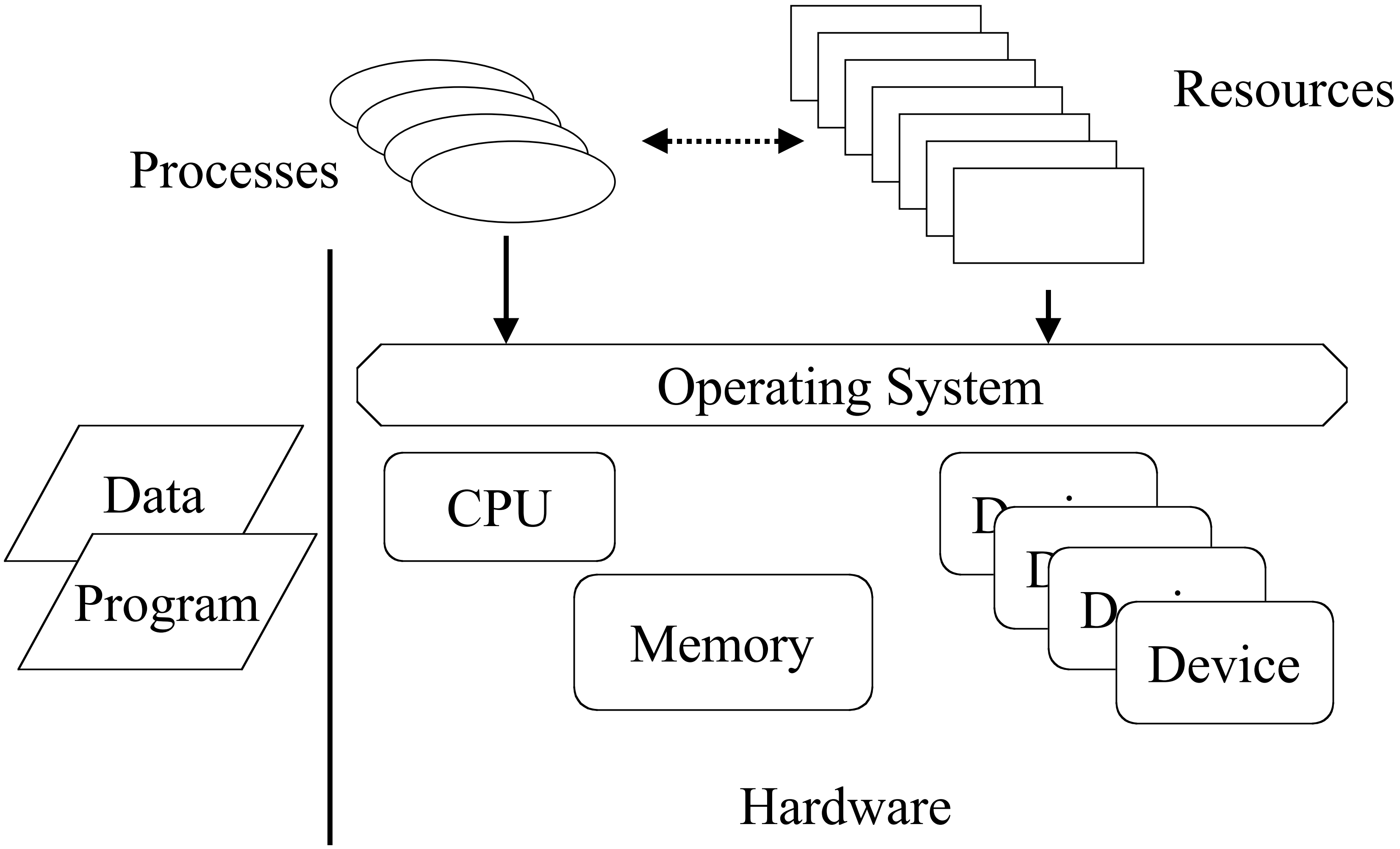
Using the OS

The Basic Abstractions

- Processes
- Files
- Other Resources

Processes & Resources

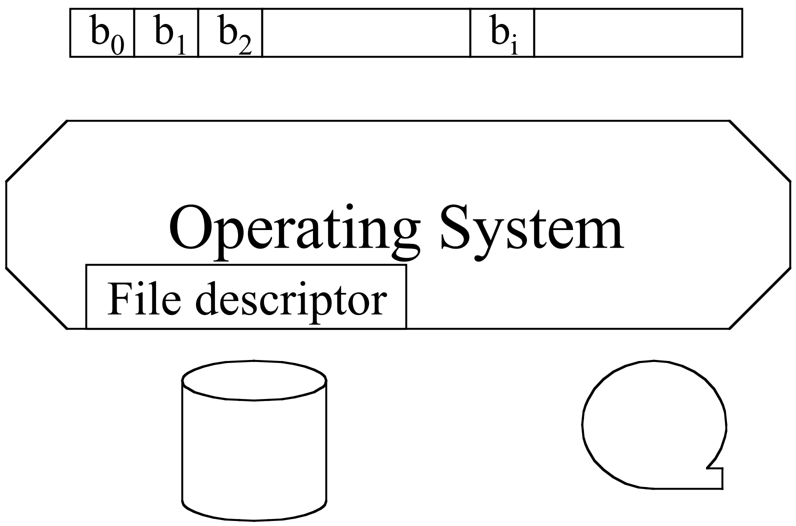


Resources

- Anything that a process requests from an OS
  - Available => allocated
  - Not available => process is blocked
- Data is a primary resource
- A *file* is a container for holding data
- Consequence: Processes & files are programmers main tools

Files

- File: A named, linear stream of records (e.g., bytes) stored on a device



UNIX Files

- UNIX and NT try to make every resource (except CPU and RAM) look like a file
- Then can use a common interface:

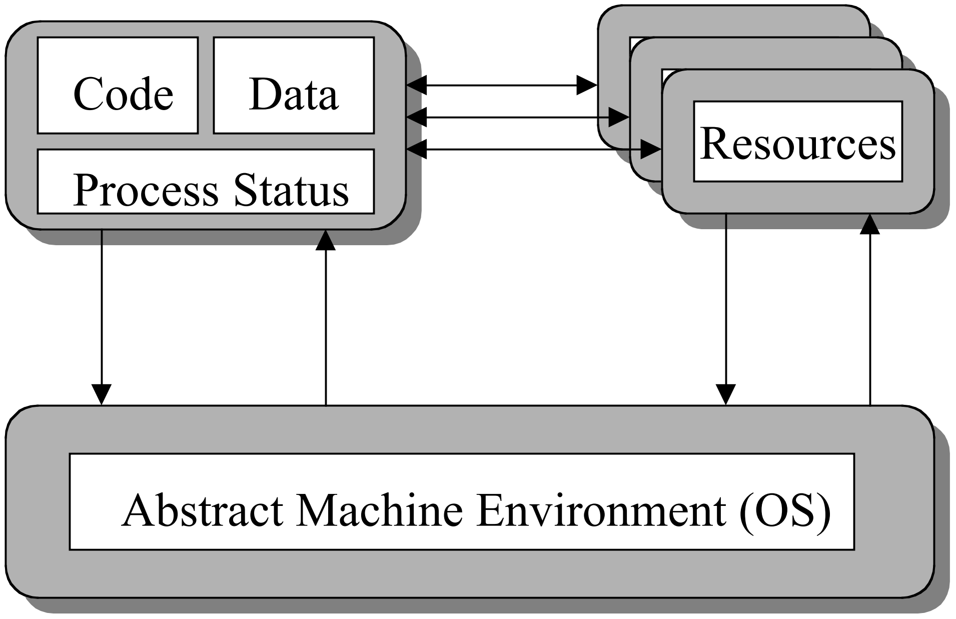
`open`	Specifies file name to be used
`close`	Release file descriptor
`read`	Input a block of information
`write`	Output a block of information
`lseek`	Position file for read/write
`ioctl`	Device-specific operations

Example

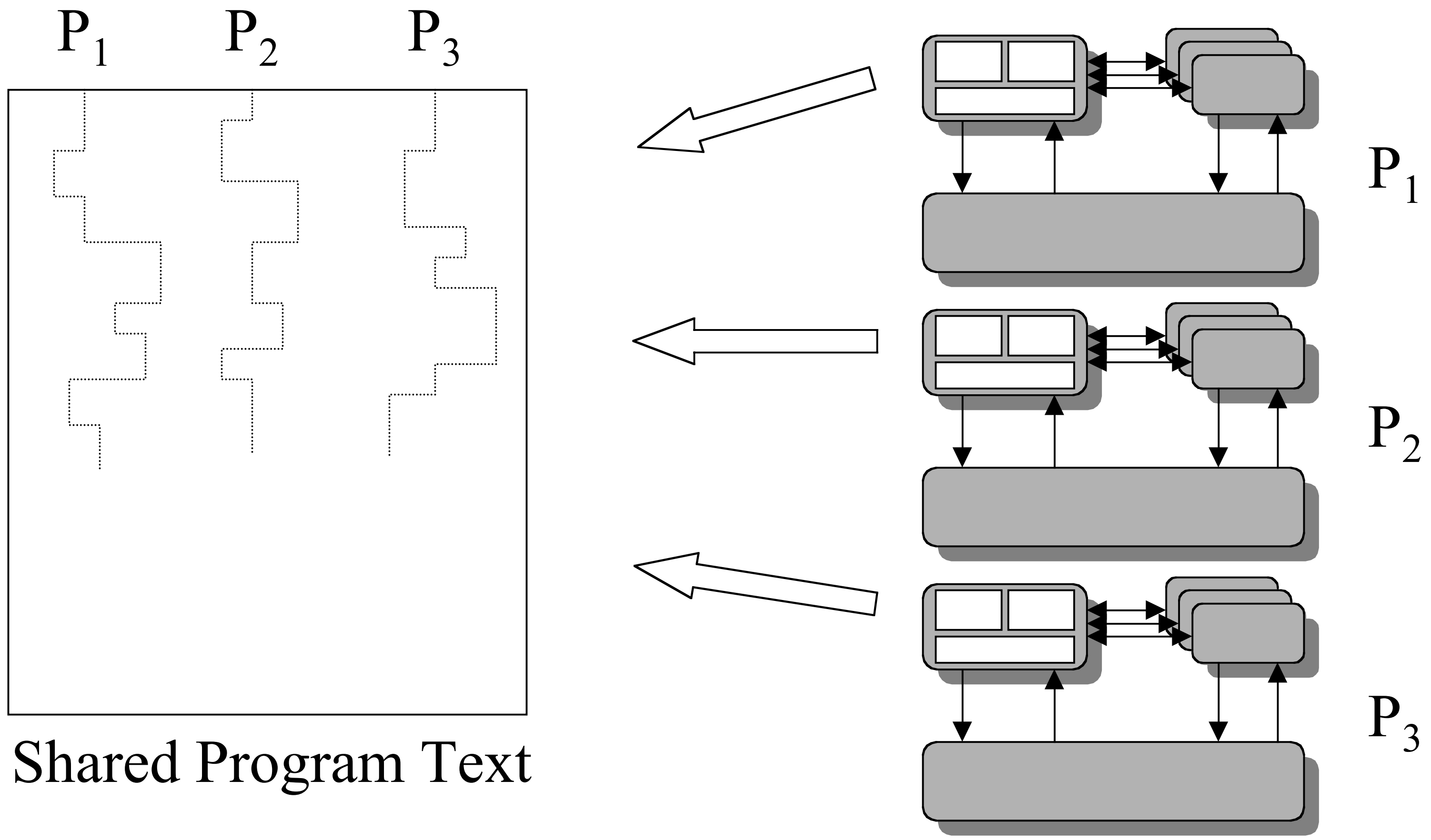
```
#include    <stdio.h>
#include    <fcntl.h>
int main() {
    int inFile, outFile;
    char *inFileName = “in_test”;
    char *outFileName = “out_test”;
    int len;
    char c;

    inFile = open(inFileName, O_RDONLY);
    outFile = open(outFileName, O_WRONLY);
/* Loop through the input file */
    while ((len = read(inFile, &c, 1)) > 0)
        write(outFile, &c, 1);
/* Close files and quite */
    close(inFile);
    close(outFile);
}
```

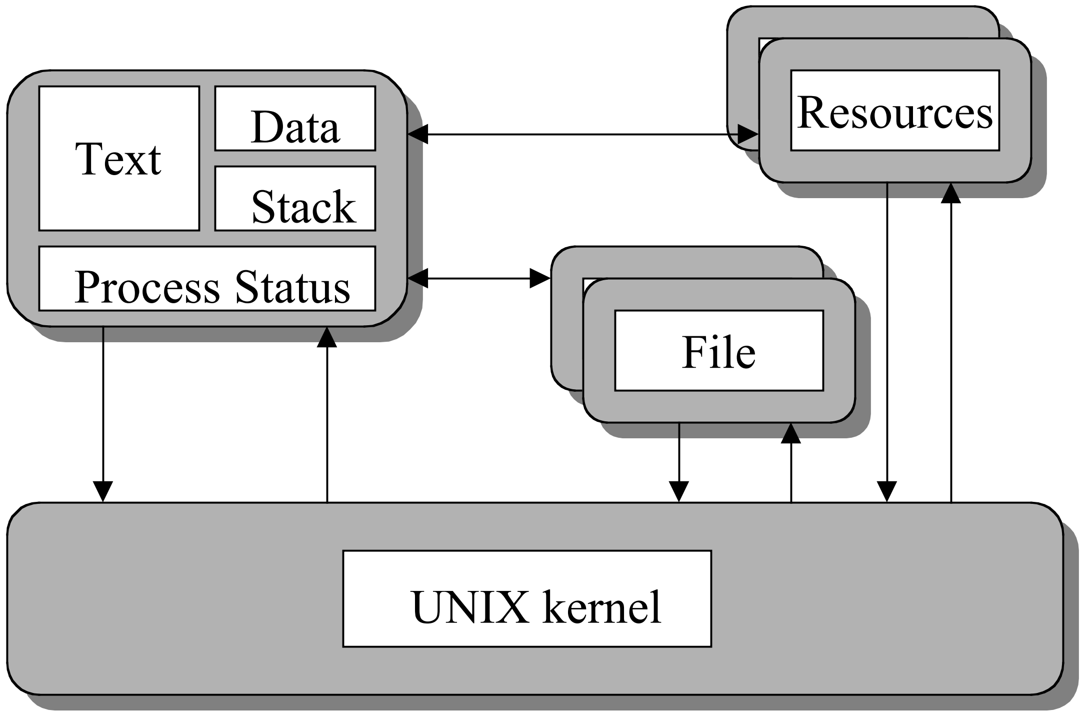
A Process


Code
Data
Process Status
Resources
Abstract Machine Environment (OS)

Processes Sharing a Program



UNIX Process



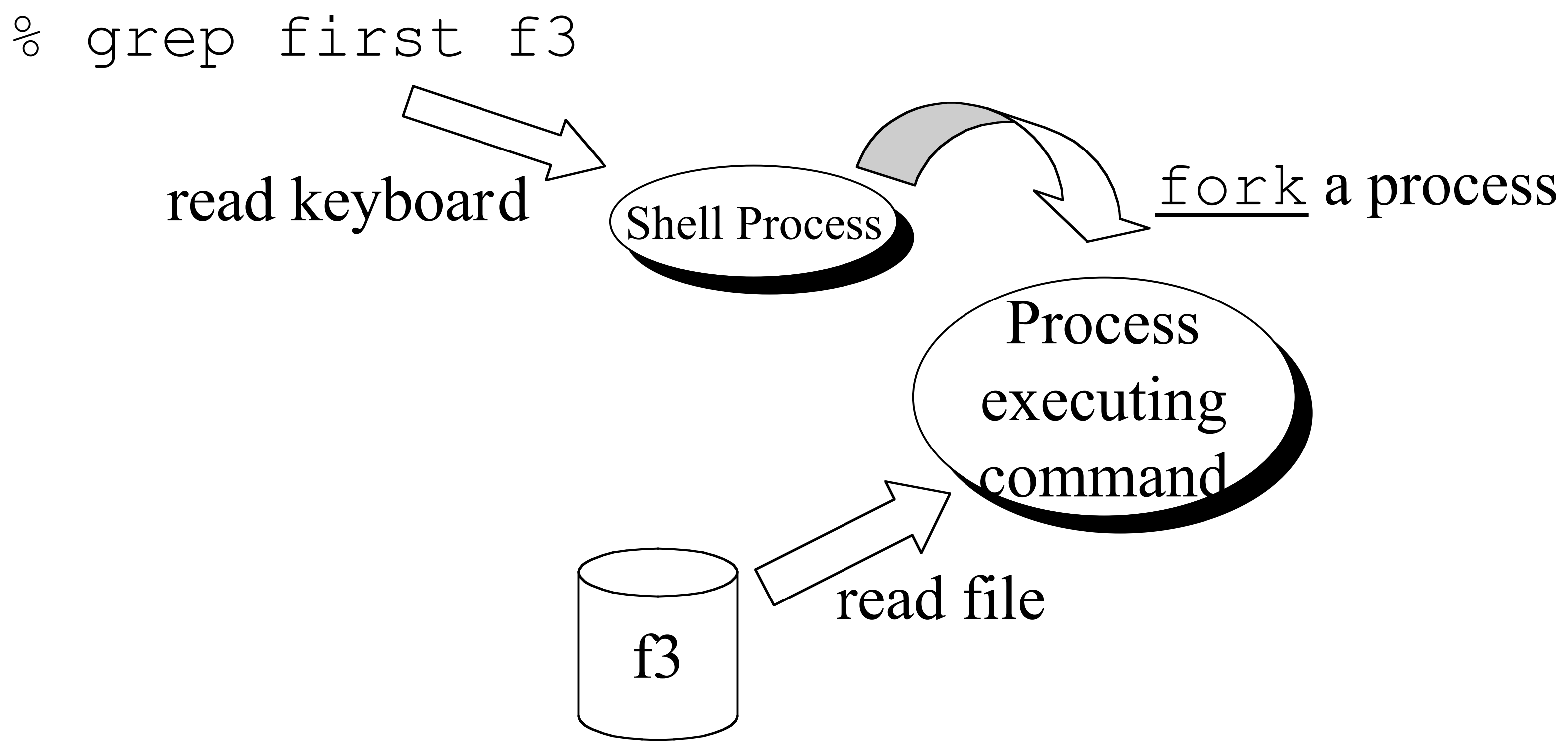
More on UNIX Processes

- Each process has its own address space
  - Subdivided into text, data, & stack segment
  - `a.out` file describes the address space
- OS creates ***descriptor*** to manage process
- ***Process identifier*** (PID): User handle for the process (descriptor)
- Try “`ps`” and “`ps -aux`” (read man page)

Creating/Destroying Processes

- UNIX `fork` creates a process
  - Creates a new address space
  - Copies text, data, & stack into new adress space
  - Provides child with access to open files
- UNIX `wait` allows a parent to wait for a child to terminate
- UNIX `exec` allows a child to run a new program

Executing a UNIX Command



Creating a UNIX Process

```
int pidValue;
 ...
pidValue = fork();       /* Creates a child process */
if(pidValue == 0) {
  /* pidValue is 0 for child, nonzero for parent */
  /* The child executes this code concurrently with parent */
    childsPlay(…);      /* A procedure linked into a.out */
    exit(0);
}
/* The parent executes this code concurrently with child */
parentsWork(..);
wait(…);
 ...
```

Executing a Different Program

```
int pid;
 ...
/* Set up the argv array for the child */
 ...
/* Create the child */
if((pid = fork()) == 0) {
  /* The child executes its own absolute program */
    execve(childProgram.out, argv, 0);
  /* Only return from an execve call if it fails */
    printf("Error in the exec … terminating the child …");
    exit(0);
}
 ...
wait(…);    /* Parent waits for child to terminate */
 ...
```

Example: Parent

```
#include        <sys/wait.h>

#define NULL    0

int main (void)
{
    if (fork() == 0){   /* This is the child process */
        execve("child",NULL,NULL);
        exit(0);        /* Should never get here, terminate */
    }
/* Parent code here */
    printf("Process[%d]: Parent in execution ...\n", getpid());
    sleep(2);
    if(wait(NULL) > 0) /* Child terminating */
        printf("Process[%d]: Parent detects terminating child \n",
                                    getpid());
    printf("Process[%d]: Parent terminating ...\n", getpid());
}
```

Example: Child

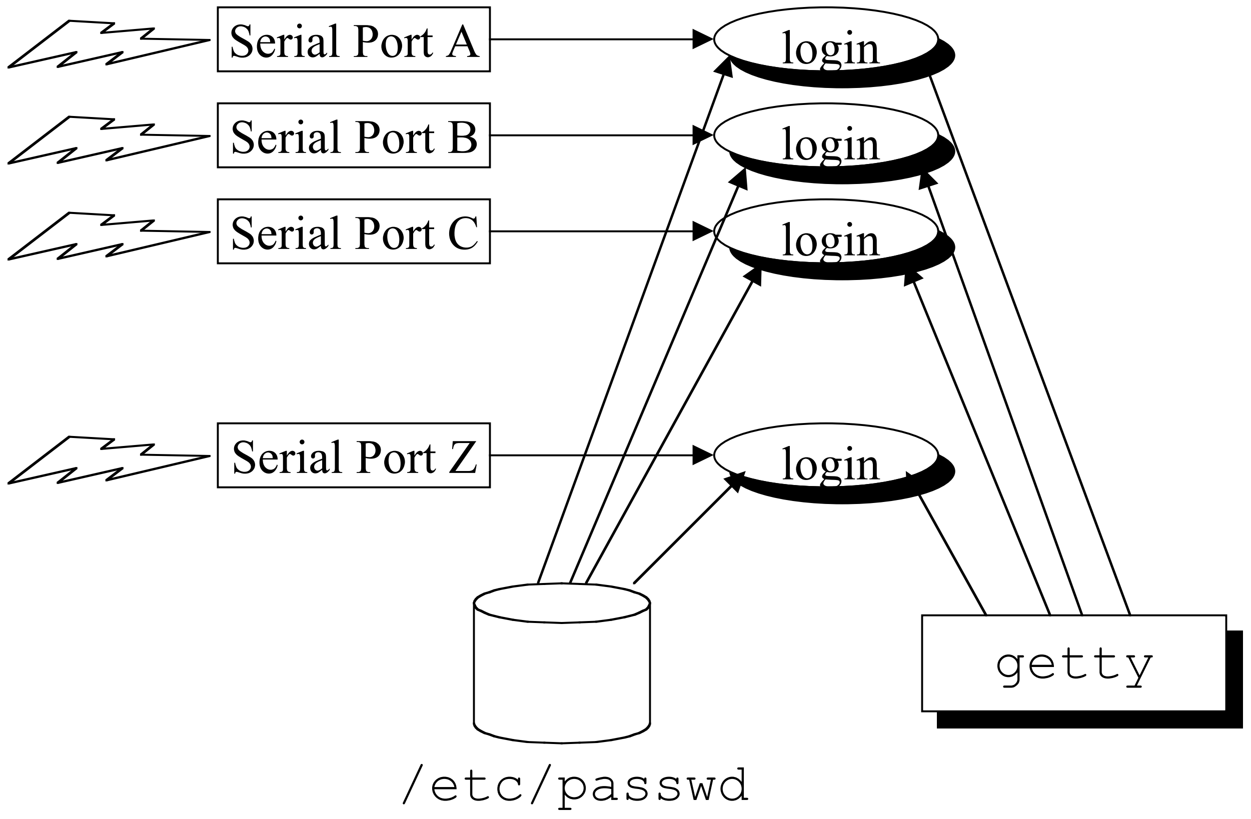
```
int main (void)
{
/* The child process's new program
   This program replaces the parent's program */

    printf("Process[%d]: child in execution ...\n", getpid());
    sleep(1);
    printf("Process[%d]: child terminating ...\n", getpid());
}
```

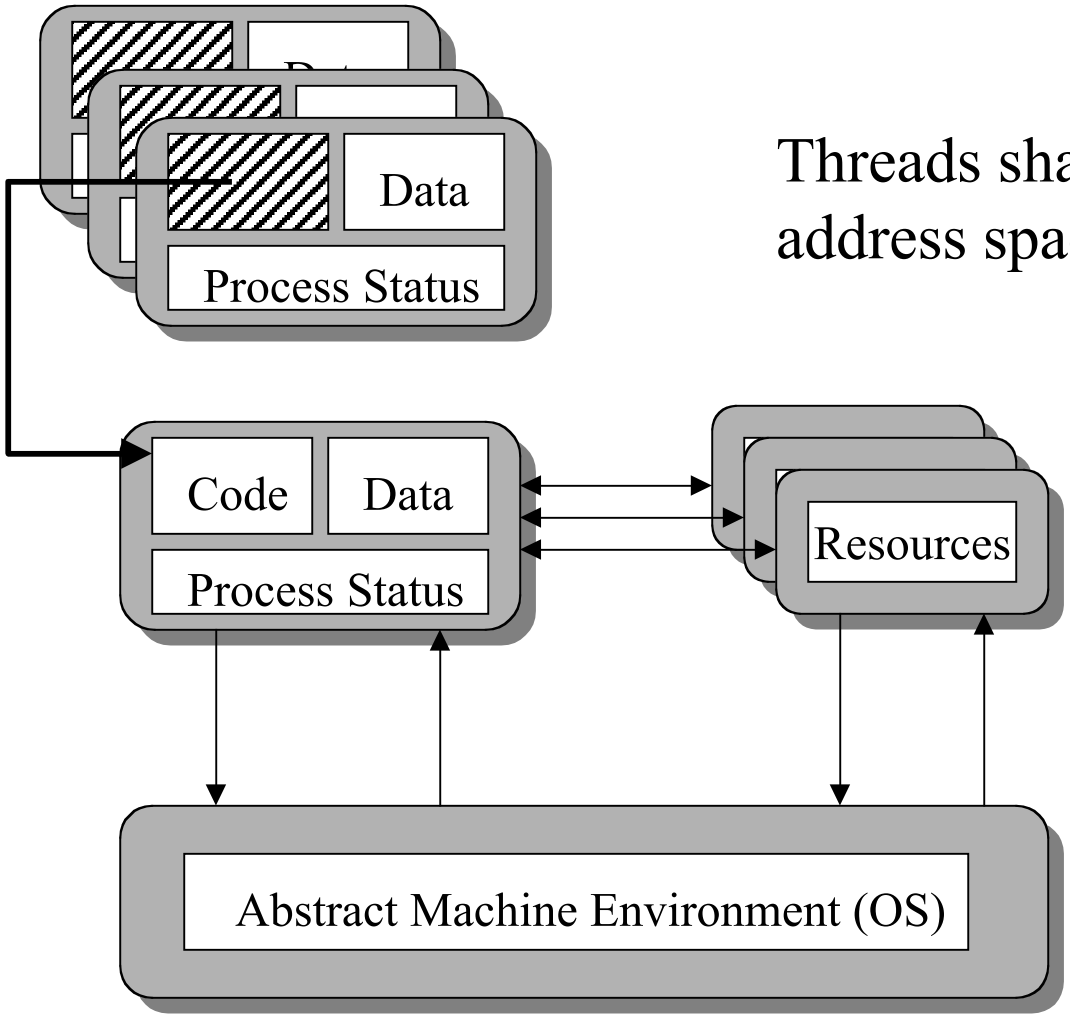
Bootstrapping

- Computer starts, begins executing a *bootstrap program* -- *initial process*
- Loads OS from the disk (or other device)
- Initial process runs OS, creates other processes

Initializing a UNIX Machine



Threads -- The NT Model



Threads share process's address space

NT Threads

```
#include      <cthreads.h>
 ...
int main(int argv, char *argv[]) {
    t_handle = CreateThread(…, tChild, &i, …);
/* A new child thread is now executing the tChild function */
    Sleep(100)    /* Let another thread execute */
}


DWPRD WINAPI tChild(LPVOID me) {
/* This function is executed by the child thread */
 ...
    SLEEP(100);   /* Let another thread execute */
 ...
}
```

Objects

- A recent trend is to replace processes by objects
- Objects are autonomous
- Objects communicate with one another using messages
- Popular computing paradigm
- Too early to say how important it will be ...