

Input/Output and Files

- background reading to support this material can be found in
 - Ritchie Chapter 8
 - Silberschatz 13.3, 11.1 and 11.6

Input/Output

- I/O Device Characteristics
 - range of characteristics/performance to be handled by OS
 - Data Rates
 - disks: 100 MB/sec
 - keyboards: 10 B/sec
- Transfer units
 - disks: blocks of data
 - serial interfaces: byte streams

Input/Output

- Example operations
 - disks: read/write/seek
 - graphics card: resolution, refresh rate
- Error conditions
 - disks: read errors
 - monitor: refresh rate not supported
- I/O devices use interrupt techniques and DMA transfer to allow data transfer to be undertaken without direct processor control

I/O System Objectives

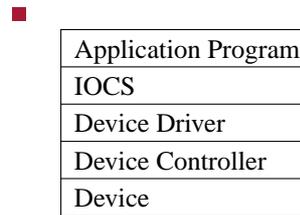
- Efficiency
 - I/O systems are often mechanical and operate in millisecond range
 - processors operate in fractions of microsecond range
- I/O system must attempt to minimise problems caused by this disparity and allow system to operate at maximum speed where possible

Device independence and abstraction

- Inherent complexities of I/O devices should be hidden from users/programmers by OS.
- Software written to interface to devices should be independent of inherent complexities of devices.

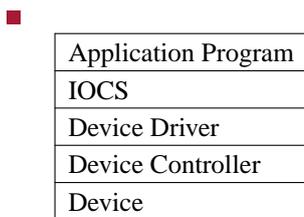
I/O System Structure

- Layers of software exist in OS which interface application to device.



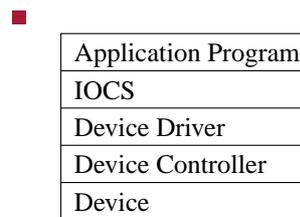
- Application program
 - I/O activity is initiated by programming language features such as read and writing to files.
 - High level language statements are translated into system calls.

I/O System Structure



- IOCS
 - Input-Output Control System is the part of the operating systems which deals with I/O related system calls.
 - Validates requests and passes them on to appropriated handler.

I/O System Structure



- Device drivers
 - software modules which manage communication with, and control a specific I/O device or in some cases deal with a range of similar devices.

I/O System Structure

- | |
|---------------------|
| Application Program |
| IOCS |
| Device Driver |
| Device Controller |
| Device |

- Device controller
 - hardware interface between computer and I/O device.

I/O System Structure

- | |
|---------------------|
| Application Program |
| IOCS |
| Device Driver |
| Device Controller |
| Device |

- Device
 - Wide range of these, sometimes complex.
 - The role of the above I/O system structure (hence OS) is to provide a simpler interface, hiding the complexities of the device from applications.

I/O Buffering

- Buffering is a common method of improving efficiency of I/O transfers.
- A comparison of un-buffered transfers and buffered will be undertaken in the accompanying lab session.

File Management

- Examples of file types in Unix
 - Regular
 - Directory
 - Special
 - Pipe

Example of File access control in Unix

- Unix file permissions
 - r - read
 - w - write
 - x - execute
- relating to
 - u - user
 - g - group
 - o - others

Example of File access control in Unix

- Use ls to view permissions, eg. ls -la

```
$ ls -la
total 28
drwxr-xr-x  2 fred fred  4096 Feb 23 12:22 .
drwxr-xr-x  3 fred fred  4096 Feb 23 12:20 ..
-rwxr-xr-x  1 fred fred  4722 Feb 23 12:20 a.out
-rw-r--r--  1 fred fred   68 Feb 23 12:20 hello.c
-rw-r--r--  1 fred fred    0 Feb 23 12:22 ls.listin
-rwxr--r--  1 fred fred   85 Feb 23 12:21 phone
-rw-r--r--  1 fred fred   30 Feb 23 12:22 phonedata
```

- use chmod to change, eg. chmod 666 file_name

```
$ chmod 666 a.out
$ ls -l a.out
-rw-rw-rw-  1 fred fred  4722 Feb 23 12:20 a.out
```

Example of File access control in Unix

- we could also have used

```
$ chmod ugo+rw a.out
$ chmod ugo-x a.out
$ ls -l a.out
-rw-rw-rw-  1 fred fred  4722 Feb 23 12:20 a.out
```

- using the octal approach is quicker!

File Identification

- File Id
 - Traditionally Name + Extension
 - Extension attempts to standardise the "type" of file eg. .txt, .doc, .html,...
- Path
 - Method used to locate a file or directory
 - In Unix the directories to search for executables is defined eg.


```
PATH=/usr/bin:/usr/local/bin:/.
```
- The pathname of a file is its full name, including its directory hierarchy ie. /home/user1/file1.ext.
- In some contexts only the file name is required.

File System Services

- On-line facilities
- file manipulation commands
- tools
 - Programming services

Using system calls to access files in UNIX

- | system call | summary |
|-------------|----------------------------------|
| open | open a file or device |
| creat | open and create a file or device |
| read | read from an open file or device |
| write | write to an open file or device |
| close | close a file or device |
| exit | return control to OS |
- see man pages for more detail (also recall the lecture on system calls)

Using system calls to access files in UNIX

- File descriptor
 - Each process has a number of file descriptors associated with it.
 - These are small integers used to access open files or devices.
 - Standard file descriptors are:
 - 0 - standard input
 - 1 - standard output
 - 2 - standard error
- Other values of file descriptor are returned by open system call

Example copy program using these system calls

- ```

#include <sys/stat.h>
#include <stdlib.h>
#include <fcntl.h>

int main ()
{
 char c;
 int in, out;
 in = open ("in.dat", O_RDONLY);
 out = creat ("out.dat", 0666);
 while (read(in, &c, 1) == 1)
 {
 write (out, &c, 1);
 }
 close (in);
 close (out);
 exit (0);
}

```

## Example copy program using these system calls

- basic system calls follow

## open system call

- ```
int open(const char *pathname, int oflags);
```
- pathname is path of file name to open, oflags specifies file access mode eg. O_RDONLY (open read only), O_WRONLY (open write only), O_RDWR (open read and write).
- A file descriptor is returned if call was successful.

close system call

- ```
int close(int fildes)
```
- Terminates association between file and file descriptor, fildes.

## creat system call

- ```
int creat(const char *pathname, int mode);
```
- pathname is path of file name to create and open for writing, mode specifies file permissions to be set.

read system call

```
int read(int filedes, char *buf, int count);
```

- Reads up to count bytes of data from filedes and places in data area called buf.
 - Returns number of bytes actually read. 0 is returned on EOF.

write system call

```
int write(int filedes, const char *buf, int count);
```

- First count bytes are written from buf to filedes.
 - Number of bytes actually written is returned. 0 is returned on EOF.
- Performance of file I/O can be greatly improved by use of buffering.
- This will be explored in the accompanying lab session

exit system call

```
void exit(int status);
```

- `exit` returns control to the host operating system.
 - status contains 0 on normal termination
 - non-zero if an error is required to be indicated
 - (see `fork/exec` example from a previous lecture).

Example copy program

```

#include <sys/stat.h>
#include <stdlib.h>
#include <fcntl.h>

int main ()
{
    char c;
    int in, out;
    in = open ("in.dat", O_RDONLY);
    out = creat ("out.dat", 0666);
    while (read(in, &c, 1) == 1)
    {
        write (out, &c, 1);
    }
    close (in);
    close (out);
    exit (0);
}

```