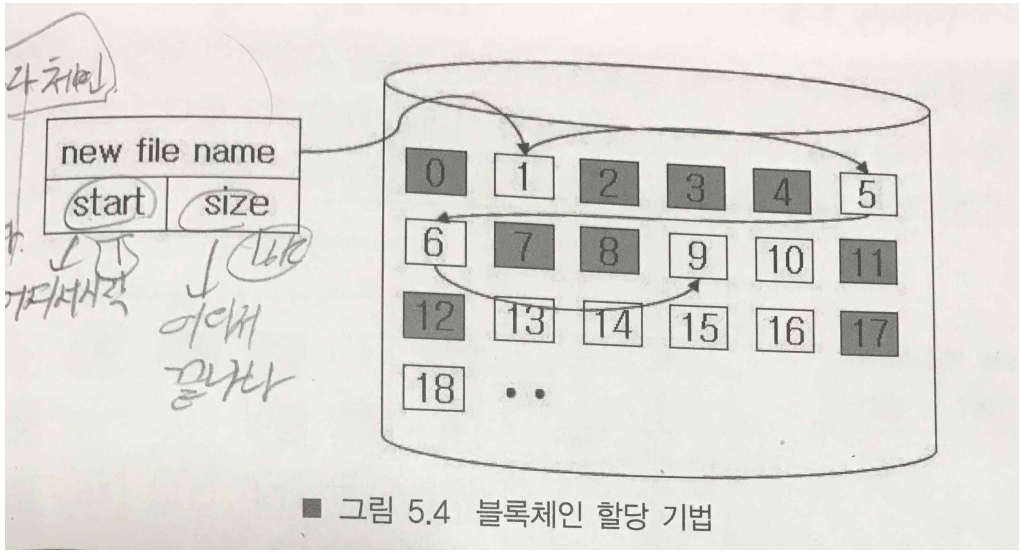
1. 연속 할당 방법
14KB의 데이터를 요청했을 경우 diskblock 한 개에 4KB이므로 총 4개의 연속된 diskblock을 할당해 줄수 있다. 즉 위에서 13,14,15,16 번을 할당하는 방법을 말한다.

2. 불연속 할당 방법
위의 diskblock에서는 1,5,6,9를 할당하는 방법을 말한다.
이렇게 할당했을때는 하나의 diskblock이 다음에 올 diskblock을 가르켜야 한다.

연속 할당 방법이였을 경우에는 불연속 할당에 비해 파일을 읽는 속도가 빠르다. 하지만 디스크 용량의 효율성을 떨어진다.

여기서 불연속 할당 방법은 블록체인 기법, 인덱스 블록 기법, FAT 기법 세가지가 존재한다.

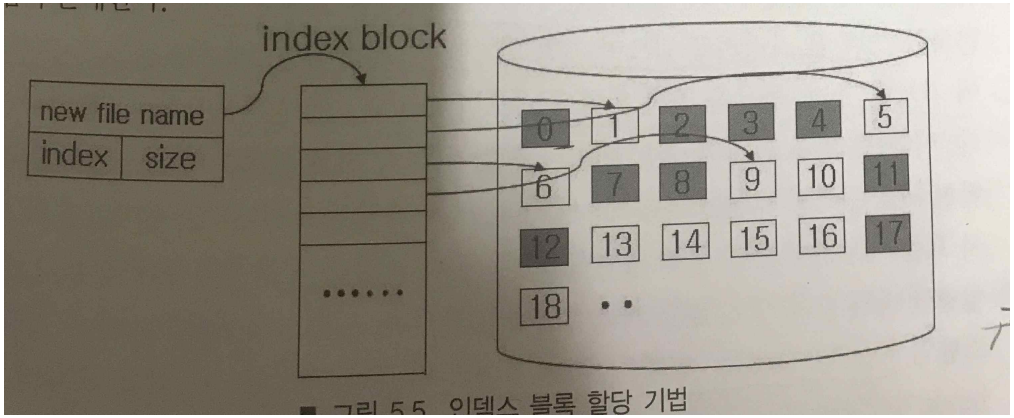
1.블록체인기법



■ 그림 5.4 블록체인 할당 기법

블록체인 기법은 같은 파일에 속한 디스크 블록들을 체인으로 연결해 놓는 방법이다. 즉 linked_list와 비슷하다. 체인에는 파일의 시작과 파일의 크기(끝나는 지점)이 저장되어있다. 그러나 이 경우 lseek()같은 시스템콜로 끝에서부터 읽으려는 경우에 어쩔 수 없이 앞부분의 데이터 블록을 읽어야 하며 중간 한 블록이 유실된 경우 나머지 데이터까지 모두 잃게 된다.

2. Index block 할당 기법

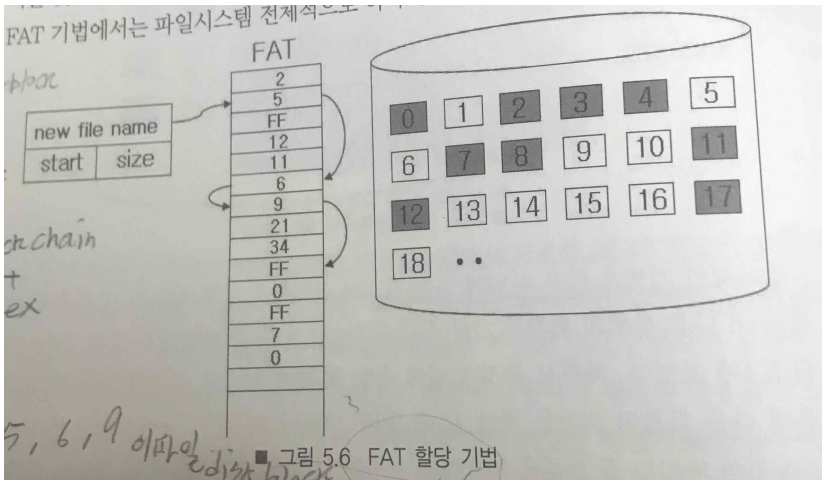


■ 그림 5.5 인덱스 블록 할당 기법

인덱스 블록 기법은 블록들에 대한 위치 정보들을 기록한 인덱스 블록을 따로 사용하는 방법이다.
lseek()를 사용하여 파일의 끝 부분을 접근하는 경우에 블록체인과 같은 단점은 없지만 만약 인덱스 블록이 유실되며 파일의 데이터 전체가 소실된다는 단점이 있다.

ex) index 0번이 날아가게 되면 다음을 찾을수 없다.

3.FAT 기법



■ 그림 5.6 FAT 할당 기법

FAT 기법은 blockchain + index 기법의 합작이다.
FF는 파일의 끝을 의미하며 0은 free상태를 나타낸다. FAT구조 1번 블록을 읽어보면, 다음 데이터 블록인 5번을 가리키고 있으며, 5번 위치에 가보면 6번 데이터 블록을 6번위치에 가보면 9번 9번위치에 가보면 파일의 끝임을 나타내는 FF가 쓰여있다. 중간에 파일이 유실되어도 start 와 size 다음에 올 블록의 데이터를 갖고있기 때문에 복원이 가능하다.
(## 언제나 루트파일 시스템은 2번에 걸림)

가상 파일시스템(Virtual File System)

가상 파일시스템은 서로 다른 파일시스템에서 read write open 같은 동일한 함수로 파일시스템을 이용하기 위해 도입된 방법이다.

가상 파일시스템이 존재하지 않는다면 msdos 파일시스템에서 ext2파일시스템으로 파일을 읽어서 쓸 경우 msdos 함수와 ext2함수를 따로따로 써줘야하는 불편함이 있다. 이것을 가상파일시스템으로 해결할 수 있다.

여기서 중요한 개념은 가상 파일시스템에 존재하는 구조체와 파일시스템에 들어있는 superblock이다.

file_struct 안에 path inode file de entry superblock 을알아야한다.

가상파일 시스템의 구조체에 모든 파일시스템의 인자들을 갖고 있다.
수퍼블록으로부터 그 인자를 받아 읽어 함수포인터로 그 값에대한 함수를 실행시켜 일반 open write read 와 같은 방식으로 실행할 수 있게 해준다.

(너무 졸려서 다음에 제대로 정리하겠습니다.)