CS 1951E: Intro to Operating Systems
What are Operating Systems?

• Possible definitions:
  – the code that {Microsoft, Apple, Linus, Google} provides
  – the code that you didn’t write
  – the code that runs in privileged mode
  – the code that makes things work
  – the code that makes things crash
  – etc.
Operating Systems

• Abstraction
  – providing an “appropriate” interface for applications

• Concerns
  – performance
    - time and space
  – sharing and resource management
  – failure tolerance
  – security
  – marketability
Hardware

Network

Memory

Processor

Disk

Disk

Processor

Keyboard

Mouse
Abstractions

• Hardware
  – disks
  – memory
  – processors
  – network
  – monitor
  – keyboard
  – mouse

• Operating system
  – files
  – programs
  – threads of control
  – communication
  – windows, graphics
  – input
  – locator
Files
Issues

• Naming
• Allocating space on disk (permanent storage)
  – organized for fast access
  – minimize waste
• Shuffling data between disk and memory (high-speed temporary storage)
• Coping with crashes
Programs

Data

Code

Memory

Disk

Disk

Processor

Processor
Memory Sharing (1)
Virtual Memory

Program 1

Program 2

Program 3

Memory

Disk

Disk
Concurrency
Parallelism
OS Components: Functional

Processor Management
- Scheduling
- Interrupt management

Memory Management
- Processes and threads
- Virtual memory
- Real memory

I/O Management
- Human-interface devices
- Network protocols
- File system
- Logical I/O management
- Physical device drivers
OS Components: Portability

Hardware Independent

- Hardware Dependent
- Hardware Dependent
- Hardware Dependent
- Hardware Dependent
- Hardware Dependent
OS Components: Importance

- Trusted Security Base
OS Components: Flow of Control

- Applications
  - system calls
  - OS Kernel
    - kernel threads
  - External Hardware
    - interrupts
Idiot Proof ...

Can I clobber David's program?

int main() {
    int i;
    int A[1];

    for (i=0; ; i++)
        A[rand()] = i;
}
Can I monopolize the processor?

Aisha's Program

```c
int runforever() {
    while(1) {
    }

int main() {
    runforever();
}
```
Simple User Address Space

- **text**
- **data**
- **dynamic**
- **bss**
- **stack**
Creating a Process: Before

parent process

fork( )
Creating a Process: After

parent process

fork( )
// returns p

child process

fork( )
// returns 0
Loading a New Image

Before

exec(prog, args)

After

args

prog’s bss

prog’s data

prog’s text
Address-Space Representation
Somewhat Simplified

- task_struct
  - mm_struct
    - vm_area_struct 0–7fff x, shared
    - vm_area_struct 8000–1afff rw, private
    - vm_area_struct 1b000–1bfff rw, private
    - vm_area_struct 7fffd000–7ffffff rw, private
- struct file
Adding a Mapped File

stack

mapped file

bss & dynamic

data

text
Address-Space Representation:
More Areas

- task_struct
- mm_struct
- vm_area_struct 0–7fff
  - x, shared
- vm_area_struct 8000–1afff
  - rw, private
- vm_area_struct 1b000–1bff
  - rw, private
- vm_area_struct 200000–201fff
  - rw, shared
- vm_area_struct 7fffd000–7fffffff
  - rw, private

- struct file
- struct file
Adding More Stuff

- stack 1
- stack 2
- stack 3
- mapped file 1
- mapped file 2
- mapped file 3
- bss & dynamic
- data
- text
- mapped file 117
Address-Space Representation: Reality

- task_struct
- mm_struct
- 200000–201fff
  - 1b000–1bff
    - 0–7fff
    - 8000–1afff
  - 202000–203fff
    - 204000–204fff
    - 7fffd000–7fffffff
    - 208000–210fff

Copyright © 2015 Thomas W. Doeppner. All rights reserved.
System Calls

- `write(fd, buf, len)`

- Kernel portion of address space
  - Kernel text
  - Kernel stack
  - Other stuff

- User portion of address space

- Trap into kernel
Multiple Threads

- other stuff
- kernel stack
- other stuff
- kernel stack
- other stuff
- kernel stack
- other stuff
- kernel stack
- kernel text
Layering

User Applications

System Calls

Virtual File System (VFS)

disk file systems (e.g. S5FS)  terminals  networking  other things (e.g. /proc)
Standard File Descriptors

main( ) {
    char buf[BUFSIZE];
    int n;
    const char* note = "Write failed\n";

    while ((n = read(0, buf, sizeof(buf))) > 0) {
        if (write(1, buf, n) != n) {
            (void)write(2, note, strlen(note));
            exit(EXIT_FAILURE);
        }
    }
    return(EXIT_SUCCESS);
}

File-Descriptor Table

File-descriptor table

User address space

Kernel address space

File descriptor

ref count
access mode
file location
inode pointer
A Program

```c
int nprimes;
int *prime;
int main(int argc, char *argv[]) {
    ...
    for (i=1; i<nprimes; i++) {
        ...
    }
    if (write(1, prime, nprimes*sizeof(int)) == -1) {
        perror("primes output");
        exit(1);
    }
    return(0);
```
int nprimes;
int *prime;
int main(int argc, char *argv[]) {
    ...
    for (i=1; i<nprimes; i++) {
        ...
    }
    for (i=0; i<nprimes; i++) {
        printf("%d\n", prime[i]);
    }
    return(0);
}
Running It

```c
if (fork() == 0) {
    /* set up file descriptor 1 in the child process */
    close(1);

    if (open("/home/twd/Output", O_WRONLY) == -1) {
        perror("/home/twd/Output");
        exit(1);
    }

    execl("/home/twd/bin/primes", "primes", "300", 0);
    exit(1);
}

/* parent continues here */

while(pid != wait(0)) /* ignore the return code */
;
```

Security
Redirected Output After Dup

File-descriptor table

File descriptor 1
File descriptor 2

User address space
Kernel address space

2  WRONLY  100  inode pointer
File Descriptors After Fork

Parent’s address space

Child’s address space

Kernel address space

file descriptor

file descriptor

inode pointer

2

WRONLY

0
Attacks

• Trap doors
• Trojan horses
• Viruses and worms
• Exploit bugs
• Exploit features
Trap Doors

- You supply a CD driver
  - `ioctl(cd_file_descriptor, 0x5309)`
    - standard command to eject the CD
  - `ioctl(cd_file_descriptor, 0xe311)`
    - second argument is passed to your driver
    - on receipt, your driver sets UID of current process to zero
Trap-Door Prevention

- Make sure everything that goes into kernel is ok
  - the Linux kernel has over 19,000 source-code files
  - also must worry about all setuid programs
  - Windows probably has more files

- How?
  - Windows
    - really careful management
  - Linux
    - thousands of eyes checking things out
Not Good Enough

• Paul Karger and Roger Schell
  – wrote 1974 paper suggesting compiler could add trap doors

• Ken Thompson was inspired and did it
  – his C compiler created trap door in login program
  – C compiler added code to do this whenever it compiled itself
  – “feature” in C-compiler binary
  – self replicates — not in source code!