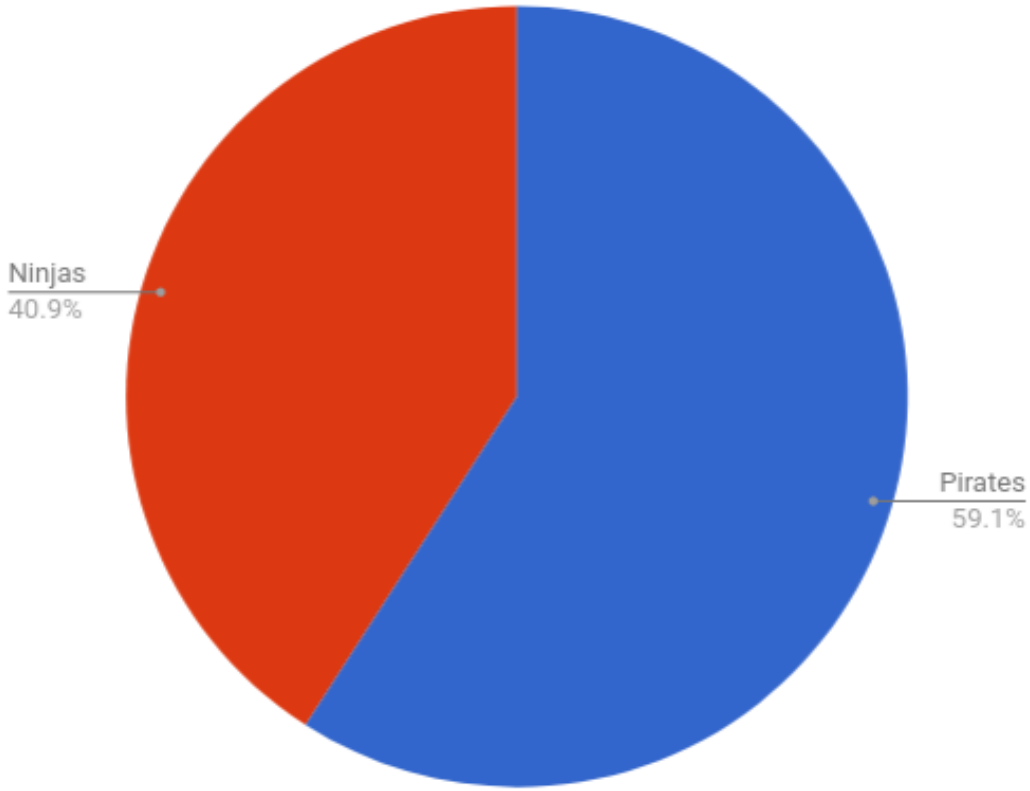
Programovanie v operačných systémoch

02 - Filesystem IO

Jozef Šiška

Department of Applied Informatics
Comenius University in Bratislava

2025/2026


Ninjas
40.9%
Pirates
59.1%

Intro: standards

- ANSI C, ISO C (C89, C99, C11)
- POSIX (IEEE)
- Single Unix Specification (The Open Group Base Specifications 2018 idetical to POSIX, other "standards" for what a UNIX system should be / should have)

- Linux Standard Base (LSB) - linux specific, for binary interoperability between linux distros (filesystem hierarchy, standard libraries, file locations)

Filesystem

- VFS (virtual filesystem)
- mounted "real" filesystems (`mount(2)`)
- files
  - name, data, metadata
  - inode
  - `open` (`creat`), `close`, `read`, `write`, `stat`, ...
- directories ("folders")
  - list of entries (files, directories)
  - POSIX C: `opendir(3)`, `readdir(3)`, `closedir(3)`
  - syscalls: (`open(2)`, `close(2)`,) `getdents(2)`, `readdir(2)`, no C wrapper

Digression: man

```
man man
```

- 1 User Commands
- 2 System Calls
- 3 C Library Functions
- ...

```
man open       # open(2)
man close      # close(2)
man 1 mkdir    # mkdir(1): command
man 2 mkdir    # mkdir(2): C function / syscall
man 1 printf   # printf(1): command
man 3 printf   # printf(2): C function
```

Maybe not a file

- Regular files, directories
- Block and character devices
- Symbolic links
- Pipes
- Sockets

`mknod(2)`	Create "special" (device) files
`symlink(2)`	Create a symbolic link
`pipe(2)`, `pipe2(2)`	Creates a pipe
`socket(2)`	Create a network socket

Filesystem related calls

fd = open(path, flags)	Open/create a file
close(fd)	Close a file
count = read(fd, &buf, count)	Read from file
count = write(fd, &buf, count)	Write to a file
pos = lseek(fd, offset, whence)	Change position
stat(path, &buf), fstat(fd, &buf)	Get file (status) information
access(name, mode)	Check accessibility / existence
chmod(n, mode), chown(n, u, g)	Change permissions, owner
utime(name, times)	Change file access / modification times
umask(mode)	Change file creation mode mask
rename(old, new)	Rename file
mkdir(name, mode), rmdir(name)	Create / remove empty directory
link(n1, n2), unlink(name)	Create link, remove dir entry
mount(special, name, flags)	Mount fs
umount(special)	Unmount fs
sync(), syncfs(), fsync(), fdatasync()	Synchronize file(system) to disk
chdir(dirname), chroot(dirname)	Change current / root directory

. . . and more

dup(fd), dup2(fd,newfd), dup3(..., flags)	Duplicate fd
fcntl(fd, cmd, args. . .)	File "operations"
ioctl(fd, request, args...)	"Special" operations on fd

. . . and more

dup(fd), dup2(fd,newfd), dup3(..., flags)	Duplicate fd
fcntl(fd, cmd, args. . .)	File "operations"
ioctl(fd, request, args...)	"Special" operations on fd

```
int fdIn = open("a.txt", O_RDONLY);
if (fdIn == -1) {
    perror("open"); // prints an error depending on errno
    exit(EXIT_FAILURE);
}

int fdOut;
if ((fdOut = open("a.txt", O_WRONLY | O_CREAT)) == -1) {
    ...
}
```

Aside: error handling

This class will also be about religiously handling every error / return value.

However: because space is precious on slides, examples on slides will almost always ommit it `;-)`.

Get output of a program

```c
int pipefd[2];
pid_t pid;

pipe(pipefd);

pid = fork();
if (pid == 0) { // child
        dup2(pipefd[1], 1);
        execvp(argv[1], argv+1);
} else { // parent
        char buffer[1024];
        ssize_t br = read(pipefd[0], buffer, sizeof(buffer));
        printf("Output: %.*s\n", (int)br, buffer);
        int wstatus;
        if (waitpid(pid, &wstatus, 0) == -1) {
                perror("waitpid");
                exit(EXIT_FAILURE);
        }
        if (WIFEXITED(wstatus)) {
                printf("Exited: %d\n", WEXITSTATUS(wstatus));
        }
}
return 0;
```

Non-blocking IO

- open(..., O_NONBLOCK)
- fcntl on already open fds

```
int flags = fcntl(fd, F_GETFL, 0); // check for -1
fcntl(fd, F_SETFL, flags | O_NONBLOCK); // check for -1
```

- read/write: return EAGAIN, EWOULDBLOCK
- select, poll, epoll: Wait for events on file descriptors
- ioctl(fd, FIONREAD, &bytes_available)

Filesystem io always returns the full file size as available, and will block (if data needs to be read from disk etc.).