CS 471 Operating Systems

Yue Cheng

George Mason University
Fall 2017
Introduction

- Instructor of Section 002
  - Dr. Yue Cheng (web: cs.gmu.edu/~yuecheng)
  - Email: yuecheng@gmu.edu
  - Office: 5324 Engineering
  - Office hours: W 3-5pm
  - Research interests: Distributed and storage systems, cloud computing, operating systems

- Teaching assistant
  - Panneer Selvam Santhalingam
  - Email: psanthal@gmu.edu
  - Office hours: MF 3-5pm (tentative)
Logistics

- **Textbook**
  - *Operating Systems Principles & Practices*
    By T. Anderson and M. Dahlin
  - [Highly recommended] *Operating Systems: Three Easy Pieces*
    By Remzi H. Arpaci-Dusseau, Andrea C. Arpaci-Dusseau

- **Prerequisites are enforced!!**
  - **CS 310** Data Structures
  - **CS 367** Computer Systems & Programming -OR-
  - **ECE 445** Computer Organization
  - **Be comfortable with C programming language**

- **Class web page**
  - [https://cs.gmu.edu/~yuecheng/teaching/cs471_fall17.html](https://cs.gmu.edu/~yuecheng/teaching/cs471_fall17.html)
  - Class materials will all be available on the class web page
Logistics (cont.)

- Syllabus
  - https://cs.gmu.edu/media/syllabi/Fall2017/CS_471ChengY.html

- Grading
  - 40% programming projects + homework assignments
  - 25% mid-term exam
  - 35% final exam

- Reminders
  - Honor code
  - Late policy: 15% deducted each day. No credit after 3 days
3 Major Topics

CPU

Memory

Disk
Some Demo’s
#include <stdio.h>
#include <stdlib.h>
#include "common.h"

int main(int argc, char *argv[]) {
    if (argc != 2) {
        fprintf(stderr, "usage: cpu <string>\n");
        exit(1);
    }
    char *str = argv[1];

    while (1) {
        printf("%s\n", str);
        Spin(1);
    }
    return 0;
}

double GetTime() {
    struct timeval t;
    int rc = gettimeofday(&t, NULL);
    assert(rc == 0);
    return (double)t.tv_sec + (double)t.tv_usec/1e6;
}

void Spin(int howlong) {
    double t = GetTime();
    while (((GetTime() - t) < (double)howlong) || (GetTime() - t) > (double)howlong) {
        // do nothing in loop
    }
}
```c
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include "common.h"

int main(int argc, char *argv[])
{
    if (argc != 2) {
        fprintf(stderr, "usage: mem <value>\n");
        exit(1);
    }
    int *p;  // memory for pointer is on "stack"
    p = malloc(sizeof(int));  // malloc'd memory is on "heap"
    assert(p != NULL);
    printf("(%d) addr of p: %llx\n", (int) getpid(), (unsigned long long) &p);
    printf("(%d) addr stored in p: %llx\n", (int) getpid(), (unsigned long long) p);
    *p = atoi(argv[1]);  // assign value to addr stored in p
    while (1) {
        Spin(1);
        *p = *p + 1;
        printf("(%d) value of p: %d\n", getpid(), *p);
    }
}
return 0;
```
```c
#include <stdio.h>
#include <stdlib.h>
#include "common.h"

volatile int counter = 0;
int loops;

void *worker(void *arg) {
    int i;
    for (i = 0; i < loops; i++) {
        counter = counter + 1;
    }
    pthread_exit(NULL);
}

int main(int argc, char *argv[]) {
    if (argc != 2) {
        fprintf(stderr, "usage: threads <loops>\n");
        exit(1);
    }
    loops = atoi(argv[1]);
pthread_t p1, p2;
printf("Initial value : %d\n", counter);
pthread_create(&p1, NULL, worker, NULL);
pthread_create(&p2, NULL, worker, NULL);
pthread_join(p1, NULL);
pthread_join(p2, NULL);
printf("Final value : %d\n", counter);
return 0;
}
```
OS Provides Virtualization on Underlying Hardware

- CPU
- Memory
- Disk
Topic 1: Process/Thread and CPU Scheduling

- Process/thread
- Synchronization, deadlocks
- CPU scheduling
Topic 2: Memory Management and Virtual Memory

- Process/thread
- Synchronization, deadlocks
- CPU scheduling

- Memory management
- Virtual memory
Topic 3: Storage, I/O, and Filesystems

- Process/thread
- Synchronization, deadlocks
- CPU scheduling

- Memory management
- Virtual memory

- Persistent storage
- File and I/O systems
Advanced Topics (Miscellaneous)

- Process/thread
- Synchronization, deadlocks
- CPU scheduling

- Memory management
- Virtual memory

- Persistent storage
- File and I/O systems

- Virtual machines
- Distributed systems
- Linux
- Protection and security
Why do you take this course?

Course outcomes

Upon completion of this course, the students should be able to:

- Demonstrate knowledge about the role and purpose of the operating systems
- Be able to explain the features of operating systems found in different types of computer systems (e.g., mainframe systems, personal computers, multiprocessor systems, handheld systems)
- Demonstrate basic knowledge about the evolution of operating system concepts from early batch systems to today's sophisticated multiuser/multiprocessor systems
- Demonstrate knowledge about different approaches to operating system design and the involved trade-offs
- Be able to explain the main performance evaluation criteria for computer systems and how the operating system design can have an impact on these
- Show an understanding of the need for the concurrent operation of multiple tasks (processes/threads) and an ability to solve basic process synchronization problems that arise from concurrent operation settings
- Demonstrate the knowledge about process scheduling, basic memory management, and file system management techniques and their impact on the overall performance
- Demonstrate the basic knowledge about distributed systems and how the ever increasing networking features have affected (and are affecting) operating system design
- Be able to implement a suite of basic algorithms proposed for the main OS services such as memory management and process scheduling
General Learning Goals

1. Grasp **basic** knowledge about **Operating Systems** and **Computer Systems** software
2. Learn **important systems concepts** in general
   - Multi-processing/threading, synchronization
   - Scheduling
   - Caching, memory, storage
   - And more…
3. Gain **hands-on** experience in **writing/hacking/designing** large systems software
General Learning Goals

1. Grasp basic knowledge about Operating Systems and Computer Systems software

And YES, I will add a note when I reach something which IMO has a high chance of being asked on some interview.

So… please pay attention. 😊

– And more…

3. Gain hands-on experience in writing/hacking/designing large systems software
Outline

- Introduction
  - What’s an OS?

- Computer Systems

- Operating Systems
What is an OS?

- A software layer that acts as an intermediary between the user and the hardware.

- Operating system goals
  - **Convenience**: Make the computer system convenient to use.
  - **Efficiency**: Manage the computer system resources in an efficient manner

- “The one program running at all times on the computer” is the **kernel**. Everything else is either a system program (ships with the operating system) or an application program

- The **government metaphor**
Computer System Components

User 1

User 2

User 3

⋯

User n

Compiler

Text editor

Filesystem

Database system

System and Application Programs

Operating System

Hardware
Computer System Components

1. **Hardware** – provides basic computing resources (CPU, memory, I/O devices).

2. **Operating system** – controls and coordinates the use of the hardware among various application programs for various users.

3. **Application programs** – define the ways in which the system resources are used to solve the computing problems of the users (compilers, database systems, video games, business programs).

4. **Users**
Operating System Definitions

- **Resource allocator** – manages and allocates resources.
- **Control program** – controls the execution of user programs and operations of I/O devices.
- **Kernel** – the one program running at all times (all else being application programs).

- I/O is a critical component of any computer system
  - For interaction with users/external world
  - For correct operation
  - For performance
Operating System Structure

- System Components
- System Calls
- Virtual Machines
Process Abstraction

- **A process is a program in execution**
  - It is a unit of work within the system. A program is a *passive entity*, a process is an *active entity*.

- Process needs resources to accomplish its task
  - CPU, memory, I/O, files
  - Initialization data

- Process termination requires reclaim of any reusable resources

- Single-threaded process has one program counter specifying location of next instruction to execute
  - Process executes instructions sequentially, one at a time, until completion

- Multi-threaded process has one program counter per thread

- A software system may have many processes, some user, some operating system running concurrently on one or more CPUs
  - Concurrency by multiplexing the CPUs among the processes / threads
Loading from Program to Process

- **CPU**
- **Memory**
  - code
  - static data
  - heap
  - stack
- **Process**
- **Disk**
  - code
  - static data
  - Program

**Loading:** Takes on-disk program and reads it into the address space of process
Process Abstraction (cont.)

- Introduced to obtain a systematic way of monitoring and controlling program execution
- A process is an **executable** program with:
  - associated data (variables, buffers…)
  - execution context: i.e. all the information that
    - the CPU needs to execute the process
      - content of the processor registers
    - the OS needs to manage the process:
      - priority of the process
      - the event (if any) after which the process is waiting
      - other data (that we will introduce later)
A Simple Implementation of Processes

- The process index register contains the index into the process list of the currently executing process (B).
- A process switch from B to A consist of storing (in memory) B’s context and loading (in CPU registers) A’s context.
- A data structure that provides flexibility (to add new features).
Memory Management

- All data in memory before and after processing
- All instructions in memory in order to execute
- Memory management determines what is in memory when
  - Optimizing CPU utilization and computer response to users
- Memory management activities
  - Keeping track of which parts of memory are currently being used and by whom
  - Deciding which processes (or parts thereof) and data to move into and out of memory
  - Allocating and deallocating memory space as needed
- **Virtual memory** management is an essential part of most operating systems
Storage Management

- OS provides a uniform, logical view of information storage
  - Abstracts physical properties to logical storage unit - file
  - Each medium is controlled by device type (i.e., disk drive, tape drive)
    - Varying properties include access speed, capacity, data-transfer rate, access method (sequential or random)

- Filesystem management
  - Files usually organized into directories
  - Access control on most systems to determine who can access what
  - OS activities include
    - Creating and deleting files and directories
    - Primitives to manipulate files and dirs
    - Mapping files onto secondary storage
    - Backup files onto stable (non-volatile) storage media
Storage Structure

- **Main memory** – only large storage media that the CPU can access directly
  - Small CPU cache memories are used to speed up average access time to the main memory at run-time.

- **Secondary storage** – extension of main memory that provides large nonvolatile storage capacity.
  - Magnetic disks
  - Electronic disks -- Solid state disks (SSDs)
Storage Systems Tradeoffs

- Storage systems organized in hierarchy
  - Speed
  - Cost
  - Volatility

- Faster access time, greater cost per bit
- Greater capacity, lower cost per bit
- Greater capacity, slower access speed
Performance Tradeoffs

<table>
<thead>
<tr>
<th>Level</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>registers</td>
<td>cache</td>
<td>main memory</td>
<td>solid state disk</td>
<td>magnetic disk</td>
</tr>
<tr>
<td>Typical size</td>
<td>&lt; 1 KB</td>
<td>&lt; 16MB</td>
<td>&lt; 64GB</td>
<td>&lt; 1 TB</td>
<td>&lt; 10 TB</td>
</tr>
<tr>
<td>Implementation technology</td>
<td>custom memory with multiple ports CMOS</td>
<td>on-chip or off-chip CMOS SRAM</td>
<td>CMOS SRAM</td>
<td>flash memory</td>
<td>magnetic disk</td>
</tr>
<tr>
<td>Access time (ns)</td>
<td>0.25 - 0.5</td>
<td>0.5 - 25</td>
<td>80 - 250</td>
<td>25,000 - 50,000</td>
<td>5,000,000</td>
</tr>
<tr>
<td>Bandwidth (MB/sec)</td>
<td>20,000 - 100,000</td>
<td>5,000 - 10,000</td>
<td>1,000 - 5,000</td>
<td>500</td>
<td>20 - 150</td>
</tr>
<tr>
<td>Managed by</td>
<td>compiler</td>
<td>hardware</td>
<td>operating system</td>
<td>operating system</td>
<td>operating system</td>
</tr>
<tr>
<td>Backed by</td>
<td>cache</td>
<td>main memory</td>
<td>disk</td>
<td>disk</td>
<td>disk or tape</td>
</tr>
</tbody>
</table>

- Movement between levels of storage hierarchy can be explicit or implicit
The CPU-Memory Gap

The gap widens between memory, disk, and CPU speeds.
Caching

- Skew rule: 80% requests hit on 20% hottest data
- Important principle, performed at many levels in a computer (in hardware, operating system, software)
- Information in use copied from slower to faster storage temporarily
- Faster storage (cache) checked first to determine if information is there
  - If it is, information used directly from the cache (fast)
  - If not, data copied to cache and used there
- Cache smaller than storage being cached
  - Cache management important design problem
  - Cache size and replacement policy
Intel Core i7 Cache Hierarchy

Core 0

Regs

L1 d-cache

L1 i-cache

L2 unified cache

L3 unified cache
(shared by all cores)

Main memory

Core 3

Regs

L1 d-cache

L1 i-cache

L2 unified cache

L3 unified cache
(shared by all cores)

L1 i-cache and d-cache:
32 KB, 8-way,
Access: 4 cycles

L2 unified cache:
256 KB, 8-way,
Access: 11 cycles

L3 unified cache:
8 MB, 16-way,
Access: 30-40 cycles

Block size: 64 bytes
Migration of Integer A from Disk to Register

- Multitasking environments must be careful to use most recent value, no matter where it is stored in the storage hierarchy

- Multiprocessor environment must provide cache coherency in hardware such that all CPUs have the most recent value in their cache

- Distributed environment situation even more complex
  - Several copies of a piece of data can exist
File Management

- A file is a collection of related information defined by its creator
- Commonly, files represent programs (both source and object forms) and data
- The operating system responsibilities
  - File creation and deletion
  - Directory creation and deletion
  - Support of primitives for manipulating files and directories
  - Mapping files onto secondary storage
  - File backup on stable (non-volatile) storage media
I/O System Management

- The Operating System will hide the peculiarities of specific hardware from the user.

- In Unix, the I/O subsystem consists of:
  - A buffering, caching and spooling system
  - A general device-driver interface
  - Drivers for specific hardware devices

- Interrupt handlers and device drivers are crucial in the design of efficient I/O subsystems.
Protection and Security

- Protection – any mechanism for controlling access of processes or users to resources defined by the OS

- Security – defense of the system against internal and external attacks
  - Huge range, including denial-of-service, worms, viruses, identity theft, theft of service

- Systems generally first distinguish among users, to determine who can do what
  - User identities (user IDs, security IDs) include name and associated number, one per user
  - User ID then associated with all files, processes of that user to determine access control
  - Group identifier (group ID) allows set of users to be defined and controls managed, then also associated with each process, file
  - Privilege escalation allows user to change to effective ID with more rights
User Operating System Interface

- Two main approaches
  - *Command-line interpreter* (a.k.a. *command interpreter*, or *shell*)
  - *Graphical User Interfaces* (GUI)

- The shell
  - allows users to directly enter commands that are to be performed by the operating system
  - is usually a system program (not part of the kernel)

- GUI allows a mouse-based window-and-menu system

- Most of today’s desktop OS’s allow both (Windows, Mac OSX, Linux)
System Calls

- System calls provide the interface between a running program and the operating system.
  - Generally available in routines written in C and C++
  - Certain low-level tasks may have to be written using assembly language.

- Typically, application programmers design programs using an *application programming interface (API)*.

- The run-time support system (run-time libraries) provides a *system-call interface*, that intercepts function calls in the API and invokes the necessary system call within the operating system.

- Major differences in how they are implemented (e.g., Windows vs. Unix)
Example System Call Processing
Syscall: write(fd, buf, nbytes)

*source: http://r00tkit.me/?p=46
Major System Calls in Unix: File Management

- `fd = open(file, how, …)`
  - Open a file for reading, writing, or both
- `s = close(file)`
  - Close an open file
- `n = read(fd, buf, nbytes)`
  - Read data from a file into a buffer
- `n = write(fd, buf, nbytes)`
  - Write data from a buffer into a file
- `position = lseek(fd, offset, whence)`
  - Move the file pointer
- `s = stat(name, &buf)`
  - Get a file’s status info
Perspective of a Programmer
Demo 4: Under the hood of a syscall

```c
#include <stdio.h>
#include <unistd.h>
#include <assert.h>
#include <fcntl.h>
#include <sys/types.h>
#include <string.h>

void do_work() {
    int fd = open("/tmp/file", O_WRONLY | O_CREAT | O_TRUNC,
                S_IRUSR | S_IWUSR);
    assert(fd >= 0);
    char buffer[20];
    sprintf(buffer, "hello world\n");
    int rc = write(fd, buffer, strlen(buffer));
    assert(rc == strlen(buffer));
    printf("wrote %d bytes\n", rc);
    fsync(fd);
    close(fd);
}

int main(int argc, char *argv[]) {
    do_work();
    return 0;
}
```