The University for
world-class professionals

Manchester Metropolitan University

Advanced Operating Systems

Lecture 1: An overview on Operating Systems

Soufiene Djahel
Office: John Dalton E114

Email: s.djahel@mmu.ac.uk

Telephone: 0161 247 1522

This lecture was originally designed by Emma Norling, MMU, UK

Learning Outcomes

- Critically assess the principles of modern operating systems, particularly with respect to distributed systems;
- Research, evaluate and critically analyse current trends in operating systems research and implementation;

Learning Outcomes

- Critically assess the technologies and architectural principles of modern large-scale computer communications systems, both wired and wireless;
- Implement, research, evaluate and critically analyse current challenges and future technological trends in the field of computer networking;

Outline (first 6 weeks)

- Overview on Operating Systems
- Concurrency
- Cloud computing
- Distributed file systems
- Virtual machines
- Fault tolerance

Assessment

- 40% coursework, 60% exam
- Coursework consists in
  - Writing a survey paper on a chosen topic in either Operating System or Computer Networks

Today’s Objectives

- To review background knowledge:
  - Operating systems principles
  - Processes, threads and scheduling
  - Concurrency
  - Memory management
  - File system management
  - Privacy and Security

Background Text Books

- Operating Systems: Internals and Design Principles (8th ed.), Stallings
  - This one is available through the library as an e-resource
- Operating System Concepts (9th ed.), Silberschatz, Galvin, & Gagne – hereafter referred to as *OSC*
- Earlier editions of either should be fine, the main difference is in the examples provided.

Principles of Operating Systems

A Little History

- Phase 1: no operating system
  - Programmers interacted directly with the hardware
  - Time allocated in blocks
- Phase 2: monitors
  - Programmers submitted jobs to operators, who bundled them into batches
  - Monitor processed all jobs in a batch, with minimal gaps between them

History (continued)

Read one record from file	15 μs
Execute 100 instructions	1 μs
Write one record to file	15 μs
Total	31 μs

$$\text{Percent CPU Utilization} = \boxed{\frac{1}{31} = 0.032 = 3.2\%}$$

- Phase 3: Multi-programmed batch systems
  - sequential batch systems spend a lot of time idle, waiting for I/O
  - multi-programmed batch systems can switch to a different job when the current job is waiting for I/O, minimising idle time

History (continued)

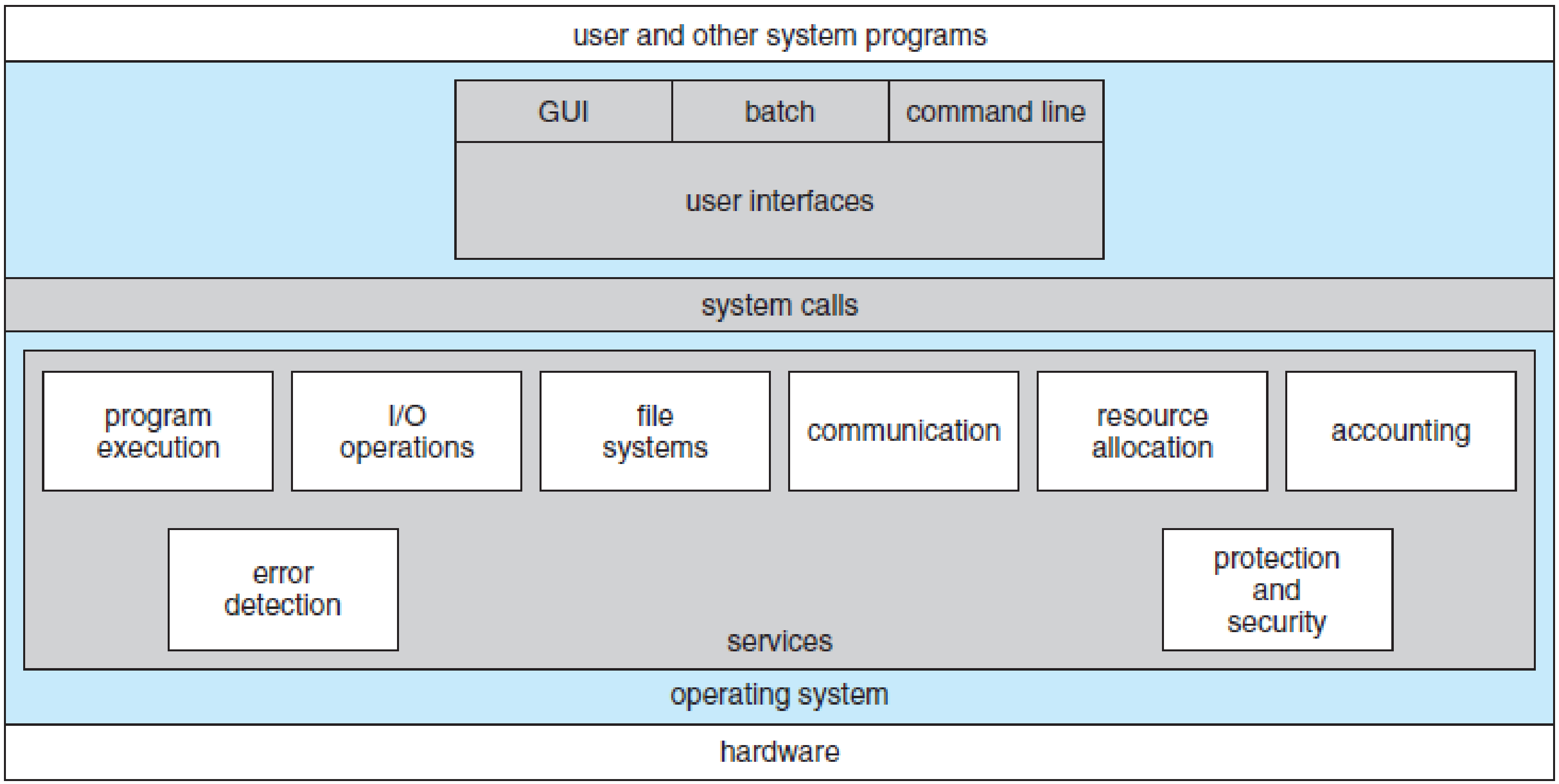
- Phase 4: Interactive systems
  - modern operating systems
  - allows user interaction with programs

Multi-programmed batch systems	Interactive systems
Main goal is to maximise the CPU use	Main goal is to minimize the response time

The Purpose of Operating Systems

- An interface between the user/programmer and underlying hardware
- Resource management
  - Efficient
  - Fair
  - Secure

A View of Operating System Services



(From slides accompanying *Operating Systems Concepts, 9th Ed.* Silberschatz, Galvin and Gagne © 2013)

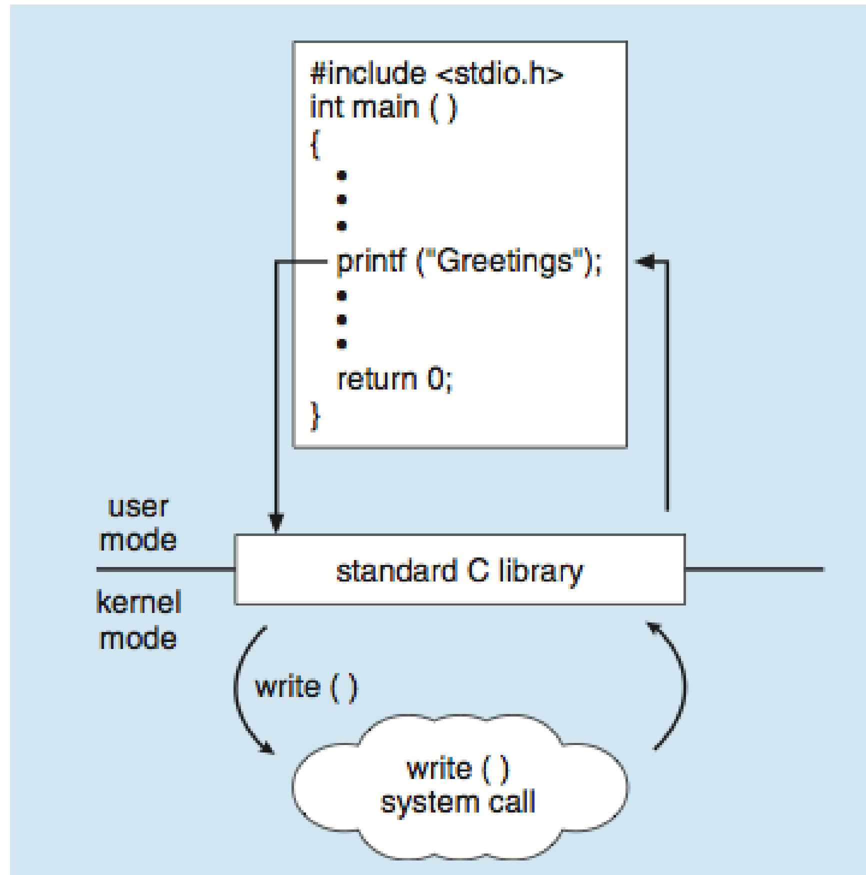
System Calls

- Programming interface to the OS services
- Mostly accessed by programs via a high-level *Application Program Interface (API)* rather than direct system calls
  - Three most common:
    - Win32 API for Windows
    - POSIX API for all POSIX-based systems (*nix, OS X)
    - Java API for the Java virtual machine (JVM)
- Programming languages can add further abstraction

An example:

C program invoking printf() library call, which calls write() system call

(From slides accompanying *Operating Systems Concepts, 9th Ed.* Silberschatz, Galvin and Gagne © 2013)



System Programs

System programs provide a convenient environment for program development and execution

- Some of them are simply user interfaces to system calls; others are considerably more complex

File management (create, delete, copy, rename, print, dump, list, and generally manipulate files and directories)

Status information (system info – date, time, memory usage, etc; performance, logging and debugging)

File modification (text editors, commands to search and transform text)

Programming-language support (compilers, assemblers, debuggers and interpreters)

System Programs

Program loading and execution (absolute loaders, relocatable loaders, linkage editors, and overlay-loaders, debugging systems for higher-level and machine language)

Communications (provide the mechanism for creating virtual connections among processes, users, and computer systems)

Background Services (disk checking, scheduling, logging, print managers)

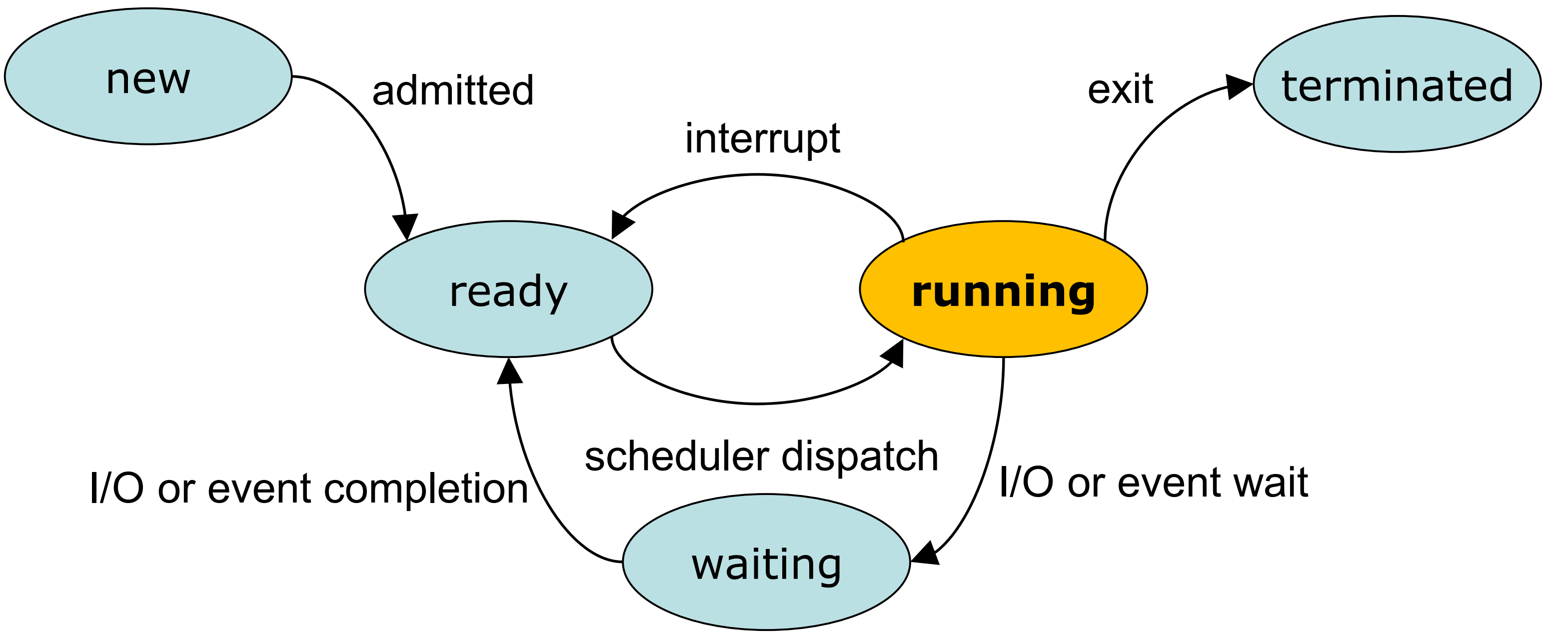
Implementing Operating Systems

- There are many different examples – Windows, Unix(es), Linux, OS X, iOS, Android…
- Two key questions:
  - Policy: *What* should the OS provide?
  - Mechanism: *How* should it provide it?
- Our focus is on these two questions, not particular examples of operating systems

Process Management

Processes

- A **program** is a passive entity, stored on disk (or other media), waiting to be executed
- Once active, we refer to its instantiation as a **process**
- Active processes typically move through a range of different states during their lifecycle



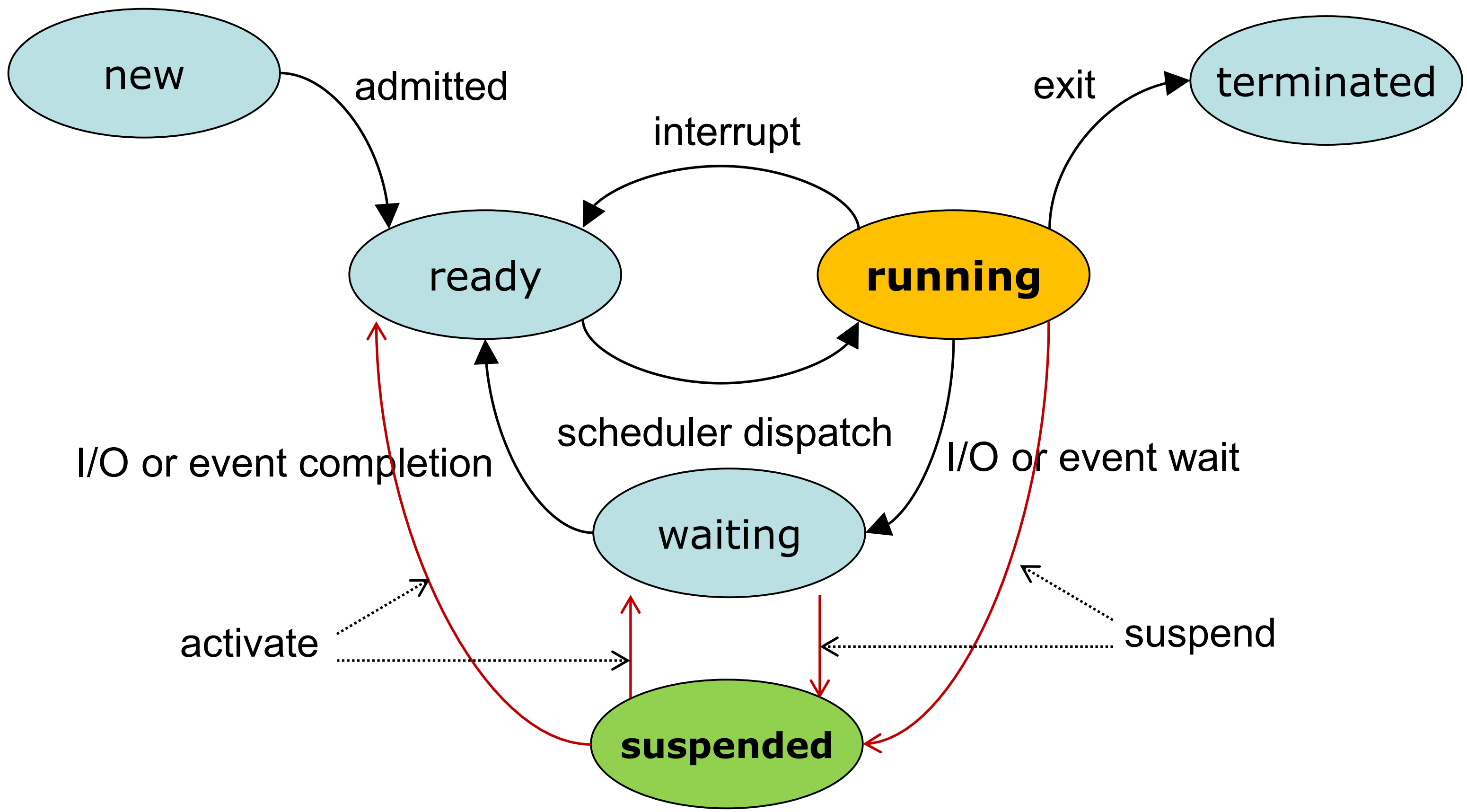
Adapted from Fig. 3.2, *Operating Systems Concepts, 9th Ed.* Silberschatz, Galvin and Gagne © 2013)

New process

- New batch job
  - OS manages a batch job control stream, reading next job when it is ready
- Interactive log-on
  - User logs on to the system
- OS service
  - E.g. print manager
- Spawned by existing process

Terminated process

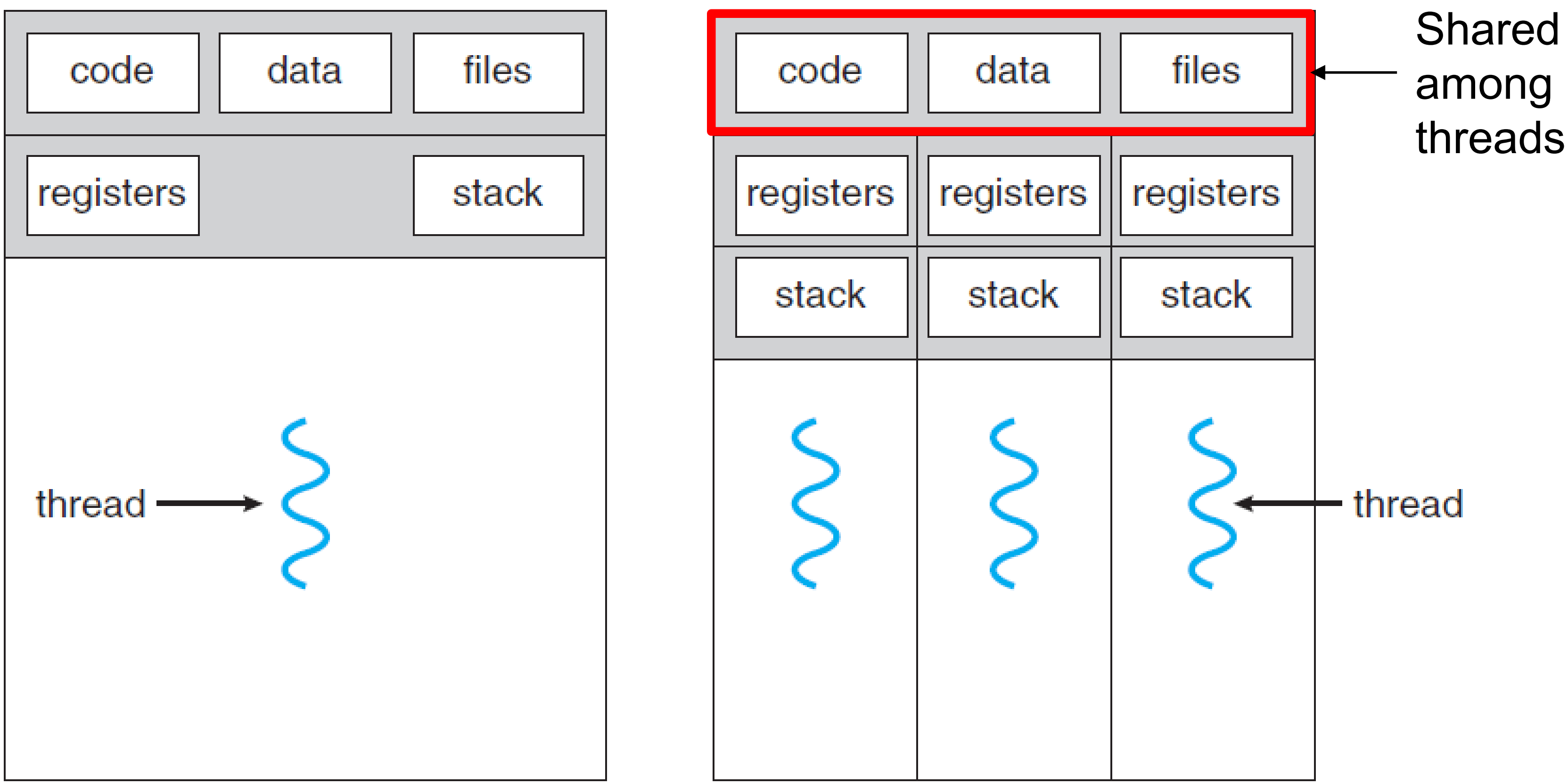
- Normal termination
- Time limit exceeded
- Memory unavailable
- Protection error
- Arithmetic error
- Timeout
- I/O failure
- Invalid instruction
- Data misuse
- Operator/OS intervention
- Parent termination
- Parent request



Adapted from Fig. 3.2, *Operating Systems Concepts, 9th Ed.* Silberschatz, Galvin and Gagne © 2013)

Threads

- A process
  - Has an associated set of resources (memory, files, I/O channels, etc.) that it "owns"
  - Also follows a particular execution path (trace)
  - These two characteristics are independent
- A **thread** is a particular path of execution within a larger process



single-threaded process

multithreaded process

(From slides accompanying *Operating Systems Concepts, 9th Ed.* Silberschatz, Galvin and Gagne © 2013)

Scheduling

- One of the resources that must be managed by the OS is the CPU
  - When it becomes idle, which process should be run next?
  - If a process is using the CPU, maybe it should sometimes be forced to relinquish it
- These decisions form the basis of CPU scheduling

Some Scheduling Criteria

User Oriented

- Performance-related:
  - Turnaround time
  - Response time
  - Deadlines

System Oriented

- Performance-related:
  - Throughput
  - Processor utilization
  - Fairness
  - Enforcing priorities
  - Balancing resources

Scheduling Algorithms

- Cannot hope to optimise for all the scheduling criteria
- Range of algorithms, each having different strengths and weaknesses
- **Non-pre-emptive** algorithms make choices when there is no currently running process
- **Pre-emptive** algorithms can send an interrupt to a currently-running process to switch to a different process

Scheduling Algorithms

- First Come First Served (FCFS)
- Shortest Job First (SJF)
- Shortest Remaining Time (SRT)
- Round Robin (RR)
- Priority scheduling

Evaluating Scheduling Algorithms

- Average response ratio is most common measure
  - Response ratio = T_r/T_s

    where

    T_r is the response time,

    time of completion – time of arrival

    T_s is the service time

    amount of time process actually uses processor
- Another consideration is starvation

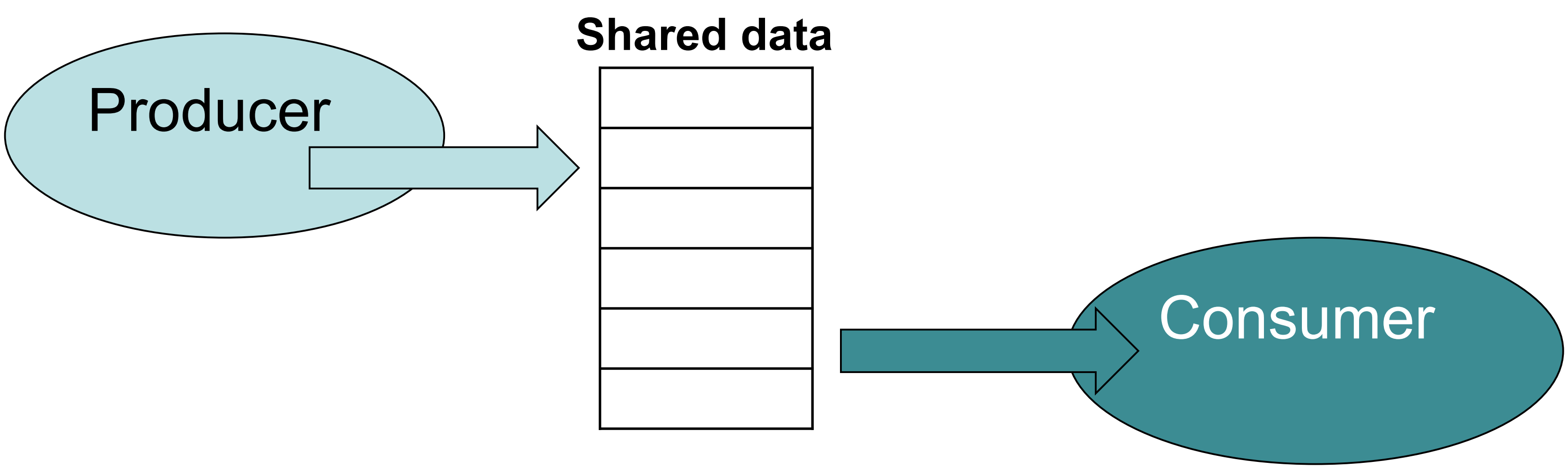
Additional Complications

- Basic scheduling algorithms assume a single processor system
- Multiprocessor systems add extra considerations
  - Assignment of processes to processors
  - Multiprogramming of individual processors

Concurrency

A Classic Problem

- Producers and consumers: one process produces data, the other consumes it



Shared data:

```
#define BUFFER_SIZE 10

typedef struct {
    …
} item;

item buffer[BUFFER_SIZE];
int in = 0;
int out = 0
int counter = 0;
```

Producer:

```
while (true) {
    // produce next_value
    while (counter == BUFFER_SIZE)
        ; // do nothing
    buffer[in] = next_value;
    in = (in + 1) % BUFFER_SIZE;
    counter++;
}
```

Consumer:

```
while (true) {
    while (counter == 0)
        ; // do nothing
    next_out = buffer[out];
    out = (out+ 1) % BUFFER_SIZE;
    counter--;
    // consume next_out
}
```

The Problem

Race Condition

- counter++

  $register_1$ = counter

  $register_1$ = $register_1$ + 1

  counter = $register_1$

- counter--

  $register_2$ = counter

  $register_2$ = $register_2$ - 1

  counter = $register_2$

- Possible execution (assume counter =5):

P	$register_1$ = counter	$register_1$ = 5
P	$register_1$ = $register_1$ + 1	$register_1$ = 6
C	$register_2$ = counter	$register_2$ = 5
C	$register_2$ = $register_2$ - 1	$register_2$ = 4
P	counter = $register_1$	counter = 6
C	counter = $register_2$	counter = 4

Race Conditions and Critical Sections

- Race condition is when multiple processes read and write data so that the final result depends on the order of execution of instructions
- Critical section is the section of code that must be protected in order to avoid a race condition

Hardware and Software solutions

- Interrupt disabling
  - Only suitable for uniprocessors
- Special machine instructions
  - Applicable to multiple processor systems
  - Possibility of starvation and deadlock
- Mutex locks
- Semaphores
- Monitors
  - Data encapsulation

Memory Management

Background: Memory Structure

- Program must be brought (from disk) into memory and placed within a process for it to be run
- Main memory and registers are only storage CPU can access directly
- Memory unit only sees a stream of addresses + read requests, or address + data and write requests

Background: Memory Structure

- Register access in one CPU clock (or less)
- Main memory can take many cycles, causing a **stall**
- **Cache** sits between main memory and CPU registers
- Protection of memory required to ensure correct operation

Memory Management Unit

- Memory management unit (MMU) maps logical addresses to physical addresses
- Many different mechanisms for this, some involving specialist hardware
  - Base and limit registers
  - Segmentation
  - Paging

Virtual Memory

- Virtual memory allows programs to be only partially loaded into memory
  - Can allow programs to use more memory than the computer actually has
  - Can speed up multiprogramming
- Program's memory divided into sections; only some sections are in memory at any given time
  - Remainder in secondary storage

Demand Paging

- Variation on simple paging, where pages are only loaded into memory when required
- When program tries to access an address which is not loaded, a *page fault* is generated
- Leads to following actions:
  - Get empty frame
  - Swap page into empty frame
  - Change to valid bit in page table
  - Restart operation that caused page fault

Page and Frame Replacement

- **Page replacement algorithm**
  - Used to decide which page to replace when no free frame
  - Want lowest page faults
- **Frame allocation algorithm determines**
  - How many frames to give each process
  - Which frames to replace
- In general, **more frames ⇒ less page faults**

File System Management

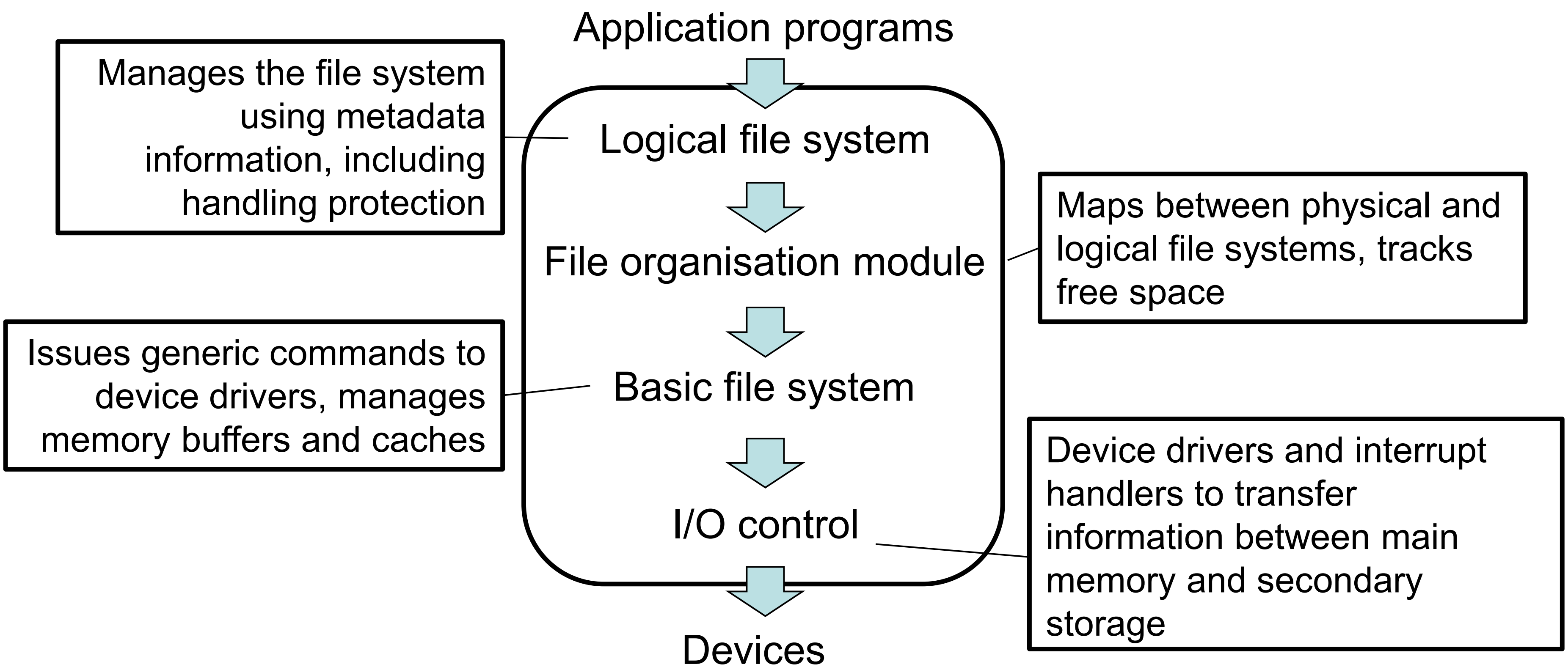
Files and File Systems

- OS provides both low-level disk (or other storage) management, and high-level structure to facilitate data storage
- A file is a contiguous logical address space
  - Many different types
    - Data or program
    - Structured or unstructured (or semi-structured)
    - Sequential or random access

File System Implementation

- Two independent design problems:
  - How the file system should look to the user
  - Algorithms and data structures to map logical file system to physical storage
- Typically implemented as a layered approach

Layered File System

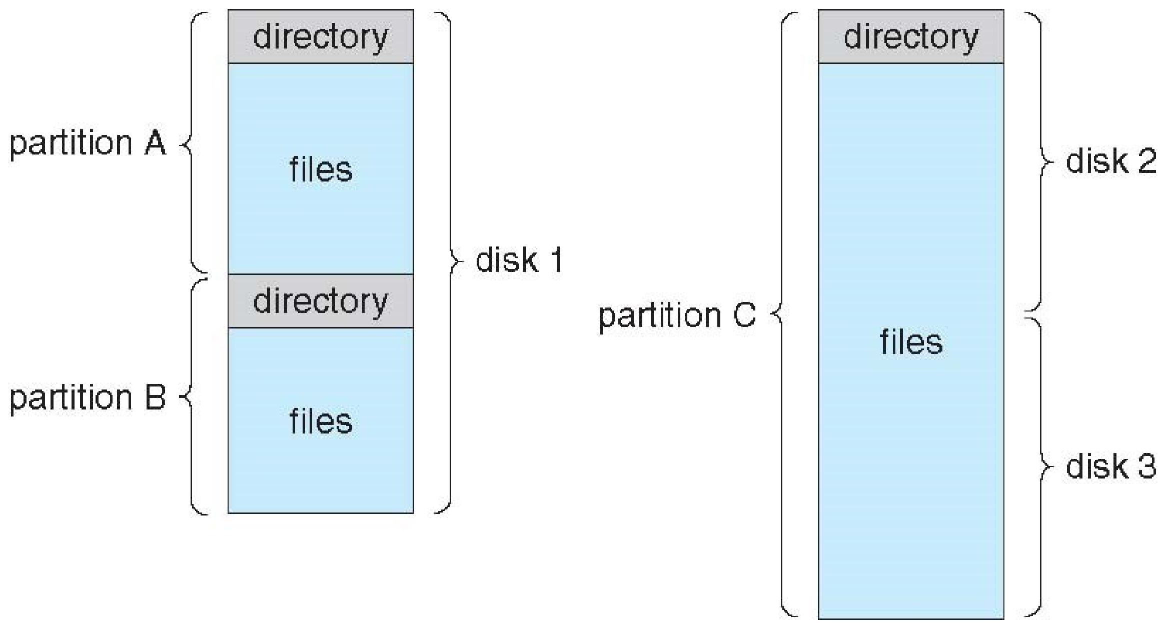


File System Data Structures

- *Boot control block* contains info needed by system to boot OS from that volume
- *Volume control block* (superblock, master file table) contains volume details
  - For mapping from logical to physical file structure
- Directory structure organizes the files
- Per-file attributes

Volumes and Partitions

- (Physical versus logical structure)



Directory Structures

- Requirements:
  - Efficiency (quickly locate a file)
  - Naming
  - Grouping
    - Logical grouping by properties
      - Type, size, date of creation/modification, etc.

File Operations

- Operating System needs to provide means to:
  - Create
  - Write (at write point)
  - Read (at read point)
  - Reposition within file (seek)
  - Delete
  - Truncate
  - **Open** (load into memory)
  - **Close** (store to disk)

System calls

File Attributes

- Name
- Identifier
- Type
- Location
- Size
- Protection
- Time, date
- User identification
- Extended attributes

Typical File Control Block

file permissions
file dates (create, access, write)
file owner, group, ACL
file size
file data blocks or pointers to file data blocks

Privacy and Security

Principles of Computer Security

Computer security is “the protection afforded to an automated information system in order to attain the applicable objectives of preserving the integrity, availability, and confidentiality of information system resources (includes hardware, software, firmware, information/data, and telecommunications).”

(from *Operating Systems: Internals and Design Principles, 7th Ed.*, Stallings © 2012)

Key Objectives

- Confidentiality
  - data confidentiality
  - privacy
- Integrity
  - Data integrity
  - System integrity
- Availability

Security Threats

- Unauthorised disclosure
  – entity gains access to data for which entity is not authorised
- Deception
  – entity receives false data that it believes to be true
- Disruption
  – system services prevented from running correctly
- Usurpation
  – system services under control of unauthorised entity

Security and Protection

- In order to meet these objectives, a key role of operating systems is *protection*, aiming to minimise the risk from these threats through:
  - authentication
  - access control
  - intrusion detection
  - malware protection

Authentication

- Basis for access control
  - who can access which files, processes, devices, etc.
- Basis of accountability
  - system use is logged on a user-by-user basis
- Three forms:
  - Something the user knows (password)
  - Something the individual possesses (smart tokens)
  - Something the individual *is* (biometrics)

Access Control

- Dictates what types of access are permitted, by whom, and under what circumstances
- Three categories:
  – Discretionary: rules describe entity's access. Entities may have ability to grant access to other entities.
  – Mandatory: Fixed rules describe entity's access.
  – Role-based: Access based on entity's role within the system

Intrusion Detection

- Motivated by:
  1. If an intruder is detected quickly enough, they can be quarantined before causing damage
  2. An efficient intruder detection system can serve as a deterrent
  3. The more intruders that are detected, the better overall picture is gathered, leading to better prevention measures
- Relies on recognising typical patterns of behaviour

Malware Defence

- Antivirus approaches
  - generic decryption
  - digital immune system
  - behaviour-blocking system
- Worm countermeasures
- Bot countermeasures
- Rootkit countermeasures

Next lecture

- Concurrency, particularly deadlocks