Principles of Computer System

1

Lecture 1: Welcome to CS110

★ Introduction to Unix file system

⇒In C, a file can be created using the open system call, and you can set the permissions at the time, as well.

```
int open(const char *pathname, int flags);
int open(const char *pathname, int flags, mode_t mode);
```

⇒There are many flags, and they can be bitwise or'd together:

↳You must include only one of the following flags:

O_RDONLY --- read only
O_WRONLY --- write only
O_RDWR --- read and write

↳We will generally only care about the following other flags when creating a file:

O_CREATE --- If the file doesnot exist, it will be created.
O_EXCL --- Ensures that this call creates the file, and fails if the file already exists.

⇒ The third argument, mode is used to attempt to set the permissions.

⇒ There is a default permissions mask, called umask, that limits the permissions.

⇒The umask can be set with the following system call:

```
mode_t umask(mode_t mask);
```

⇒ Note:
- An integer literal that starts with 0 is an octal number, much like a number starting with 0xis a hexadecimal number.

⇒ If mode == 0644

↳Other: 4 => 100 => r--

↳Group: 4 => 100 => r--

↳Owner: 6 => 110 => rw-

	rwx
0	--- 000
1	--- 001
2	--- 010
3	--- 011
4	--- 100
5	--- 101
6	--- 110
7	--- 111

⇒ command $ errno -l will give list of error numbers and there meaning.

⇒ read and write are defined as follows:

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

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

- It is decleared inside unist.h
- buf is just char* array.
- count is the number of bytes to read or write
- returned value is a ssize_t, which is the same magnitude as size_t, but with the ability to have negative value.
- A return value -1 indicates an error, and errno is set appropriately.

⇒ The return value is not always the same as count, but only the number of bytes successfully read or written.

⇒ FILE pointers and c++ iostreams works well when you know you're interacting with standard output, standard input, and local files.

- They are less useful when the stream of bytes is associated with a network connections.

(2)

Lecture 2: Files System

Umask

- umask is set for the user in the shell, and when the program is run it inherits the users umask settings.
- The bits that are set in the umask disable creating permissions for the permission bits that the program is attempting to set.

★ Using Stat and lstat

⇒ stat and lstat are system calls, that populates a struct stat with info about some file name (regular file, a directory, a symbolic link etc)

```
int stat(const char *pathname, struct stat *st);
int lstat(const char *pathname, struct stat *st);
```

⇒ stat and lstat works exactly the same way, except when the named file is a link stat returns information about the file the link reffers, and lstat returns information about the link itself.

- the **`struct stat`** contains the following fields (source)

```
struct stat {
  dev_t st_dev;        // ID of device containing file
  ino_t st_ino;        // file serial number
  mode_t st_mode;      // mode of file
  // many other fields (file size, creation and modified times, etc)
};
```

⇒ S_ISDIR is a macro which examins the upper four bits of the st_mode field to determine whether the named file is a directory.

- S_ISREG decides whether file is regular file
- S_ISLNK decides whether file is a link

opendir

- Input: directory
- Output: pointer to an opaque iterable that surfaces a series of struct dirent via a sequence or readdir call.
  - If **`opendir`** accepts anything other than an accessible directory, it'll return **`NULL`**.
  - When the **`DIR`** has surfaced all of its entries, **`readdir`** returns **`NULL`**.
- opendir returns access to a record that evantually must be released via a call to closedir.

Lecture 3: Layering, Naming and Filesystem design

⇒ Just like RAM, hard drives provides us with a continuous streach of memory where we can store information.

⇒ Information in RAM is byte addressable.

⇒ A similar concept exist in the world of hard drives.

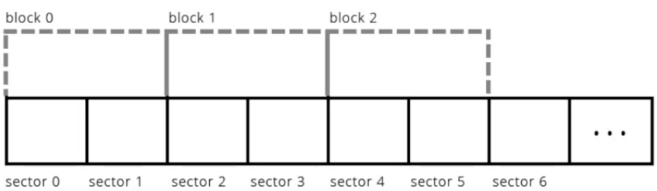
- Hard drives are divided into sectors (often 512 bytes) and are sector addressable.

- Conceptually, a hard drive might be viewed like this:

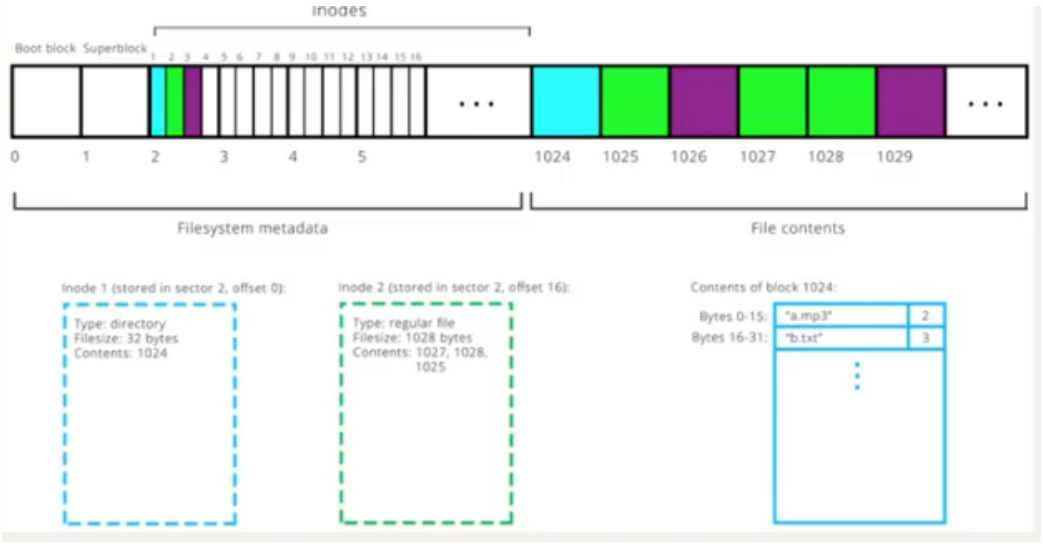
⇒ The drive itself exports an API that allows us to read a sector into main memory or update an entire sector with a new payload.

```
1 class Drive {
2 public:
3     size_t getNumSectors() const;
4     void readSector(size_t num, unsigned char data[]) const;
5     void writeSector(size_t num, const unsigned char data[]);
6 };
```

⇒ Operating system generally frames its operations in terms of blocks



{ Comprises of one or more sectors }



BootBlock

- Contains information about the hard drive itself
- It is named so because its content are tapped when booting

SuperBlock

- It contain information about the filesystem imposing itself onto the hardware

Inode

- It is a 32 byte data structure that stores metadata about a single file.
- In unix V6 filesystem, inodes can only store a maximum of 8 block numbers.

⇒ File payload are stored in quantums of 512bytes

⇒ Directore is a file which has list of name of files and inode numbers associated with it.

- 16 byte per file (14 for name and 2 for inode numbers)

⇒ If inode can only store max of 8 block numbers, then this limits the total file size to 8*512=4096 bytes.

- This is way too small for any reasonable file.

⇒ To resolve this problem we use a scheme called indirect addressing.

- We set a flag specifying that we are using indirect addressing
- In indirect addressing the blocks in an inode points to block which contains pointers pointing to other 264 bolcks.
- This increases max filesize to 8*256*512=1,048,576(1MB)

⇒ 1MB is still not that big. To make the max file size even bigger, Unix V6 uses the 8th block number of the inode to store a doubly indirect block number.

- This increases max filesize to (7+256)*256*512=34,471,936(34MB)
- This is still not big as per present standards but Unix V6 file system was designed in 1975, at that time it was ok.

(4)

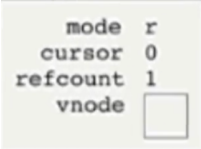
Lecture 4: File System data Structure & System Calls

⇒ Linux maintain a data structure for each active process called "process control block".

- → These are stored in "process table"
- → It contain many things such who launched it etc and descriptor table.
  - → Descriptor 0,1,2 are understood to be treated as standard input, standard output and standard error.
  - → Program sees descriptor as the identifier needed to interact with a resource via read, write and close system calls.

⇒ If a descriptor table entry is in use, it maintains a link to an open file table entry.

- → It maintains information about an active sesson with a file.



⇒ Each process maintain its own descriptor table, but there is only one, system wide open file table.

⇒ vnode itself is a structure housing information about a file.

⇒ There is one system-wide vnode table

★ System Calls

⇒ System calls are functions that our program use to interact with the os and request some core service be executed on there behalf.

⇒ A modern computer operating system usually segregates virtual memory into:

- → Kernel space — memory reserved for running a privileged operating system kernel
- → User space — memory area where application software executes

⇒ For system call system issues a software interrupt by executing syscall, which prompts an intterupt handler to execute in superuser mode.

★ Multiprocessing

process

- When a program runs its a single process
- It has a process id that the os assigns
- A program can get its own pid with getpid system call

fork

- when a program wants to launch a second process, it uses the fork syatem call
- It creates an new process that starts on the following instruction after the original, parent process
- fork call returns returns a pid to both process
  - Nither is the actual pid of the process
  - The parent process gets a return value that is pid of the child process
  - The child process gets a return value of zero, indicating that it is child
- All memory is identical between the parent and the child, though it is not shared (its copied)
- getpid and getppid returns the process id of the caller and of the callers parent.

(5) Lecture 5: Understanding execvp

waitpid

- Can be used to temporarly block one process until a child process exits.

```
pid_t waitpid(pid_t pid, int *status, int options);
```

execvp

- Most often a programmer wants to run a compleatly separate program, but wants to maintain cotrol over the program.
  - May also want to send data to the program through stdin and capture the output of the program through its stdout.
- execvp effectively reboots a process to run a different program from scratch.

```
int execvp(const char *path, char *argv[]);
```

6 execvp, Pipes and Interprocess Communication

pipe

- pipe system call takes an uninitialized array of two integers (lets call it fda) and populates it with two file descriptors such that everything written to fds[1] can be read from fds[0].

```
int pipe(int fds[]);
```

- pipe is particularly useful for allowing parent process to comminicate with spawned child process.

⇒In Unix-like operating systems, dup (short for "duplicate") and dup2 system calls create a copy of a given file descriptor.

★ Signals

⇒ A Signal is a small message that notifies a process that an event of some type occurred.

⇒ A Signal handler is a function that executes in response to the arrival and consumption of the signal.

⇒ Each signal catagory is represented internally by some number.

7 Signals

SIGSEGV

- Known as segmentation fault.
- Kernel delivers this signal to the program when there is segmentation fault.
- Default action: Terminate the program and generate core dump.

SIGFPE

- Whenever a process commits an divide by zero error.

SIGINT

- When you press ctrl+c, the kernel sends this message to the foreground process.
- Default: Forground process will be terminated.

SIGTSTP

- When you press ctrl+z, the kernel sends this message to the foreground process.
- Default: Foreground process is haulted until a subsequent SIGCONT signal.

SIGCHILD

↳ Whenever a child process changes state: it exits, crashes, stops, or resumes from a stopped state, the kernel sends a SIGCHILD signal to the process parent.

⇒ The kernel provides directives that allow a process to temporarly ignore signal delivery.

```
int sigemptyset(sigset_t *set);
int sigaddset(sigset_t *additions, int signum);
int sigprocmask(int op, const sigset_t *delta, sigset_t *existing);
```

⇒ Processes can message other processes using signals via the kill system call.

⇒ And process can send themselves signals using raise.

```
int kill(pid_t pid, int signum);
int raise(int signum); // equivalent to kill(getpid(), signum);
```

(8)

Race Conditions, Deadlock, and Data Integrity

sigsuspend(sigset_t* set)

↳ Asks the OS to change the blocked set to the one provided, but only after the caller has been forced off the CPU.

↳ When some unblocked signal arrives, the process gets the CPU, the signal is handled, the orignal blocked set is restored and sigsuspend returns.

atomic: If something is atomic means, it happens without something is able to interrupt it.

(10)

Introduction to Threads

⇒ Thread is an independent execution sequence within a single process.

⇒ The stack segment is subdivided into multiple miniature stacks one for each thread.

⇒ Thread is often called a lightweight processes.

⇒ Each thread maintains its own stack, but all threads share the same heap segments and globals.

⇒ One thread can share its stack space (via pointers) with other.

⇒ Virtual address space is shared between threads.

⇒ ANIS C dosent provide support for threads.

⇒ But pthread, comes with all standard UNIX and LINUX installations of gcc.

⇒ The primary pthreads data type is pthread_t.

⇒ pthread functions

- pthread_create
- pthread_join

11 From C thread to C++ thread

std::thread

std::mutex

12 Multithreading and Conditional Variables

std::lock_guard

- The class lock_guard is a mutex wrapper that provides a convenient RAII-style mechanism for owning a mutex for the duration of a scoped block.
- When a lock_guard object is created, it attempts to take ownership of the mutex it is given.
- When control leaves the scope in which the lock_guard object was created, the lock_guard is destructed and the mutex is released.
- Example use: `lock_guard<mutex> lg(m);`

```
class condition_variable_any {
public:
   void wait(mutex& m);
   template <typename Pred> void wait(mutex& m, Pred pred);
   void notify_one();
   void notify_all();
};
```

Semaphore

- A semaphore is a variable or abstract data type used to control access to a common resource in a concurrent system.

(14) Introduction to Networking

⇒ Networking is simply communicating between two computers connected on a network.

↳ You can actually set up a network connection on a single computer within processes.

⇒ Network requires one computer to act as:

Server

↳ Waiting patiently for an incoming connecton from another computer called:

Client

⇒ Server side applications set up a socket that listens to a particular port.

↳ The server socket is an integer identifier associated with a local IP address.

↳ Port number is 16 bit integer.

⇒ netstat -plnt : To see the ports your computer is listning to.

⇒ Port number in the range 0 - 1023 are the well known ports.

⇒ Port number in the range 1024 - 49151 are the registered ports.

⇒ telnet <machine> <port_number> to connect to the <machine> using port <port_number> using terminal.

ANSI Escape sequences

↳ ANSI escape sequences are a standard for in-band signaling to control the cursor location, color, and other options on video text terminals and terminal emulators.