IN2140 – Introduksjon til operativsystemer og datakommunikasjon

Obligatorisk oppgave 2: Ditt eget filsystem

Vår 2022

1 Introduksjon

I denne obligatoriske oppgaven skal du implementere en skisse av ditt eget filsystem. Vi kaller det en skisse fordi løsningen din selvsagt ikke behøver å omfatte all funksjonalitet som et komplett filsystem bør ha. Hvis du senere tar kurset IN3000/IN4000 – Operativsystemer, så vil du implementere en mer avansert filsystemløsning. Mye av det du lærer i denne oppgaven kan komme til nytte. Du gis en struktur inspirert av inoder på Unix-liknende operativsystemer, som OpenBSD, Linux og MacOS. Strukturen inneholder metadata om filer og kataloger, herunder pekere til andre inoder i filsystemet. På denne måten dannes et tre med én rotnote, som vist i eksempelet under.

```
/
├── etc/
│   ├── httpd/
│   │   └── conf/
│   │       └── httpd.conf
│   ├── host.conf
│   └── hostname
└── var/
    └── log/
        └── messages
```

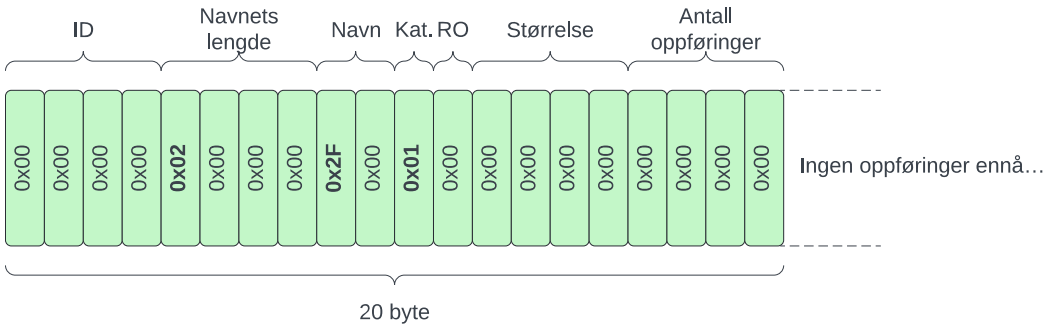
Du skal implementere funksjoner for å arbeide med filsystemet. Hver fil og hver katalog er representert ved en inode, som har en type, et navn, noen andre attributer og informasjon om innhold i filen eller katalogen. For å kunne lese og forandre inoder, så må de hentes fra disken til minnet. Når den ligger i minnet har en inode følgende struktur:

```
struct inode
{
    int     id;
    char*   name;
    char    is_directory;
    char    is_readonly;
    int     filesize;
    int     num_entries;
    size_t* entries;
};
```

I det følgende forklarer vi feltene i `struct inode`:

- Hver inode har et ID-nummer `id` unikt for hele filsystemet og et fil- eller katalognavn `name`. Rotkatalogen vil for eksempel ha navn `"/"`. Vilkårlig lange fil- og katalognavn tillates. På disk lagres navnets lengde inkludert den avsluttende nullbyten etterfulgt av bokstavene. Å lagre navnets lengde på disk vil gjøre arbeidet litt enklere.
- Byteflagget `is_directory` bestemmer om inoden representerer en fil eller en katalog. Dersom `is_directory` er 0, så representerer inoden en fil.
- Et annet byteflagg, `is_readonly` bestemmer om filen eller katalogen, som inoden representerer, er lesbar og skrivbar eller kun lesbar. Den er tatt med som eksempelattributt, men har ingen praktisk betydning for oppgaven.
- `Filesize` gir filstørrelsen i byte; for kataloger er denne alltid 0.
- Hver inode har en peker `entries` til en array med `num_entries` oppføringer à 64 bit. Oppføringene tolkes forskjellig avhengig om inoden representerer en fil eller en katalog: For filer tolkes oppføringene som blokknumre (heltallstype), hvor hver blokk har størrelse 4096 byte. For kataloger inneholder hver entry en peker til inode, som representerer en fil eller underkatalog i denne katalogen.

Figuren under viser hvordan inoden som representerer rotkatalogen ser ut når den ligger på disk. Legg merke til antall oppføringer er 0 så lenge ingen andre filer eller kataloger er opprettet. Senere vil hver oppføring fylle 8 byte. En oppføring i en filinode inneholder et blokknummer som beskrevet over, mens en oppføring i en kataloginode inneholder **ID-en** til en annet inode (og ikke noen peker til den). `0x2F` er forresten den heksadesimale ASCII-verdien for foroverskråstrek.



2 Oppgaver

2.1 Design

Skriv en README-fil hvor du forklarer følgende punkter:

- Hvordan du leser superblokkfilen fra disk og laster inodene inn i minnet.
- Eventuelle implementasjonskrav som ikke er oppfylt
- Eventuelle deler av implementasjonen som avviker fra prekoden. Dersom du for eksempel oppretter egne filer, forklar hva hensikten er.
- Eventuelle tester som feiler og hva du tror årsaken kan være.

2.2 Implementasjon

Vi leverer ut prekode og forventer at dere implementerer følgende fire funksjoner, som finnes som skjeletter i filen `inode.c`. Det vil være hensiktsmessig å skrive hjelpefunksjoner i tillegg.

Opprett fil

```
struct inode* create_file(
    struct inode* parent,
    char* name,
    char readonly,
    int size_in_bytes
);
```

Funksjonen tar som parameter en peker til inoden til katalogen som skal inneholde den nye filen. Innenfor denne katalogen må navnet være unikt. Dersom det finnes en fil med samme navn der fra før, så skal funksjonen returnere `NULL` uten å gjøre noe.

Opprett katalog

```
struct inode* create_dir(
    struct inode* parent,
    char* name
);
```

Funksjonen tar som parameter inoden til katalogen som skal inneholde den nye katalogen. Innenfor denne katalogen må navnet være unikt. Dersom det finnes en katalogen med samme navn der fra før, så skal funksjonen returnere `NULL` uten å gjøre noe.

Finn inode ved navn

```
struct inode* find_inode_by_name(
    struct inode* parent,
    char* name
);
```

Funksjonen sjekker alle inoder som refereres direkte fra foreldrenoden. Hvis én av dem har navnet `name`, så returnerer funksjonen dens inodepeker. `Parent` må peke på en kataloginode. Dersom ingen slik inode finnes, så returnerer funksjonen `NULL`.

Last inn

```
struct inode* load_inodes();
```

Funksjonen leser superblokkfilen og oppretter i minnet en inode for hver tilsvarende oppføring i filen. Funksjonen setter pekere mellom inodene på riktig vis. Superblokkfilen forblir uforandret.

Avslutt

Skriv følgende funksjon for å koble ned filsystemet.

```
void fs_shutdown( struct inode* node );
```

Funksjonen frigjør alle inode-datastrukturer og alt minne som de refererer til. Dette kan utføres rett før et testprogram avslutter programmet.

2.3 Viktige utleverte funksjoner

```
void debug_fs( struct inode* node );
```

I prekoden leveres det en funksjon `debug_fs` som skriver ut en inode og hvis denne inoden er en katalog, rekursivt også alle fil- og katalog-inoder under den. Det skrives ID, navn og tilleggsinformasjon.

```
void debug_disk( );
```

I tillegg leveres funksjonen `debug_disk`, som skriver ut enten 0 eller 1 for alle blokker på den simulerte disken vår. Har en blokk verdi 1 så finnes det en fil-inode som har reservert blokken til sin fil. Blokker representert av ved 0 er ikke i bruk.

```
int allocate_block();
```

Funksjonen allokerer en diskblokk på 4096 bytes. Man kan ikke allokere mindre enn én diskblokk for fildata, og blokken kan ikke deles mellom filer. Funksjonen returner en blokindeks eller -1 ved ingen ledige blokker. Vi holder denne informasjonen oppdatert på disk i en fil som heter `block_allocation_table`. Dette enkle filsystemet implementerer ikke extent-prinsippet. Man må allokere én blokk á 4096 byte om gangen ved å kalle funksjonen. Vi antar at superblokken ikke forvaltes på samme måte.

```
int free_block(int block);
```

Når filen slettes skal diskblokkene frigjøres. Returnerer også -1 ved feil (blokk ikke allokert og så videre).

3 Råd

Her er noen råd som kan gjøre arbeidet med oppgaven litt enklere.

- Det kan være hensiktsmessig å implementere funksjonene i denne rekkefølgen:

1. `load_inodes`
2. `create_dir`,
3. `create_file`
4. `find_inode_by_name`
5. `fs_shutdown`.

- For å sjekke minnelekkasjer, kjør Valgrind med følgende flagg:

```
valgrind
    --track-origins=yes   \
    --malloc-fill=0x40    \
    --free-fill=0x23      \
    --leak-check=full     \
    --show-leak-kinds=all \
    DITT_PROGRAM
```

4 Levering

1. Legg alle filene i en mappe med ditt brukernavn. Bortsett fra Makefile behøver du ikke opprette noen nye filer selv, men kan klare deg med filene som du har fått utlevert som prekode (etter at du har implementert funksjonene). Hvis du oppretter egne filer, så husk å kopiere dem inn i mappen også.

```
$ mkdir brukernavn
$ cp -r
        create_example1/ \
        create_example2/ \
        create_example3/ \
        load_example1/   \
        load_example2/   \
        load_example3/   \
        allocation.c     \
        allocation.h     \
        inode.c          \
        inode.h          \
        Makefile         \
        brukernavn/
```

2. Lag et komprimert arkiv, som du leverer.

```
$ tar czf brukernavn.tar.gz brukernavn/
```

3. Sterkt anbefalt! Test innleveringen ved å laste tar.gz-filen ned til en Ifi-maskin. Pakk ut, kompiler og kjør.