

설계과제 3 개요 : SSU-EXT2

Linux System Programming, School of CSE, Soongsil University, Spring 2025

○ 개요

- ext2 파일 시스템의 이미지 파일을 분석하여 루트 디렉토리부터 하위 경로 및 파일을 파악하는 프로그램

○ 목표

- ext2 파일 시스템의 원리를 이해하고, 다양한 시스템 자료구조, 시스템 콜 및 라이브러리를 활용하여 디렉토리 구조를 분석하는 쉘 프로그램을 작성함으로써 파일 시스템과 쉘에 대한 이해를 높이고 디렉토리 구조를 링크드 리스트로 구현함으로써 시스템 프로그래밍의 설계 및 응용 능력을 향상

○ 팀 구성

- 개인별 프로젝트

○ 보고서 제출 방법 및 배점 기준 (강의계획서와 동일, 강의계획서 참고)

○ 보고서 및 소스코드 배점

- 보고서는 다음과 같은 양식으로 작성(강의계획서 FAQ 참고)

1. 과제 개요 (1점) // 명세에 주어진 개요를 더 상세하게 작성
2. 구현 기능 (1점) // 함수 프로토타입 반드시 포함
3. 상세 설계 (10점) // 함수 기능별 흐름도(순서도) 반드시 포함
4. 실행결과 (3점) // 테스트 프로그램의 실행결과 캡처 및 분석
5. (텍스트 형태의) 소스코드 // 주석 포함, 캡처해서 넣지 말고, 텍스트 형태로.

- 소스코드 및 실행 여부 (85점) // 주석 (5점), 실행 여부 (80점)

○ ssu_ext2 프로그램 기본 사항

- 리눅스 시스템에서 생성한 자신의 사용자 아이디에 대해 사용자 홈 디렉토리를 확인 (예. 자신의 리눅스 시스템상 계정 아이디가 oslab일 경우 \$HOME은 /home/oslab임. (%echo \$HOME, 또는 %cat /etc/passwd 파일에서 해당 아이디의 홈 디렉토리 확인)
 - ✓ 리눅스 상에서 파일 경로의 최대 크기는 4,096 바이트이며, 파일 이름의 최대 크기는 255 바이트임
- 본 명세에서 특별히 설명하거나 지정하지 않은 것들을 모두 학생들이 스스로 판단하여 구현하면 됨
- **ssu_ext2 프로그램 실행 시 내장명령어(tree, print, help, exit)에 따라 해당 기능 실행**
- ssu_ext2 프로그램은 첫 번째 인자로 전달된 ext2 파일시스템의 이미지 내용을 읽고 파일 시스템을 분석함
 - ✓ ext2 파일시스템 이미지 생성 방법은 명세 하단 참고

예제 0. ssu_ext2 실행 결과

```
% ./ssu_ext2 ~/ext2disk.img
20250000>
20250000> exit

% ./ssu_ext2
Usage Error : ./ssu_ext2 <EXT2_IMAGE>

%
```

- 프로그램 전체에서 system() 절대 사용 불가. 사용 시 0점 처리
- 프로그램 전체에서 libext2fs.h, ext2_fs.h 등 ext2 파일시스템과 관련된 모든 헤더파일 및 관련 라이브러리 함수 절대 사용 불가. 사용 시 0점 처리
 - ✓ 헤더파일 내의 구조체나 변수는 필요 시 직접 작성하여 사용
- getopt() 라이브러리를 사용하여 옵션 처리 권장
- ssu_ext2 프로그램 자체는 foreground로만 수행되는 것으로 가정함(& background 수행은 안되는 것으로 함)

○ 설계 및 구현

1. ssu_ext2

- 1) Usage : ssu_ext2 <EXT2_IMAGE>

- ssu_ext2 프로그램 실행 시 사용자 입력을 기다리는 프롬프트 출력

2) 인자 설명

- EXT2_IMAGE : ext2 파일 시스템의 이미지 파일

3) 실행결과

- ssu_ext2 프로그램의 실행결과는 “학번”이 표준 출력되고 내장명령어(tree, print, help, exit) 입력 대기. 학번이 20250000일 경우 “20250000” 형태로 출력

예제 1. ssu_ext2 실행결과

```
% ./ssu_ext2
Usage Error : ./ssu_ext2 <EXT2_IMAGE>

% ./ssu_ext2 ~/ext2disk.img
20250000>
20250000> asd
Usage:
  > tree <PATH> [OPTION]... : display the directory structure if <PATH> is a directory
    -r : display the directory structure recursively if <PATH> is a directory
    -s : display the directory structure if <PATH> is a directory, including the size of each file
    -p : display the directory structure if <PATH> is a directory, including the permissions of each
        directory and file
  > print <PATH> [OPTION]... : print the contents on the standard output if <PATH> is file
    -n <line_number> : print only the first <line_number> lines of its contents on the standard output if <PATH>
                      is file
  > help [COMMAND] : show commands for program
  > exit : exit program
20250000> exit

%
```

4) 예외처리(미 구현 시 아래 점수만큼 감점, 필수 기능 구현 여부 판단과는 상관 없음)

- 첫 번째 인자 없이 ssu_ext2 프로그램 실행 시 Usage 출력
- 프롬프트 상에서 엔터만 입력 시 프롬프트 재출력
- 프롬프트 상에서 지정한 내장명령어 외 기타 명령어 입력 시 help 명령어의 결과를 출력 후 프롬프트 재출력

2. 내장명령어 1. tree

1) Usage : tree <PATH> [OPTION]...

- 경로(PATH)를 입력받아 해당 디렉토리의 구조를 표준 출력으로 출력

2) 인자 설명

- 첫 번째 인자 <PATH>은 내용을 출력할 디렉토리의 경로로, 루트 디렉토리(.)를 기준으로 상대경로로 입력 해야 함
- 두 번째 인자 [OPTION]은 '-r', '-s', '-p'가 있으며 동시 사용 가능하고, 생략 가능함('-r', '-s', '-p' 옵션은 아래 설명 확인)

3) 실행결과

- 첫 번째 인자 <PATH>가 디렉토리일 시 해당 경로에 있는 파일과 디렉토리들의 이름을 출력
- “.”, “..”, “lost+found” 디렉토리는 출력 생략

예제 2-1. tree 내장명령어 실행결과

```
% ./ssu_ext2 ~/ext2disk.img
20250000> tree .
.
├ A
├ B
├ ssu_open.c
└ ssu_test.txt
```

3 directories, 2 files

```
20250000> tree B
B
├ BB
└ hello.py
```

2 directories, 1 files

```
20250000> tree ssu_open.c
Error: 'ssu_open.c' is not directory
```

예제 2-1-1. /home/oslab/mnt/ext2disk 디렉토리 트리 구조

```
% tree .
.
├ A
│ └ AA
│   └ hello.c
├ B
│ └ BB
│   └ bye.cpp
├ hello.py
├ lost+found
├ ssu_open.c
└ ssu_test.txt
```

20250000>	
-----------	--

- '-r' 옵션 입력 시에는 디렉토리의 하위 파일과 하위 디렉토리들의 이름도 추가로 출력

예제 2-2. tree 내장명령어 '-r' 옵션 실행결과

```
% ./ssu_ext2 ~/ext2disk.img
```

```
20250000> tree . -r
```

```
.
├── A
│   ├── AA
│   └── hello.c
├── B
│   ├── BB
│   │   └── bye.cpp
│   └── hello.py
├── ssu_open.c
└── ssu_test.txt
```

```
5 directories, 5 files
```

```
20250000> tree B -r
```

```
B
├── BB
│   └── bye.cpp
└── hello.py
```

```
2 directories, 2 files
```

```
20250000>
```

'-s' 옵션 입력 시에는 파일 또는 디렉토리의 크기와 함께 출력

예제 2-3. tree 내장명령어 '-s' 옵션 실행결과

```
% ./ssu_ext2 ~/ext2disk.img
```

```
20250000> tree . -s
```

```
[4096] .
├── [4096] A
├── [4096] B
├── [339] ssu_open.c
└── [75] ssu_test.txt
```

```
3 directories, 2 files
```

```
20250000> tree B -s
```

```
[4096] B
├── [4096] BB
└── [15] hello.py
```

```
2 directories, 1 files
```

```
20250000>
```

- ‘-p’ 옵션 입력 시에는 파일 또는 디렉토리의 권한과 함께 출력

예제 2-4. tree 내장명령어 ‘-p’ 옵션 실행결과

```
% ./ssu_ext2 ~/ext2disk.img
20250000> tree . -p
[drwxr-xr-x] .
├ [drwxr-xr-x] A
├ [drwxr-xr-x] B
├ [-rw-r--r--] ssu_open.c
└ [-rw-r--r--] ssu_test.txt

3 directories, 2 files

20250000> tree B -p
[drwxr-xr-x] B
├ [drwxr-xr-x] BB
└ [-rw-r--r--] hello.py

2 directories, 1 files

20250000>
```

- ‘-r’, ‘-s’, ‘-p’ 세 가지 옵션은 중복해서 사용 가능함

예제 2-5. tree 내장명령어 옵션 중복 사용 실행결과

```
% ./ssu_ext2 ~/ext2disk.img
20250000> tree . -s -p
[drwxr-xr-x 4096] .
├ [drwxr-xr-x 4096] A
├ [drwxr-xr-x 4096] B
├ [-rw-r--r-- 339] ssu_open.c
└ [-rw-r--r-- 75] ssu_test.txt

5 directories, 5 files

20250000> tree B -rsp
[drwxr-xr-x 4096] B
├ [drwxr-xr-x 4096] BB
├ [-rw-r--r-- 339] bye.cpp
└ [-rw-r--r-- 75] hello.py

2 directories, 2 files

20250000>
```

4) 예외 처리(미 구현 시 아래 점수만큼 감점, 필수 기능 구현 여부 판단과는 상관 없음)

- 첫 번째 인자로 올바르지 않은 경로(존재하지 않는 디렉토리) 입력 시 Usage 출력 후 프롬프트 재출력(5점)
- 첫 번째 인자가 디렉토리가 아닐 경우 에러 메시지 출력 후 프롬프트 재출력(5점)
- 두 번째 인자로 올바르지 않은 옵션이 들어왔을 경우 Usage 출력 후 프롬프트 재출력(3점)

3. 내장명령어 2. print

1) Usage : print <PATH> [OPTION]...

- 경로(PATH)를 입력받아 해당 파일에 대한 내용을 표준 출력으로 출력

2) 인자 설명

- 첫 번째 인자 <PATH>은 내용을 출력할 파일의 경로로, 루트 디렉토리(.)를 기준으로 상대경로로 입력 해야 함
- 두 번째 인자 [OPTION]은 ‘-n’가 있으며, 생략 가능함(‘-n’ 옵션은 아래 설명 확인)

3) 실행결과

- 첫 번째 인자 <PATH>가 파일일 시 해당 경로에 있는 파일의 내용을 출력

예제 3-1. print 내장명령어 실행결과
<pre>% ./ssu_ext2 ~/ext2disk.img 20250000> print ssu_test.txt Linux System Programming! Unix System Programming! Linux Mania Unix Mania 20250000> print B/BB/bye.cpp #include <iostream> using namespace std; int main() { cout << "bye" << endl; } 20250000> print A Error: 'A' is not file 20250000></pre>

- '-n' 옵션 입력 시에는 파일의 내용을 입력받은 라인 수 만큼만 출력

예제 3-2. print 내장명령어 '-n' 옵션 실행결과
<pre>% ./ssu_ext2 ~/ext2disk.img 20250000> print ssu_test.txt -n 2 Linux System Programming! Unix System Programming! 20250000> print A/hello.c -n print: option requires an argument -- 'n' 20250000></pre>

4) 예외 처리(미 구현 시 아래 점수만큼 감점, 필수 기능 구현 여부 판단과는 상관 없음)

- 첫 번째 인자로 올바르지 않은 경로(존재하지 않는 파일) 입력 시 Usage 출력 후 프롬프트 재출력(5점)
- 첫 번째 인자가 파일이 아닐 경우 에러 메시지 출력 후 프롬프트 재출력(5점)
- 두 번째 인자로 올바르지 않은 옵션이 들어왔을 경우 Usage 출력 후 프롬프트 재출력(3점)

4. 내장명령어 3. help

1) Usage : help [COMMAND]

- 프로그램 내장명령어에 대한 설명(Usage) 출력

2) 인자 설명

- 첫 번째 인자 [COMMAND]에 대해 해당 내장명령어에 대한 설명(Usage)를 출력. 첫 번째 인자는 생략 가능하며, 생략 시 모든 내장명령어에 대한 설명(Usage) 출력

3) 실행결과

예제 4. help 내장명령어 실행결과
<pre>% ./ssu_ext2 ~/ext2disk.img 20250000> help Usage : > tree <PATH> [OPTION]... : display the directory structure if <PATH> is a directory -r : display the directory structure recursively if <PATH> is a directory -s : display the directory structure if <PATH> is a directory, including the size of each file -p : display the directory structure if <PATH> is a directory, including the permissions of each directory and file > print <PATH> [OPTION]... : print the contents on the standard output if <PATH> is file -n <line_number> : print only the first <line_number> lines of its contents on the standard output if <PATH> is file > help [COMMAND] : show commands for program > exit : exit program</pre>

```

20250000> help tree
Usage : tree <PATH> [OPTION]... : display the directory structure if <PATH> is a directory
-r : display the directory structure recursively if <PATH> is a directory
-s : display the directory structure if <PATH> is a directory, including the size of each file
-p : display the directory structure if <PATH> is a directory, including the permissions of each
    directory and file

20250000> help asd
invalid command -- 'asd'
Usage :
> tree <PATH> [OPTION]... : display the directory structure if <PATH> is a directory
-r : display the directory structure recursively if <PATH> is a directory
-s : display the directory structure if <PATH> is a directory, including the size of each file
-p : display the directory structure if <PATH> is a directory, including the permissions of each
    directory and file
> print <PATH> [OPTION]... : print the contents on the standard output if <PATH> is file
-n <line_number> : print only the first <line_number> lines of its contents on the standard output if <PATH>
    is file
> help [COMMAND] : show commands for program
> exit : exit program

20250000>

```

5. 내장명령어 4. exit

- 1) Usage : exit
 - 현재 실행중인 ssu_ext2 프로그램 종료
- 2) 실행결과
 - 프로그램 종료

예제 5. exit 내장명령어 실행
<pre> % ./ssu_ext2 ~/ext2disk.img 20250000> exit % </pre>

○ ext2 파일시스템 이미지 생성 방법

- dd if=/dev/zero of=ext2disk.img bs=1M count=100
 - ✓ if=/dev/zero : 빈 데이터를 입력
 - ✓ of=ext2disk.img : ext2disk.img 라는 이름으로 저장
 - ✓ bs=1M : 한 번에 1MB씩 쓰기
 - ✓ count=100 : 100MB 생성
- mkfs.ext2 ext2disk.img
 - ✓ ext2disk.img 파일을 ext2 파일 시스템으로 포맷
- mkdir -p ~/mnt/ext2disk
 - ✓ 파일 시스템을 사용할 디렉토리 생성
- sudo mount -o loop ext2disk.img ~/mnt/ext2disk
 - ✓ 파일 기반 가상 디스크(ext2disk.img)를 loopback 장치로 마운트
 - o loop 옵션으로 일반 파일을 가상의 디스크(loopback 장치)로 인식해 마운트
 - ✓ 마운트한 뒤에 ~/mnt/ext2disk 경로에 디렉터리나 파일을 생성하면 ext2disk.img에 반영됨
- sudo umount ~/mnt/ext2disk
 - ✓ 마운트 해제(필요 시 사용)

○ 보고서 제출 시 유의 사항

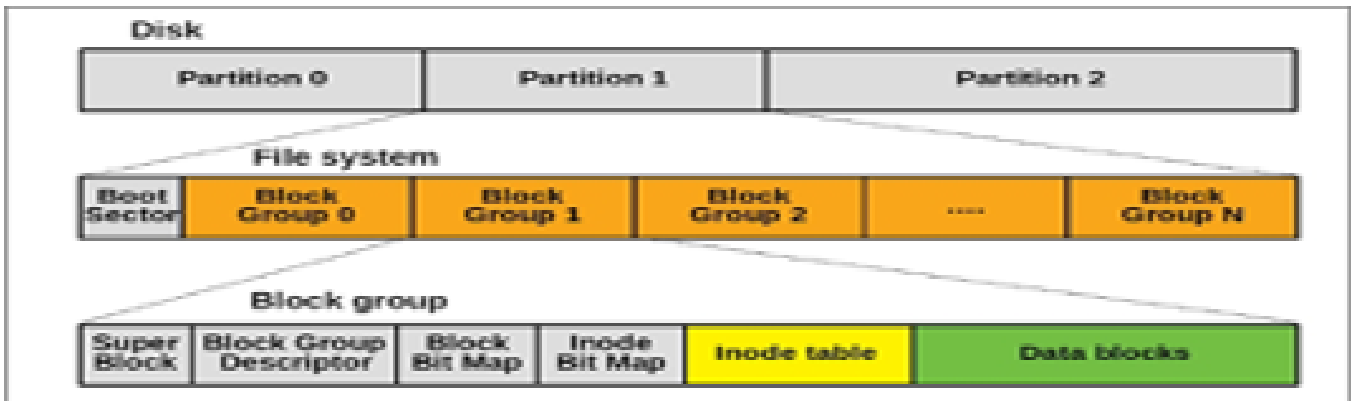
- 보고서 제출 마감은 **6월 1일(일) 오후 11시 59분 59초**
 - ※ 구글클래스룸에서 11:59:00에서 1초라도 지연되면 1일 지연제출로 나올 것임. 이 부분은 추후 제출 시간 확인 후 11:59:59 이내인 경우 정상 제출로 인정 예정임. 단, 1초 지연제출도 0점 처리
- 압축 예러, 파일 누락 관련 감점 syllabus 참고
- 필수구현 : 1. ssu_ext2, 2. 내장명령어 tree, print, help, exit (예외처리 포함 모든 명령어가 필수구현임)
- 배점(100점 만점. 실행 여부 배점 80점으로 최종 환산하며, 보고서 15점과 소스코드 주석 5점은 별도, 강의계획서 확인)
 - 1. ssu_ext2 - 30점
 - 2. 내장명령어 1. tree - 37점, 아래 옵션 미구현 시 : 7점
 - ✓ '-r' 옵션 : 10점
 - ✓ '-s' 옵션 : 10점
 - ✓ '-p' 옵션 : 10점
 - 3. 내장명령어 2. print - 30점, 아래 옵션 미구현 시 : 25점
 - ✓ '-n' 옵션 : 5점
 - 4. 내장명령어 3. help - 1점
 - 5. 내장명령어 4. exit - 1점
- makefile 작성(매크로 사용하지 않아도 됨) - 1점

※ (가산점 부여기준 1) 5월 18일(일) 밤 11시59분까지 명세서에서 모든 기능을 모두 구현하고 보고서를 제출한 경우 본과제 총점에 +30점, 5월 25일(일) 밤 11시59분까지 제출한 경우 본과제 총점(100점)에 +10점을 (가산점으로)추가 부여함.

ext2 filesystem walkthrough

(1) ext2 Background

- Linux용 파일 시스템
- 1993 by Rémy Card
- Block group : block 을 여러 개의 block group으로 나눔 => 헤드의 seek time 최소화



(2) Basic Commends for making ext2

- fdisk : 디스크 파티션 생성 및 확인 % fdisk -l
- mkfs : 파티션의 파일 시스템 생성 % mkfs -t ext2 /dev/sdb1
- mount 파일 시스템의 연결 % mount /dev/sdb1 /home/usb %unmount /dev/sdb1
- automount /media/
- fsck

(3) Boot Block = 1KB

- include/linux/ext2_fs.h
- Super Block = 1 Block, Block Descriptor = n Block, Block Bit Map 1 Block, inode Bit Map = 1 Block
- 각 block 그룹에 대한 정보는 super block 바로 뒤에 있는 block 에 저장된 디스크립터 테이블에 보관

```
struct ext2_group_desc
{
    __u32    bg_block_bitmap; /* Blocks bitmap block */
    __u32    bg_inode_bitmap; /* Inodes bitmap block */
    __u32    bg_inode_table; /* Inodes table block */
    __u16    bg_free_blocks_count; /* Free blocks count */
    __u16    bg_free_inodes_count; /* Free inodes count */
    __u16    bg_used_dirs_count; /* Directories count */
    __u16    bg_pad;
    __u32    bg_reserved[3];
};
```

- 각 그룹의 앞에 있는 2 blocks : 사용 중인 block과 inode를 표시하는 Block Bit Map / inode Bit Map
- 하나의 파티션은 다수의 블록그룹으로 나눌 수 있음
- 각 블록의 상태의 블록비트맵 0, 1로 표시. 블록비트맵은 블록그룹당 1개가 존재
- 블록비트맵은 블록그룹당 1개가 존재하며 설정한 블록 크기(4KB) 하나 사용 => 블록비트맵은 8 bit * 4096 = 32,768 bit 개의 블록 상태를 나타낼 수 있음. => 파티션에서 블록 크기를 4 KB 로 설정하면 한 블록그룹당 블록의 갯수는 32,768임 => 각 비트맵은 하나의 block에 들어가야 하기 때문에 block group 의 최대 block의 갯수= block 크기의 8배
 - (if one block is 4KB) 4KB * 8 = 32,768 (32KB)
 - 하나의 block 그룹이 가질 수 있는 최대 block 의 갯수 : 32KB * 4KB = 128MB
 - 10GB 파티션의 최대 block 그룹 수 : 10GB(10240MB) / 128 MB = 80개 그룹

(4) Super block

- struct ext2_super_block in include/linux/ext2_fs.h
- offset 1024 에 위치
 - ✓ lseek(fd, 1024, SEEK_SET); /* position head above super-block */
- Magic number

```

struct ext2_super_block {
    __u32    s_inodes_count;        /* Inodes count */
    __u32    s_blocks_count;       /* Blocks count */
    ...
    __u32    s_free_blocks_count;  /* Free blocks count */
    __u32    s_free_inodes_count;  /* Free inodes count */
    __u32    s_first_data_block;   /* First Data Block */
    __u32    s_log_block_size;     /* Block size */
    ...
    __u32    s_blocks_per_group;   /* # Blocks per group */
    ...
    __u16    s_magic;              /* Magic signature, EXT2_SUPER_MAGIC */
    ...
}

```

✓ if (super.s_magic != EXT2_SUPER_MAGIC) exit(1); /* bad file system */

```

Filesystem volume name: ssuos_fs
Last mounted on: /media/su/ssuos_fs
Filesystem UUID: 5c93ecd7-e708-4e47-a6bd-e1363cf077e2
Filesystem magic number: 0xEF53
Filesystem revision #: 1 (dynamic)
Filesystem features: ext_attr resize_inode dir_index filetype sparse_super large_file
Filesystem flags: signed_directory_hash
Default mount options: user_xattr acl
Filesystem state: not clean
Errors behavior: Continue
Filesystem OS type: Linux
Inode count: 243840
Block count: 975264
Reserved block count: 48763
Free blocks: 958038
Free inodes: 243829
First block: 0
Block size: 4096
Fragment size: 4096
Reserved GDT blocks: 238
Blocks per group: 32768
Fragments per group: 32768
Inodes per group: 8128
Inode blocks per group: 508
Filesystem created: Tue Nov 15 22:57:48 2022
Last mount time: Tue Nov 15 23:01:39 2022
Last write time: Tue Nov 15 23:01:39 2022
Mount count: 1
Maximum mount count: -1
Last checked: Tue Nov 15 22:57:48 2022
Check interval: 0 (<none>)
Reserved blocks uid: 0 (user root)
Reserved blocks gid: 0 (group root)
First inode: 11
Inode size: 256
Required extra isize: 32
Desired extra isize: 32
Default directory hash: half_md4
Directory Hash Seed: 876563a7-f575-4216-9eff-aca668a3af5c

Group 0: (Blocks 0-32767)
Primary superblock at 0, Group descriptors at 1-1
Reserved GDT blocks at 2-239
Block bitmap at 240 (+240)
Inode bitmap at 241 (+241)
Inode table at 242-749 (+242)
32810 free blocks, 8115 free inodes, 2 directories
Free blocks: 756-1026, 1028-1535, 1537-32767
Free inodes: 12-13, 15-16, 18-8128

```

• **\$fdisk -l**

• **\$dumpe2fs /dev/sdb1**

- 슈퍼블럭, 그룹 정보 등 표시
- *dumpfs (BSD)*

975,264 blocks * 4,096 bytes per block = 약 3.72 GB (4,096 = 0x1000)

8,128 inodes per group (0x1fc0)

256 bytes per inode (0x100)

Groups form an intermediate structure within an ext2 volume

```

su@su-virtual-machine:~$ sudo mkfs.ext2 -L ssuos_fs /dev/sdb1
mkfs2fs 1.45.5 (07-Jan-2020)
/dev/sdb1 contains a ext2 file system
  last mounted on /media/su/0776e6b2-0be3-405b-ab62-1f4785230ad6 on Tue Nov 15 22:42:39 2022
Proceed anyway? (y,N) y
Creating filesystem with 975264 4k blocks and 243840 inodes
Filesystem UUID: 5c93ecd7-e708-4e47-a6bd-e1363cf077e2
Superblock backups stored on blocks:
    32768, 98304, 163840, 229376, 294912, 819200, 884736

Allocating group tables: done
Writing inode tables: done
Writing superblocks and filesystem accounting information: done

```

(5) 현재 디렉토리(루트 디렉토리)에 존재하는 디렉토리 및 파일 구조

```
root@su-virtual-machine:/media/su/ssuos_fs# tree
```

```

.
├── fs.txt
├── helloworld
├── helloworld.c
├── linux.txt
├── lost+found
└── ssuos
    └── a.txt

```

2 directories, 5 files

```
root@su-virtual-machine:/media/su/ssuos_fs# ls -al
```

```

total 60
drwxr-xr-x  4 root root  4096 11월 30 15:44 .
drwxr-xr-x+ 4 root root  4096 11월 30 15:46 ..
-rw-r--r--  1 root root    26 11월 15 23:02 fs.txt
-rwxr-xr-x  1 root root 16704 11월 30 15:44 helloworld
-rw-r--r--  1 root root    80 11월 30 15:44 helloworld.c
-rw-r--r--  1 root root  3283 11월 15 23:03 linux.txt
drwx----- 2 root root 16384 11월 15 22:58 lost+found
drwxr-xr-x  2 root root  4096 11월 18 16:06 ssuos

```

(6) fs.txt 파일 내용

```
root@su-virtual-machine:/media/su/ssuos_fs# cat fs.txt
SSUOS File System : EXT2
```

```
root@su-virtual-machine:/media/su/ssuos_fs#
```

○ \$ hexdump /dev/sdb1

- file contents를 hexadecimal, decimal, octal, ascii로 표시
- 00 bytes 연속은 "*"
- 오른쪽에 아스키값 표시 가능 -C ("canonical") 옵션(One-byte character display)

```

00000000  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | .....|
*
00000400  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | .....|
00000410  75 b8 03 00 00 00 00 00 02 00 00 00 02 00 00 00 | W.....|
00000420  00 00 00 00 00 00 00 00 c0 1f 00 00 c3 9b 73 63 | .....SC|
00000430  c3 9b 73 63 01 00 ff ff 53 ef 00 00 01 00 00 00 | ..SC...S....|
00000440  dc 9a 73 63 00 00 00 00 00 00 00 00 01 00 00 00 | ..SC.....|
00000450  00 00 00 00 00 00 00 00 00 01 00 00 38 00 00 00 | .....S....|
00000460  02 00 00 00 03 00 00 00 50 93 00 07 e7 08 4e 47 | .....NG....|
00000470  a6 bd e1 36 3c f0 77 e2 73 73 25 61 73 5f 66 73 | ...6c..h...fs|
00000480  00 00 00 00 00 00 00 00 2f 6d 65 04 00 04 2f 73 | ...../media/s|
00000490  75 2f 73 73 75 6f 73 5f 66 73 00 00 00 00 00 00 | u/ssuos_fs...|
000004a0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | .....|
*
000004c0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 ee 00 | .....|
000004d0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | .....|
000004e0  00 00 00 00 00 00 00 00 00 00 00 00 07 65 63 a7 | .....6C....|
000004f0  f5 75 42 16 9e ff ac a6 68 a3 af 5c 01 00 00 00 | .U8.....h...|
00000500  0c 00 00 00 00 00 00 00 dc 9a 73 63 00 00 00 00 | .....SC....|
00000510  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | .....|
*
00000550  00 00 00 00 00 00 00 00 00 00 00 00 20 00 20 00 | .....  . |
00000560  01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | .....|
00000570  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | .....|
*
00010000  f0 00 00 00 f1 00 00 00 f2 00 00 00 0a 7d b5 1f | .....).|

```

```

struct ext2_super_block {
    __le32 s_inodes_count; /* Inodes count */
    __le32 s_blocks_count; /* Blocks count */
    __le32 s_free_blocks_count; /* Free blocks count */
    __le32 s_free_inodes_count; /* Free inodes count */
    __le32 s_first_data_block; /* First data block */
    __le32 s_log_block_size; /* Block size */
    __le32 s_log_frag_size; /* Fragment size */
    __le32 s_blocks_per_group; /* # blocks per group */
    __le32 s_frags_per_group; /* # frags per group */
    __le32 s_inodes_per_group; /* # inodes per group */
    __le32 s_mtime; /* Mount time */
    __le32 s_wtime; /* Write time */
    __le16 s_mnt_count; /* Mount count */
    __le16 s_max_mnt_count; /* Maximal mount count */
    __le16 s_magic; /* Magic signature */
    __le16 s_state; /* File system state */
    __le16 s_errors; /* Behaviour when detecting errors */
    __le16 s_minor_rev_level; /* minor revision level */
    __le32 s_lastcheck; /* time of last check */
    __le32 s_checkinterval; /* time between checks */
    __le32 s_creator_os; /* OS */
    __le32 s_rev_level; /* Revision level */
    __le16 s_def_resuid; /* default uid for reserved blocks */
    __le16 s_def_resgid; /* default gid for reserved blocks */
    __le32 s_first_ino; /* First non-reserved inode */
    __le32 s_inode_size; /* size of inode structure */
    __le32 s_block_group_nr; /* block group # of this superblock */
    __le32 s_feature_compat; /* compatible feature set */
    __le32 s_feature_incompat; /* incompatible feature set */
    __le32 s_feature_ro_compat; /* readonly-compatible feature set */
    __u8 s_uuid[16]; /* 128-bit uuid for volume */
    char s_volume_name[16]; /* volume name */
    char s_last_mounted[64]; /* directory where last mounted */
    __le32 s_algorithm_usage_bits; /* for compression */
    __u8 s_journal_uuid[16]; /* uuid of journal superblock */
    __u32 s_journal_inum; /* inode number of journal file */
    __u32 s_journal_dev; /* device number of journal file */
    __u32 s_last_orphan; /* start of list of inodes to delete */
    __u32 s_hash_seed[4]; /* NBWEE hash seed */
    __u8 s_def_hash_algorithm; /* default hash version to use */
    __u8 s_reserved_char_dev;
    __u32 s_reserved_word_dev;
    __le16 s_default_mount_opts;
    __le32 s_first_meta_bg; /* First meta-block block group */
    __u32 s_reserved[190]; /* padding to the end of the block */
}

```

(7) group descriptor, Block bitmap, inode bitmap, inode table

```
00001000 f0 00 00 00 f1 00 00 00 f2 00 00 00 0a 7d b3 1f |.....}..|
00001010 02 00 04 00 00 00 00 00 00 00 00 00 00 00 |.....|
00001020 f0 80 00 00 f1 80 00 00 f2 80 00 00 12 7d c0 1f |.....}..|
00001030 00 00 04 00 00 00 00 00 00 00 00 00 00 00 |.....|
```

```
struct ext2_group_desc
{
    __le32 bg_block_bitmap; /* Blocks bitmap block */
    __le32 bg_inode_bitmap; /* Inodes bitmap block */
    __le32 bg_inode_table; /* Inodes table block */
    __le16 bg_free_blocks_count; /* Free blocks count */
    __le16 bg_free_inodes_count; /* Free inodes count */
    __le16 bg_used_dirs_count; /* Directories count */
    __le16 bg_pad;
    __le32 bg_reserved[3];
};
```

```
000f0000 ff ff ff ff ff ff ff ff ff ff ff ff ff ff |.....|
*
000f0050 ff ff ff ff ff ff ff ff ff ff ff ff ff ff |.....|
000f0060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |.....|
*
000f0080 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |.....|
000f0090 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |.....|

000f1000 ff 27 01 00 00 00 00 00 00 00 00 00 00 00 |..'......|
000f1010 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |.....|
*
000f13f0 00 00 00 00 00 00 00 00 ff ff ff ff ff ff ff ff |.....|
000f1400 ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff |.....|
```

- inode table은 0xf2에서 시작, 한개의 block은 4,096 bytes 이기 때문에 해당 inode는 0xf2000
- inode structure 당 256바이트. 256 = 0x100이므로 F2000, F2100, F2200, F2300
- ✓ 단, 앞의 몇개 inode는 시스템적으로 이미 예약
- 계속해서 볼륨을 읽기 위해 루트 디렉토리의 inode인 inode 2를 확인
- inode 1이 F2000에 있으므로 F2100에 inode 2가 있음

```
/*
 * Special inode numbers
 */
#define EXT2_BAD_INO 1 /* Bad blocks inode */
#define EXT2_ROOT_INO 2 /* Root inode */
#define EXT2_BOOT_LOADER_INO 5 /* Boot loader inode */
#define EXT2_UNDEL_DIR_INO 6 /* Undelete directory inode */

/* First non-reserved inode for old ext2 filesystems */
#define EXT2_GOOD_OLD_FIRST_INO 11
```

```
000f2000 00 00 00 00 00 00 00 00 17 9b 73 63 17 9b 73 63 |.....9C..9C|
000f2010 17 9b 73 63 00 00 00 00 00 00 00 00 00 00 |...9C.....|
000f2020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |.....|
*
000f2100 0d 41 00 00 00 18 00 00 ca 9c 73 63 00 73 63 |..A.....9C..9C|
000f2110 00 5c 73 63 00 00 00 00 00 00 04 00 00 00 00 |...5C.....|
000f2120 00 00 00 00 11 00 00 00 00 00 00 00 00 00 00 |.....|
000f2130 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |.....|
*
000f2180 20 00 00 00 38 92 85 d6 38 92 85 d6 c0 c0 00 |...8..8.....|
000f2190 17 9b 73 63 00 00 00 00 00 00 00 00 00 00 |...9C.....|
000f21a0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |.....|
*
000f2200 00 01 00 00 c0 00 00 00 17 9b 73 63 17 9b 73 63 |.....9C..9C|
000f2210 17 9b 73 63 00 00 00 00 00 00 00 00 00 00 |...9C.....|
000f2220 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |.....|
*
000f2300 00 00 00 00 00 00 00 00 00 00 00 00 f3 02 00 00 |.....|
000f2310 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |.....|
000f2320 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |.....|
000f2330 20 00 00 00 00 00 00 00 00 00 00 00 00 00 |.....|
000f2340 17 9b 73 63 00 00 00 00 00 00 00 00 00 00 |...9C.....|
000f2350 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |.....|
*
000f2400 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |.....|
000f2410 20 00 00 00 00 00 00 00 00 00 00 00 00 00 |.....|
000f2420 17 9b 73 63 00 00 00 00 00 00 00 00 00 00 |...9C.....|
000f2430 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |.....|
```

- 루트 디렉토리의 i_mode는 0x41ed (0100 0001 1110 1101), directory file, S_IFDIR, rwxr_xr_x
- 첫번째 Data block : 0x02ee * 1000

```
000f2100  ed 41 00 00 00 10 00 00  ca 9c 73 63 ae 9c 73 63 |.A.....sc..sc|
000f2110  ae 9c 73 63 00 00 00 00  00 00 04 00 08 00 00 00 |..sc.....|
000f2120  00 00 00 00 11 00 00 00  ee 02 00 00 00 00 00 00 |.....|
000f2130  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 |.....|
```

```
struct ext2_inode {
    __le16 i_mode; /* File mode */
    __le16 i_uid; /* Low 16 bits of Owner Uid */
    __le32 i_size; /* Size in bytes */
    __le32 i_atime; /* Access time */
    __le32 i_ctime; /* Creation time */
    __le32 i_mtime; /* Modification time */
    __le32 i_dtime; /* Deletion Time */
    __le16 i_gid; /* Low 16 bits of Group Id */
    __le16 i_links_count; /* Links count */
    __le32 i_blocks; /* Blocks count */
    __le32 i_flags; /* File flags */
    union {
        struct {
            __le32 l_i_reserved1;
        } linux1;
        struct {
            __le32 h_i_translator;
        } hurd1;
        struct {
            __le32 m_i_reserved1;
        } msvc1;
    } osd1; /* OS dependent 1 */
    __le32 i_block[EXT2_N_BLOCKS]; /* Pointers to blocks */
    __le32 i_generation; /* File version (for NFS) */
    __le32 i_file_acl; /* File ACL */
    __le32 i_dir_acl; /* Directory ACL */
    __le32 i_faddr; /* Fragment address */
    union {
        struct {
            __u8 l_i_frag; /* Fragment number */
            __u8 l_i_fsize; /* Fragment size */
            __u16 i_pad1;
            __le16 l_i_uid_high; /* these 2 fields */
            __le16 l_i_gid_high; /* were reserved2[0] */
            __u32 l_i_reserved2;
        } linux2;
        struct {
            __u8 h_i_frag; /* Fragment number */
            __u8 h_i_fsize; /* Fragment size */
            __le16 h_i_node_high;
            __le16 h_i_uid_high;
            __le16 h_i_gid_high;
            __le32 h_i_author;
        } hurd2;
        struct {
            __u8 m_i_frag; /* Fragment number */
            __u8 m_i_fsize; /* Fragment size */
            __u16 m_pad1;
            __u32 m_i_reserved2[2];
        } msvc2;
    } osd2; /* OS dependent 2 */
};
```

(8) Directory

- 디렉토리의 데이터 블록은 디렉토리 항목의 배열
- ✓ 데이터 블록 0x02ee (또는 오프셋 0x002ee000에 있는 루트 디렉토리용)

- ✓ 현재 디렉토리(".")과 상위 디렉토리("..")도 명시적 디렉토리 항목으로 저장

```

struct ext2_dir_entry_2 {
    __le32 inode;      /* Inode number */
    __le16 rec_len;    /* Directory entry length */
    __u8  name_len;    /* Name length */
    __u8  file_type;   /* File type */
    char  name[];      /* File name, up to EXT2_NAME_LEN */
};

```

002ee000	02 00 00 00 0c 00 01 02	2e 00 00 00 02 00 00 00
002ee010	0c 00 02 02 2e 2e 00 00	0b 00 00 00 14 00 0a 02
002ee020	6c 6f 73 74 2b 66 6f 75	6e 64 00 00 01 fa 02 00	lost+found.....
002ee030	2c 00 05 02 73 73 75 6f	73 00 00 00 00 00 00 00	ssuos.....
002ee040	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
002ee050	00 00 00 00 00 00 00 00	11 00 00 00 14 00 09 01
002ee060	6c 69 6e 75 78 2e 74 78	74 00 00 00 0e 00 00 00	linux.txt.....
002ee070	94 0f 06 01 66 73 2e 74	78 74 00 00 00 00 00 00	fs.txt.....
002ee080	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
002ef000	0b 00 00 00 0c 00 01 02	2e 00 00 00 02 00 00 00
002ef010	f4 0f 02 02 2e 2e 00 00	00 00 00 00 00 00 00 00
002ef020	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00

- 0x0 unknown
- 0x1 regular file
- 0x2 diectory
- 0x3 character device file
- 0x4 blokc device file
- 0x5 FIFO
- 0x6 Socket
- 0x7 Symbolic Link

```

root@su-virtual-machine:/media/su/ssuos_fs# ls -li
14 fs.txt          17 linux.txt

```

(9) fs.txt

- 파일 크기 : 26 bytes
- Data block의 시작점 : 0x0600 * 1000

000f2d00	a4 81 00 00 1a 00 00 00	e1 ea 86 63 ae 9c 73 63c..sc
000f2d10	05 9c 73 63 00 00 00 00	00 00 01 00 08 00 00 00	..sc.....
000f2d20	00 00 00 00 01 00 00 00	00 06 00 00 00 00 00 00
000f2d30	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00

00600000	53 53 55 4f 53 20 46 69	6c 65 20 53 79 73 74 65	SSUOS File Syste
00600010	6d 20 3a 20 45 58 54 32	0a 0a 00 00 00 00 00 00	m : EXT2.....

(10) linux.txt

- 파일 크기 : 3,283 bytes
- Data block의 시작점 : 0x0403 * 1000

```
000f3000  a4 81 00 00 d3 0c 00 00  a6 fd 86 63 e6 2e 77 63  |.....C..wC|
000f3010  29 9c 73 63 00 00 00 00  00 00 01 00 08 00 00 00  |).sc.....|
000f3020  00 00 00 00 01 00 00 00  03 04 00 00 00 00 00 00  |.....|
000f3030  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |.....|

00403000  4c 69 6e 75 78 20 28 2f  cb 88 6c 69 cb 90 6e ca  |Linux (/..ll..n.|
00403010  8a 6b 73 2f 20 28 6c 69  73 74 65 6e 29 20 4c 45  |.ks/ (listen) LE|
00403020  45 2d 6e 75 75 6b 73 20  6f 72 20 2f cb 88 6c c9  |E-nuuks or /..l.|
00403030  aa 6e ca 8a 6b 73 2f 20  4c 49 4e 2d 75 75 6b 73  |.n..ks/ LIN-uuks|

***

00403cc0  74 20 70 72 6f 67 72 61  6d 2e 5b 34 30 5d 5b 32  |t program.[40][2|
00403cd0  31 5d 0a 00 00 00 00 00  00 00 00 00 00 00 00 00  |1].....|
00403ce0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |.....|
*
```

○ 실제 파일 내용 확인

- 텍스트 파일은 찾기 쉽고 추가 디렉토리는 루트 디렉토리와 동일한 방식으로 찾으면 됨
- fs.txt
 - ✓ 오프셋 0xf2d00으로 변환되는 inode e에 있으며, 여기서 파일의 첫 번째 데이터 블록은 전체 데이터 블록 0x0600있음
 - ✓ Inode #'s offset = F2000 + (100 * (# - 1))

```
000f2d00  a4 81 00 00 1a 00 00 00  05 9c 73 63 ae 9c 73 63  |.....sc..sc|
000f2d10  05 9c 73 63 00 00 00 00  00 00 01 00 08 00 00 00  |..sc.....|
000f2d20  00 00 00 00 01 00 00 00  00 06 00 00 00 00 00 00  |.....|
000f2d30  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |.....|
```

- ✓ 실제 데이터는 오프셋 0x1000 * 0x0600 = 0x600000에서 찾을 수 있음

```
00600000  53 53 55 4f 53 20 46 69  6c 65 20 53 79 73 74 65  |SSUOS File Syste|
00600010  6d 20 3a 20 45 58 54 32  0a 0a 00 00 00 00 00 00  |m : EXT2.....|
00600020  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |.....|
```

- linux.txt
 - ✓ 오프셋 0xf300으로 변환되는 Inode 11에 있으며, 첫번째 데이터 블록은 0x0403
 - ✓ Inode #'s offset = F2000 + (100 * (# - 1))

```
000f3000  a4 81 00 00 d3 0c 00 00  29 9c 73 63 95 9c 73 63  |.....).sc..sc|
000f3010  29 9c 73 63 00 00 00 00  00 00 02 00 08 00 00 00  |).sc.....|
000f3020  00 00 00 00 01 00 00 00  03 04 00 00 00 00 00 00  |.....|
000f3030  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |.....|
```

- ✓ 실제 데이터는 오프셋 0x1000 * 0x0403 = 0x403000에서 찾을 수 있음

```
00403000  4c 69 6e 75 78 20 28 2f  cb 88 6c 69 cb 90 6e ca  |Linux (/..ll..n.|
00403010  8a 6b 73 2f 20 28 6c 69  73 74 65 6e 29 20 4c 45  |.ks/ (listen) LE|
00403020  45 2d 6e 75 75 6b 73 20  6f 72 20 2f cb 88 6c c9  |E-nuuks or /..l.|
00403030  aa 6e ca 8a 6b 73 2f 20  4c 49 4e 2d 75 75 6b 73  |.n..ks/ LIN-uuks|
00403040  29 5b 31 31 5d 20 69 73  20 61 6e 20 6f 70 65 6e  |)[11] is an open|
00403050  2d 73 6f 75 72 63 65 20  55 6e 69 78 2d 6c 69 6b  |-source Unix-lik|

***

00403cc0  74 20 70 72 6f 67 72 61  6d 2e 5b 34 30 5d 5b 32  |t program.[40][2|
00403cd0  31 5d 0a 00 00 00 00 00  00 00 00 00 00 00 00 00  |1].....|
00403ce0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |.....|
*
```

Dec	Hex	Name	Char	Ctrl-char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	0	Null	NUL	CTRL-@	32	20	Space	64	40	@	96	60	`
1	1	Start of heading	SOH	CTRL-A	33	21	!	65	41	A	97	61	a
2	2	Start of text	STX	CTRL-B	34	22	"	66	42	B	98	62	b
3	3	End of text	ETX	CTRL-C	35	23	#	67	43	C	99	63	c
4	4	End of xmit	EOT	CTRL-D	36	24	\$	68	44	D	100	64	d
5	5	Enquiry	ENQ	CTRL-E	37	25	%	69	45	E	101	65	e
6	6	Acknowledge	ACK	CTRL-F	38	26	&	70	46	F	102	66	f
7	7	Bell	BEL	CTRL-G	39	27	'	71	47	G	103	67	g
8	8	Backspace	BS	CTRL-H	40	28	(72	48	H	104	68	h
9	9	Horizontal tab	HT	CTRL-I	41	29)	73	49	I	105	69	i
10	0A	Line feed	LF	CTRL-J	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	VT	CTRL-K	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	FF	CTRL-L	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage feed	CR	CTRL-M	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	SO	CTRL-N	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	SI	CTRL-O	47	2F	/	79	4F	O	111	6F	o
16	10	Data line escape	DLE	CTRL-P	48	30	0	80	50	P	112	70	p
17	11	Device control 1	DC1	CTRL-Q	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	DC2	CTRL-R	50	32	2	82	52	R	114	72	r
19	13	Device control 3	DC3	CTRL-S	51	33	3	83	53	S	115	73	s
20	14	Device control 4	DC4	CTRL-T	52	34	4	84	54	T	116	74	t
21	15	Neg acknowledge	NAK	CTRL-U	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	SYN	CTRL-V	54	36	6	86	56	V	118	76	v
23	17	End of xmit block	ETB	CTRL-W	55	37	7	87	57	W	119	77	w
24	18	Cancel	CAN	CTRL-X	56	38	8	88	58	X	120	78	x
25	19	End of medium	EM	CTRL-Y	57	39	9	89	59	Y	121	79	y
26	1A	Substitute	SUB	CTRL-Z	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	ESC	CTRL-[59	3B	;	91	5B	[123	7B	{
28	1C	File separator	FS	CTRL-\	60	3C	<	92	5C	\	124	7C	
29	1D	Group separator	GS	CTRL-]	61	3D	=	93	5D]	125	7D	}
30	1E	Record separator	RS	CTRL-^	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	US	CTRL-~	63	3F	?	95	5F	_	127	7F	DEL