# Отчет по работе с ФС XFS

Выполнил: Куприянов А.А, P33113

Лабораторная работа #1

Пример работы программы: https://youtu.be/wSnewFKoC9E

## Создание файла ФС (loop device)

Для создания файла я использовал самописный скрипт, который создвавал нулевой файл и монтировал его на первый свободный dev/loop**

```
magicnum = 0x58465342
blocksize = 4096
dblocks = 16384
rblocks = 0
rextents = 0
uuid = b65c0299-6c57-42b9-a6c4-8998af3d937a
logstart = 8198
rootino = 128
rbmino = 129
rsumino = 130
rextsize = 1
agblocks = 4096
agcount = 4
rbmblocks = 0
logblocks = 1368
versionnum = 0xb4a5
sectsize = 512
inodesize = 512
inopblock = 8
fname = "\000\000\000\000\000\000\000\000\000\000\000\000"
blocklog = 12
sectlog = 9
inodelog = 9
inopblog = 3
agblklog = 12
rextslog = 0
inprogress = 0
imax_pct = 25
icount = 64
ifree = 61
fdblocks = 14984
frextents = 0
uquotino = null
gquotino = null
qflags = 0
flags = 0
shared_vn = 0
inoalignmt = 8
unit = 0
width = 0
```

```
dirblklog = 0
logsectlog = 0
logsectsize = 0
logsunit = 1
features2 = 0x18a
bad_features2 = 0x18a
features_compat = 0
features_ro_compat = 0x5
features_incompat = 0x3
features_log_incompat = 0
crc = 0x88e4fba1 (correct)
spino_align = 4
pquotino = null
lsn = 0x100000008
meta_uuid = 00000000-0000-0000-0000-000000000000
```

Далее, его можно было смонтировать и добавить туда некоторые файлы для валидации работоспособности приложения.

## Начало работы. Архитектура

При продумывании архитектуры приложения необходимо было учесть, что во второй лабораторной работе нужно будет использовать эту же библиотеки с другим приложением (на языке высокого уровня).

Поэтому нужно было жестко ограничить модули и выделить из них две основные - для UI и Core, которая бы работала с файловой системой.

### Раздельные модули - залог успеха

Было решено создать две отдельные директории со своими Makefile'ами, которые бы отвечали только за свой модуль. Далее, из этого исходила проблема того, как их собрать вместе.

Чтобы решить эту проблему я использовал отдельно команду `ld`, с relocatable:

```
OBJ_FILES = main interactive_mode info device xfs commands

.SILENT: clean-temp

all: $(OBJ_FILES)
    ld -relocatable $(OBJ_FILES) -o core.o
    make clean-temp

$(OBJ_FILES):
    gcc -c src/$@.c -o $@

clean:
    rm *.o

clean-temp:
    for file in $(OBJ_FILES); do \
        rm $$file;  \
    done;
```

Таким образом, на каждом модуле мы получали объектный файл, который можно было использовать в финальной компиляции:

```
SUBDIRS = cli core

all: subdirs
    gcc -o super-fs core/core.o cli/cli.o

.PHONY: subdirs $(SUBDIRS)

subdirs: $(SUBDIRS)

$(SUBDIRS):
            $(MAKE) -C $@

clean:
    find . -type f -name '*.out' -delete
    rm super-fs

    for dir in $(SUBDIRS); do \
    $(MAKE) clean -C $$dir; \
    done
```

Минусом данной схемы является то, что для итерации нужно знать список файлов и постоянно синхронизировать его в Makefile (тоже самое с субдиректориями в финальном Makefile). Возможно, эту проблему можно решить красиво, но я решил не тратить на это много времени.

## Програмист на Си рожден, чтобы страдать

> Мне не нравится Си. И лично я убежден, что Си - это пережиток прошлого и никакой низкоуровневой романтики я в нем не нахожу. Во время написания этой ~чертовой~ лабораторной работы я не раз его проклинал

Для начала были прочитаны следующие источники:

- http://ftp.ntu.edu.tw/linux/utils/fs/xfs/docs/xfs_filesystem_structure.pdf
- https://xfs.org/docs/xfsdocs-xml-dev/XFS_Filesystem_Structure/tmp/en-US/html/index.html
- Исходники ядра Linux версий 3.15 (считается, что с этой версии xfs работает стабильно) и 5.11.4
  - https://elixir.bootlin.com/linux/v5.11.4/source/fs/xfs
  - https://elixir.bootlin.com/linux/v3.15/source/fs/xfs
- Linux Conf Au 2018, Sydney, Australia
- How to create XFS virtual block device
- RedHat XFS
- https://web.archive.org/web/20171031110501/http://oss.sgi.com/projects/xfs/papers/xfs_filesystem_structure.pdf#page=25
- https://web.archive.org/web/20150904032700/http://www.ibm.com/developerworks/library/l-fs9/index.html
- И множество других, которые я уже не смог найти по истории браузера

Некоторые куски кода были взяты из некоторых уже готовых сэмплов в интернете, например, некоторые типы в структуре XFS, к примеру тот же суперблок:

```
typedef struct PACKED xfs_sb {
```

```
    __uint32_t sb_magicnum;
    __uint32_t sb_blocksize;
    xfs_rfsblock_t sb_dblocks;
    xfs_rfsblock_t sb_rblocks;
    xfs_rtblock_t sb_rextents;
    uuid_t sb_uuid;
    xfs_fsblock_t sb_logstart;
    xfs_ino_t sb_rootino;
    xfs_ino_t sb_rbmino;
    xfs_ino_t sb_rsumino;
    xfs_agblock_t sb_rextsize;
    xfs_agblock_t sb_agblocks;
    xfs_agnumber_t sb_agcount;
    xfs_extlen_t sb_rbmblocks;
    xfs_extlen_t sb_logblocks;
    __uint16_t sb_versionnum;
    __uint16_t sb_sectsize;
    __uint16_t sb_inodesize;
    __uint16_t sb_inopblock;
    char sb_fname[12];
    __uint8_t sb_blocklog;
    __uint8_t sb_sectlog;
    __uint8_t sb_inodelog;
    __uint8_t sb_inopblog;
    __uint8_t sb_agblklog;
    __uint8_t sb_rextslog;
    __uint8_t sb_inprogress;
    __uint8_t sb_imax_pct;
    __uint64_t sb_icount;
    __uint64_t sb_ifree;
    __uint64_t sb_fdblocks;
    __uint64_t sb_frextents;
    xfs_ino_t sb_uquotino;
    xfs_ino_t sb_gquotino;
    __uint16_t sb_qflags;
    __uint8_t sb_flags;
    __uint8_t sb_shared_vn;
    xfs_extlen_t sb_inoalignmt;
    __uint32_t sb_unit;
    __uint32_t sb_width;
    __uint8_t sb_dirblklog;
    __uint8_t sb_logsectlog;
    __uint16_t sb_logsectsize;
    __uint32_t sb_logsunit;
    __uint32_t sb_features2;
    __uint32_t sb_bad_features2;

    /* version 5 superblock fields start here */
    __uint32_t sb_features_compat;
    __uint32_t sb_features_ro_compat;
    __uint32_t sb_features_incompat;
    __uint32_t sb_features_log_incompat;
    __uint32_t sb_crc;
    xfs_extlen_t sb_spino_align;
    xfs_ino_t sb_pquotino;
    xfs_lsn_t sb_lsn;
    uuid_t sb_meta_uuid;
    xfs_ino_t sb_rrmapino;
```

```
} xfs_sb_t;
```

Как видно из комментария он может отличаться в зависимости от версиии ядра Linux.

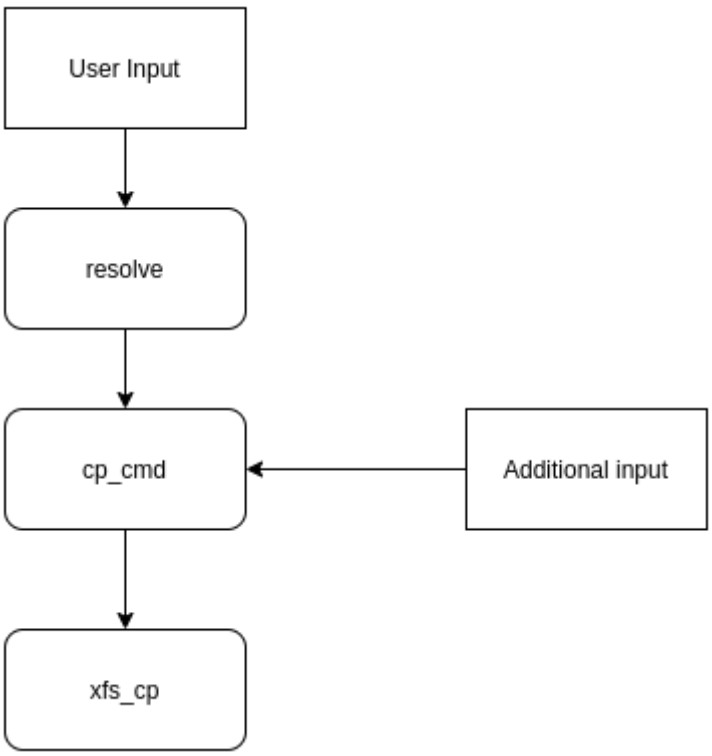
Чтобы просмотреть все типы используемые для работы с XFS, посмотрите файл `xfs_types.h`, некоторые из них наприямую связаны с типами в организации XFS, некоторые же касаются только данного приложения.

Далее, после очень мучительных дней работы с XFS был создан `commands`, который просто являлся прослойкой обработки пользовательского ввода и его исполнением в среде XFS.

```
int resolve(char *cmd, xfs_t *fm);
```

```
  if (strcmp("ls", cmd) == 0)
    res = ls_cmd(fm);
  else if (strcmp("cd", cmd) == 0)
    res = cd_cmd(fm);
  else if (strcmp("cp", cmd) == 0)
    res = cp_cmd(fm);
  // next part of cp command will be executed inside cp_cmd
  else if (strcmp("exit", cmd) == 0)
    res = 0;
  else if (strcmp("help", cmd) == 0)
    res = help_cmd();
  else if (strcmp("dog", cmd) == 0) {
    res = dog_cmd(fm);
  } else
    printf("Unknown command '%s'. Use help command\n", cmd);
```

При этом обработка некоторых команд происходила внутри вызовов методов оберток



```
static int cp_cmd(xfs_t *fm) {
  char *from, *to;
  scanf("%ms %ms", &from, &to);
  wrap_err__(xfs_cp(fm, from, to));
  free(from);       // чистим все
  free(to);         // свидетелей не оставляем
  return 1;
}
```

# Вывод

## Мое личное мнение про эту лабораторную работу и сам предмет

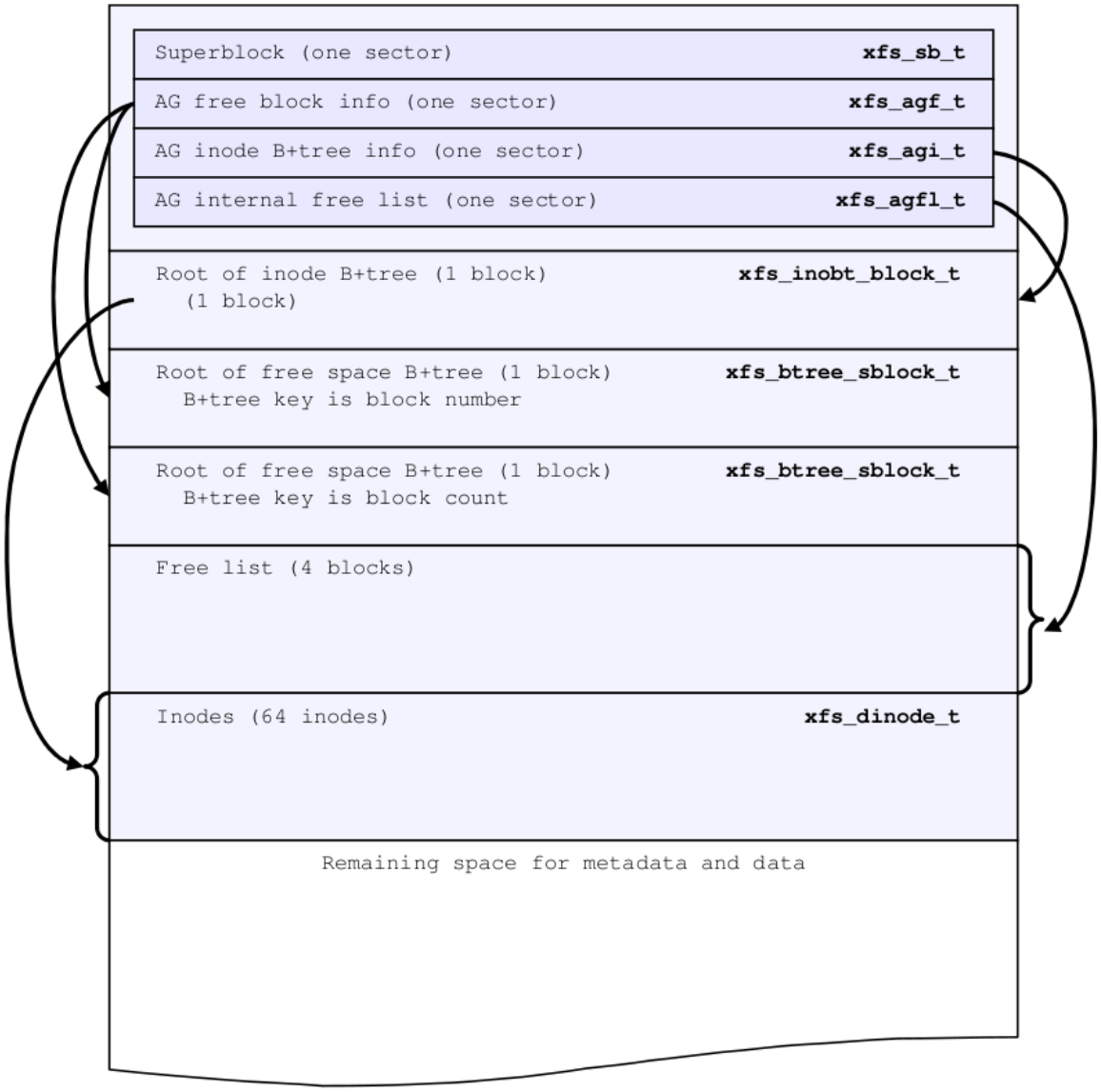
Я ненавижу Си. Помимо этого мне пришлось работать с файловой системой в сыром виде (вопрос того насколько это было необходимо и будет ли мне полезно в будущем, конечно, оставим в стороне).

Конкретно этой лабораторной работы - она сложная, при том слишком (что заставляет задуматься о качестве верификации таких лабораторных работ). Тем не менее, мне в какой степени сильно помогло то, что я раньше также ковырял байт-код для JVM: Моя статья на хабре .. И лично я считаю, что это было бы намного полезнее и в принципе в какой-то степени легче (из-за более разумной документации и комьюнити).

## XFS

### AG

Файловая системы XFS довольно интересна. Во-первых, она делится на несколько чанков Allocation Groups (AG). И каждый AG почти является индивидуальной файловой системой со своим суперблоком, менеджентом свободный памяти и выделением инодов.




### Free Space Managment

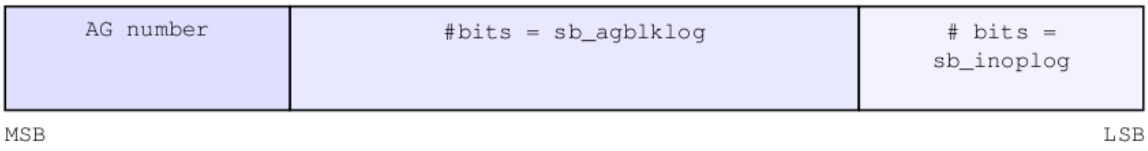
Также лично мне интересным показалась работы с свободной памятью. Так как в XFS менеджмент памяти происходит с использованием двух B+tree, один из которых работает с номером блока, а другой по размеру свободной памяти. Это позволяет быстро находить нужные участки свободной памяти.

### Inode managment

Иноды имеют две вариации - абсолютная и относительная. Относительная инода имеет длину в 32 бита, абсолютная же имеет дополнительно спереди номер AG

**Relative Inode number format**


**Absolute Inode number format**



**B+tree везде! Нам нужно больше!**

B+tree в XFS используется везде. Из примеров выше, например, при трэкинге свободной памяти или при менеджементе инодов.

**В целом вроде неплохо**

XFS в принципе оставляет приятные впечатления, о чем также и говорит его комьюнити, которое его использует