Operating system interfaces

- Share hardware across multiple processes
  - Illusion of private CPU, private memory
- Abstract hardware
  - Hide details of specific hardware devices
- Provide services
  - Serve as a library for applications
Typical UNIX OS
System calls

- Provide user to kernel communication
  - Effectively an invocation of a kernel function

Interface for:

- Processes
  - Creating, exiting, waiting, terminating
- Memory
  - Allocation
- Files and folders
  - Opening, reading, writing, closing
- Inter-process communication
  - Pipe
Shell

- Normal process
- Interacts with the kernel through system calls
  - Creates new processes
fork() -- create new process

int pid;

pid = fork();
if(pid > 0){
    printf("parent: child=%d\n", pid);
    pid = wait();
    printf("child %d is done\n", pid);
} else if(pid == 0){
    printf("child: exiting\n");
    exit();
} else {
    printf("fork error\n");
}
More process management

- `exit()` -- terminate current process
- `wait()` -- wait for the child to exit
- `exec()` -- replace memory of a current process with a memory image (of a program) loaded from a file

```c
char *argv[3];
argv[0] = "echo";
argv[1] = "hello";
argv[2] = 0;
exec("/bin/echo", argv);
printf("exec error\n");
```
Xv6 demo
File descriptors

- An index into a table, i.e., just an integer
- The table maintains pointers to “file” objects
  - Abstracts files, devices, pipes
  - In UNIX everything is a pipe – all objects provide file interface
- Process may obtain file descriptors through
  - Opening a file, directory, device
  - By creating a pipe
  - Duplicating an existing descriptor
Standard file descriptors

• Just a convention
  • 0 – standard input
  • 1 – standard output
  • 2 – standard error

• This convention is used by the shell to implement I/O redirection and pipelines
File I/O

- `read(fd, buf, n)` – read `n` bytes from `fd` into `buf`
- `write(fd, buf, n)` – write `n` bytes from `buf` into `fd`
Example: cat

    char buf[512]; int n;
    for(;;) {
        n = read(0, buf, sizeof buf);
        if(n == 0)
            break;
        if(n < 0) {
            fprintf(2, "read error\n");
            exit(); }
        if(write(1, buf, n) != n) {
            fprintf(2, "write error\n");
            exit();
        }
    }
File I/O redirection

- `close(fd)` – closes file descriptor
  - The next opened file descriptor will have the lowest number
- `fork` replaces process memory, but
  - leaves its file table (table of the file descriptors untouched)
Example: `cat < input.txt`

```c
char *argv[2];
argv[0] = "cat";
argv[1] = 0;
if(fork() == 0) {
    close(0);
    open("input.txt", O_RDONLY);
    exec("cat", argv);
}
pipe - interprocess communication

- Pipe is a kernel buffer exposed as a pair of file descriptors
  - One for reading, one for writing
- Pipes allow processes to communicate
  - Send messages to each other
int p[2];
char *argv[2]; argv[0] = "wc"; argv[1] = 0;
pipe(p);
if(fork() == 0) {
    close(0);
    dup(p[0]);
    close(p[0]);
    close(p[1]);
    exec("/bin/wc", argv);
} else {
    write(p[1], "hello world\n", 12);
    close(p[0]);
    close(p[1]);
}
Pipelines

- Shell implements pipelines with pipes, e.g.
  
  ```
  grep fork sh.c | wc -l
  ```

- Create a pipe and connect ends
Files

- Files
  - Uninterpreted arrays of bytes
- Directories
  - Named references to other files and directories
Creating files

- **mkdir()** – creates a directory
- **open(O_CREATE)** – creates a file
- **mknod()** – creates an empty files marked as device
  - Major and minor numbers uniquely identify the device in the kernel
- **fstat()** – retrieve information about a file
  - Named references to other files and directories
**Fstat**

- `fstat()` – retrieve information about a file

```c
#define T_DIR 1 // Directory
#define T_FILE 2 // File
#define T_DEV 3 // Device

struct stat {
    short type; // Type of file
    int dev; // File system’s disk device
    uint ino; // Inode number
    short nlink; // Number of links to file
    uint size; // Size of file in bytes
};
```
Links, inodes

- Same file can have multiple names – links
  - But unique inode number
- `link()` – create a name
- `unlink()` – create a name
- Example, create a temporary file

```c
fd = open("/tmp/xyz", O_CREAT|O_RDWR);
unlink("/tmp/xyz");
```
This is essentially all of UNIX, which you run today.