Operating Systems
CS-C3140, Lecture 2

Alexandru Paler

Grading

- The grade is based on points
    - The exercises: n (8?) rounds and 28 points/round -> n*28 points from exercises
    - The exam points are max. 400 - n*28
- To pass the course you have to pass both the exam and the exercises
    - A minimum number of points for the exam (to be announced separately)
    - Final grade = round ((exercises + exam + 100) / 100)
        - 449 -> 400 -> 4
        - 451 -> 500 -> 5
    - Per round limits for the exercises (you have to pass (n-2)/n rounds)
        - 50% of max points for passing the assignments
        - 75% of max points if submitted one week later
        - 2 trials per exercise
            - Do not solve the exercises in the A+ interface
            - Use A+ only for submission
            - Solve in different software (e.g. Notepad)

Resource sharing is typical

- The basic view of a computer
  - Processor, memory, peripheral devices
- Typically, there are several programs active
  - Even with a single processor (note the concept of concurrency)
  - They have to share the processor, the memory, and the devices
  - This calls for management resource sharing
- Further
  - Each program typically consists of several parts
    - How they can be active simultaneously?
  - The computer can be part of a distributed systems
    - Plenty of sharing of resources, plenty of concurrency

https://www.computer-history.info/Page4.dir/pages/PDP.1.dir/
https://www.computerhistory.org/pdp-1/timesharing/

Getting a program to run and Running it

- Source code is compiled into machine code
  - Compilers usually come with a runtime system (or similar)
  - Mechanisms for memory allocation are often there
- The compilation result has
  - Program code (text),
  - static data, meta data (e.g., symbol table)
  - other things (like debug info)
- Program parts linked together:
  - executable program
  - stored into a file, and loaded into memory
- Note that
  - There can be other phases
  - In modern systems, the phases are mixed (e.g., a compiler does some initial linking and a loader does some final compilation)

- Getting a program to run takes some time
  - Copying bytes, but also setting values, etc.
  - The CPU must be set
    - Especially the special registers like PSW, PC, FP, SP, etc.
    - This is typically rather fast (compared to memory)

The major parts (segments) in memory are
  - Text (the program)
  - Data (what we are computing)
  - Stack (the control)
    - The stack together with the CPU state forms the context of the computation, which is essential for control
  - Note that
    - Specific formats, padding, etc. is used
    - Typically, a small fraction of the total memory address space is used

The control stack vs. execution context

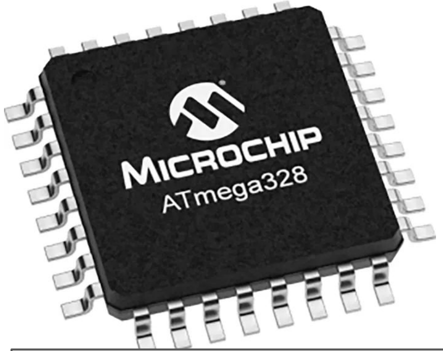
- Programs typically have subroutines (or “methods”, etc.)
  - These can be called (or “invoked”, etc.)
  - Usually, the caller remains activated as the callee is executing
    - After the callee execution, the control returns to the caller
- This is usually implemented with a control stack
  - The stack contains frames (abstractly: subroutine activations)
  - The most recent is at the top of the stack
    - Note typically stacks grow downwards (i.e., the “top” is at low end)
  - The FP points to the topmost frame and the frames are linked
    - The FP is used for accessing data
  - The SP point to the stack top
    - The SP is used for allocating (or deallocating space)

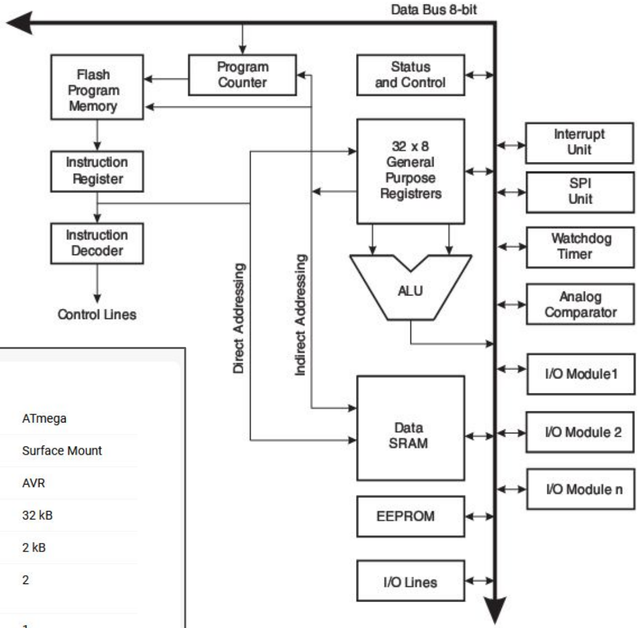
- When code is executed, the execution context defines
  - The code and data bindings that are not in the executed code itself (e.g., variable bindings)
  - Is dependent on the source language, on the hardware, on all the layers participating the computation
- There are several levels of execution context
  - CPU state and a stack imply the context for single thread
  - Note that there can be several threads within a program
- In a wider view
  - There are open files, active network connections, etc.
  - Much of such context resides in the OS (or middleware layers)

System Types

Microcontrollers

- Single-chip computers
  - Designed for embedding
  - CPU
  - Memory
  - I/O subsystem
- CPU
  - For RT (predictable)
- Memory
  - Different types
- I/O subsystem
  - Analog/digital
  - Programmable





Tekniset tiedot

Merkki	Microchip	Family Name	ATmega
Package Type	TQFP	Mounting Type	Surface Mount
Pin Count	32	Device Core	AVR
Data Bus Width	8bit	Program Memory Size	32 kB
Maximum Frequency	20MHz	RAM Size	2 kB
Number of PWM Units	1	Number of SPI Channels	2
Number of UART Channels	1	Number of I2C Channels	1
Typical Operating Supply Voltage	5.5 (Maximum) V	Number of USART Channels	1
Maximum Operating Temperature	+85 °C	Length	7.1mm
ADCs	8 x 10 bit	Height	1.05mm
Width	7.1mm	Minimum Operating Temperature	-40 °C
Dimensions	7.1 x 7.1 x 1.05mm	Pulse Width Modulation	1 (6 Channel)
Instruction Set Architecture	RISC	Number of ADC Units	1
Program Memory Type	Flash		

Credit:
Microchip

Boards

- Instead of buying a microcontroller,
  you typically buy a “single board computer”
  - With suitable capabilities (especially the interfaces)
  - Can be very cheap… or somewhat more expensive…
- Example: an ODROID system
  - An I/O shield at the top
  - Physical I/O (rich electric IFs)
  - Real-time processing (RT)
  - Slow but predictable
- ODROID-U3
  - Has processing power
  - Fast but unpredictable (not RT)
  - The cyber-side
  - Runs easily a full Linux
  - Network I/O

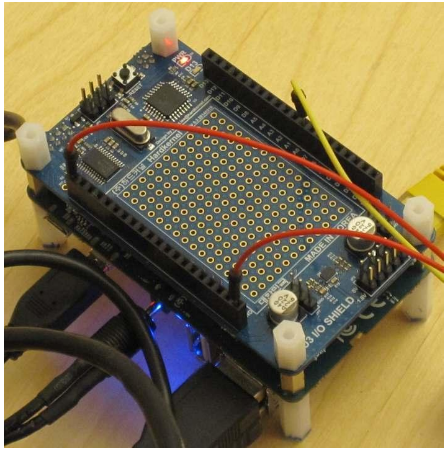
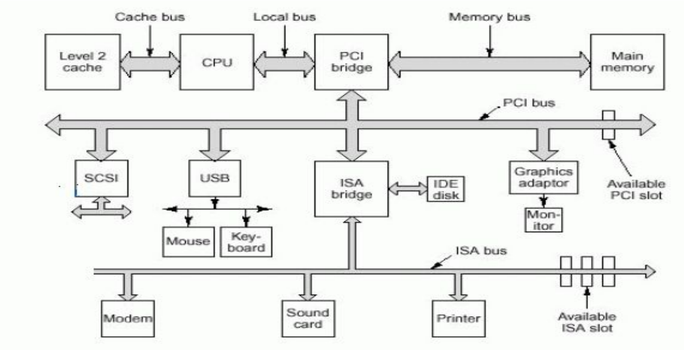
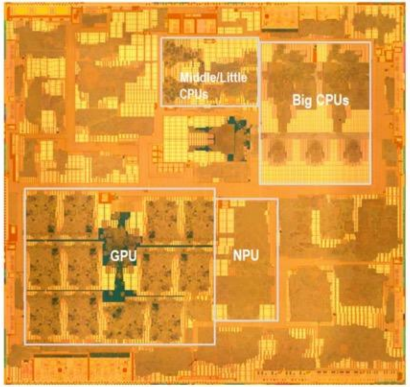


Photo: Vesa Hirvisalo

The classical PC (Personal Computer)

- Centered around the CPU-MEM axis
  - Busses, bridges, controllers
  - peripheral devices
- PCs have (more or less) evolved into laptops
  - Basically, more integration and compactness
  - resemble older PCs a lot
- Smartphones with touch screens dominate the outlook
  - share much with the PC, laptops, etc.





Source DOI: 10.1109/ISSCC19947.2020.9062907

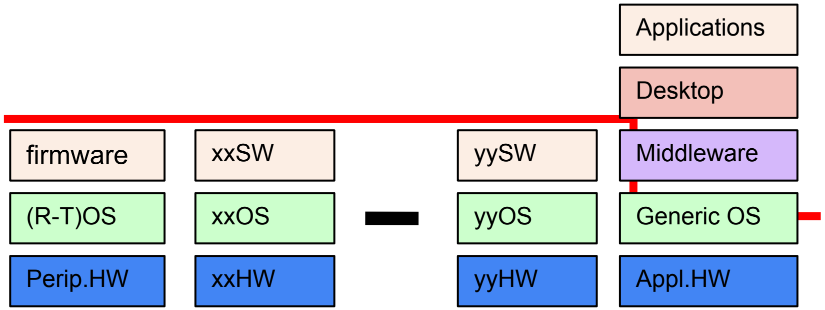
Embedded computers

- In embedded systems
  - A computer is embedded into a host
  - A host can be a classical physical system, i.e., a car
- A traditional computer (e.g. PC)
  - The focus on computation
  - The processor (CPU) and memory
  - The software computes something (of interest)
- An embedded computer
  - The focus is on interaction
  - Sensors and actuators attached
  - … but there is a lot of variance!
- Industrially
  - An embedded computer is typical for a specific purpose
  - Very often “custom made” (ASICs, etc.)
  - Embedded computer = special purpose computer
  - Are expensive (not only HW manufacturing, but the testing, etc.)

Much of software is embedded
- OS theory ~= “how to construct a SW system/stack”
- single process -> single stack -> a system

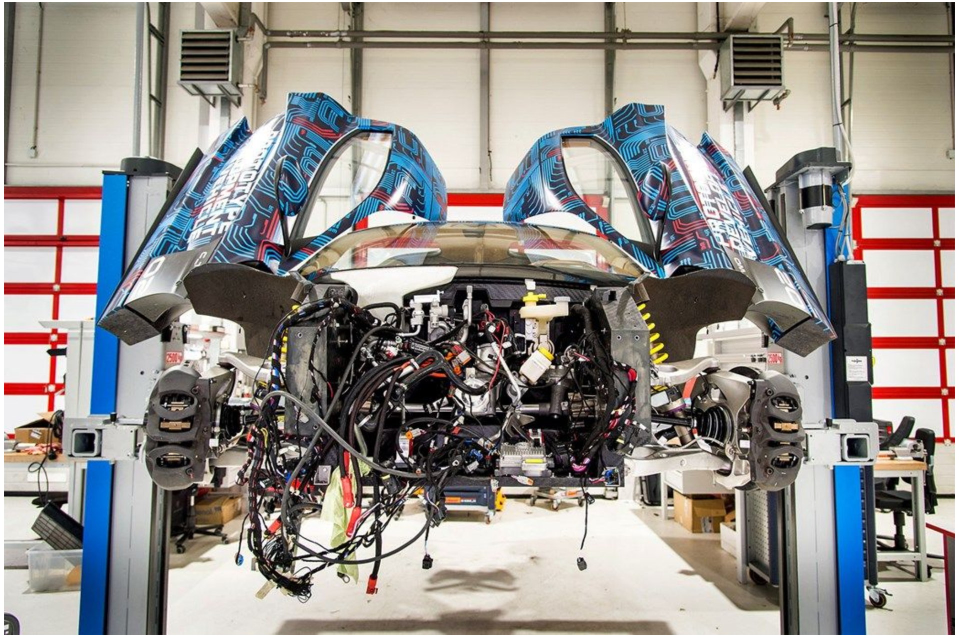
In a device we have
- many processors (inside peripheral devices)
- on them, many SW stacks and OSes
- the application stack is special

Distribution makes things very complex



Modern cars

- Power train & basic driving
  - Engine and transmission control
  - Steering etc. (ABS, ESP, …)
  - Monitoring systems (TPMS, OBD-II, ...)
- Dashboard, infotainment, etc.
  - Lights, signals, doors, windows, locking, …
  - Heating, ventilation, and air conditioning (HVAC)
  - Anti-theft systems, …
- Actually, there are a lot of things
  - Tens of processors, huge number of sensors and actuators
  - Cars are connected to the cyber-world outside
  - Several communication links, at least one SIM, …



https://www.carmagazine.co.uk/car-news/industry-news/rimac/mate-rimac-electric-cars/

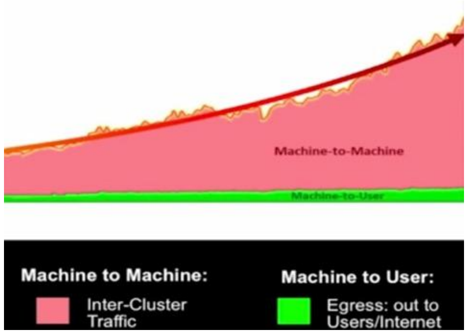
Clusters and Data centers

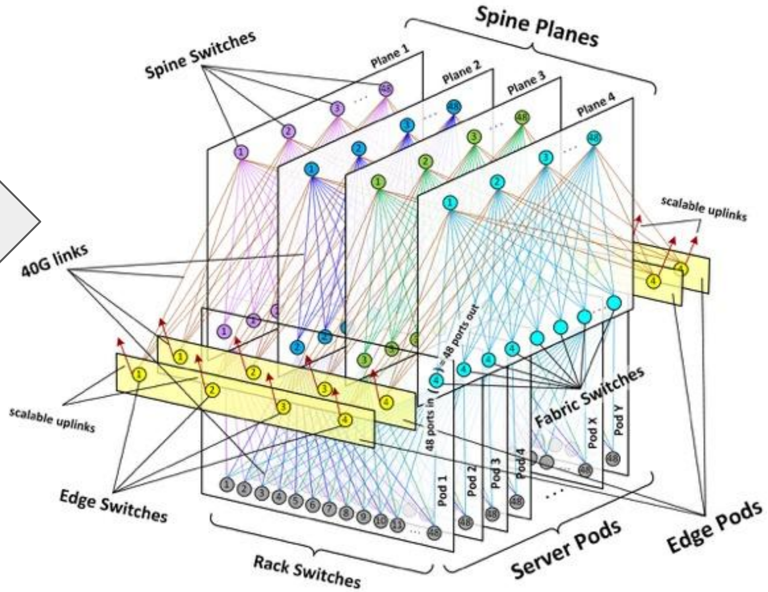
- A computing cluster is a set of connected computer working together for a joint purpose
  - Each node (computer) is usually set to perform the same task
- There is a wide variety of computing clusters
  - This is because of the variety of reasons for having clusters
- Better performance, dependability, efficiency, costs, etc. than what can be achieved with a single computer
  - Note that many modern “single computer systems” have merged plenty of cluster technology inside them
    - In practice, the mechanisms of distributed computing are used
- HPC (High-Performance Computing) has been the main driving for in many respects

- Typical data centers are warehouse size computers
  - Built around a set of networks
- Usually, there is
  - A set of networks
    - Technically, networking is the central thing for data centers
  - A hierarchy of memory and storage
    - Computing typically happens by altering their contents
  - The processing units
    - The processors sit on the top of the complex networking and memory/storage systems
  - In addition
    - Plenty issues on power, cooling, etc.
    - It really is a house!

Data center architectures

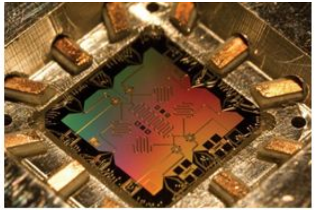
- Datacenter to Internet traffic is huge
  - Traffic inside the data centers is several orders of magnitude larger
  - Datacenter technology is much about communication
- Traditional hierarchical cluster based designs do not scale with growing traffic
  - New approach is to make the entire data center building one high-performance network
- Automated tools for management
  - A large network with a complex topology and many devices and interconnects cannot be configured and operated in a manual way



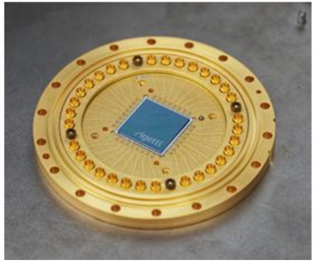


Source: Facebook

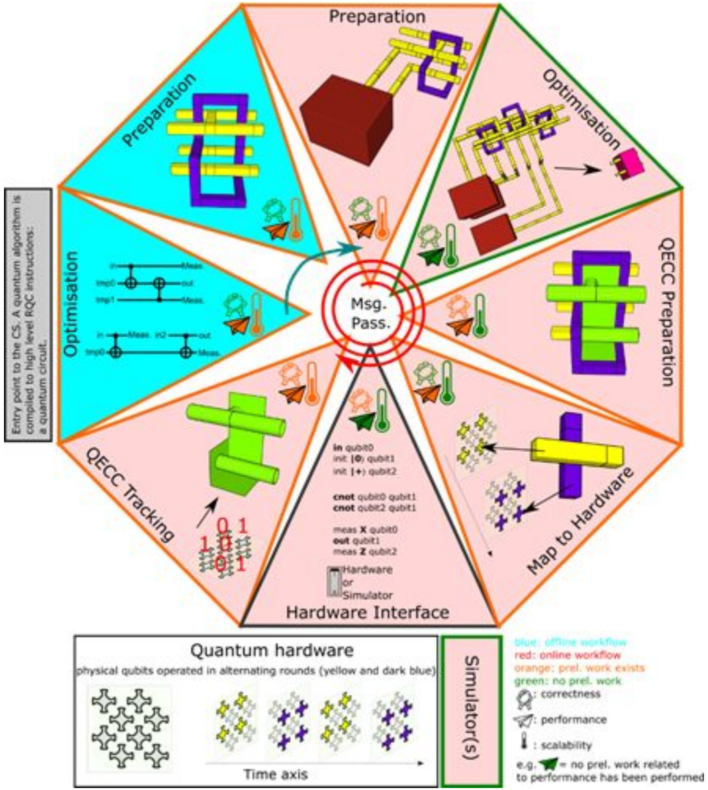
Quantum Computers



The era of NISQ?
Noisy Intermediate-Scale Quantum



Quantum Error-Correction
Quantum Algorithms
Design Automation
Computer Architecture
Machine Learning (?)



Operating Systems Overview

Computers and operating systems

- A computer is a device consisting of
  - CPU
  - Memory
  - I/O peripherals: disk, display, network card
- A computer is executing programs
  - Perform arithmetic or logical computations
  - Have control
  - Do input and output (I/O)
- An operating system is a special program
  - Controls access to computer peripherals
  - Enables other programs to run
- How many different operating systems have you used?

Operating System

- A program that controls the execution of application programs
- An interface between applications and hardware

Main objectives of an OS:

- Convenience
- Efficiency
- Ability to evolve

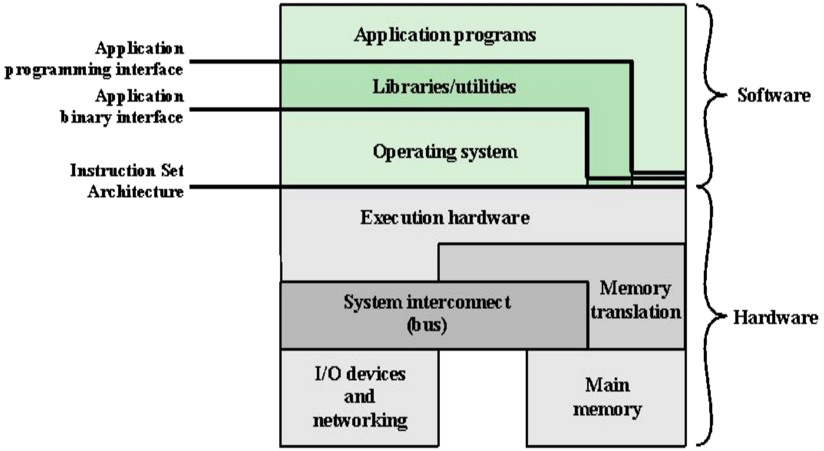


Figure 2.1 Computer Hardware and Software Structure

Operating System Services

- Program development
- Program execution
- Access I/O devices
- Controlled access to files
- System access
- Error detection and response
- Accounting

- Key Interfaces
  - Instruction set architecture (ISA)
  - Application binary interface (ABI)
  - Application programming interface (API)
- **POSIX - Portable Operating System Interface**
  - system- and user-level application programming interfaces (API),
  - command line shells and utility interfaces, for software compatibility (portability) with variants of Unix and other operating systems
  - Certified: e.g macOS
  - Mostly compliant: e.g. Linux, OpenBSD

Operating System as Resource Manager

- The OS is responsible for controlling the use of a computer's resources, such as
  - I/O
  - main and secondary memory
  - processor execution time
- Functions in the same way as ordinary computer software
  - Program, or suite of programs, executed by the processor
  - Frequently relinquishes control and must depend on the processor to allow it to regain control

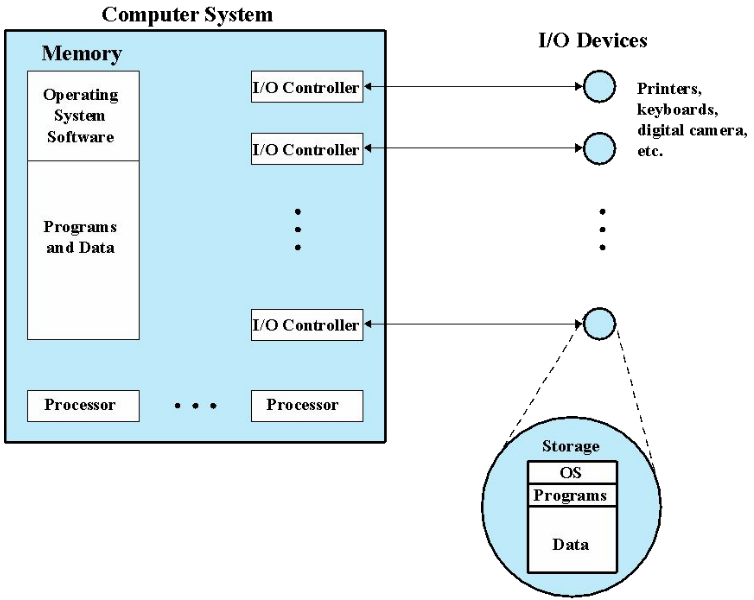


Figure 2.2 The Operating System as Resource Manager

Evolution of Operating Systems

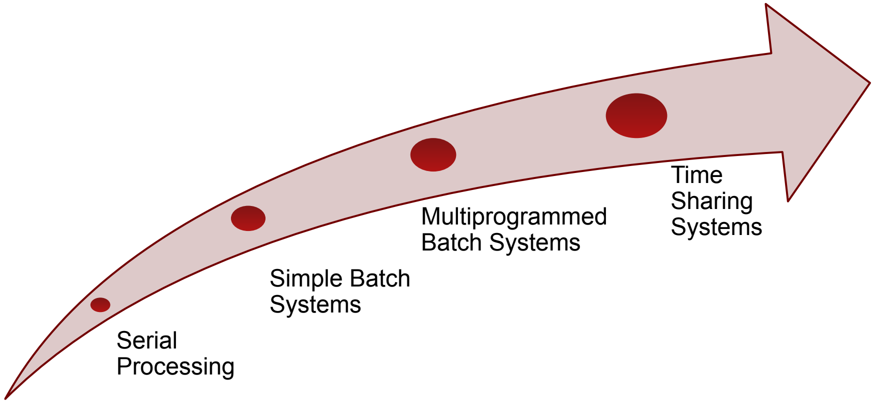
- A major OS will evolve over time for a number of reasons:

Hardware upgrades

New types of hardware

New services

Fixes



Serial Processing

Earliest Computers:

- No operating system
- Programmers interacted directly with the computer hardware
- Computers ran from a console with display lights, toggle switches, some form of input device, and a printer
- Users have access to the computer in "series"

- Scheduling
  - Most installations used a hardcopy sign-up sheet to reserve computer time
  - Time allocations could run short or long, resulting in wasted computer time
- Setup time
  - A considerable amount of time was spent on setting up the program to run

Simple Batch Systems

- Early computers were very expensive
  - Important to maximize processor utilization
- Monitor
  - User no longer has direct access to processor
  - Monitor controls the sequence of events
  - Resident Monitor is software always in memory
  - Job is submitted to computer operator who batches them together and places them on an input device
  - Program branches back to the monitor when finished

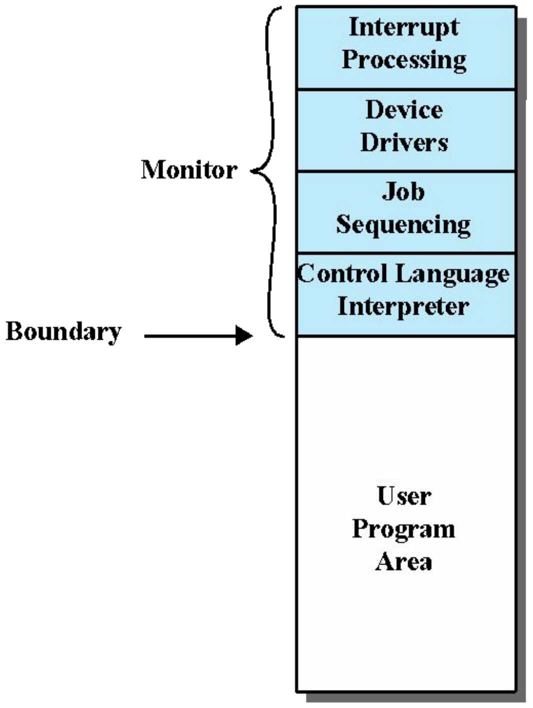


Figure 2.3 Memory Layout for a Resident Monitor

Modes of Operation

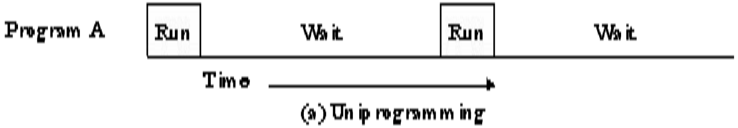
User Mode

- User program executes in user mode
- Certain areas of memory are protected from user access
- Certain instructions may not be executed

Kernel Mode

- Monitor executes in kernel mode
- Privileged instructions may be executed
- Protected areas of memory may be accessed

Uniprogramming



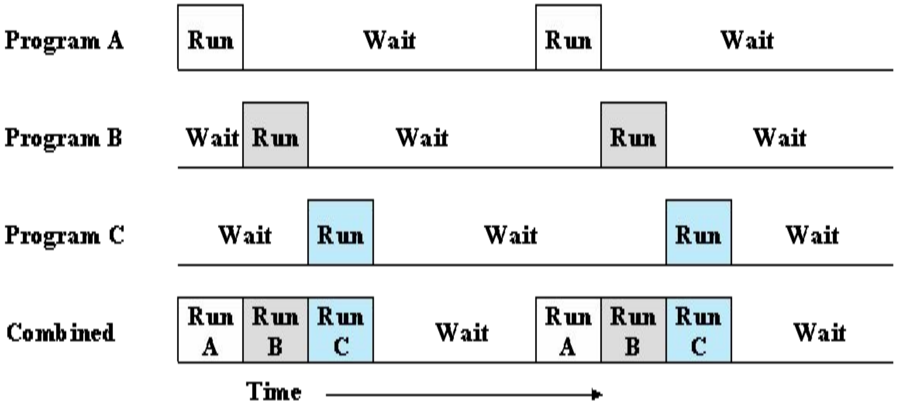
Read one record from file	15 µs
Execute 100 instructions	1 µs
Write one record to file	15 µs
TOTAL	31 µs

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

Figure 2.4 System Utilization Example

The processor spends a certain amount of time executing, until it reaches an I/O instruction; it must then wait until that I/O instruction concludes before proceeding

Multiprogramming aka multitasking



(c) Multiprogramming with three programs

- There must be enough memory to hold the OS (resident monitor) and one user program
- When one job needs to wait for I/O, the processor can switch to the other job, which is likely not waiting for I/O
- Memory is expanded to hold three, four, or more programs and switch among all of them

Multiprogramming Example

	JOB1	JOB2	JOB3
Type of job	Heavy compute	Heavy I/O	Heavy I/O
Duration	5 min	15 min	10 min
Memory required	50 M	100 M	75 M
Need disk?	No	No	Yes
Need terminal?	No	Yes	No
Need printer?	No	No	Yes

	Uniprogramming	Multiprogramming
Processor use	20%	40%
Memory use	33%	67%
Disk use	33%	67%
Printer use	33%	67%
Elapsed time	30 min	15 min
Throughput	6 jobs/hr	12 jobs/hr
Mean response time	18 min	10 min

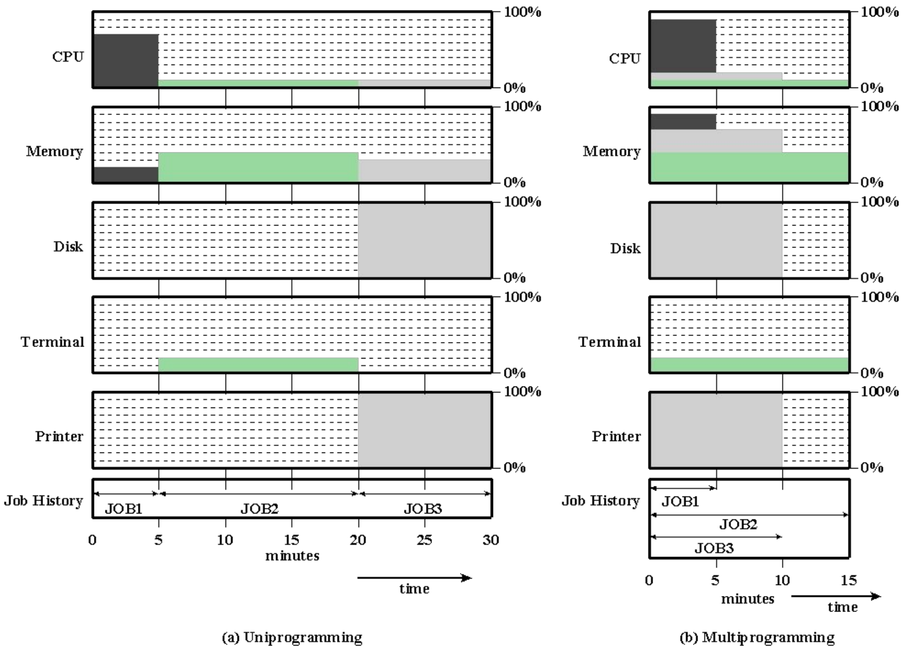


Figure 2.6 Utilization Histograms

Time-Sharing Systems

- Can be used to handle multiple interactive jobs
- Processor time is shared among multiple users
- Multiple users simultaneously access the system through terminals,
- The OS interleaves the execution of each user program in a short burst or quantum of computation

	Batch Multiprogramming	Time Sharing
Principal objective	Maximize processor use	Minimize response time
Source of directives to operating system	Job control language commands provided with the job	Commands entered at the terminal

Compatible Time-Sharing System (CTSS)

- One of the first time-sharing operating systems
  - Developed at MIT by a group known as Project MAC
  - The system was first developed for the IBM 709 in 1961
  - Ran on a computer with 32,000 36-bit words of main memory, with the resident monitor consuming 5000 of that
- Utilized a technique known as **time slicing**
  - System clock generated interrupts at a rate of approximately one every 0.2 seconds
  - At each clock interrupt the OS regained control and could assign the processor to another user
  - At regular time intervals the current user would be preempted and another user loaded in
- To preserve the old user program status for later resumption, the old user programs and data were written out to disk before the new user programs and data were read in
- Old user program code and data were restored in main memory when that program was next given a turn

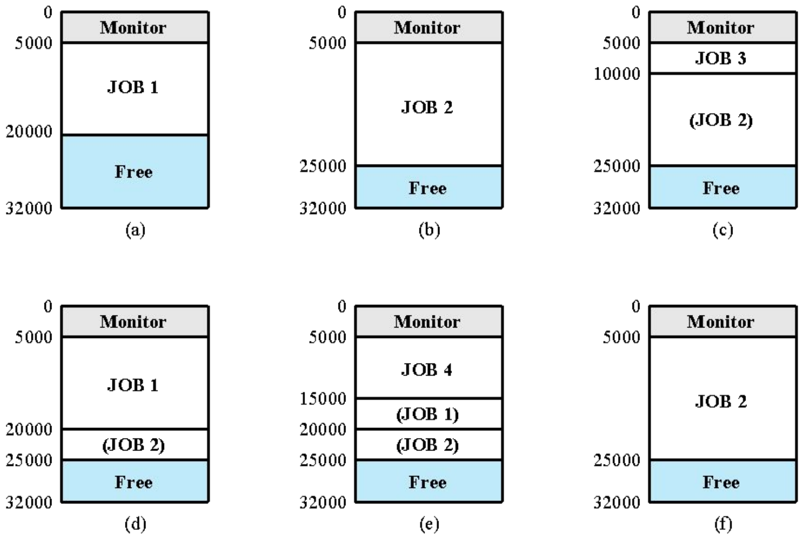


Figure 2.7 CTSS Operation

Major Achievements in OS

Major Achievements

- Operating Systems are among the most complex pieces of software ever developed
- Major advances in development include:
    i. Processes
    ii. Memory management
    iii. Information protection and security
    iv. Scheduling and resource management
    v. System structure

1. Process

Fundamental to the structure of operating systems

A *process* can be defined as:

- A program in execution
- An instance of a running program
- The entity that can be assigned to, and executed on, a processor
- A unit of activity characterized by a single sequential thread of execution, a current state, and an associated set of system resources

Components of a Process

- A process contains three components:
  - An executable program
  - The associated data needed by the program (variables, work space, buffers, etc.)
  - The execution context

- The execution context is essential:
  - It is the internal data by which the OS is able to supervise and control the process
  - Includes the contents of the various process registers
  - Includes information such as the priority of the process and whether the process is waiting for the completion of a particular I/O event

Process Management

- The entire state of the process at any instant is contained in its context
- New features can be designed and incorporated into the OS by expanding the context to include any new information needed to support the feature

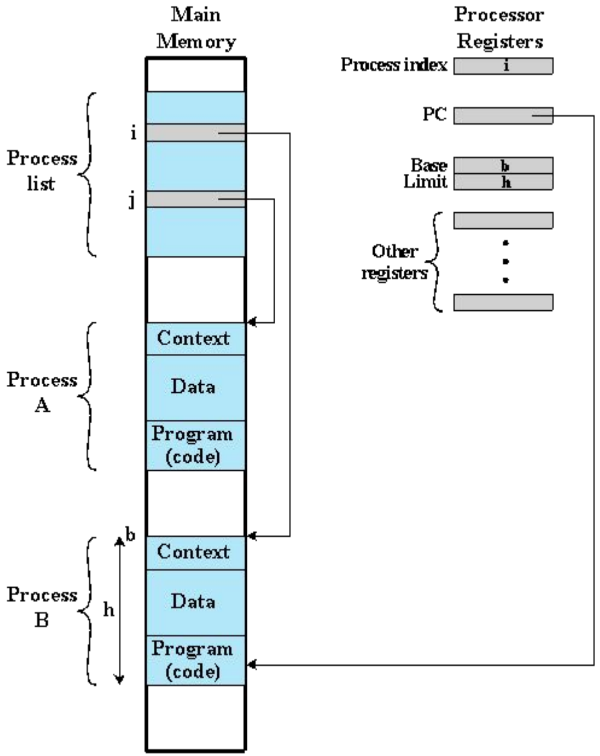


Figure 2.8 Typical Process Implementation

2. Memory Management

- The OS has **five** principal storage management responsibilities:

Process isolation

Automatic allocation and management

Support of modular programming

Protection and access control

Long-term storage

2. Virtual Memory and Paging

- A facility that allows programs to address memory from a logical point of view, without regard to the amount of main memory physically available
- Conceived to meet the requirement of having multiple user jobs reside in main memory concurrently

- Allows processes to be comprised of a number of fixed-size blocks, called pages
- Program references a word by means of a virtual address, consisting of a page number and an offset within the page
- Each page of a process may be located anywhere in main memory
- The paging system provides for a dynamic mapping between the virtual address used in the program and a real address (or physical address) in main memory

2. Virtual Memory and Paging

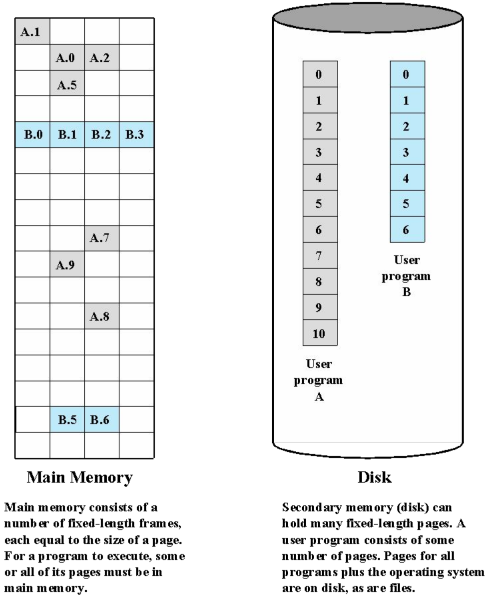


Figure 2.9 Virtual Memory Concepts

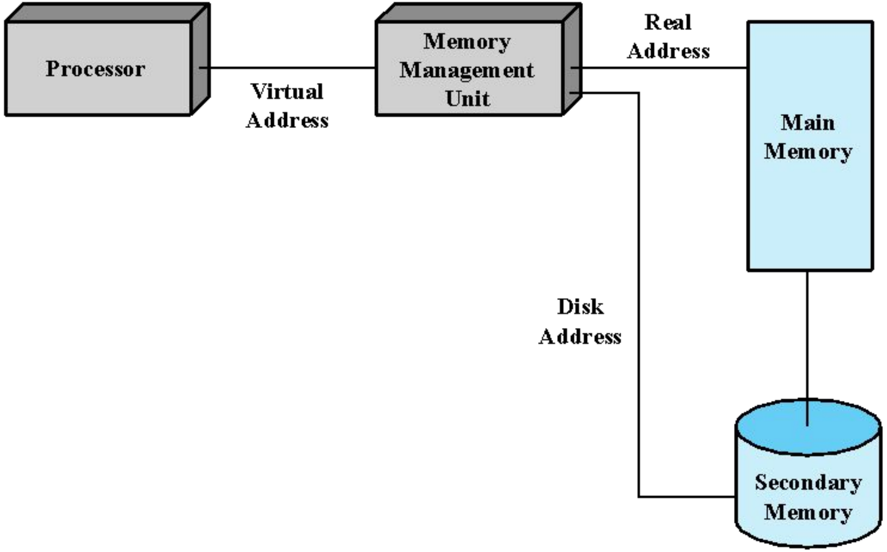
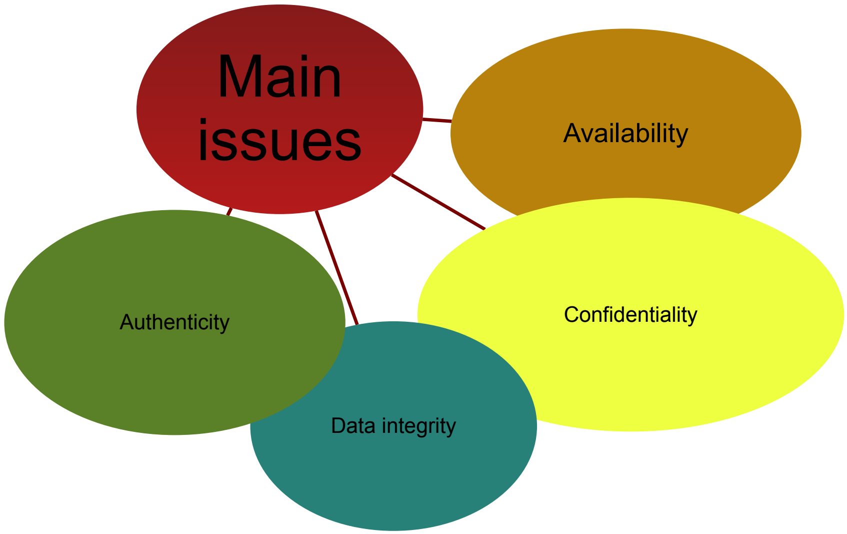


Figure 2.10 Virtual Memory Addressing

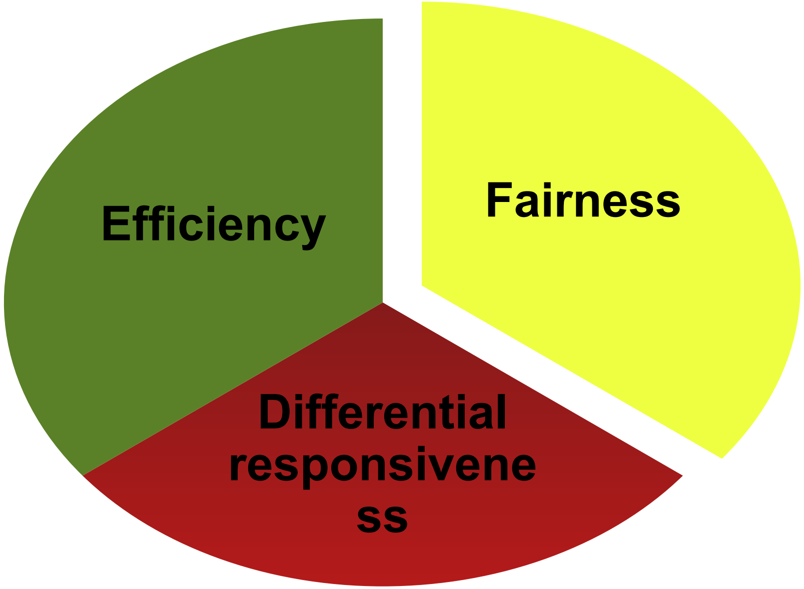
3. Information Protection and Security

- The nature of the threat that concerns an organization will vary greatly depending on the circumstances
- The problem involves controlling access to computer systems and the information stored in them



4. Scheduling and Resource Management

- Key responsibility of an OS is managing resources
- Resource allocation policies must consider:



5. Different Architectural Approaches

- Demands on operating systems require new ways of organizing the OS

Different approaches and design elements have been tried:

- Microkernel architecture
- Multithreading
- Symmetric multiprocessing
- Distributed operating systems
- Object-oriented design

Microkernel Architecture

- Assigns only a few essential functions to the kernel:

Address space management

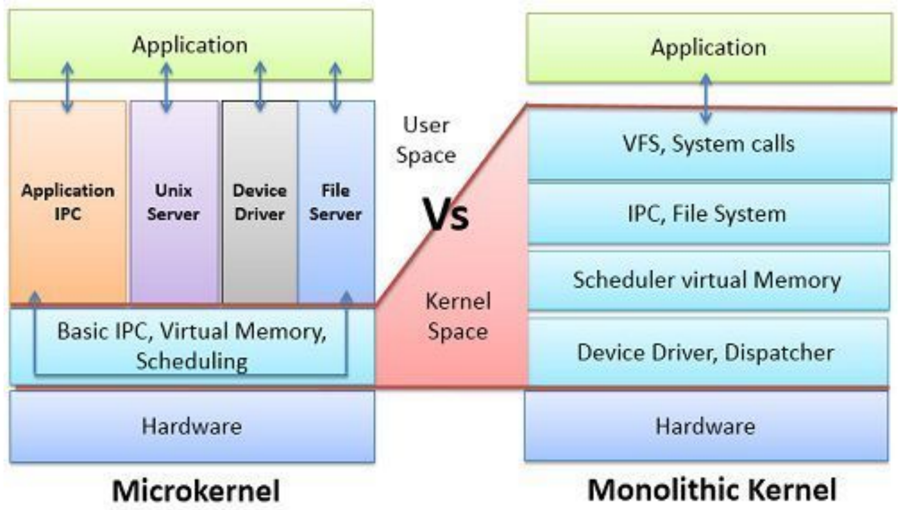
Interprocess communication (IPC)

Basic scheduling

Simplifies implementation

Provides flexibility

Well suited to a distributed environment

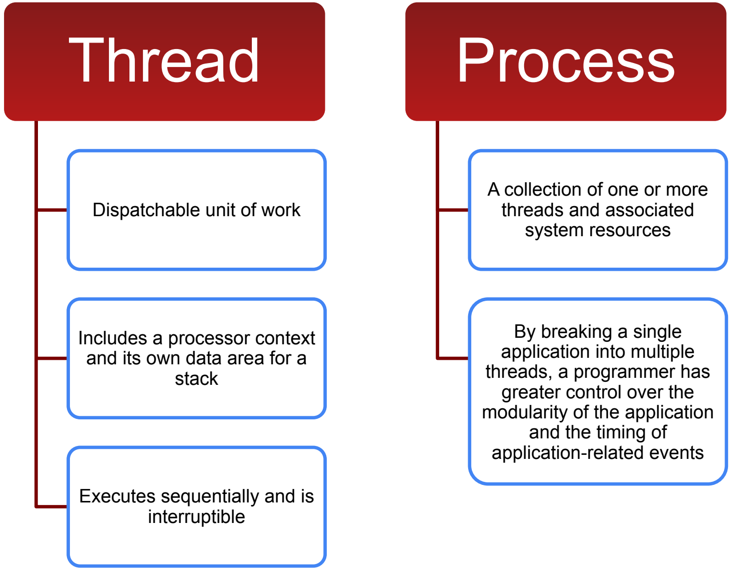


https://techdifferences.com/difference-between-microkernel-and-monolithic-kernel.html

Symmetric Multiprocessing and Multithreading

- Term that refers to a computer hardware architecture and also to the OS behavior that exploits that architecture
- The OS of an SMP schedules processes or threads across all of the processors
- The OS must provide tools and functions to exploit the parallelism in an SMP system
- Multithreading and SMP are often discussed together, but the two are independent facilities
- An attractive feature of an SMP is that the existence of multiple processors is transparent to the user

- Technique in which a process, executing an application, is divided into threads that can run concurrently



Multiprogramming and Multiprocessing