# Input & Output

Matt Warner

# Contents

# 1 UNIX System Call

- a system call is how a program prequest a service from the operating system, i.e. UNIX kernal
- System call executes code in the kernal and makes direct use of facilities provided by the kernal.

***Versus:***

- libray function is linked to executable, becomes part of the executable.

## 1.1 System Call Categories

- **File management**
  - create/delete file, open/close, read/write, get/set attributes.
- **Process control**
  - create/terminate process, wait/signal event, allocate/free memory
- **Communication**
  - create/delete connection, send/recieve messages, remote devices
- **Information management**
  - get/set system data and attributes, e.g. time
- **Device management**
  - attach/request/release/detach device, read/write/position.

## 1.2 System Call Invocation

- Declare system call via appropriate C header file
  - Read man page carefully (section 2)
- Prepare parameters using basic C data types
  - No c++ classes, i.e. string
- Prepare suitable return value variable

***Note: Call like any other function***

# 2 File Management

- **open** open a file
- **read** read data from a file
- **write** write data to a file
- **close** close a file
- **creat** make a new file
- **unlink** remove file
- **stat** remove file
- **chmod** change permissions
- **dup** duplicate file descriptor

***All calls share file descriptor, i.e. number, to identify file***

## 2.1 System Call: open

***Syntax:***

```
int open(const char* pathname, int flags)
```

- opens file specified as **pathname** for access
- **flags** determine access types
  - **O_RDONLY** read only
  - **O_WRONLY** write only
  - **O_RDWR** read and write
- returns file descriptor, to be used in read/write/close
- returns -1 on error

- additional **flags,** used with $\mathbf{O}_W RONLY$
  - **O_APPEND** to append to an existing file
  - **O_TRUNC** existing file will overwritten (default)

***Ex:***

**O_WRONLY — O_TRUNC**
**O_WRONLY — O_APPEND**

- additional flag: **O_CREAT**
  - creates file, if file does not exist

## 2.2 System Call: read

***Syntax:***

```
ssize_t read(int fd, void *buf, size_t count)
```

- attempts to read **count** bytes from file descriptor **fd** into the buffer starting at **buf**
  - **ssize_t** and **size_t** are system specific integer types
- returns the number of bytes read
  - maybe smaller than count, zero indicates end of file
  - file position is advanced by this number
- returns -1 on error

## 2.3 System Call: close

***Syntax:***

```
int close(int fd)
```

Closes file specified by **fd** file descriptorMakes file descriptor available

- returns zero on success

## 2.4 System Call: write

***Syntax:***

```
ssize_t write(int fd, const void *buf, size_t count)
```

- Writes up to count bytes from buffer starting at **buf** to the file referred to by file descriptor **fd**
  - **ssize_t** and **size_t** are system specific integer types
- Returns the number of bytes written
  - may be smaller than count
- returns -1 on error

# 3 System Call: open with O_CREAT

- must specify file mode, i.e. permissions, of type **mode_t**
  - **S_IRWXU** 00800 user has read, write and execute permission
  - **S_IRUSR** 00400 user has read permission
  - **S_IWUSR** 00200 user has write permission
  - **S_IXUSR** 00100 user has execute permission
  - **S_IRWXG** 00070 group has read, write and execute permission
  - **S_IRWXO** 00007 others have read, write and execute permission
- C++ requires leading '0' for octal numbers
- actual file mode is further modified by umask

**Example:**

```
open("ex.txt", O_WRONLY | O_APPEND | O_CREAT, 00666)
```

## 3.1 System Call: creat

***Syntax:***

```
int creat(const char *pathname, mode_t mode)
```

- creates new file specified as **pathname** and opens file for write access
- returns file descriptor
- returns -1 on error

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <iostream>

int main(int argc, char** argv) {

  int ifd, ofd, count, sum=0;
  char buffer[1024];

  // check command line args
  if (argc < 3) {
    std::cerr << "Usage: copy fromFile toFile\n";
    exit(EXIT_FAILURE);
  }

  // open file to read
  ifd = open(argv[1], O_RDONLY);
  if(ifd == -1) {
    perror(argv[1]);
    exit(EXIT_FAILURE);
  }

  // open file to write
  ofd = open(argv[2], O_WRONLY | O_TRUNC | O_CREAT, 00666);

  // loop to read/write buffer
```

```
  while ((count = read(ifd, buffer, sizeof(buffer))) != 0) {
    if ((count == -1)) {
      perror(argv[1]);
      exit(EXIT_FAILURE)
    }
    count = write(ofd, buffer, count);
    if (count == -1) {
      perror(argv[2]);
      exit(EXIT_FAILURE)
    }
  }
  cout << "copied " << sum << "bytes from " << argv[1] << " to " << argv[2] << endl;
  // close all files
  close(ifd);
  close(ofd);
}
```