# Low-Level File I/O

Marco Gruteser

Advanced Programming for Linux



# File I/O Functions

- C Function / Macro for every system call
  - Open
  - Read
  - Write
  - Lseek
  - Close
- Unbuffered I/O
- Every read or write call invokes system call
- Open files are identified by *file descriptor* (nonnegative integer)
- Shells follow this convention
  - 0 stdin
  - 1 stdout
  - 2 stderr



# open

- Open(const char* pathname, int oflag, ...)
- Oflags
  - Required: O_RDONLY, O_WRONLY, or O_RDWR
  - Optional O_APPEND, O_CREAT, O_SYNC O_EXCL (w/ O_CREAT atomic test and create)
- Creat no longer needed



# Close /seek

- Close
  - Cleans up kernel data structures
  - Files automatically closed if program ends
- Seek
  - Moves file pointer/offset
  - Returns current offset
  - Returns -1 if cannot seek (pipe / fifo)

# File with hole

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

char buf1[] = "abcdefghij";
char buf2[] = "ABCDEFGHIJ";

int
main(void)
{
    int                 fd;

    if ( (fd = creat("file.hole", FILE_MODE)) < 0)
        err_sys("creat error");

    if (write(fd, buf1, 10) != 10)
        err_sys("buf1 write error");
    /* offset now = 10 */

    if (lseek(fd, 40, SEEK_SET) == -1)
        err_sys("lseek error");
    /* offset now = 40 */

    if (write(fd, buf2, 10) != 10)
        err_sys("buf2 write error");
    /* offset now = 50 */

    exit(0);
}
```



# I/O Efficiency



# File copy

```
#include    "ourhdr.h"

#define     BUFFSIZE    8192

int
main(void)
{
    int                 n;
    char    buf[BUFFSIZE];

    while ( (n = read(STDIN_FILENO, buf, BUFFSIZE)) > 0)
        if (write(STDOUT_FILENO, buf, n) != n)
            err_sys("write error");

    if (n < 0)
        err_sys("read error");

    exit(0);
}
```



# Buffer effect (read only)

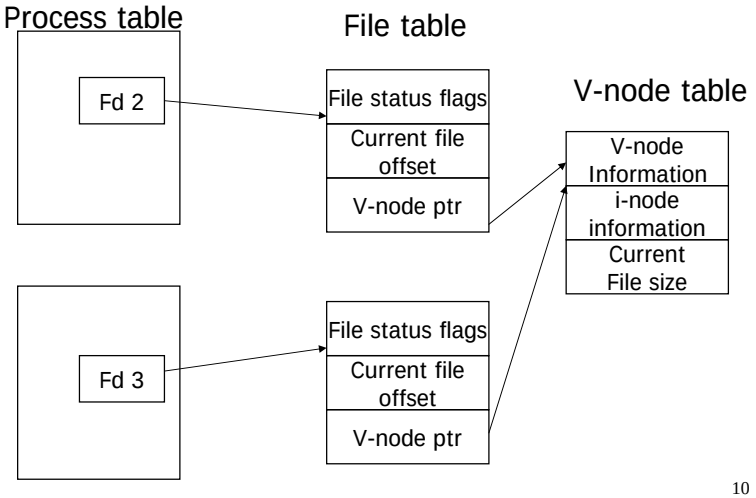
- Size usr(s) sys(s) clk(s)
  1 23.8 397.9 423.4
  8 3 50.7 54
  64 0.3 6.6 7.0
  512 0 1 1.1
  64K 0 0.3 0.3

# File Sharing

- Files can be opened multiple times
  - By same or different processes



# Kernel File Data Structures






# Atomic Operations

- Append to file
  - Lseek(fd, 0L, 2)
  - Write(fd, buf, 100)
- Not atomic:
  - P1: Lseek(fd, 0L, 2)
  - P2: Lseek(fd, 0L, 2)
  - P1: Write(fd, buf1, 100)
  - P2: Write(fd, buf2, 100)
- Solution: O_APPEND



# Atomic Operations (2)

```
Fd = open(pathname, O_WRONLY)
If (errno == ENOENT) {
  Fd= creat(pathname, mode)
```

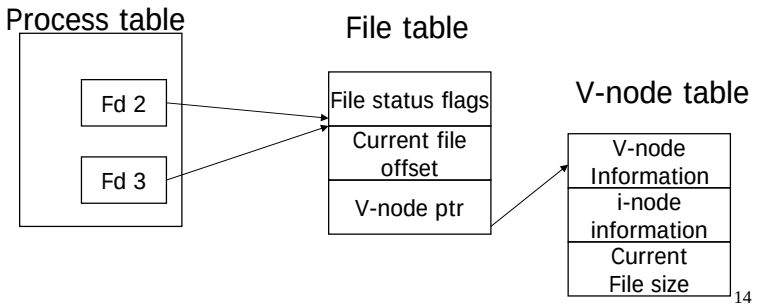
Solution?

# Solution?



# Dup / fcntl

- newFD=Dup(FD) *returns lowest*
- newFD=Dup2(srcFD,destFD) *atomic*






# Difference to calling open twice?



# Fcntl to modify file status flags

```
#include    <sys/types.h>
#include    <fcntl.h>
#include    "ourhdr.h"

int
main(int argc, char *argv[])
{
    int                 accmode, val;

    if (argc != 2)
        err_quit("usage: a.out <descriptor#>");

    if ( (val = fcntl(atoi(argv[1]), F_GETFL, 0)) < 0)
        err_sys("fcntl error for fd %d", atoi(argv[1]));

    accmode = val & O_ACCMODE;
    if  (accmode == O_RDONLY)   printf("read only");
    else if (accmode == O_WRONLY)   printf("write only");
    else if (accmode == O_RDWR)         printf("read write");
    else err_dump("unknown access mode");

    if (val & O_APPEND)                 printf(", append");
    if (val & O_NONBLOCK)               printf(", nonblocking");
#if  !defined(_POSIX_SOURCE) && defined(O_SYNC)
    if (val & O_SYNC)                   printf(", synchronous writes");
#endif
    putchar('\n');
    exit(0);
}
```

Using bitmasks

# Setting File Status Flags

```
#include  <fcntl.h>
#include  "ourhdr.h"

void
set_fl(int fd, int flags) /* flags are file status flags to turn on */
{
    int                 val;

    if ( (val = fcntl(fd, F_GETFL, 0)) < 0)
        err_sys("fcntl F_GETFL error");

    val |= flags;               /* turn on flags */

    if (fcntl(fd, F_SETFL, val) < 0)
        err_sys("fcntl F_SETFL error");
}
```

GETFL, SETFL necessary



# O_SYNC - Read times

- Useful for reliability (ensure that data is on disk)
- Performance penalty
- Measurements 1.5MB, 8K Buffer (old)

| Operation | Usr(s) | Sys(s) | Clk(s) |
|---|---|---|---|
| read only | 0 | 0.3 | 0.3 |
| read/write | 0 | 1.0 | 2.3 |
| r/w O_SYNC | 0 | 1.4 | 13.4 |



# Why access mode flags?

- O_RDONLY
- O_WRONLY
- O_RDWR



# Why access mode flags?

- Access control checks only on open, fcntl
- File locking?

# Bash redirection

- > redirect stdout to file
- < redirect stdin to file
- 2> redirect stderr to file
- >> append stdout to file
- 2>&1 redirect fd2 to fd1



# What will y contain?

Rm *; Touch b

Ls a b >out
Ls a b >out 2>out
Ls a b >out 2>>out
Ls a b >>out 2>out
Ls a b >out 2>&1



# Next Class

- Filesystem structures
- Read Chapter 4