**1. 프로그램의 원리**

**1. myfs 파일**

MY 파일시스템의 정보들은 전부 myfs 파일에 저장함.

**1) myfs 파일의 구조**boot block, super block, inode list, data block으로 구성됨.

boot block -> 16 비트 (2 바이트).  
super block -> 128 + 256 비트 (12 바이트).  
inode list -> 20 바이트(설계 시 정한 값) \* 128 (2560 바이트).  
data block -> 256 바이트 \* 256 (65536 바이트).

inode와 data block 각각은 번호를 가짐. (inode는 1~128, data block은 1~256.)  
inode와 data block에 접근할 때는 해당되는 번호를 사용함.

**2) inode list의 구조**파일 종류, 파일 생성 날짜, 파일 크기, data block 정보로 구성된 inode들의 집합으로 구성됨.

파일 종류 -> 1 바이트.  
 -> 0인 경우 디렉토리, 1인 경우 일반 파일.

파일 생성 날짜 -> 6 바이트.  
 -> 연, 월, 일, 시, 분, 초를 각각 1바이트씩 사용하여 정수로 저장함.

파일 크기 -> 4바이트.  
 -> 파일의 크기(바이트 단위).

data block 정보 -> 9바이트.  
 -> 1~8번째 바이트에는 direct 블럭 번호와 관련된 정수를 저장.  
 -> 9번째 바이트에는 single indirect 블럭 번호와 관련된 정수를 저장.  
 -> 해당 정수에 1을 더한 것이 블럭 번호임.  
 -> 저장될 내용이 없는 공간에는 정수 0이 저장되어 있음.

파일의 크기가 커지면 파일의 내용은 여러 개의 블럭으로 나누어 저장함.  
direct를 우선 전부 사용하고, 공간이 부족한 경우 single indirect를 사용함.  
indirect의 data block에는 최대 8개의 data block 번호가 저장되어 있음.

**2. myfs 파일 관리**

이 프로그램의 핵심은 myfs 파일의 정보들을 다루는 것임.  
myfs 파일은 이진 파일로 함. (그 이유는 본 페이지 아래쪽에 적어 놨음.)  
myfs 파일의 정보를 다루기 위해 파일 위치 지시자, 이진 파일 관련 함수, 구조체를 사용함.

**1) myfs 파일 관리 과정**1. 파일 위치 지시자를 통해 파일 내에서 특정 위치로 이동한다.  
2. 이진 파일 관련 함수로 해당 위치에서 데이터를 입출력함.

데이터 입출력에는 구조체를 사용함.

**2) 구조체의 사용**파일의 데이터 별 구조체를 만들어서 파일 내의 데이터를 통째로 읽고 저장함.

구조체를 사용하는 이유.  
1. 파일 위치 지시자 사용 시 편리하고, 실수를 줄일 수 있음.  
2. fseek() 함수는 바이트 단위로만 이동이 가능해서 비트 단위로 수정한 데이터는 반영하기 까다로움.

super block, inode list는 각각의 데이터를 저장할 수 있는 구조체를 사용함.  
boot block은 다루지 않으므로 구조체를 사용하지 않음.  
data block은 그 크기가 비교적 크므로 구조체를 사용하지 않음.

super block에 사용하는 구조체는 super block 전체를 저장할 수 있는 구조체임.  
inode list에 사용하는 구조체는 inode list에서 하나의 inode를 저장할 수 있는 구조체임.

--------------------------------------------------------------------------------------------------------------------------

**+ 이진 파일의 필요성**아래와 같은 이유들 때문에 myfs 파일을 이진 파일로 함.  
1. 텍스트 파일의 경우 파일 내에서 데이터의 위치가 고정적이지 않기 때문에 필요한 데이터를 읽기 어려움.  
(특히 inode list에서.)  
2. 텍스트 파일의 경우 읽은 값을 매번 사용할 자료형으로 변환해야 함.

**+ 이진 파일의 확장자**.bin 으로 함.

**2. 구체적 설계**

<프로그램 코드 설계도>

텍스트, 영수증이(가) 표시된 사진

자동 생성된 설명

**1. 함수 단위 설계도**

전체 프로그램의 코드 상 구조를 함수 단위로 구성한 설계도. (위의 <프로그램 코드 설계도> 참고.)

각 함수 별 세부적 구현 방법은 담당자가 관련 내용을 자세히 확인한 후 직접 정해야 함.  
이 과정에서 설계는 팀원들 간 논의를 거쳐 수정될 수 있음.

myfs 파일은 각 함수에서 개별적으로 열어서 사용/처리함.

**2. 파일 구조**

myfs 파일과 프로그램 파일로 나뉨.

프로그램 파일은 사용자 정의 헤더파일과 각 팀원에 해당하는 소스 파일로 나뉨.

기본적으로 각 팀원은 자신의 파일 내에서 코드를 작성하고, 헤더파일이나 다른 사람의 파일을 수정할 때는 다른 팀원들과 논의를 거쳐야 함.

**3. 디버깅 관련 코드 추가.**

본 프로그램에서는 많은 버그가 발생할 것으로 보임.  
이를 대비하기 위해 각 함수 구현 시 버그 방지를 위한 코드를 추가해야 함.

함수 작성 시 진단 코드로 예외 처리하기.  
static, const 등 사용하기.

--------------------------------------------------------------------------------------------------------------------------

**+ 공통적으로 사용될 것 같은 기능**  
파일 구현 중 다른 함수에서도 공통적으로 사용될 것 같은 기능을 발견했으면 팀원들 간 논의를 거쳐 함수로 만들어야 함.

(ex. inode 번호를 인자에 지정하면 해당 구조체 값을 리턴하는 함수.  
super block에 대한 구조체 값을 리턴하는 함수.  
등등...)

**4. 함수 인자 작성/전달 방법**

구현하려는 명령어가 필요로 하는 최대 인자의 개수와, 함수 인자의 개수를 맞춤.  
명령어에서 인자가 작성된 경우 char \* 포인터가 전달되고, 인자가 작성되지 않은 경우 NULL 값이 전달됨.

char \* 포인터에는 작성된 인자가 문자열로 저장되어 있음.

**5. 디렉토리 관련 세부사항**

**1) 디렉토리 파일 구조**  
디렉토리에 저장되는 파일 목록에서 파일 하나는 9바이트의 크기를 가짐.

파일명 8바이트 + inode 1바이트

파일명 끝에는 NULL 문자를 붙임.  
파일명은 각 글자 7바이트와 NULL 문자 1바이트로, 최대 8바이트를 가지기 때문에 8바이트로 함.

**2) 디렉토리 처리 시 유의점**.과 . .파일도 디렉토리 안에 들어있어야 함.

. 과 . .은 이름이 각각 . 과 . .인 파일임.  
. 은 작업 디렉토리의 inode 번호를 가지고, . . 은 부모 디렉토리의 inode 번호를 가짐.  
단, root는 .과 . . 모두 inode 번호가 작업 디렉토리의 inode 번호임.

**6. linked list 사용**

Queue 형태의 linked list를 사용하여 작업 디렉토리의 절대경로 상에 있는 디렉토리들의 이름과 inode를 저장함.

front 쪽에 상위 디렉토리를 저장하고, rear 쪽에 하위 디렉토리를 저장함.

linked list를 가리키는 front, rear 포인터는 전역변수로 정의함.

노드의 포인터들은 front->rear 방향으로 가리켜야 함.  
이름은 문자열(NULL 문자로 끝나야 됨.)로 저장하고, inode는 정수로 저장함.  
주소가 저장되어 있지 않은 next\_ptr 멤버에는 NULL을 배정해 두어야 함. (중요.)

초기 값(루트 디렉토리 관련 값)은 mymkfs 명령어에서 배정함.