Lecture materials on system software development

Zhmylev Sergei

Autumn 2019

УНИВЕРСИТЕТ ИТМО

Course structure I

System software

Software interface

Code structure

Code compilation

Error code

A role of the OS

OS booting process

Syscalls

Course structure II

Calling the OS

File

File descriptor

I/O streams of a process

Standard I/O streams

open(2)

Common access modes

lseek(2)

Course structure III

read(2)

write(2)

close(2)

dup(2) и dup2(2)

stat(2)

Errors in syscalls

Error code standardization

Error example

Course structure IV

Headers

Read/write example

Makefile

make utility

Useful functions

I/O with offsets

Useful functions

iovec structure (I/O vector)

Course structure V

Cache flushing

Working with file descriptors

fcntl(2) commands

Access rights check

Access rights modification

Changing file owner

File creation mask

Working with links

Course structure VI

Symbolic links

Working with directories

Current working directory

Reading directory

dirent structure

Device files

Memory model of a process

Memory allocation

Course structure VII

pmap(1) utility

Process

Process states

Useful links

`http://src.illumos.org/`

`https://github.com/mit-pdos/xv6-public`

`https://se.ifmo.ru/~korg/`

`https://vk.com/korglings`

Books :

1. Uresh Vahalia. UNIX Internals
2. A. S. Tanenbaum, A. S. Woodhull. Operating Systems: Design and Implementation

System software

- System software languages
- Syscalls
- I/O
- Threads and processes

Software interface

Each program receives arguments and environment variables

Error code is an integer describing the correctness of program termination

Code structure

```
int main(
   int argc,
   char *argv[],
   char *envp[]
) {
   /* ... */

   return 0;
}
```

Code structure

```
#include <stdlib.h>

int main(
   int argc,
   char *argv[],
   char *envp[]
) {
   /* ... */

   return EXIT_SUCCES;
}
```

Code compilation

```
# gcc -c program.c
# gcc -o program program.o

# gcc -o program program.c

# cc -o program program.c
```

Error code

```
# rm -f /etc/passwd 2<&-
# echo $?
1
# echo Hello, world!
Hello, world!
# echo $?
0
```

A popular mistake

Using void main() is inappropriate!

```
# cat void.c
void main(void) {}
# ./void
# echo $?
16
```

A role of the OS

- Multitasking;
- Memory virtualization;
- Device management;
- Interrupt handling;
- Extending the available set of application-level operations.

OS booting process

- Reset vector: UEFI, BIOS, . . .
- I/O, *PIC(IRQ), VGA
- POST + PCI BIOS
- Boot device detection
- Bootloader stage0 (boot sector)
- Bootloader stage1
- OS kernel

Syscalls

- Kernel functions calling
- Using hardware via the common API
- Have libc interfaces
- Have kernel privileges

Calling the OS

```
/* program termination with errcode 2 */
_exit(2);
```

```
.globl _start
_start:
pushq $2
movq $1, %rax
int $0x80
```

```
# cc -m64 -Wall -Wextra -Wno-comment \
   -nostdlib -o main main.S
```

https://pastebin.com/knTdpZRe

File

What is a file?

Everything is a file!

Apart from threads and the kernel

File descriptor

`http://src.illumos.org/source/xref/illumos-gate/usr/src/uts/common/syscall/open.c#54`

`http://src.illumos.org/source/xref/illumos-gate/usr/src/uts/common/fs/vnode.c#940`

- A file descriptor number is a positive integer that abstracts processes from files they are using.

I/O streams of a process

Number	File	Flags
0	/dev/tty	O_RDWR | O_LARGEFILE
1	/dev/tty	O_RDWR | O_LARGEFILE
2	/dev/tty	O_RDWR | O_LARGEFILE
3	/etc/passwd	O_RDONLY
4	/dev/mtdblock3	O_RDWR
…	…	…
255	…	…

Standard I/O streams

```
# grep FILENO /usr/include/unistd.h
#define STDIN_FILENO 0
#define STDOUT_FILENO 1
#define STDERR_FILENO 2
```

open(2)

```
int open(
   const char *path,  /* file path */
   int oflag,         /* access mode */
   /* mode_t mode */ /* access rights */
);
```

Returns a file descriptor number or an error code

Common access modes

O_RDONLY – Read-only

O_WRONLY – Write-only

O_RDWR – Read-write

O_CREAT – Create if not exists

O_APPEND – Append to the end of the file

O_TRUNC – Write from the beginning of the file

O_LARGEFILE – Long file position

O_EXCL – Long file position

lseek(2)

```
off_t lseek(
   int fildes, /* номер открытого файла */
   off_t offset,    /* смещение позиции */
   int whence               /* действие */
);
```

Returns an updated offset in bytes or an error code

read(2)

```
ssize_t read(
   int fildes, /* номер открытого файла */
   void *buf,           /* буфер чтения */
   size_t nbyte      /* количество байт */
);
```

Returns the amount of bytes read successfully or an error code

write(2)

```
ssize_t write(
   int fildes, /* номер дескриптора */
   const void *buf, /* буфер записи */
   size_t nbyte  /* количество байт */
);
```

Returns the amount of bytes written successfully or an error code

close(2)

```
int close(
   int fildes, /* номер дескриптора */
);
```

Returns zero or an error code

dup(2) и dup2(2)

```
int dup(
   int fildes /* номер открытого файла */
);
int dup2(int fildes, int fildes2);
```

Returns a number of a new file descriptor or an error code

stat(2)

```
int stat(
   const char *restrict path,
      /* путь к файлу*/
   struct stat *restrict buf
      /* результат */
);
```

Returns zero or an error code

Errors in syscalls

A return code of the syscall:

- below zero – an error occured while syscall processing
- equals to zero – successful execution of the syscall
- above zero - the result of the successful exection

Error code standardization

- Error code unification
- `errno` variable
- `perror(3)` function
- `strerror(3)` function

Error example

```
if (read(7, buf, 1) < 0) {
   fprintf(stderr, "%d␣", errno);
   perror("read");
   _exit(1);
}

/* 9 read: Bad file number */
```

Headers

- unistd.h – UNIX declarations
- stdio.h – standard input/output
- fcntl.h – file operations
- sys/types.h – system types
- sys/stat.h – system statuses

Read/write example

```
#include <unistd.h>
int main(int argc, char *argv[]) {
   int bytes;
   char buf[256];
   while((bytes = read(STDIN_FILENO, buf,
   ↪   sizeof(buf))) > 0) {
      if (write(STDERR_FILENO, buf, bytes)
   ↪    < 0) {
         return 1;
      }
   }

   return bytes;
}
```

Makefile

```
PROJS=main
CC=gcc
CFLAGS=-m64

all: $(PROJS)
   @echo Done!

$(PROJS):
   $(CC) $(CFLAGS) -o $@ $(@:=.c)
```

make utility

```
# make
gcc -m64 -o main main.c
Done!
# ./main
Hello, world!
```

Useful functions

- isatty(3C)
- gethostbyname(3NSL)
  gethostbyaddr(3NSL)
- htons(3SOCKET)
  htonl(3SOCKET)
  ntohs(3SOCKET)
  ntohl(3SOCKET)
- usleep(3C)

I/O with offsets

```
ssize_t pread(int fildes, void *buf,
   ↪  size_t nbyte, off_t offset);

ssize_t pwrite(int fildes, const void *buf
   ↪  , size_t nbyte, off_t offset);
```

Returns the amount of bytes or an error code

Useful functions

```
ssize_t readv(
   int fildes,   /* дескриптор файла */
   const struct iovec *iov,
                  /* массив структур */
   int iovcnt /* количество структур */
);
```

Returns the amount of bytes or an error code

iovec structure (I/O vector)

```c
#include <sys/uio.h>
typedef struct iovec {
void *iov_base;
/* start address */
size_t iov_len;
/* segment length */
} iovec_t;
```

Cache flushing

```
void sync(
   void /* не принимает аргументов */
);
```

Return code is meaningless

Working with file descriptors

```
int fcntl(
   int fildes,          /* дескриптор */
   int cmd,                /* команда */
   ... /* переменное число аргументов */
);
```

Return code meaning depends on a particular command

fcntl(2) commands

F_DUPFD / F_DUP2FD	similar to dup/dup2
F_FREESP	free up some space
F_GETFD / F_SETFD	close on exec flag
F_GETFL / F_SETFL	access flags
F_GETLK / F_SETLK	file locking
F_GETLKW / F_SETLKW	

F_RDLCK / F_WRLCK / F_UNLCK

Access rights check

```
int access(const char *path, int amode);
```

R_OK – read
W_OK – write
X_OK – execute
F_OK – existence

Returns zero or an error code

Access rights modification

```
int chmod(const char *path, mode_t mode);
int fchmod(int fildes, mode_t mode);
```

S_ISUID 04000
S_IRWXU 00700
(S_ISUID | S_URWXU) 04700

Returns zero or an error code

Changing file owner

```
int chown(
   const char *path,
   uid_t owner,
   gid_t group
);
int fchown(
   int fildes,
   uid_t owner,
   gid_t group
);
```

Returns zero or an error code

File creation mask

```
mode_t umask(
   mode_t cmask      /* значение маски */
);
```

Returns the previous mask value

How should you get the current value?

Working with links

```
int link(
   const char *existing, /* путь к файлу*/
   const char *new     /* путь к ссылке */
);
int unlink(const char *path);
```

Returns zero or an error code

Symbolic links

```
int symlink(
   const char *name1,
   const char *name2
);

ssize_t readlink(
   const char *restrict path, /* ссылка*/
   char  *restrict buf,       /* буфер */
   size_t bufsiz        /* размер буфера */
);
```

Working with directories

```
int mkdir(
   const char *path, /* путь к каталогу */
   mode_t mode       /* режим доступа */
);
int rmdir(const char *path);
```

Returns zero or an error code

Current working directory

```
int chdir(const char *path);
int fchdir(int fildes);
```

`getcwd(3)` returns either a pointer to a buffer or a -1; has the following prototype

```
char *getcwd(char *buf, size_t size);
```

Reading directory

```
DIR *opendir(const char *dirname);

struct dirent *readdir(DIR *dirp);

void rewinddir(DIR *dirp);

int closedir(DIR *dirp);
```

dirent structure

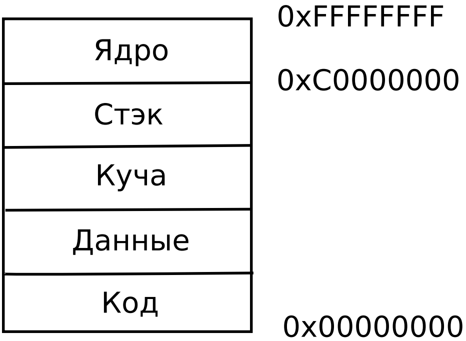
```
typedef struct dirent {
   ino_t d_ino;
      /* номер индексного дескриптора */
   off_t d_off; /* смещение от начала */
   unsigned short reclen;
                      /* длина записи */
   char d_name[];        /* имя файла */
} dirent_t;
```

Device files

```
int mknod(
   const char *path,   /* путь к файлу */
   mode_t mode, /* режим доступа и тип */
   dev_t dev             /* устройство */
);
```

Returns zero or an error code

Memory model of a process



0xFFFFFFFF

Ядро

0xC0000000

Стэк

Куча

Данные

Код

0x00000000

Memory allocation

Data segment extension:

```
int brk(void *endds);
void *sbrk(intptr_t incr);
```

New segment allocation from an Anonymous Memory:

```
void *mmap(
   void *addr,
   size_t len,
   int prot,
   int flags,
   int fildes,
   off_t off
);
```

pmap(1) utility

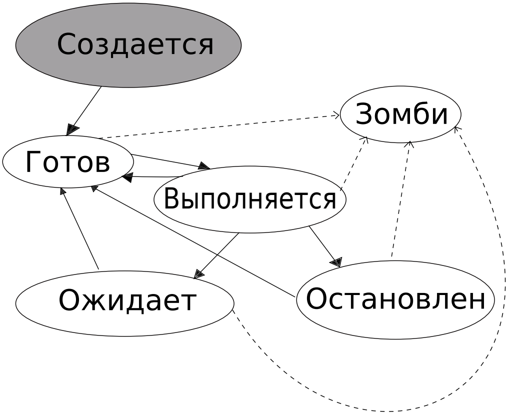
```
helios$ pmap $$
08043000   20K   rw---    [ stack ]
08050000   552K  r-x--    /usr/bin/bash
080E9000   76K   rwx--    /usr/bin/bash
080FC000   300K  rwx--    [ heap ]
FEB20000   64K   rwx--    [ anon ]
FEB40000   56K   r-x--    /lib/module.so
```

Process

A process is an aggregation of a program and a metadata describing the program’s runtime ⓒKorG

Run in parallel; technically independent from each other

Process states



Special thanks to

- Dmitry Borisovich Afanasyev
- Aleksandra Andreevna Gorskaia
- Ksenia Nikolaevna Khovalkina
- Valery Yuryevich Kireev
- Nikita Andreevich Tomilov
- ...and others

Thanks!

```
# perl ’−es!!),−#(−.?{<>−8#=..#<−∗}>;∗7−86)!;y!#() −?{}!\x20/‘−v;<!;s++$_+ee’
```