CAOS 6 – low level IO

Manakov Danila

MIPT

12 октября 2022 г.

Prelude

Представим себе ситуацию: вы пишите операционку и уже почти написали ядро.

Prelude

Представим себе ситуацию: вы пишите операционку и уже почти написали ядро.
Теперь нужно как-то научить программы общаться между собой, обращаться к файловой системе, обращаться к модулям ядра, общаться по сети, ...

Prelude

Представим себе ситуацию: вы пишите операционку и уже почти написали ядро.
Теперь нужно как-то научить программы общаться между собой, обращаться к файловой системе, обращаться к модулям ядра, общаться по сети, ...
Что делаем?

Prelude

Представим себе ситуацию: вы пишите операционку и уже почти написали ядро.
Теперь нужно как-то научить программы общаться между собой, обращаться к файловой системе, обращаться к модулям ядра, общаться по сети, ...
Что делаем?
???

Prelude

Windows на каждый вопрос дала свой ответ: реестр, WinAPI, драйвера, ещё фигня какая-то ...

Prelude

Windows на каждый вопрос дала свой ответ: реестр, WinAPI, драйвера, ещё фигня какая-то ...
Если решите с этим разобраться, то сядьте...

Prelude

Windows на каждый вопрос дала свой ответ: реестр, WinAPI, драйвера, ещё фигня какая-то ...
Если решите с этим разобраться, то сядьте... заварите себе чай...

Prelude

Windows на каждый вопрос дала свой ответ: реестр, WinAPI, драйвера, ещё фигня какая-то ...
Если решите с этим разобраться, то сядьте... заварите себе чай... и подумайте, в какой момент вы приняли неверное решение...

Prelude

Windows на каждый вопрос дала свой ответ: реестр, WinAPI, драйвера, ещё фигня какая-то ...
Если решите с этим разобраться, то сядьте... заварите себе чай... и подумайте, в какой момент вы приняли неверное решение...

UNIX дала на этот вопрос всего 1 ответ – ЧЕРЕЗ ФАЙЛЫ!!!

Linux files

Ядро отвечает за распределение ресурсов компьютера (в том числе и за место на диске), ну и оно имеет встроенные функции для работы с файлами)

Linux files

Ядро отвечает за распределение ресурсов компьютера (в том числе и за место на диске), ну и оно имеет встроенные функции для работы с файлами)
Внутренняя "штука"для работы с фалами называется *filedescriptor*(*fd*).

Linux files

Ядро отвечает за распределение ресурсов компьютера (в том числе и за место на диске), ну и оно имеет встроенные функции для работы с файлами)
Внутренняя "штука"для работы с фалами называется *filedescriptor*(*fd*).
Их внутреннее устройство выходит за рамки курса, нам о них надо знать следующее:

- Каждый открытый файл связан с одним fd

Linux files

Ядро отвечает за распределение ресурсов компьютера (в том числе и за место на диске), ну и оно имеет встроенные функции для работы с файлами)
Внутренняя "штука"для работы с фалами называется *filedescriptor*(*fd*).
Их внутреннее устройство выходит за рамки курса, нам о них надо знать следующее:

- Каждый открытый файл связан с одним fd
- fd обозначаются числами, начиная с 0

Linux files

Ядро отвечает за распределение ресурсов компьютера (в том числе и за место на диске), ну и оно имеет встроенные функции для работы с файлами)
Внутренняя "штука"для работы с фалами называется *filedescriptor*(*fd*).
Их внутреннее устройство выходит за рамки курса, нам о них надо знать следующее:

- Каждый открытый файл связан с одним fd
- fd обозначаются числами, начиная с 0
- при открытии нового файла fd получает минимальный свободный номер
- их количество ограничено на каждый процесс

File types

```
$ ls -l a.out
-rwxrwxr-x 1 dmanakov dmanakov 16784 Oct 10 23:10 a.out

$ ls -l ../
drwxrwxr-x 2 dmanakov dmanakov 4096 Oct 11 00:57 01-enumerate

$ ls -l /dev/sda
brw------- 1 root root 8,  0 Sep 29 00:13 /dev/sda

$ ls -l a.out.link
lrwxrwxrwx 1 dmanakov dmanakov 5 Oct 11 00:56 a.out.link -> a.out

$ ls -l /dev/null
crw-rw-rw- 1 root root 1, 3 Sep 29 00:13 /dev/null

$ ls -l some.fifo
prw-rw-r-- 1 dmanakov dmanakov 0 Oct 11 00:57 some.fifo
```

File types

```
$ ls -l a.out
-rwxrwxr-x 1 dmanakov dmanakov 16784 Oct 10 23:10 a.out

$ ls -l ../
drwxrwxr-x 2 dmanakov dmanakov 4096 Oct 11 00:57 01-enumerate

$ ls -l /dev/sda
brw------- 1 root root 8,  0 Sep 29 00:13 /dev/sda

$ ls -l a.out.link
lrwxrwxrwx 1 dmanakov dmanakov 5 Oct 11 00:56 a.out.link -> a.out

$ ls -l /dev/null
crw-rw-rw- 1 root root 1, 3 Sep 29 00:13 /dev/null

$ ls -l some.fifo
prw-rw-r-- 1 dmanakov dmanakov 0 Oct 11 00:57 some.fifo
```

Ну и во все из них можно писать / читать

File descriptors amout

```
$ ulimit -a  # текущие лимиты
...
-m: resident set size (kbytes)       unlimited
-u: processes                        50288
-n: file descriptors                 1024
-l: locked-in-memory size (kbytes)   64
-v: address space (kbytes)           unlimited
-x: file locks                       unlimited
-i: pending signals                  50288
...

$ ulimit -aH  # hard limits (yes, but actually no)
...
-m: resident set size (kbytes)       unlimited
-u: processes                        50288
-n: file descriptors                 4096
-l: locked-in-memory size (kbytes)   64
-v: address space (kbytes)           unlimited
-x: file locks                       unlimited
-i: pending signals                  50288
...
```

File descriptors amout

```
$ ulimit -a  # текущие лимиты
...
-m: resident set size (kbytes)       unlimited
-u: processes                        50288
-n: file descriptors                 1024
-l: locked-in-memory size (kbytes)   64
-v: address space (kbytes)           unlimited
-x: file locks                       unlimited
-i: pending signals                  50288
...

$ ulimit -aH  # hard limits (yes, but actually no)
...
-m: resident set size (kbytes)       unlimited
-u: processes                        50288
-n: file descriptors                 4096
-l: locked-in-memory size (kbytes)   64
-v: address space (kbytes)           unlimited
-x: file locks                       unlimited
-i: pending signals                  50288
...
```

Какая из этого мораль?)

File descriptors amout

```
$ ulimit -a   # текущие лимиты
...
-m: resident set size (kbytes)       unlimited
-u: processes                        50288
-n: file descriptors                 1024
-l: locked-in-memory size (kbytes)   64
-v: address space (kbytes)           unlimited
-x: file locks                       unlimited
-i: pending signals                  50288
...

$ ulimit -aH   # hard limits (yes, but actually no)
...
-m: resident set size (kbytes)       unlimited
-u: processes                        50288
-n: file descriptors                 4096
-l: locked-in-memory size (kbytes)   64
-v: address space (kbytes)           unlimited
-x: file locks                       unlimited
-i: pending signals                  50288
...
```

Какая из этого мораль?) - Файлы надо закрывать))

Давайте что-нибудь напишем

```c
#include <fcntl.h>
#include <unistd.h>
#include <stdio.h>

int main() {
        int fd3 = open("/tmp/files/1.txt", O_RDONLY);
        int fd4 = open("/tmp/files/2.txt", O_RDONLY);

        printf("fd3: %d\n", fd3);
        printf("fd4: %d\n", fd4);

        close(fd3);

        int fd5 = open("/tmp/files/3.txt", O_RDONLY);
        printf("fd5: %d\n", fd5);
        close(fd4);
        close(fd5);
        return 0;
}
```

Какой вывод?)

Давайте что-нибудь напишем

```c
#include <fcntl.h>
#include <unistd.h>
#include <stdio.h>

int main() {
        int fd3 = open("/tmp/files/1.txt", O_RDONLY);
        int fd4 = open("/tmp/files/2.txt", O_RDONLY);

        printf("fd3: %d\n", fd3);
        printf("fd4: %d\n", fd4);

        close(fd3);

        int fd5 = open("/tmp/files/3.txt", O_RDONLY);
        printf("fd5: %d\n", fd5);
        close(fd4);
        close(fd5);
        return 0;
}
```

Какой вывод?)

```
$ gcc main.c
$ ./a.out
fd3: 3
fd4: 4
fd5: 3
```

Открываем fd при запуске

Прошу заметить! Внутри программы файл я НИГДЕ НЕ ОТКРЫВАЛ

```c
#include <unistd.h>

int main() {
    char str[] = "Hello!!!\n";
    write(7, str, sizeof(str) - 1);
    return 0;
}
```

```
$ make && make run
gcc main.c
./a.out 7>my_output.txt

$ cat my_output.txt
Hello!!!
```

Struct

Структура - берем кучу данных, складываем их вместе.

```
1  struct {
2      field_0
3      field_1
4      field_2
5  }
```

Внимание вопрос!) Как посчитать размер структуры?

Struct

Проведем викторинку)))

```
1  struct {
2          char c0;
3          char c1;
4          char c2;
5  } ccc;
6  printf("%ld\n", sizeof(ccc));
```

Размер структуры -

Struct

Проведем викторинку)))

```
1  struct {
2          char c0;
3          char c1;
4          char c2;
5  } ccc;
6  printf("%ld\n", sizeof(ccc));
```

Размер структуры - 3

Struct

Проведем викторинку)))

```
1  struct {
2          char c0;
3          char c1;
4          char c2;
5  } ccc;
6  printf("%ld\n", sizeof(ccc));
```

Размер структуры - 3

```
1  struct {
2          char c0;
3          char c1;
4          int16_t i16;
5  } cci;
6  printf("%ld\n", sizeof(cci));
```

Размер структуры -

Struct

Проведем викторинку)))

```
1  struct {
2          char c0;
3          char c1;
4          char c2;
5  } ccc;
6  printf("%ld\n", sizeof(ccc));
```

Размер структуры - 3

```
1  struct {
2          char c0;
3          char c1;
4          int16_t i16;
5  } cci;
6  printf("%ld\n", sizeof(cci));
```

Размер структуры - 4

Struct

```
1  struct {
2          char c0;
3          int16_t i16;
4          char c1;
5  } cic;
6  printf("%ld\n", sizeof(cic));
```

Размер структуры -

Struct

```
1  struct {
2          char c0;
3          int16_t i16;
4          char c1;
5  } cic;
6  printf("%ld\n", sizeof(cic));
```

Размер структуры - 6!!!

Правила размещения полей структуры в памяти

- порядок полей в памяти соответствует порядку полей в описании структуры
- размер структуры должен быть кратен размеру наибольшего поля структуры
- поля структуры располагаются в памяти так, чтобы быть прижатыми к её границам
- поля структуры располагаются по адресам, кратным их размерам

Правила размещения полей структуры в памяти

- порядок полей в памяти соответствует порядку полей в описании структуры
- размер структуры должен быть кратен размеру наибольшего поля структуры
- поля структуры располагаются в памяти так, чтобы быть прижатыми к её границам
- поля структуры располагаются по адресам, кратным их размерам

Пы.Сы. В ридингах это уже было

Правила размещения полей структуры в памяти

- порядок полей в памяти соответствует порядку полей в описании структуры
- размер структуры должен быть кратен размеру наибольшего поля структуры
- поля структуры располагаются в памяти так, чтобы быть прижатыми к её границам
- поля структуры располагаются по адресам, кратным их размерам

Пы.Сы. В ридингах это уже было
Читайте ридинги!)

Struct

```
1  struct {
2          char c0;
3          // unused byte
4          int16_t i16;
5          // one more unused byte
6          char c1;
7  } cic;
8  printf("%ld\n", sizeof(cic));
```

Размер структуры - 6

Struct

The last one викторинка

```
1  struct {
2          uint64_t u64;
3          char c;
4  } u64c;
5  printf("%ld\n", sizeof(u64c));
6  } cic;
7  printf("%ld\n", sizeof(cic));
```

Размер структуры -

Struct

The last one викторинка

```
1  struct {
2          uint64_t u64;
3          char c;
4  } u64c;
5  printf("%ld\n", sizeof(u64c));
6  } cic;
7  printf("%ld\n", sizeof(cic));
```

Размер структуры - 16

Struct

The last one викторинка

```
1  struct {
2          uint64_t u64;
3          char c;
4  } u64c;
5  printf("%ld\n", sizeof(u64c));
6  } cic;
7  printf("%ld\n", sizeof(cic));
```

Размер структуры - 16

```
1  struct {
2          uint64_t u64;
3          char c;
4  } __attribute__((packed)) u64cp;
5  printf("%ld\n", sizeof(u64cp));
```

Размер структуры -

Struct

The last one викторинка

```
1  struct {
2          uint64_t u64;
3          char c;
4  } u64c;
5  printf("%ld\n", sizeof(u64c));
6  } cic;
7  printf("%ld\n", sizeof(cic));
```

Размер структуры - 16

```
1  struct {
2          uint64_t u64;
3          char c;
4  } __attribute__((packed)) u64cp;
5  printf("%ld\n", sizeof(u64cp));
```

Размер структуры - 9

Основные ф-ии для работы с файлами

- **open** - открывает / создает файл
- **write** - пишет в файл
- **read** - читает из файла
- **lseek** - reposition read/write file offset (двигаем курсор)
- **close** - закрывает fd

Нулевка

Программе в аргументах командной строки передаются три имени файла. Первый аргумент - входной файл, два остальных - выходные. Реализуйте программу, которая читает символы из первого файла, во второй файл записывает только цифры, а в третий - всё остальное. Разрешается использовать только низкоуровневый ввод-вывод POSIX.
Если входной файл не существует, то нужно завершить работу с кодом 1.
Если не возможно создать один из выходных файлов, то завершить работу с кодом 2.
При возникновении других ошибок ввода-вывода - завершить работу с кодом 3.

Нулевка

Программе в аргументах командной строки передаются три имени файла. Первый аргумент - входной файл, два остальных - выходные. Реализуйте программу, которая читает символы из первого файла, во второй файл записывает только цифры, а в третий - всё остальное. Разрешается использовать только низкоуровневый ввод-вывод POSIX.
Если входной файл не существует, то нужно завершить работу с кодом 1.
Если не возможно создать один из выходных файлов, то завершить работу с кодом 2.
При возникновении других ошибок ввода-вывода - завершить работу с кодом 3.

~~Растягиваем время семинара~~ Life-кодим нулевку))