Project. 3

Simple File System Implementation

Ⅰ. Introduction

Ⅱ. Motivation

Ⅲ. Concept

Ⅳ. Build environment

Ⅴ. Problem & Solution

Ⅵ. Personal Feelings

Ⅶ. Code

32143153 이대훈
32140301 권태완

Ⅰ. Introduction

파일시스템이란 하드 디스크나 CD-ROM같은 저장매체에서 파일이나 자료를 쉽고 빠르게 발견 및 접근할 수 있도록 관리하는 시스템입니다. 대용량의 메모리를 저장할 수 있는 저장매체가 보편화되면서 저장매체에는 상당히 큰 양의 파일이 저장될 것입니다. 이러한 대용량의 파일을 관리를 할 수 있어야 하는데 운영체제는 파일시스템을 이용하여 파일을 효율적으로 관리하게 됩니다. 운영체제는 이 파일시스템을 이용하여 사용자에게 읽기, 쓰기, 삭제 등의 기능을 수행할 수 있도록 합니다.

Meta Data Area (파일이름, 위치, 크기..)	Data Area (실제 데이터)

파일시스템은 사용자에게 기능을 제공하기 위해 메타영역과 데이터영역 두 가지의 추상화된 구조를 가지게 됩니다. 메타데이터 영역에는 파일이름, 파일위치, 파일크기, 파일유형 등의 정보들이 담겨 있고, 데이터 영역에는 해당 파일의 실제 데이터들이 담겨 있습니다. 메타영역은 데이터영역에 기록된 수많은 파일들의 정보들을 가지고 있어 메타영역의 접근만으로 데이터 영역에서 해당 파일의 정보를 얻어 올 수 있습니다. 실제로 윈도우 탐색기도 파일에 직접 접근하여 정보를 가져오는 것이 아니라 메타영역을 확인하여 파일의 정보를 빠르게 얻을 수 있는 것입니다.

Ⅱ. Motivation

지금까지 제작한 Multi-process execution 을 기반으로 하여 하드 디스크로부터 데이터를 받아와 파일시스템으로 데이터 정보를 관리할 수 있도록 하였습니다. 디스크의 4MB 크기의 파일 데이터를 Partition이라는 형태로 구조화시켜 파일의 데이터에 접근할 수 있도록 파일 시스템을 제작하였습니다. partition은 다음과 같은 형태로 되어 있습니다.

* Partition(4MB)

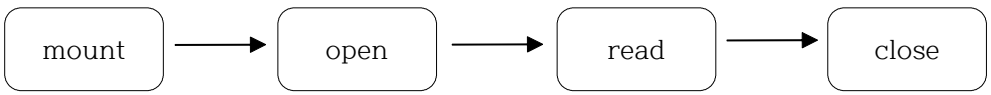
0 1KB 8KB 4MB

Super Block(1kb)	Inode_table(7kb)	Data_Block(4088kb)

Inode - Meta area에 해당되는 것으로 파일의 meta-data를 가지고 있습니다. 디스크 데이터 블락의 위치나 파일 크기, 모드 등 파일에 대한 정보를 가지고 있어 이 inode를 보고 데이터의 정보를 확인 할 수 있습니다. 하나의 inode는 32byte이고 244개(7Kb)가 모여 Inode_table이 됩니다.

Block - 파일의 실질적인 데이터에 해당되는 것으로 하나의 block은 1024byte(1kb)이고 Partition 총 4Mb중에 super_block + inode_table 8Kb를 제외한 4088Kb의 크기에 해당됩니다.

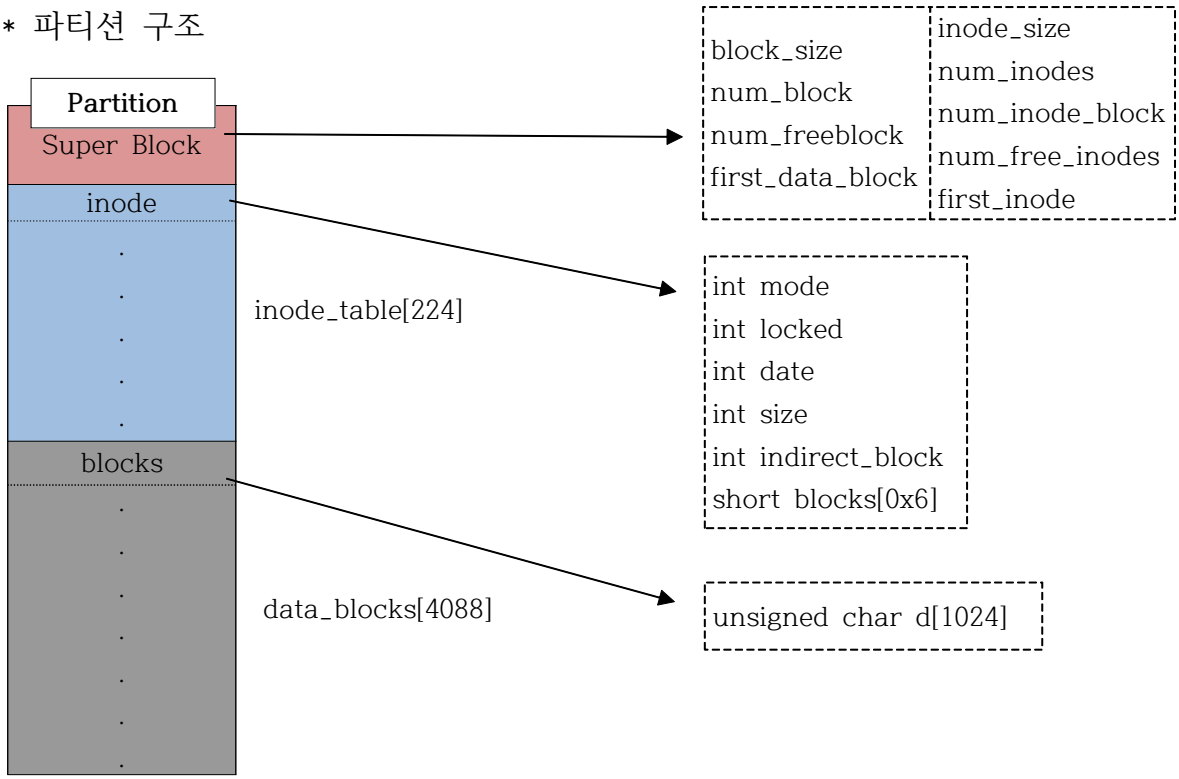
Super_Block - 파일시스템의 전반적인 정보를 담고 있는 special한 block으로 inode 사이즈, inode 개수, block 사이즈, block 개수 등을 포함하고 있습니다.



파일시스템은 위와 같이 4단계의 과정이 있는데 처음에 모든 파일을 받아오는 mount이고, mount된 파일들 중에 읽고 싶은 파일을 찾고 파일의 Inode를 알아내는 것이 open, 찾은 Inode를 통해 Inode table에서 원하는 파일의 정보를 얻는 것을 read고, 작업이 끝나면 파일 시스템을 종료해주는 것을 close라고 합니다.

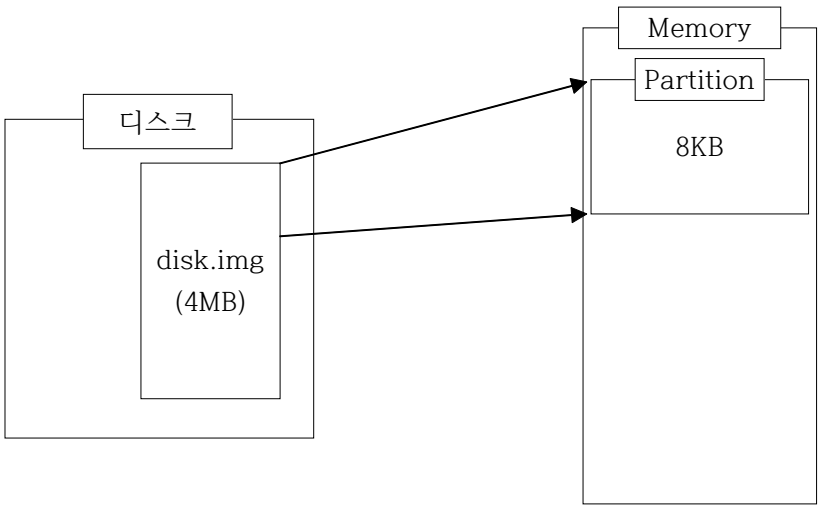
Ⅲ. Concept

* 파티션 구조



1. Mount

디스크에 저장되어 있는 파일들을 효율적으로 관리하기 위하여 파일 시스템에서 사용되는 디렉토리는 하나의 파일시스템에서 서로 연관된 파일들을 한곳에 체계적으로 저장 될 수 있도록 하는 방법입니다. 이 디렉토리와 디스크를 연결해주는 작업을 마운트라고 합니다. 결국 마운트는 디스크에 있는 모든 파일을 불러오는 작업이고 모든 파일 목록을 사용자에게 보여주는 것으로 ls -al 와 유사합니다.

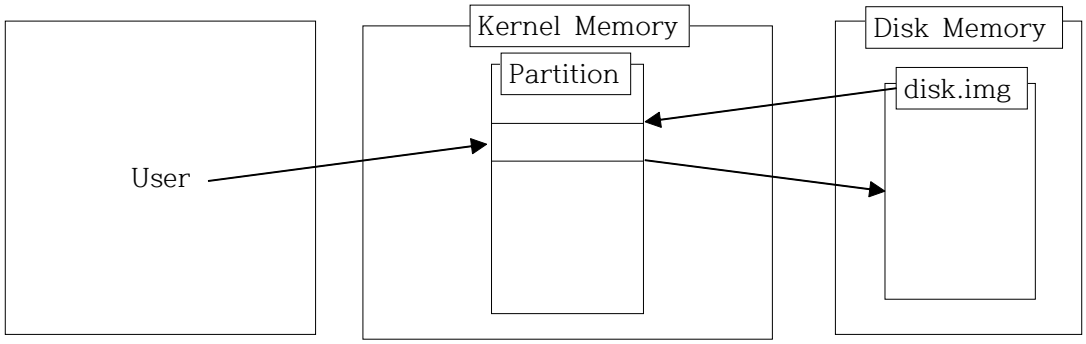


==> 4Mb 크기의 disk.img 파일을 partition라는 구조형태로 메모리에 적재를 시킵니다. 우선 partition에서 super_block안의 first_inode_number를 확인하고 Inode_table에서 first_inode_number번째의 inode(root inode)로 가서 사이즈를 확인합니다. 이 사이즈가 inode의 데이터 크기이고 이 사이즈를 보고 1024byte(1kb)블락 단위로 blocks[0x6]에 얼마나(몇 칸) 담을 수 있는지 확인할 수 있습니다. 확인해보면 사이즈는 3264이고 이는 1024로 3번(0,1,2)이 들어가고 나머지는

3번 째 칸에 들어가는 것을 알 수 있습니다. 이를 통해 0부터 3번까지의 block에 모든 파일의 목록 담겨 있는 것을 알 수 있습니다. 그리고 0x2000부터 시작되는 directory entry에서의 name이 곧 파일의 이름이 되고 0x400 크기만큼 옮겨가면서 directory entry에서 name을 위에서 구한 0~3까지 출력하면 모든 파일의 목록을 출력할 수 있습니다. 이는 곧 ls -al가 하는 일과 같습니다. 또한 mount는 os가 부팅하는 단계에서 이뤄지는 작업입니다. 그렇기 때문에 파일이 os가 실행함과 함께 kenerl에서 작업하도록 하였습니다.

2. Open

한 명의 user가 프로그램을 사용한다고 가정하고, 마운트가 되면 이제 메모리에 디스크의 데이터가 적재되었을 것이고 모든 파일의 목록을 확인 가능 할 것입니다. user가 원하는 파일을 읽으려면 inode의 blocks에서 파일의 데이터를 찾아야 하는데 파일의 inode number를 모르기 때문에 읽기 전에 파일이 디렉토리 파일에 존재하는 지 확인을 해야 하고 존재한다면 해당 파일의 inode를 알아야 합니다. 이 inode number는 해당 파일의 디렉토리 엔트리에서 확인이 가능합니다.



==> 모든 file system은 I/O burst에서 실행하게 됩니다. 이를 통해 스케줄링에서 cpu burst가 끝나고 I/O에 들어갈 때마다 Open을 하여 disk에 접근할 수 있도록 하였습니다. User process에서 file 이름을 입력받아 입력받은 file 이름을 kernel에 전달해줍니다. Msgq를 사용하여 User가 file 이름을 얻었을 때 전달해 주었습니다. 이후 kernel에서 Open을 하면 disk의 data_block의 directory file을 찾아 다시 dentry 구조체로 구분하여 strncmp를 사용하여 입력한 file 이름과 dentry의 name이 같은 것을 찾습니다. 비교하여 같은 것이 있으면 원하는 file을 찾은 것이고 없으면 directory file안에 없는 file입니다. 원하는 file을 찾은 경우 file의 meta-data를 PCB에서 기억할 수 있도록 linked inode를 만들어 주었습니다. 찾은 inode num을 이곳에 저장하고 다음번에 같은 file이 open되었을 경우에 이미 있으므로 생략하고 새로운 file이 open하면 다시 이곳에 저장합니다. 이를 통해 지금까지 open 했던 file들에 대해 정보를 가지고 있을 수 있습니다.

3. Read

파일이 open되었으면 이제 해당파일의 inode값을 알고 inode_table에서 inode번째로 가서 block에서 데이터를 찾을 수 있게 됩니다. 이 때 이전에 파일실행이 어디까지 하고 종료된 것인지를 알고 있어야 하며 blocks에서 실제로 데이터가 저장된 블락을 찾아가면 그 블락에서 원하는 파일의 데이터를 얻을 수 있습니다.

==> open해서 얻은 해당 파일의 inode를 inode_table에서 찾아가 size를 확인하고 1024byte(1kb) 단위로 몇 개의 block을 사용하고 있는지를 확인합니다. Open에서 얻은 file의 inode를 사용하여 block의 data를 읽으면 해당 file의 data를 알 수 있습니다. 이 때 disk에서 가지고 온 data block의 정보를 user buffer에 임시적으로 저장하여 user process가 사용할 수 있도록 하였습니다.

4. Close

Open 했던 file의 실행이 끝나고 더 이상 사용하지 않는 경우에 대해서 memory에 file의 data_block을 그대로 사용하는 것은 유한한 memory에 대해 비효율적인 작업입니다. 이런 점을 해결 하기위해 open 하여 read까지 마친 file은 close를 사용하여 해당 file의 data_block을 다시 disk에 돌려보냅니다. Close를 할 경우 PCB에서 저장하고 있던 file의 meta-data인 inode 또한 지워 file이 open하기전의 상태로 돌려줍니다.

* Program Structure

1. Structure & Global value initialize
2. mount
3. fork
    - 3-1. parent
        - 3-1-1. get data(pid, TTBR, timequantum, execution time)
        - 3-1-2. enqueue qready
    - 3-2. child
        - 3-2-1. child exit handler
        - 3-2-2. check(exec_time == 0)?
        - 3-2-3. get input file value
        - 3-2-4. msgsend
4. kernel
    - 4-1 SIGALRM handler
    - 4-2. msgget
    - 4-3 check wait queue empty?
    - 4-4 do_wait for I/O burst
    - 4-5 open
    - 4-6 read
    - 4-7 close
    - 4-8 check (I/O burst time == 0)?
    - 4-9 check ready queue empty?
    - 4-10 RR
    - 4-11 check (time quantum == 0), (execution time == 0)?

Ⅳ. Build environment

Compilation : Linux Assam, with GCC
Header file : msg.h, t.h, queue.h, fs.h
Input file : disk.img
To compile : gcc Filesystem.c
To run : ./a.out

Ⅳ. Problem & Solution

1. problem) Inode-size를 어떻게 구분해서 blocks[0x6]에 넣어 줄지 어려움이 있었습니다.
☞ solution) 한 블락의 크기가 1024byte(1KB)이므로 inode-size가 1024로 나눠서 떨어질 때와 그러지 않을 때 경우로 나눠서 나머지가 남지 않는 경우에는 그 몫의 값이 블락을 사용하는 개수가 될 것이고 나머지가 남는 경우에는 몫에 +1을 해준 값이 블락을 사용하는 개수가 되도록 하였습니다.

2. problem) 전 프로젝트 Multi-Process execution에서 10명의 user가 있었다면 이번 파일시스템에서는 1명의 user가 사용되는 execution이라 user를 1명으로 줄었더니 cpu_burst time이 종료되고 I/o _burst time이 시작되어야 하는 시점에서 실행이 멈추는 현상이 발생했습니다.
☞ solution) user가 보낸 메시지 큐를 kernel이 제때 받지 못하는 경우가 있어서 메시지를 받을 때 msgflg의 변수에 MSG_NOERROR에서 IPC_NOWAIT으로 바꿔줌으로서 메시지를 잘 못 받을 경우에도 기다리지 않고 진행하도록 하였습니다.

3. problem) fs.h에 있는 super block에서 first_data_block의 값이 0x8으로 설정이 되어 있어서 이 값이 의미하는 것이 data_block이 0x2000부터 시작하고 여기서 8만큼 더 간 값이 첫 data_block의 시작인 것인지 헷갈렸습니다.
☞ solution) first_data_block 0x8이 의미하는 것이 block 단위(1kb)로 봤을 때 8kb 다음부터인 0x2000부터가 data_block의 시작점이라는 것을 알려주는 것이었습니다.

4. problem) dist.image.hex파일을 읽을 때 있는 그대로 의 값을 읽었더니 예상했던 값이 나오지 않았습니다.
☞ solution) 비트를 그대로 읽으면 안 되고 비트를 메모리에 저장할 때에는 Big-edian과 Little-edian 두 가지 방식으로 저장할 수 있는데 Big-edian은 상위비트가 먼저 저장되고 Little-edian은 하위비트가 먼저 저장되는 방식으로 여기서는 Little-edian방식이 쓰이는 것을 알았습니다.

5. problem) 코딩을 하면서 disk.img.hex파일을 보면서 값을 확인하면서 진행하려고 하였지만 disk.img.hex파일에 있는 값과 실제 disk.img값을 돌렸을 때의 값이 다르게 나왔습니다.
☞ solution) 다른 사람들과 비교해본 결과 disk.img.hex파일과 disk.img의 파일이 일치하지 않다는 것을 알았습니다.

6. problem) Scheduling과 합치기 위해 file system이 언제 실행하는지에 대해 명확히 해야 했습니다. 특히 어려웠던 부분은 mount 부분이 어려웠습니다. File system이 disk에 가서 data를 memory로 가져온다고 하는 부분에 대해서 I/O 요청이 있을 때마다 실행한다고 생각하였습니다.
☞ solution) 실제로 Open, read, close 등 다른 부분들에 대해서는 I/O 요청이 있는 경우에 실행

합니다. 하지만 mount의 경우 os가 booting하기 전에 먼저 실행하는 작업으로 처음 한번 작업을 해줌으로써 가장 효율적으로 필요한 block들만 가져왔습니다.

7. problem) 처음 partition structure에 disk.img를 mount할 때 disk.img의 크기인 4MB 만큼 한번에 mount 하여 memory에 할당하였습니다. 하지만 나중에 지금과 다르게 여러 개의 disk.img를 사용하는 경우에 대해 memory 부족현상이 발생합니다.
☞ solution) 이러한 부분을 해결하기위해 data_block을 뺀 super_block과 inode_block만을 가지고 memory에 들어갑니다. Data_block에 대한 meta-data를 가지고 있는 inode_block을 가지고 있기 때문에 실제로 disk에 있는 data_block에 필요에 의해 찾아갈 수 있습니다. 이를 통해 현재 사용하는 memory의 공간도 추가로 확보 할 수 있습니다.

8. problem) User process이 어떤 file을 사용하는지 알고 open과 read의 작업은 kernel에서 하기 때문에 User process에서 kernel에 값을 전달해 주어야 합니다. 반대로 User process가 원하는 file의 meta data인 inode를 알기 위해서는 kernel에서 다시 user process에 inode를 전달해 주어야합니다.
☞ solution) 이런 부분을 해결하기 위해 msgq를 사용하여 각각에 맞는 값들을 넘겨주는 작업을 하였습니다.

Ⅴ. Personal Feelings

주어진 disk.img.hex 파일을 fs.h 헤더파일을 보고 superblock, data_block, directory_entry 구조체 마다 각자하는 역할이 무엇인지 구조체 안의 값들이 무엇을 의미하는지, disk.img.hex 파일과 비교해봐서 매치되는 값이 무엇을 뜻하는지 분석하고 이해하는데 시간이 걸렸습니다. 분석하고 나서는 mount, open, read, close 순서대로 파일을 찾아내는 logic에 따라 코딩을 하는 데는 큰 문제가 없었습니다. 그렇지만 또 이것을 이전 프로젝트들을 기반으로 작성하려고 하니 어려움이 많았습니다. 실제 운영체제에서 돌아가는 것처럼 구현하지 못한 부분도 많고 다른 것들과 겹쳐져 많이 못했던 아쉬움이 남습니다. File system은 앞으로도 많은 주제로 다루어 질수 있는 영역이라고 생각합니다. 이렇게 1학기 운영체제 수업을 마치면서 많이 힘들었지만 즐겁고 알찬 수업을 경험했다고 생각합니다.

Ⅶ. Code

```
/* signal test */
/* sigaction */
#include "t.h"
#include "queue.h"
#include "msg.h"
#include "fs.h"

QUEUE *qwait;
QUEUE *qready;
QUEUE *qinode;
FILE *stream;
```

```
int ptr;
int wptr;
int msgq;
int key = 0x32143153;
int num = 0;
int num_file = 0;
char buf[256];

struct sigaction old_sa;
struct sigaction new_sa;
struct msgbuf msg;
struct partition party;
struct dentry dentry;
struct blocks userbuf;
pcb_t PCB[MAXPROC];
user uProc;

int main()
{
        memset(&new_sa, 0, sizeof(new_sa));
        msgq = msgget( key, IPC_CREAT | 0666);
        memset(&msg, 0, sizeof(msg));
        qwait = CreateQueue();
        qready = CreateQueue();
        qinode = CreateQueue();
        srand((unsigned)time(NULL));
        mount();

        printf("---------------SuperBlock Information----------------\n");
        printf("Partition type:--------- %d\n", party.s.partition_type);
        printf("Block size:------------- %d\n", party.s.block_size);
        printf("Inode size:------------- %d\n", party.s.inode_size);
        printf("Directory file inode:--- %d\n", party.s.first_inode);
        printf("Number of inodes:------- %d\n", party.s.num_inodes);
        printf("Number of inode blocks:- %d\n", party.s.num_inode_blocks);
        printf("Number of free inodes:-- %d\n", party.s.num_free_inodes);
        printf("Number of blocks:------- %d\n", party.s.num_blocks);
        printf("Number of free blocks:-- %d\n", party.s.num_free_blocks);
        printf("First data block:------- %d\n", party.s.first_data_block);
        printf("Partition name:--------- %s\n", party.s.volume_name);
        printf("\n");

        for(int i = 0; i < 224; i++)
        {
                printf("---inode #%d---\n", i);
```

```
		printf("Mode:---------- %d\n", party.inode_table[i].mode);
		printf("Locked:--------- %d\n", party.inode_table[i].locked);
		printf("Date:----------- %d\n", party.inode_table[i].date);
		printf("File Size:------ %d\n", party.inode_table[i].size);
		printf("Indirect block:- %d\n", party.inode_table[i].indirect_block);
		printf("Blocks:--------- ");
		for(int j = 0; j < 6; j++)
		{
			printf("%d ", party.inode_table[i].blocks[j]);
		}
		printf("\n\n");
}

pid_t pid;

for(int k=0; k<MAXPROC; k++)
{
		uProc.exec_time = EXE;//(rand() % 10) + 1;
		uProc.ppid = getpid();
		uProc.IO = 1;
		pid=fork();
		if(pid<0){
			perror("fork fail");
		}
		else if(pid == 0){//child
			new_sa.sa_handler = &signal_handler2;
			sigaction(SIGUSR1, &new_sa, &old_sa);
			while(1);
			exit(0);
		}
		else{//parent
			PCB[k].pid = pid;
			PCB[k].TTBR = k;
			PCB[k].remaining_tq = DEFAULT_TQ;
			PCB[k].exec_time = uProc.exec_time;
		}
		Enqueue(qready, k);
}

new_sa.sa_handler = &signal_handler;
sigaction(SIGALRM, &new_sa, &old_sa);
new_sa.sa_handler = &signal_handler3;
sigaction(SIGUSR2, &new_sa, &old_sa);
fire(1);
```

```
        while(1);
}

void RR()
{
        if(emptyQueue(qready)){
                return;
        }
        else{
                ptr = qready->front->data;
                if(PCB[ptr].remaining_tq > 0)
                {
                        PCB[ptr].remaining_tq--;
                        PCB[ptr].exec_time--;
                        printf("Tick : %d, PID : %d\tPCB[%d].exec_time : %d\n", tick,
PCB[ptr].pid, ptr, PCB[ptr].exec_time);
                        kill(PCB[ptr].pid, SIGUSR1);
                        if(PCB[ptr].exec_time == 0)
                        {
                                PCB[ptr].remaining_tq = DEFAULT_TQ;
                                Enqueue(qwait, Dequeue(qready));
                                return;
                        }
                }
                if(PCB[ptr].remaining_tq == 0)
                {
                        PCB[ptr].remaining_tq = DEFAULT_TQ;
                        Enqueue(qready, Dequeue(qready));
                        return;
                }
        }
        return;
}

void do_wait()
{
        if(emptyQueue(qwait))
        {
                return;
        }
        else
        {
                for(int k=0; k<qwait->count; k++)
                {
                        wptr = qwait->front->data;
```

```
                        if(PCB[wptr].IO_time > 0)
                        {
                                int value = open();
                                if(value == 1)
                                {
                                        read_ker();
                                        close_ker();
                                }
                                PCB[wptr].IO_time--;
                                //printf("PCB[%d].IOtime : %d\n", wptr, PCB[wptr].IO_time);
                                if(PCB[wptr].IO_time == 0)
                                {
                                        Enqueue(qready, Dequeue(qwait));
                                }
                                else if(PCB[wptr].IO_time > 0)
                                {
                                        Enqueue(qwait, Dequeue(qwait));
                                }
                        }
                }
        }
}


void signal_handler(int signo)
{
        tick++;
        if((strncmp(PCB[0].pathname, "exit", 4)==0))
        {
                for(int k=0; k<MAXPROC; k++)
                {
                        kill(PCB[k].pid, SIGINT);
                }
                DestroyQueue(qready);
                DestroyQueue(qwait);
                printf("Tick : %d, OS exit\n", tick);
                fclose(stream);
                exit(0);
        }
        do_wait();
        RR();
}

void signal_handler2(int signo)
{
```

```
		uProc.exec_time--;
		if(uProc.exec_time == 0)
		{
			printf("File name : ");
			fgets(buf, sizeof(buf), stdin);
			buf[strlen(buf)-1] = '\0';
			memcpy(&uProc.pathname, &buf, sizeof(buf));
			uProc.exec_time = EXE;//(rand() % 10) + 1;
			msgsend();
			kill(uProc.ppid, SIGUSR2);
		}
}

void signal_handler3(int signo)
{

		msgrecieve();
		for(int k=0; k<MAXPROC; k++)
		{
			if(PCB[k].pid == msg.pid)
			{
				PCB[k].IO_time = msg.io_time;
				PCB[k].exec_time = msg.cpu_time;
				memcpy(&PCB[k].pathname, &msg.pathname,
sizeof(msg.pathname));
			// printf("mother CPU : %d IO : %d, index : %d\n",
PCB[k].exec_time, PCB[k].IO_time, k);
			}
		}
}

int msgsend()
{
		int ret = -1;
		msg.mtype = 0;
		msg.pid = getpid();
		msg.io_time = uProc.IO;
		msg.cpu_time = uProc.exec_time;
		memcpy(&msg.pathname, &uProc.pathname, sizeof(uProc.pathname));
		msg.O_RD = uProc.O_RD;
		while(ret == -1)
		{
			ret = msgsnd(msgq, &msg, sizeof(msg), 0);
		}
		return 0;
```

```
}

int msgrecieve()
{
	int ret = -1;
	while(ret == -1)
	{
		ret = msgrcv(msgq, &msg, sizeof(msg), 0, MSG_NOERROR);
	}
	return 0;
}

void fire(int interval_sec)
{
	struct itimerval new_itimer, old_itimer;
	new_itimer.it_interval.tv_sec = 0;
	new_itimer.it_interval.tv_usec = 1000;
	new_itimer.it_value.tv_sec = 0;//interval_sec;
	new_itimer.it_value.tv_usec = 1000;
	setitimer(ITIMER_REAL, &new_itimer, &old_itimer);
}

int mount()
{
	stream = fopen("disk.img", "r");
	if (stream == NULL) exit(EXIT_FAILURE);

	fread(&party, (sizeof(party.s)+sizeof(party.inode_table)) , 1, stream);

	if((party.inode_table[party.s.first_inode].size % 1024)==0)
	{
		num = party.inode_table[party.s.first_inode].size / 1024;
	}
	else
	{
		num = (party.inode_table[party.s.first_inode].size / 1024) + 1;
	}


	for(int i=0; i<num; i++)
	{
		fseek(stream, 0x2000 +(i * 0x400), SEEK_SET);
		for(int k=0; k<32; k++)
		{
			fread(&dentry, sizeof(dentry), 1, stream);
```

```
                        if(dentry.name_len == 0)return 0;

                        printf("name : %s\n", dentry.name);
                }
        }
}

int open()
{
        for(int i=0; i<num; i++)
        {
                fseek(stream, 0x2000 +(i * 0x400), SEEK_SET);
                for(int k=0; k<32; k++)
                {
                        fread(&dentry, sizeof(dentry), 1, stream);
                        if(strncmp(dentry.name, PCB[0].pathname, sizeof(PCB[0].pathname))
== 0)
                        {
                                //printf("path : %s\n", PCB[0].pathname);
                                printf("File name:-------- %s\n", dentry.name);
                                printf("Inode number:----- %d\n", dentry.inode);
                                printf("Dentry Size:------ %d\n", dentry.dir_length);
                                printf("File name length:- %d\n", dentry.name_len);
                                printf("File Type:-------- %d\n", dentry.file_type);
                                printf("Mode:-------------                              %d\n",
party.inode_table[dentry.inode].mode);
                                printf("Locked:-----------                              %d\n",
party.inode_table[dentry.inode].locked);
                                printf("Date:-------------                              %d\n",
party.inode_table[dentry.inode].date);
                                printf("File                    Size:--------          %d\n",
party.inode_table[dentry.inode].size);
                                printf("Indirect                    block:---          %d\n",
party.inode_table[dentry.inode].indirect_block);
                                printf("Blocks:----------- ");
                                for(int j = 0; j < 6; j++)
                                {
                                        printf("%d                                      ",
party.inode_table[dentry.inode].blocks[j]);
                                }
                                printf("\n");
                                for(int a=0; a<qinode->count; a++)
                                {
                                        if(qinode->front->data == dentry.inode)
                                        {
```

```
                                        return 1;
                                }
                                else
                                {
                                        Enqueue(qinode, Dequeue(qinode));
                                }
                        }
                        Enqueue(qinode, dentry.inode);
                        printf("Open file inode num : %d\n", dentry.inode);
                        return 1;
                }
        }
    }
    printf("Not Found\n");
    return 0;
}

int read_ker()
{
    if((party.inode_table[dentry.inode].size % 1024)==0)
    {
        num_file = party.inode_table[dentry.inode].size / 1024;
    }
    else
    {
        num_file = (party.inode_table[dentry.inode].size / 1024) + 1;
    }

    for(int i=0; i<num_file; i++)
    {
        fseek(stream, 0x2000 + ((party.inode_table[dentry.inode].blocks[i])*0x400),
SEEK_SET);
        fread(&userbuf, sizeof(userbuf), 1, stream);
        printf("%s\n", userbuf.d);
    }
}

int close_ker()
{
    for(int k=0; k<qinode->count; k++)
    {
        if(qinode->front->data == dentry.inode)
        {
            Dequeue(qinode);
        }
```

```
                else
                {
                        Enqueue(qinode,  Dequeue(qinode));
                }
        }
}
```