Семинар 30.04.

Буферизир. и небуферизир. вв/вывод

(или
(мало открытые файлы)

- Пр-сы работают с открыт. файлами, т.к. один и т.же сис. вызов может открыть ∃ файл или создать новый файл
- Сис. вызов open() имеет 2 варианта

```
int open(const char *pathname, int flags);
int open(const char *pathname, int flags, mode_t mode);
```

- Если есть флаг O_CREATE, то open() создаёт новый файл, то д.б. обязательно парам-р с правами доступа. (права доступа устанавл. при создании файла)

Флаги:

O_CREAT — обязательно используем

вместе использ.: O_EXCL — контроль существование → проверка ∃ файла, чтобы не потерять данные, запис. в него

} Эти флаги устанавл. при создании файла

O_EXEC
O_RDONLY } права доступа
O_RDWR
O_APPEND

• В си есть станд. библиотека stdio.h — библиотека буферизованного ввода/вывода

Из метод.: int main()

for I/O system call — int fd = open("foo.txt", O_RDONLY);

FILE *fs = fopen("bar.txt", "w"); — for C standard library I/O

...

• 2 пр-сса могут открыть один и тот же файл для чтения, для чтения и для записи. Возможны серьёзные проблемы как при чтении, так и при записи (прочитать не в том послед-ти) (можно затереть информацию другой инф-цией)

fopen() — буферизованная ф-ция

• Все ф-ции библиотеки stdio.h — буферизовн. ввод/вывод
• Все ф-ции из stdio.h работают с буфером

Сначала инф-ция пишется в буфер, и только затем в результ. наступление 3х событий (причины) данные из буфера перекинутся в файл:

причина ① Буфер заполнен и мы пытаемся в буферный символ записать (a-char) → в этом случае данные из буфера будут переписаны в файл, буфер освобождается и символ в буфер будет записан?

причина ② Принудительный слив данных в файл fflush().

причина ③ Закрытие файла fclose().
Когда мы вызываем fclose, данные из буфера записываются в файл.

open () - небуферизов-ое вв/вывод
fopen () - буферизов-ое вв/вывод.

дескриптор процесса
↓
struct task_struct

• Процесс открывает файлы

• В task_struct есть 2 поля, связ. с файлами:

struct task_struct { ...

1) /* filesystem information: */
   struct fs_struct *fs; - // информ-ция о ФС, которой принадлежит образ процесса, т.е. исполняемый файл

2) /* open file information: */
   struct files_struct *files; - // инф-ция об открытых процессом файлах

Прежде чем мы запустим программу, она левее файлом, а
• ∀ файл ∈ конкр. ФС (эта информация есть в дескрипторе пр-сса)

• Пр-сс может открыть много файлов ← нет ограничений
  Пр-сс может несколько раз открыть один и тот же файл

(введено много доп. типов - для доп. контроля типов (это автоматически компилятор делает))

```
struct fs_struct
{
    atomic_t count;
    spinlock_t lock;
    int umask;
    struct path root, pwd;   // указ-ль на корневой и текущий каталог
    ( {  struct dentry *dentry;
         struct vfsmount *mnt;
      }  )   - это поле стр-ры struct path
    ...
    int in_exec;
}
```

не к-во ссылок на файл

```
struct files_struct
{
    atomic_t count;
    spinlock_t file_lock;
    int max_fds;        // max кол-во файл. объектов
    int max_fdset;      // max кол-во файл. дескрипторов
    int next_fd;        // содержит номер первого незанятого дескриптора в таблице дескрипторов открытых файлов процесса
    unsigned long *close_on_exec;   // пр-сс потомок наследует дескрипт-ры батюшки файлов, но когда вызывается след. вызов exec, пр-сс переходит на выполнение новой пр-ммы и ему старые открытые файлы не нужны.
    unsigned long close_on_exec_init[1];   // первоначальное файловые для закрытие при вызове exec
    unsigned long open_fds_init[1];   // первоначальный набор файл. дескрипторов
    struct file *fd_array[NR_OPEN_DEFAULT];   // массив файловых объектов
};
```

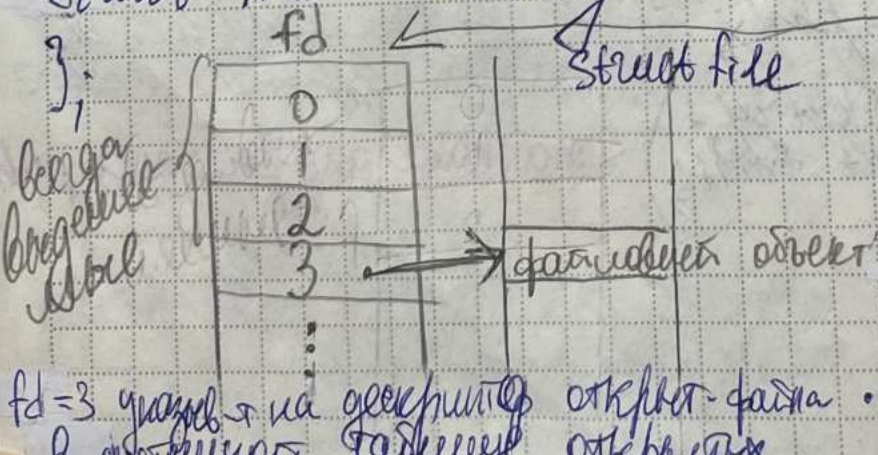
сделано чтобы пр-сс быстрее получил этот номер

ф. дескрипторы, кот. должны закрываться при вызове exec

это находится в стр-ре struct fd_table

close-on-exec

(это таблица, в кот. каждая строка явл. указателем на дескриптор открытого файла struct file)



fd = 3 указывает на дескриптор открыт. файла в системной таблице открытых файлов

• Размер этой таблицы в системе определён (512)

Структура уровня польз-ше: (библиотечная стр-ра)

// эту стр-ру вот-нет приводить надо

typedef struct _IO_FILE FILE

struct _IO_FILE FILE

// В стр-ре дескриптор, а указатели на буфера

{
...
char *_IO_write_base; /* start of put area // указ-ль на начало буфера
char *_IO_write_ptr; // текущ. ук-ль
char *_IO_write_end; // указ-ль на конец
...
int _fileno; // это номер дескриптора открытого файла в таблице открытых файлов процесса
...
}

указ-ль на FILE ≠ дескриптор файла
— это указ-ль на буфера

PFT - process file table — таблица файлов, открытых пр-сом (т.е. каждый пр-с имеет собств. таблицу открытых файлов

SFT - system file table — системная таблица открытых файлов

В системе есть 1 таблица открытых файлов в которой находятся файловые объекты struct file.

!!! Файл м.б. много раз открыт (struct file у него м.б. много), но inode будет один !!!

struct file
{
...
struct path f_path;
struct inode *f_inode; // указ-ль на inode
const struct file_operations *f_op;
fmode_t fmode;
...
atomic_long f_count; // сколько hard linkов имеет-ся на данный файл
...

uptr - указ-ль на inode

В Linux нет vnode, есть только inode

loff_t f_pos; // позиция в файле

открыв файл, pos=0

• И файл считает, что он начинается с 0-го смещения

• struct file — логическая организация файла

pos — позиция в файле — логическая организация файла

• Каждый inode описание конкретного смонтированного ФС, т.е. экземпляр (superblock)

• Мы 3 раза открыли один и тот же файл, для него будет создано 3 экземпляра структуры struct file,

• каждый будет поле pos, т.е. с каждым это будет отдельная работа независимая.

1) • если у нас блок-ориентированность файла, то каждая операция с файлом сдвигает смещение на размер блока

2) если символьный файл, то — " — на 1 байт

• в дисковом inode находятся адреса блоков, в кот. хранится информация, записанная в файл

• в struct inode есть указ-ль на super-block

• каждый экземпляр стр-ры struct file имеет поле pos

В системе нет "текстовых" файлов, есть "символьные"

Важно! запись

Пример из метода (Лаб работа 5...)

Программа № 1: • файл alphabet.txt открывается 1 раз сис. вызовом open

Вред. создаётся 2 указателя на файл

• (библиот. функция) fdopen() вызывается 2 раза (для этих указателей установлен размер буфера 20 символов)

fdopen — "fdopen" типо дублирование open

• fscanf() читает из файла (буферизованный ввод/вывод)

· 1го fscanf() считает из файла 20 символов
т.е. буфер будет заполнен

· 2ой fscanf считает в буфер оставшиеся 6 символов

· fprintf(stdout, ...) — пишем в стандартный поток вывода поочерёдно символы.

Получаем результат работы программы — скриншот + краткие комментарии

· Рисуем связь структур до super_block

User | Kernel — во всех файлах отделяем user-mode от kernel-mode

Программа № 2: · 2 раза вызываются сис. выз. open
файл alphabet 2 раза открывается

· Возвращаются fd1, fd2 — дескрипторы открытых файлов таблицы откр файлов процесса

· Выполняется чтение из файла выполняется по дескриптору fd1 и вывод на экран, потом fd2

Получаем результат + скриншот в отчёт с комментариями

7:23 Открываем в текстовом редакторе файл и это наш скрин. Комментируем + что оказ. в файле

Надо показать что в файле оказалось. скрин

Программу № 2 реализуем на потоках в 2 вариантах (там где читаем). потоки все перемешиваются

глав

разные результаты (I вар 1 осн. поток и 1 доп. поток,
II вар 2 доп. потока в кот. из файла читаем

Программа №3: · это прога л 2 но с буферизованност (fopen) (fwrite)

на каждой итерации выводим из stat значение inode, size из дерева каталогов (Кого)

Структуру struct stat приводим в отчёте.

Нет наше pos, но есть size. (но при записи инфы в файла pos эквивал. size)

Для 1го, 2го, 3го программ - сводь структур (не для подвариантов) user / kernel

Потоки создаются не сразу.

struct stat не надо?

Лаб Seq. фатальн требование: · в каждой ф-ции надо написать printk.

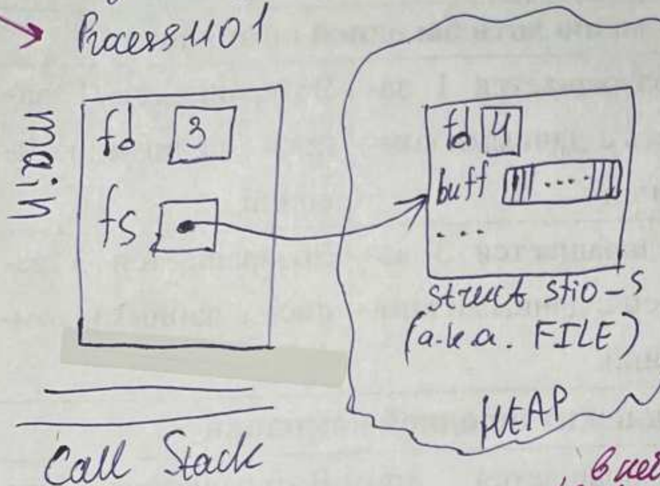
Пишем: open
read
write — одинаков
release но
exit пишем

· Указатели в ф-ции итератора.

• Когда создаётся пр-сс, для него автоматически открываются 3 файла с номерами 0, 1, 2. При этом, если пр-сс не унаследовал от предка ещё какие-то файлы, то первый свободный дескриптор открытого файла у процесса будет 3. (Но не факт, что 3. Мб были открыты какие-то файлы)

• Описание к 1й картинке из методички:

fopen() возвращает указ-ль struct FILE *fs, но в коде библиот. ф-ции fopen будет вызываться сис. вызов open(), поэтому в этом облаке упоминается fd (file descriptor)



! Стр-ры FILE не сущест.

??? • PFT содержит указатели на записи в SFT. PFT - массив файл. объектов (индексируется файловыми дескрипторами: 0 - stdin, 1 - stdout, 2 - stderr ...)

• SFT - глобальная таблица ядра, хранящая ВСЕ открытые файлы в системе.

Каждая запись в ней описывает реальный открытый файл struct file

• В stdio.h есть ф-ция file_no, кот. возвращает дескриптор файла

• Пояснение к первой картинке из методички: (kernel space)

PFT - process file table (т.е. каждый пр-сс имеет собств. таблицу открыт. файлов)

SFT - system file table

В системе есть 1 таблица открытых файлов, в которой находятся файловые объекты struct file