CSS 503 A Lecture # 9 2019-04-29

Administrivia

OSSF

- Banker's algorithm
  - safety: given max requests & current allocation, determine whether there exists a sequence of allocations (& program terminations) that will allow all processes run to completion
    ⇒ system is in safe state
  - allocation: determine whether allocation request, if granted, would keep the system in a safe state

- algorithm is "worst-case": as if processes run sequentially - actual allocation would be ~~less~~ More inefficient

OSSF (cont.)

- peripheral devices: many properties
  - character vs. block
  - sequential vs random access
  - read-only / write-only / read-write

- disk drive: (cylinder, platter, sector)
  → device driver: block 0 .. n-1

- file system: data structure on top of block sequence

- Unix-derived systems: hierarchical system of directories & files

- meta data: file attributes
  name, type, size, protection/permissions, time/date, ownership
  (disk is shared by multiple users)

- file formats
  - operating system-dependent — complex
  - Unix-derived systems: simplicity, simplicity unstructured
    - file is sequence of bytes — unstructured
    - structure is application-specific
    - convention: text files have newline-delimited records (aka lines)

OSSF (cont)

- file operations
  - open()
  - read()
  - write()
  - close()
  - tell()
  - seek()
  - ioctl()
  - fcntl()
  - ...

OSSP (Cont.)

optional, used only if newfile is created

- creat(const char* pathname, int flags)
- open(const char* pathname, int flags, mode_t mode)

⇒ more complex than you might expect
  - open only if file exists
  - open only if file does not exist (create) — atomic → used for synchronization
  - open only if exists
  - open for reading
  - open for writing
    - truncate, append
  - open for async I/O ops
  - open for non-blocking I/O (pipes, devices)

- flags: { symbolic values, powers of 2; bitwise or-ed together
  - open mode: O_RDONLY, O_WRONLY, O_RDWR
  - other flags: O_APPEND, O_CREAT, O_EXCL, O_TRUNC, ...

- mode: permission bits
  - same flags as chmod(2)

File I/F (cont.)

read() - read specified # bytes, advance file pointer
write() - write specified # bytes, advance file pointer
close() - flush kernel buffers to disk, release resources
tell() - return current position of file pointer
seek() - reposition file pointer
ioctl() - device interface
fcntl() - file (kernel data structure) parameters

repositioning the file pointer

```
off_t lseek(int fd,
            off_t offset,
            int whence)
```

Whence

SEEK_SET	0	(from beginning) +ve
SEEK_CUR	1	(from current position)
SEEK_END	2	(from end, -ve value)

↓ +ve or -ve

llseek

~~lseek64~~ : handles larger files

File Interface (cont.)

request-specific parameters

int ioctl(int fd, unsigned long request, ...)

- device drivers: section 4 of the manual

man(4) tty

sd

advisory file locks

- flock()

File Interface (cont.)

Metadata Syscalls

- unlink
- rename
- stat
- chmod

Higher-Level File I/O libraries

- buffering
  - efficiency
- formatting
  - convenience

C (stdio):

```
FILE* f = fopen(fname, "r")
          fopen(fname, "w")
fread(void* p
      size_t size,
      size_t nmemb,
      FILE* f)
fwrite(const void* p
       size_t size,
       size_t nmemb,
       FILE* f)
fclose(FILE* f)
```

C++

```
fstream f( fname, ios::in)
f( fname, ios::out)
f << buf
f >> buf
f.close()
```

Java

```
FileInputStream inputStream =
        new FileInputStream(fileName)
ObjectInputStream input =
        new ObjectInputStream( fileInputStream.getInputStream() )
Object ob = input.objectRead()

FileOutputStream outputStream =
        new FileOutputStream( fileName )
ObjectOutputStream output =
        new ObjectOutputStream(
                output.getOutputStream())
output.objectWrite( Object ob, )

f.close()
```

Memory mapped file

- mmap (2)