CSCE 351
Operating System Kernels

The UNIX/MINIX Operating System
Processes & IPC

Steve Goddard
goddard@cse.unl.edu

http://www.cse.unl.edu/~goddard/Courses/CSCE351

UNIX History

- The original UNIX
  - » An experimental operating system developed by Ken Thompson & Dennis Ritchie in the late ‘60s
- Main variants (“Standards”)
  - » System V (1983)
    - ❖ developed by AT&T
  - » 4.4 BSD (1993)
    - ❖ Open Software Foundation
  - » POSIX
    - ❖ IEEE/ISO
  - » FreeBSD
  - » Linux
- Commercial products
  - » Ultrix, DEC UNIX — DEC
  - » SunOS, Solaris — Sun
  - » HP/UX — Hewlett Packard
  - » AIX — IBM
  - » Xenix — Microsoft
  - » ...

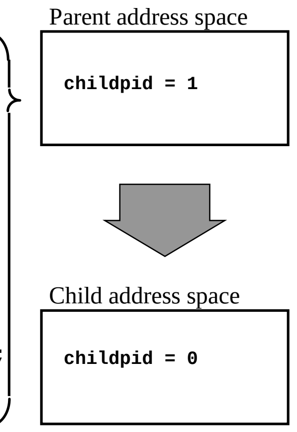
MINIX History

- The original MINIX was developed by Andrew Tanenbaum after AT&T closed the source code (1978)
  - » An educational operating system
  - » Version 1.0 was released circa 1987
  - » Version 2.0 was released circa 1997
- Version 2.0
  - » POSIX compliant
  - » Fully documented
  - » Requires only 30MB
  - » Micro-kernel design
- "Looks like UNIX from the outside"

Processes

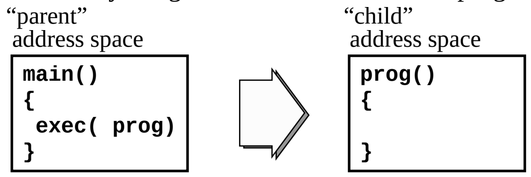
- A process is created by the **`fork()`** system call
  - » creates a new address space that is a duplicate of the callers

```
main (argc, argv)
 int childpid;
{
 switch (childpid = fork()) {
   case 0:          /* child  */
     child_func();
     exit(0);
   default:         /* parent */
     parent_func();
     while(wait((int *) 0) != childpid);
     exit(0);
   case -1:         /* oops   */
     error("fork:%s\n",sys_errlist[errno]);
  }
}
```



Processes

- Alternatively, processes can be "created" by an **`execve()`**
  - » replaces the memory image of the caller with a new program

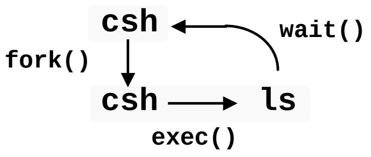


```
main()
{
 exec( prog)
}
```

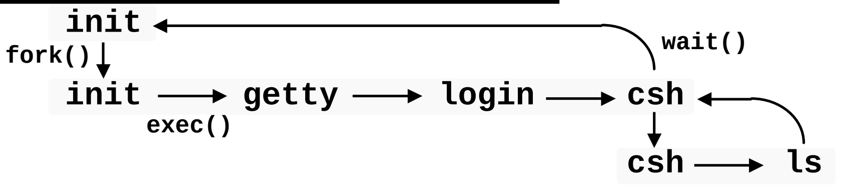
```
prog()
{

}
```

- This is how the shell executes commands
  - » a **`fork()`** followed by an **`exec()`**


$$\begin{tikzcd} \texttt{csh} \arrow[d, "\texttt{fork()}"'] & \\ \texttt{csh} \arrow[r, "\texttt{exec()}"'] & \texttt{ls} \arrow[ul, bend right, "\texttt{wait()}"'] \end{tikzcd}$$

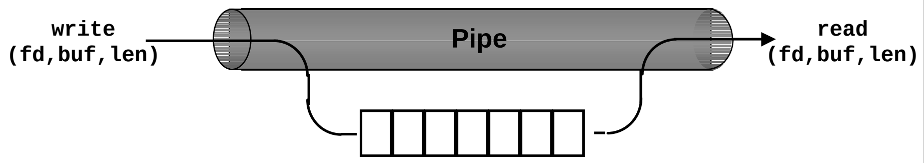
Example: How users logs in


$$\begin{tikzcd} \texttt{init} \arrow[d, "\texttt{fork()}"'] & & & \\ \texttt{init} \arrow[r, "\texttt{exec()}"'] & \texttt{getty} \arrow[r] & \texttt{login} \arrow[r] & \texttt{csh} \arrow[ulll, bend right, "\texttt{wait()}"'] \arrow[d] & \\ & & & \texttt{csh} \arrow[r] & \texttt{ls} \arrow[ul, bend right] \end{tikzcd}$$

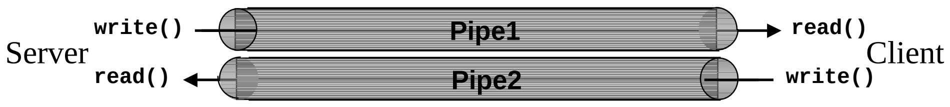
- There exists a "master process" in UNIX: **`init`**
- **`init`** forks a process for each terminal port
- each **`init`** copy execs **`getty`** which prints the login prompt and then reads the login and password
- **`getty`** then execs **`login`** which verifies the login
- **`login`** then execs **`csh`** which forks new processes for each command

(Simple) Interprocess Communication

- ◆ Like message passing except more general
- ◆ Pipes — a shared, in-memory file
  - » a queue of 4K bytes
  - » buffered, asynchronous message passing
    - ❖ blocks reader when queue is empty
    - ❖ blocks writer when queue is full



(Simple) Interprocess Communication



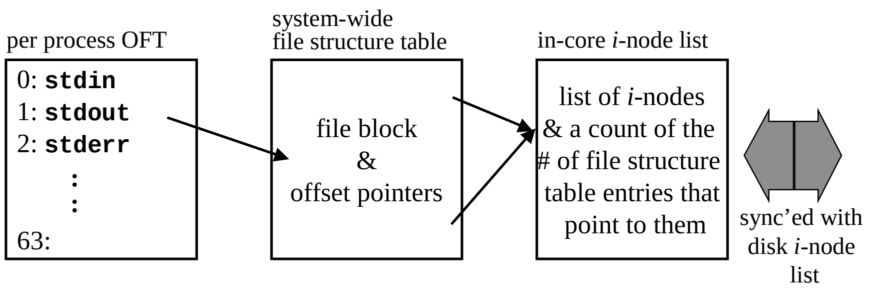
```
main() {
 int pipe1[2], pipe2[2];

 if (pipe(pipe1) == -1  ||  pipe(pipe2) == -1) error(...

 switch (childpid = fork()) {
   case 0:                      /* child */
     close(pipe1[1]);           /* write descriptor for pipe1 */
     close(pipe2[0]);           /* read  descriptor for pipe2 */
     client(pipe1[0],pipe2[1]); /* client program             */
   default :                    /* parent */
     close(pipe1[0]);           /* read  descriptor for pipe1 */
     close(pipe2[1]);           /* write descriptor for pipe2 */
     server(pipe2[0],pipe1[1]); /* server program             */
     while (wait((int *) 0) != childpid);   /* wait for child */
   }
}
```

Open File Table

- UNIX maintains an *open file table* for each process which lists each file in use by the process
  - » the OFT is copied when processes are forked

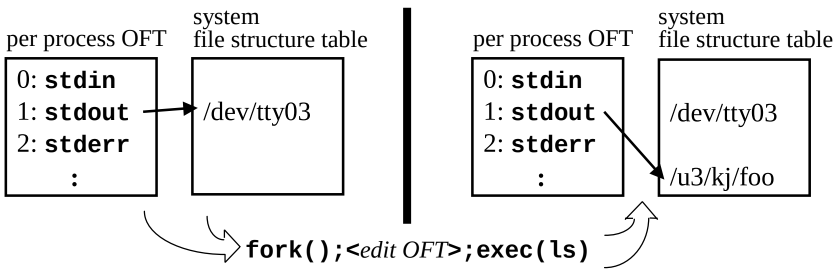


per process OFT
0: stdin
1: stdout
2: stderr
:
63:
system-wide file structure table
file block & offset pointers
in-core *i*-node list
list of *i*-nodes & a count of the # of file structure table entries that point to them
sync'ed with disk *i*-node list

The UNIX File System

Open file table examples

- I/O redirection — **`ls > foo`**
  - » just change a pointer in the OFT



(Primary) MINIX Kernel Interprocess Communication

- Message Passing is used by Kernel tasks
  - » Blocking send and receive primitives
    - ❖ Syntax:

`send(`*`dest`*`, &`*`message`*`)`

`receive(`*`dest`*`, &`*`message`*`)`

`sendrec(src_`*`dest`*`, &`*`message`*`)`

Note: page 97 shows this as `send_rec(src_`*`dest`*`, &`*`message`*`)`

but page 560, file include/minix/syslib.h shows it as `sendrec()`