L1 Operating Systems Overview

What Is An Operating System? 什么是操作系统?

- An operating system (OS) is a **large and complex software system** which **manages all of the hardware resources of a computer and makes it easy to use.**

  操作系统（OS）是一个大型且复杂的软件系统，它管理计算机的所有硬件资源，并使计算机易于使用。

- Provides common services for application software to use **processing resources, memory space, disk storage, input/output devices and network communication functions of the underlying hardware.**

  为应用软件提供通用服务，以使用底层硬件的处理资源、内存空间、磁盘存储、输入/输出设备和网络通信功能。

Application Interaction 应用程序交互

- Applications interact with the operating system through **(APIs)** and **System Calls**, which define a set of **commonly required functions**

  应用程序通过一系列应用程序编程接口（APIs）和系统调用与操作系统交互，定义了常用函数

- Applications are therefore written for **specific operating systems.**

  因此，应用程序是为特定操作系统编写的。

- APIs **lessen the development effort** required to get an application running on the hardware

  操作系统API的可用性减少了使应用程序在硬件上运行所需的开发工作。

Environment 环境

- It is a **concurrent environment** where numerous tasks and events must be handled at the same time.

  它是一个并发环境，需要同时处理众多任务和事件

User Interaction 用户交互

- User interaction with **a graphical user interface or through a text based command-line interface.**

  用户与操作系统的交互通常通过图形用户界面或基于文本的命令行界面进行。

What Is Kernel? 什么是内核?

Definition 定义

- The Kernel of an operating system refers to the core set of services associated with managing the **CPU, the electrical memory, basic interprocess communication and low level hardware devices.** The kernel executes in a privileged mode on the processor where additional instructions and permissions are available.

操作系统的内核是指与管理CPU、电记忆体、基本进程间通信和低级硬件设备相关的操作系统的核心服务集合。内核在处理器的特权模式下执行，在该模式下可以使用附加的指令和权限。

Operating System Distribution

- An Operating System Distribution will have additional components to the Kernel such as a File System, a Database Engine, Network Communication Suite, Graphics and Media functions, a Web Server, a User Interface GUI, Security and Authentication elements, Device Drivers for various external I/O devices, and Utilities for configuring the system.

  操作系统发行版除了内核之外，还包含文件系统、数据库引擎、网络通信套件、图形和媒体功能、网络服务器、用户界面（GUI）、安全和认证组件、各种外部I/O设备的设备驱动程序，以及用于配置系统的实用工具等附加组件。
- 不同公司开发的的不同操作系统

Company	OS	Company	OS
Microsoft	MS-DOS, Windows	Canonical	Ubuntu
Google	Chrome, Android	SUN	Solaris
Huawei	Harmony OS	DEC	VMS
Apple	MacOS, iOS	Xiaomi	HyperOS, MIUI
Bell Labs	Unix	Linus Torvalds	Linux

Summary of the tasks an operating must perform 操作系统必须执行的任务概述

1. **Allocation and Management of Resources**

   分配和管理资源。
2. **Maximising Resource Utilisation**

   最大化资源利用率
3. **Providing a user interface and an application interface**

   提供用户界面和应用程序界面。
4. **Coordinating the many concurrent activities and devices, handling input and output**, ensuring correct synchronisation and communication is achieved.

   协调众多并发活动和设备，处理输入和输出，并确保实现正确的同步和通信。
5. **Protect various resources of the computer system**

   保护计算机系统的各种资源
6. **Accounting for periods of resource usage** by user processes

   记录用户进程的资源使用时间
7. **Power and Thermal Management.**

   电源和热管理。

L2 Process Management Concepts 进程管理概念

Programs, Processes, Processors 程序、进程、处理器

- A program is a collection of instructions specifying a defined sequence of execution. It is the translation of an algorithm into a programming language.
  程序是一组指令的集合，用于指定执行的预定顺序。它是将算法转换为编程语言。
- A process is an instance of a program in action. When the instructions are being carried out, a process exists.
  进程是程序在执行时的实例。当指令被执行时，进程就存在了。
- The executable module of a program is loaded as a program image in main memory and processor fetches instructions from it.
  程序的可执行模块作为程序映像被加载到主存储器中，处理器从中获取指令。

Representing Process Abstractions

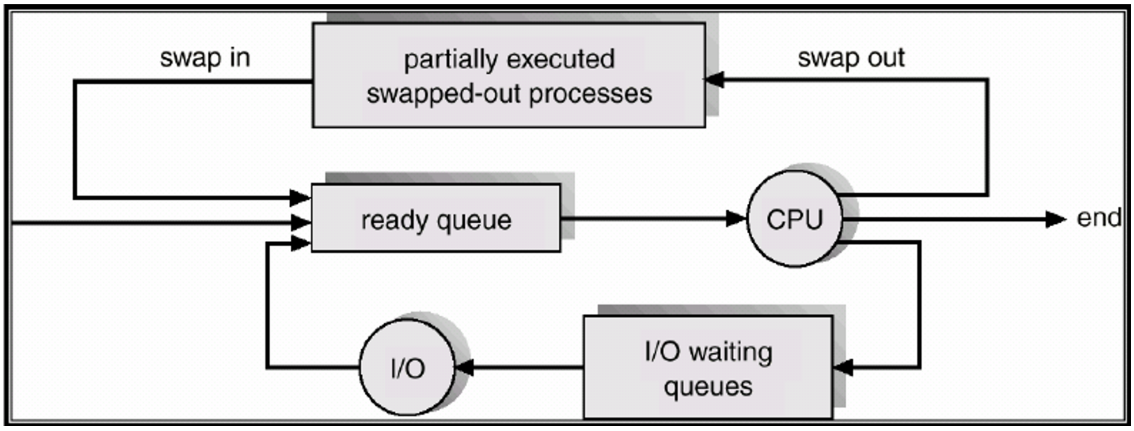
Process Control Block (PCB)

- For each process, the operating system maintains a process control block (PCB) or process descriptor to keep a clear picture of what each process is doing, what point it has reached in its execution and what resources have been assigned to it.
  为了完成这一基本任务，操作系统使用数据结构来表示其处理的进程对象的状态。对于每个进程，操作系统维护一个进程控制块（PCB）或进程描述符，以清晰地反映每个进程正在做什么、执行到了哪个阶段以及分配了哪些资源。
- A Process Control Block is used to keep track of the execution context (all resource information about the process and its activity) that can be used for independent scheduling of that process onto any available processor.
  进程控制块用于跟踪执行上下文（有关进程及其活动的所有资源信息），这些信息可用于将该进程独立地调度到任何可用的处理器上。

Process Lifecycle

Process States and Queues

- The PCB may be moved between different queues over the process lifetime depending on the priority or state of its execution.
  根据进程的优先级或执行状态，其 PCB 可能在进程生命周期内被移动到不同的队列中。

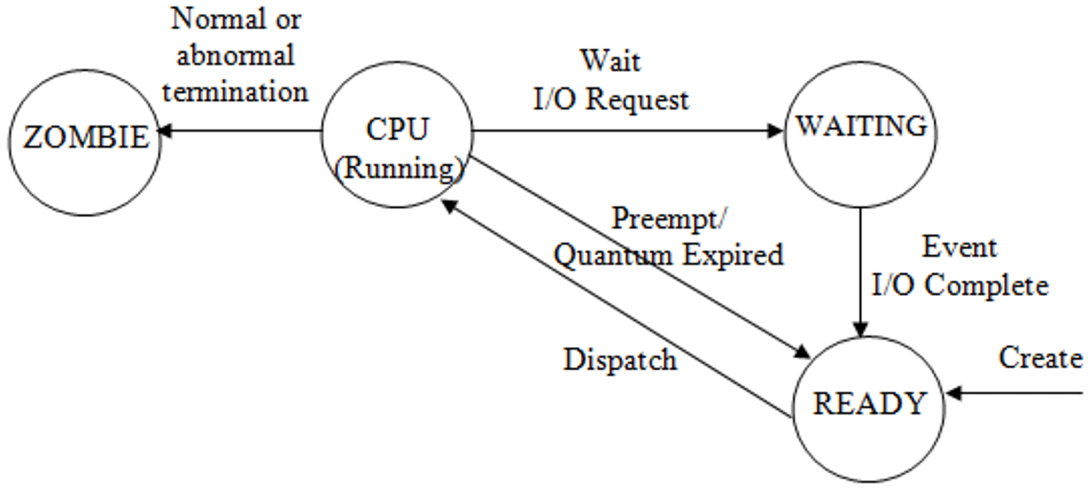


- A queued process is one which is waiting for its turn to run on the CPU.
  排队的进程是在等待其轮到在 CPU 上运行的进程。

- A zombie process is one which has terminated, but its resources, like memory and stack, have not been reclaimed by the OS.
  僵尸进程是已经终止，但其资源（如内存和栈）尚未被操作系统回收的进程。

Resource Waiting

- There will be periods where it needs the CPU to execute its instructions and there will be other periods where it is waiting on various other system resources, such as input/output devices, to provide data so that it can continue its execution.
  在某些时期，进程需要 CPU 来执行其指令，而在其他时期，它可能正在等待诸如输入/输出设备等其他系统资源提供数据以继续执行。



Simplified Process State Transition Diagram

- A running process is one that is currently executing on the CPU.
  正在运行的进程是当前正在 CPU 上执行的进程。
- A blocked process is one that is waiting for some event to occur, such as waiting for I/O operations to complete or for a resource to become available.
  被阻塞的进程是正在等待某个事件发生的进程，例如等待 I/O 操作完成或等待资源可用。

Communicating with the OS

System Calls 系统调用

- Communication with other processes in the system is regulated by using interprocess communication functions provided by the OS.
- Communication with the operating system is done via a special system call mechanism

  系统中进程之间的通信通过操作系统提供的进程间通信功能进行调节。

  与操作系统的通信通过特殊的系统调用机制完成，该机制会自动切换处理器的执行模式

Processor Modes 处理器 + 软件中断

- OS code executes in a privileged mode where a wider instruction set is available in the processor, instructions that are not available in user mode.
  操作系统代码在特权模式下执行，在该模式下处理器可用的指令集更广泛，这些指令在用户模式下不可用。

- A special processor instruction: software interrupt does this like accessing the hard disk or other hardware, creating new processes, doing interprocess communication or configuring kernel services.

  软件中断的特殊处理器指令实现例如访问硬盘或其他硬件、创建新进程、进行进程间通信或配置内核服务。

Hardware Interrupts 硬件中断

- Hardware Interrupt mechanisms are needed to implement a multitasking environment. When a task is to be scheduled to use the processor for a set amount of time, a clock timer is initialised. When the time expires an interrupt signal invokes the operating system scheduler to select another process. This scheme prevents one process from hogging the CPU indefinitely.
  要实现多任务环境，需要硬件中断机制。当任务被调度使用处理器设定时间时，会初始化一个时钟计时器。时间到期时，中断信号会调用操作系统调度程序选择另一个进程。这种方案防止一个进程无限期地占用 CPU。

*L3+4 Process Scheduling I+II

Process Scheduling Overview

Multitasking Concepts

- A multitasking operating system allows multiple processes to be resident and active on the computer system at one time.
  多任务操作系统允许多个进程同时驻留在计算机系统中并处于活动状态

Process Scheduling

- Each process must compete with others for the available resources. The operating system must decide which processes are assigned to use the processor(s) and for how long.
  每个进程必须与其他进程竞争可用资源。操作系统必须决定哪些进程被分配使用处理器以及使用多长时间

OS Control Mechanism

- When a process makes a system call or when its time quantum on the processor expires, the resulting software or hardware interrupt causes the CPU to stop fetching instructions from the currently assigned process
  当进程发出系统调用或其在处理器上的时间片用完时，由此产生的软件或硬件中断会使 CPU 停止从当前分配的进程获取指令

Scheduling Decision

- The operating system may then decide to allocate the CPU to another process, in which case it will execute code to perform a context switch. It saves the run-time state of the current process in the process control block so that it can be continued later, and then executes dispatcher code to load the run-time state of the chosen process for the CPU to continue executing instead.
  操作系统可能会决定将 CPU 分配给另一个进程，在这种情况下，它将执行代码以进行上下文切

换。它会将当前进程的运行时状态保存在进程控制块中，以便稍后继续执行，然后执行调度程序代码以加载选定进程的运行时状态，以便 CPU 继续执行。

Context Switch

- Context Switch code must be efficient. Usually supported by hardware instructions Incurs Performance Overhead, (CPU is spending time executing operating system code)
  上下文切换代码必须高效，通常由硬件指令支持会产生性能开销（CPU 花费时间执行操作系统代码）

Multiprocessing

Multiprocessing Concept

- Multiprocessing is the use of more than one CPU in a system, usually separate physical CPUs but the idea may also apply to a single CPU with multiple execution cores.
  多处理是指在系统中使用多个 CPU，通常是单独的物理 CPU，但这个概念也适用于具有多个执行核心的单个 CPU。

Asymmetric Multiprocessing

- With asymmetric multiprocessing, one processor, the master, centrally executes operating system code and handles I/O operations and the assignment of workloads to the other processors which execute user processes. Only one processor is accessing system data structures for resource control.
- The master may not have enough slaves , so maximum hardware performance is not achieved.
  在非对称多处理中，一个处理器（主处理器）集中执行操作系统代码并处理 I/O 操作以及将工作负载分配给其他处理器，这些处理器执行用户进程。只有主处理器访问用于资源控制的系统数据结构。
- 主处理器可能没有足够的从处理器，因此无法实现最大硬件性能。

Coordinating Multiprocessing Activity

- With Symmetric multiprocessing, the identical processors use a shared bus to connect to a single shared main memory and have full access to all I/O devices and are controlled by a single operating system instance. Each processor will examine and manipulate the operating system queue structures concurrently with others when selecting a process to execute. This access contention must be programmed carefully to protect the integrity of the shared data structures.
  在对称多处理中，相同的处理器使用共享总线连接到单一共享主内存，并且可以完全访问所有 I/O 设备，并由单一操作系统实例控制。每个处理器在选择要执行的进程时，会与其他处理器同时检查和操作操作系统队列结构。

  必须仔细编程这种访问竞争，以保护共享数据结构的完整性。

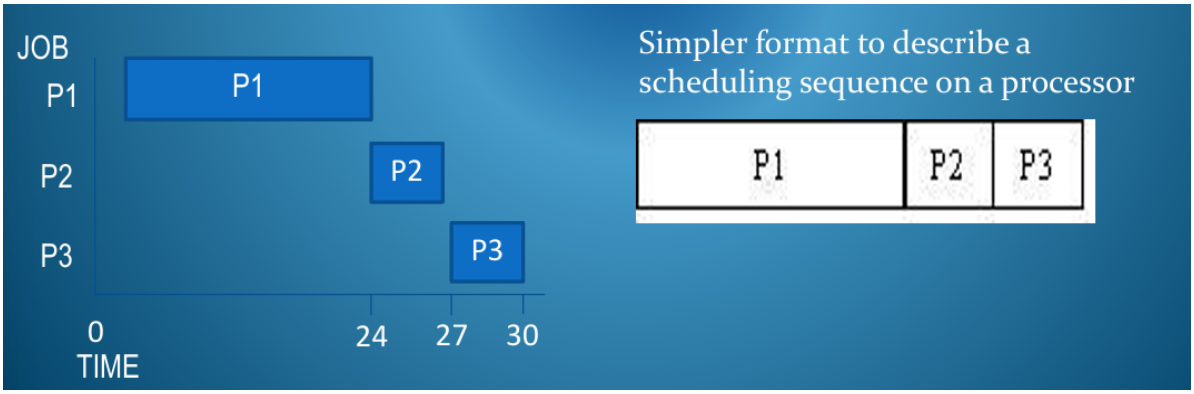
Scheduling Algorithm Performance Criteria

Criteria Metrics

- Processor Utilisation = (Execution Time) / (Total Time)
  CPU 利用率 =（执行时间）/（总时间）
- Throughput = Jobs per Unit time
  吞吐量 = 单位时间内的作业数
- Turnaround Time = (Time job finishes) - (Time job was submitted)
  周转时间 =（作业完成时间）-（作业提交时间）
- Waiting Time = Time doing nothing in a queue
  等待时间 = 在队列中无所事事的时间
- Response Time = (Time job is first scheduled on cpu) (Time job was submitted)
  响应时间 =（作业首次被调度到 CPU 上的时间）（作业提交时间）

Gantt Chart

- The order in which processes are serviced from a queue can be described by a Gantt Chart. It represents a scheduling sequence.
  进程从队列中被服务的顺序可以用甘特图描述。它表示一个调度序列。



Preemptive and Nonpreemptive Algorithms

- Scheduling algorithms can be preemptive or nonpreemptive. All multitasking systems would employ preemptive scheduling as otherwise, one process might never give up control of the CPU.
  调度算法可以是抢占式的或非抢占式的。所有多任务系统都会采用抢占式调度，否则，一个进程可能永远不会放弃对 CPU 的控制。

Analysis of Non-preemptive Algorithms (P1 = 24, P2 = 3, P3=3)

FCFS (First Come First Served)

- FCFS is an inherently fair algorithm but performs badly for interactive systems where the response time is important and in situations where the job length differs greatly.
  FCFS 本质上是一种公平的算法，但在响应时间重要的交互式系统中表现不佳，且在作业长度差异较大的情况下表现不佳。
- 使用 FCFC 算法，我们得到以下顺序：P1 P2 P3
- FCFS 得到以下性能结果：

Job	Waiting Time	Response Time	Turnaround Time

Job	Waiting Time	Response Time	Turnaround Time
1	0	0	24
2	24	24	27
3	27	27	30
Average	17	17	27

SJF (Shortest Job First)

- SJF is provably the optimal algorithm in terms of throughput, waiting time and response performance but is not fair. SJF favours short jobs over longer ones. The arrival of shorter jobs in the ready queue postpones the scheduling of the longer ones indefinitely even though they may be in the ready queue for quite some time. This is known as starvation.
  SJF 在吞吐量、等待时间和响应性能方面被证明是最佳算法，但不公平。SJF 优先考虑短作业而非长作业。较短作业的到来会无限期地推迟长作业的调度，即使它们可能在就绪队列中等待了相当长的时间，这被称为饿死。
- 使用 SJF 算法，我们得到以下顺序: P2 P3 P1
- SJF 得到以下性能结果：

Job	Waiting Time	Response Time	Turnaround Time
1	6	6	30
2	0	0	3
3	3	3	6
Average	3	3	13

- 在实践中，不可能事先知道作业的 CPU 爆发长度，但可以根据其历史性能使用平滑公式（一种启发式猜测）进行估计。设 tn = 第 n 次爆发的长度，Tn = 前 n 次爆发的平均值。可以根据公式估计进程的下一次爆发：

$$t_{n+1} = at_n + (1 - a)T_n$$

其中 0 <= a <= 1，a 是基于最近活动和过去活动选择的加权值。

HRN（highest response ratio next)

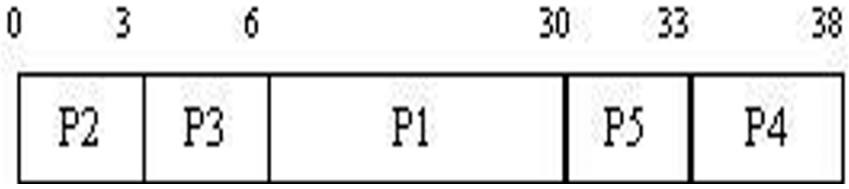
- The Response Ratio of a job is calculated as follows: Response Ratio（R） = (Waiting Time) / (Service Time)
  作业的响应比计算如下：响应比 =（等待时间）/（服务时间）
- 对于下面这个例子来说：

Arrival	Job	CPU Burst
0	P1	24
0	P2	3
0	P3	3
5	P4	5
10	P5	3

将每一个时间点的Response Ratio算出来比较，选择最小的

T	Job	R		T	Job	R		T	Job	R		T	Job	R
0	P1	0/24		3	P1	3/24		**6**	**P1**	**6/24**		30	P4	5/25
0	**P2**	**0/3**												
0	P3	0/3		**3**	**P3**	**3/3**		6	P4	1/5		**30**	**P5**	**20/3**

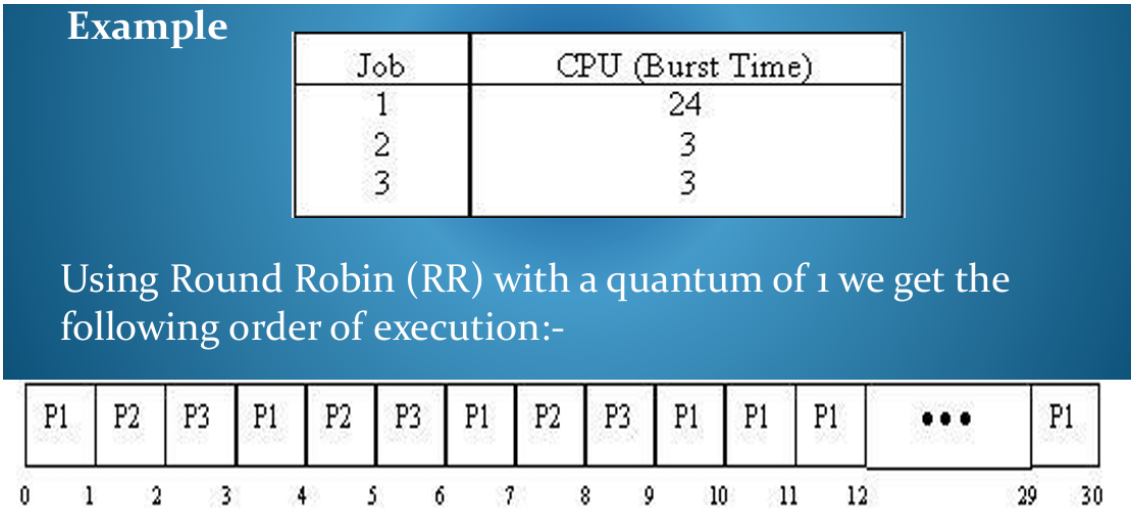
可以画出它的甘特图



Preemptive Scheduling using RR

Definition

- A common type of algorithm used as the basis for scheduling in multitasking systems is known as Round Robin.
  在多任务系统中，作为调度基础的常见算法称为轮转法。
- Round Robin is chosen because it offers good response time to all processes.
  选择轮转法是因为它为所有进程提供了良好的响应时间



- A process is initially submitted to the top level highest priority queue where it benefits from good response time. If its behaviour over time shows that it is computationally demanding then it can be demoted to a lower priority queue
  进程最初被提交到最高优先级的顶级队列，在那里它从良好的响应时间中受益。如果随着时间的推移，其行为表明它计算强度较高，那么它可以被降级到较低优先级的队列

*L5 Process Scheduling III

Scheduling on Multiprocessor Systems

Performance Issues with Multiprocessing

- doubling processors does not usually double performance.
  处理器数量翻倍通常不会使性能翻倍。
- Contention for the bus, I/O, shared memory, operating system code and maintaining cache coherency can cause some performance lag.
  争夺总线、I/O、共享内存、操作系统代码以及维持缓存一致性可能会导致一些性能滞后

Queue Organisation for Multiprocessor Systems

Modelling the Arrival and Queuing Process

- 例如，**泊松分布**可以用于我们描述的计算机系统的到达过程，以找出在给定时间段内特定数量任务到达的概率。

$$P(keventsininterval) = \frac{\lambda^k e^{-\lambda}}{k!}$$

其中λ是每个间隔的平均事件数，在这段时间内发生k次

Queuing Theory Application

- 如果系统的平均任务到达速率是 λ，服务速率是 μ，那么可以根据 Little 定律证明平均停留时间或周转时间（在系统中花费的时间）T 与 λ 和 μ 之间的关系由以下公式给出：

$$T = \frac{1}{\mu - \lambda}$$

- 当然，μ 必须大于 λ，系统才能应对负载。

Other Scheduling Environments

Distributed Systems

- A Distributed System is composed of a collection of physically distributed computer systems connected by a local area network.
  分布式系统由通过局域网连接的多个物理分布的计算机系统组成。
- From a scheduling perspective, a distributed operating system would endeavour to monitor and share or balance the load on each processing node.
  从调度的角度来看，分布式操作系统将努力监控并共享或平衡每个处理节点上的负载。

Real Time Systems

- Hard Real Time Systems are required to complete a critical task within a guaranteed amount of time.
  硬实时系统需要在保证的时间内完成关键任务
- Soft Real Time Systems are ones which endeavour to meet scheduling deadlines but where missing an occasional deadline may be tolerable.
  软实时系统是那些努力满足调度截止时间但偶尔错过截止时间可能是可以接受的系统。
  - Earliest Deadline First: When an event is detected, the handling process is added to the ready queue, serving the most urgent tasks first.
    最早截止时间优先: 当检测到事件时，处理进程被添加到就绪队列,首先服务最紧急的任务。
  - Least Laxity Algorithm: If a process requires 200msec and must finish in 250msec, then its laxity is 50msec. The Least Laxity Algorithm chooses the process with the smallest amount of time to spare.
    最小松弛时间算法 如果一个进程需要 200 毫秒并且必须在 250 毫秒内完成，那么它的松弛时间是 50 毫秒。最小松弛时间算法选择剩余时间最少的进程。

L6 Disk Operation and Scheduling

Magnetic Hard Disk Operation and Scheduling

Engineering Developments

- Smaller platter diameters offer better rigidity, weigh less and require less power to spin,, less noise and heat and improved seek performance.
  更小的盘片直径提供更好的刚度，重量更轻，旋转所需的功率更小，噪音和热量更少，寻道性能得到改善。
- Faster Signaling Interfaces improving transfer speed.
  更快的信号接口提高传输速度。

Data Organisation

- The surface of each platter is organised as a concentric group of magnetic tracks, Each track is divided into a number of blocks of fixed size called sectors
  每个盘片的表面被组织成一组同心的磁道，每个磁道被分成若干个固定大小的块，称为扇区
- One side of one platter contains space reserved for hardware track-positioning information and is not available to the operating system.
  一个盘片的一侧包含为硬件磁道定位信息保留的空间，这些空间不可供操作系统使用。
- A disk assembly containing two platters has three sides available for data. The system disk controller reads this data to settle the drive heads in the correct track position.
  包含两个盘片的磁盘组件有三个侧面可用于数据存储。系统磁盘控制器读取此数据以将驱动器磁头定位到正确的磁道位置。

Seek Time and Rotational Latency

- The slowest part of accessing a disk block is physically moving the head to the correct track.
  访问磁盘块的最慢部分是物理上将磁头移动到正确的磁道。
- In a multitasking system, requests to access different parts of the disk arrive from different processes or a file's blocks may be scattered across a platter.
  在多任务系统中，来自不同进程的请求访问磁盘的不同部分，或者文件的块可能分散在盘片上。
- Correct scheduling of these requests can reduce the average seek time by reducing the average distance the head has to travel.
  正确调度这些请求可以通过减少磁头必须移动的平均距离来减少平均寻道时间。

Zoned Bit Recording

- The number of sectors stored on inner tracks is constrained by the **bit density of the magnetic surface.**
  内磁道上存储的扇区数量受到磁表面位密度的限制。
- Tracks are grouped based on their distance from the centre with the outer zones being used to store more sectors than the inner ones.
  根据磁道与中心的距离将磁道分组，外区用于存储比内区更多的扇区。
- This also means that if the rotational velocity is constant, that data can be transferred faster from the outer tracks than the inner tracks.
  这也意味着，如果旋转速度恒定，数据可以从外磁道比内磁道更快地传输。
- As the drive is filled from the outside in, it gives its best performance **when new and empty.**
  由于驱动器从外向内填充，当它全新且为空时性能最佳。

Solid State Drives

- Solid State Drives have energy and performance benefits over hard drives but are more expensive and generally have smaller capacity.
  固态硬盘在能源和性能方面优于机械硬盘，但价格更高，通常容量更小。
- Operating system may boot faster from SSD & file opening speeds maybe 20-40x faster than a standard HDD.
  操作系统从SSD启动可能更快，文件打开速度可能比标准HDD快20-40倍。

Hard Disk Drive Performance Metrics

- **Capacity** – Highly reliable high capacity cheap solution
  容量——高可靠性、大容量、低成本的解决方案
- **Data Rate** – The bandwidth between a drive and the system, connected by a particular interface e.g. SATA 1.5Gb/s, SATA-2 3Gb/s, SATA-3 6Gb/s, SATA 3.2 16Gb/s
  数据速率——驱动器与系统之间的带宽，通过特定接口连接，例如SATA 1.5Gb/s、SATA-2 3Gb/s、SATA-3 6Gb/s、SATA 3.2 16Gb/s
- **Average Latency** – Time to access a sector e.g. 4ms-9ms
  平均延迟——访问扇区的时间，例如4毫秒-9毫秒
- **Reliability** – Mean Time to Failure
  可靠性——平均故障间隔时间
- **Scheduling Algorithms** - Seek Time
  调度算法——寻道时间

L7: Process Creation and Interprocess Communication

Unix Process Creation

Process Characteristics

- Every process has a unique ID.
  每个进程都有一个唯一的 ID。
- The initial process is the Parent and the new process is the Child.
  初始进程是父进程，新进程是子进程。
- The child process is an exact copy of the parent, including a copy of the parent's I/O descriptor table. It inherits access to all the parent's open I/O devices.
  子进程是父进程的精确副本，包括父进程的 I/O 描述符表的副本。它继承了对父进程所有打开的 I/O 设备的访问权限。

Booting a Linux System

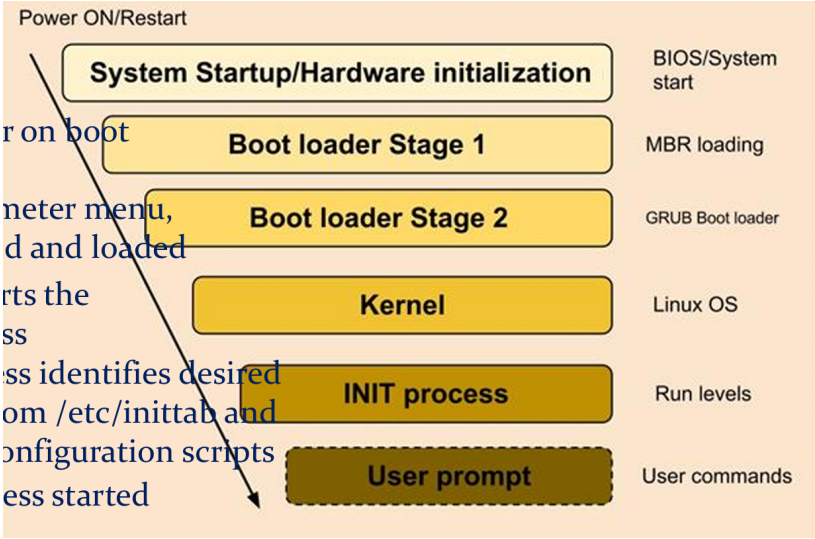
Master Boot Record

- After power-on self-test and hardware identification by firmware routines, the first boot device is selected and the Master Boot Record is read from that device.
  在固件例程进行开机自检和硬件识别后，选择第一个启动设备，并从该设备读取主引导记录（MBR）。
- The MBR contains initial bootstrapping code and information about the active partition.
  MBR 包含初始引导代码和有关活动分区的信息。

Hard Disk Partitioning

- The MBR contains initial bootstrapping code called by the BIOS and partition information about the primary boot device and the active partition.
  MBR 包含 BIOS 调用的初始引导代码和有关主启动设备和活动分区的分区信息。
- Additional bootstrapping code and device drivers may be stored in unused sectors, unaffected by filesystem formatting.
  额外的引导代码和设备驱动程序可能存储在未使用的扇区中，不受文件系统格式化的影响。

Creating the First Process



- Linux Run Levels: Init is the initial start up process. It creates processes based on the configuration in /etc/inittab depending on the selected runlevel.
  Linux 运行级别：Init 是初始启动进程。根据选定的运行级别，它基于 /etc/inittab 中的配置创建进程。

Run Level	Name	Description
0	*Halt*	Shuts down all services when the system will not be rebooted.
1	*Single User*	Used for system maintenance. No Networking capabilities.
2	*MultiUser* *No Network Support*	Used for maintenance and system testing.
3	*MultiUser* *Network Support*	Non-Graphical Text Mode operations for server systems.
4	-	Custom Mode, used by SysAdmin
5	*Graphical* *X11*	Graphical login with same usability of Run Level 3.
6	*Reboot*	Shuts down all services when the system is being rebooted.

Interprocess Communication

Design Considerations

- Interprocess communication refers to the exchange or sharing of information between separate independent schedulable tasks.
  进程间通信指的是在独立的可调度任务之间交换或共享信息。
- Reasons for IPC: 进程间通信的原因：
  - Data Parallelization for Computational Speedup, 计算加速的数据并行化
  - Modular Division of functions, 功能的模块化划分
  - Client use of a service on a network. 网络上客户端对服务的使用

Implementation Approaches

- Two basic ways to implement IPC: Shared Memory and Message Passing.
  实现进程间通信的两种基本方法：共享内存和消息传递。
  - Shared Memory Communication
    - Processes use the operating system to establish a shared region of memory.
      进程使用操作系统建立共享内存区域。
    - Communication requires processes to be on the same host.
      通信要求进程在同一主机上。
    - Implicit communication through read/write operations.
      通过读写操作进行隐式通信。
    - Highly efficient but requires synchronization mechanisms.
      高效但需要同步机制。
  - Message Passing Communication
    - Message passing facilities provided by the operating system.
      操作系统提供的消息传递设施。
    - Processes explicitly invoke send and receive primitives to exchange messages.
      进程显式调用发送和接收原语来交换消息。
    - Higher-level abstractions like remote procedure call.
      更高级的抽象，如远程过程调用。

L8: Unix Pipes and Sockets

Unix Interprocess Communication Mechanisms

Pipes

- Definition: Processes that are related by creation hierarchy and run on the same machine can use a fast kernel-based stream communication mechanism known as a pipe.
  定义：通过创建层次关系并在同一台机器上运行的进程可以使用一种快速的基于内核的流通信机制，称为管道。
- Characteristics: Pipes are FIFO byte stream communication channels implemented with finite size memory buffers maintained by the kernel.
  特点：管道是先进先出的字节流通信通道，由内核维护固定大小的内存缓冲区实现。
- Usage: A process writes data to one end of the pipe, and usually another process reads it from the other end. A pipe exists for the lifetime of the processes that have access to it.

使用：一个进程将数据写入管道的一端，通常另一个进程从另一端读取数据。管道在能够访问它的进程的生命周期内存在。

- Scope: A pipe can only be used between related processes and is generally used as a unidirectional communication channel from parent to child or vice versa. The parent would create the pipe before creating the child so that the child can inherit access to it.
范围：管道只能在相关进程之间使用，通常用作从父进程到子进程或反之的单向通信通道。父进程会在创建子进程之前创建管道，以便子进程可以继承对管道的访问权限。

Named Pipes

- Definition: A variation of this mechanism known as a named pipe can be used for communication between unrelated processes that have access to the same file name space.
定义：一种称为命名管道的变体机制，可用于在具有相同文件名空间访问权限的不相关进程之间进行通信。
- Implementation: A named pipe is implemented as a special FIFO file by the kernel, rather than as a memory buffer, and so is accessible to independent processes through the file system's shared name space.
实现：命名管道由内核实现为特殊的 FIFO 文件，而不是内存缓冲区，因此可通过文件系统的共享名称空间被独立进程访问。
- Sharing: Named pipes can be shared across machines with a common file system.
共享：命名管道可以在具有公共文件系统的机器之间共享。

Network Sockets

- Definition: Sockets are the more usual and general-purpose means of communication between independent processes and can be used when neither kernel data structures nor files can be shared, for example, communicating across the Internet.
定义：套接字是独立进程之间更常用和通用的通信方式，当既不能共享内核数据结构也不能共享文件时可以使用，例如通过互联网进行通信。
- Characteristics: A communication channel can be visualized as a pair of communication endpoints.
特点：可以将通信通道视为一对通信端点。
- Implementation: A socket is a data structure created by an application to represent a communication channel. The socket must be bound or associated to some entity (service access point/port) within the communication system. By using communication primitives of the socket interface, the process then exchanges this data from its address space to the address space of the communication subsystem which handles delivery.
实现：套接字是由应用程序创建的用于表示通信通道的数据结构。套接字必须绑定或关联到通信系统内的某个实体（服务访问点/端口）。通过使用套接字接口的通信原语，进程将其地址空间中的数据交换到处理传输的通信子系统的地址空间中。

Socket Communication

- Service: The communication service (e.g., TCP/IP) provides the glue for establishing connections between sockets belonging to different processes, which are often on different machines and networks, and for transporting and routing messages through the network to a destination. On the Internet, IP addresses and port numbers are used as a means of

identifying the endpoints of a connection to which a program's socket might be connected.
服务：通信服务（例如 TCP/IP）提供了在不同进程的套接字之间建立连接的纽带，这些进程通常位于不同的机器和网络上，并通过网络传输和路由消息到目的地。在互联网上，IP 地址和端口号被用作标识程序套接字可能连接的连接端点的手段。

- Protocols: A communication subsystem which understands the same messaging protocols must be running on every point along the network over which the message travels. In an Internet context, this subsystem implements a communication protocol known as TCP/IP.
协议：在网络上的每个节点上都必须运行理解相同消息协议的通信子系统。在互联网环境中，此子系统实现了称为 TCP/IP 的通信协议。
- Reliability: Different levels of reliability for message delivery through the socket may be specified to the transport control protocol (TCP) when it is created. Faster delivery can be traded for less reliability.
可靠性：在创建传输控制协议（TCP）时，可以指定通过套接字传输消息的不同可靠性级别。可以权衡更快的传输速度与较低的可靠性。
- Connection: A socket may be 'connected' to any named remote socket by the underlying communication protocol. After connection, bytes which are sent to the local socket are delivered to the remote socket. The remote process 'listens' to its socket and receives the data sent into the byte stream.
连接：底层通信协议可以将套接字连接到任何命名的远程套接字。连接后，发送到本地套接字的字节将被传输到远程套接字。远程进程监听其套接字并接收发送到字节流中的数据。

Socket-based Client/Server Communication

- Server Socket: Marks the socket as a passive socket, a socket that will be used to accept incoming connection requests using accept.
服务器套接字：将套接字标记为被动套接字，该套接字将用于使用 accept 接受传入的连接请求。
- Binding: Associate socket with named IP port.
绑定：将套接字与命名的 IP 端口关联。

L9: Unix Message Queues and Remote Procedure Call

Interprocess Communication Mechanisms

Remote Procedure Call (RPC)

- Definition: RPC is a more convenient communication abstraction built on top of an underlying endpoint connection mechanism like sockets over TCP/IP which is intended to make client-server programming easier. It enables you to call a remote function in a similar manner to a local one.
定义：RPC 是一种更方便的通信抽象，建立在底层端点连接机制（如 TCP/IP 上的套接字）之上，旨在简化客户端-服务器编程。它使你能够以类似于调用本地函数的方式调用远程函数。
- Interface: An RPC service will define an interface enabling communications stubs to be generated automatically from that interface.
接口：RPC 服务将定义一个接口，使得可以从该接口自动生成通信存根。
- Code Complexity: The implementation of the RPC mechanism is hidden within the communication stub code. The stubs are then linked statically with the client and server

code.
代码复杂性：RPC 机制的实现隐藏在通信存根代码中。存根然后与客户端和服务器代码静态链接。

- Working Process: A client process calls the desired procedure and supplies arguments to it. The client RPC stub mechanism packs the arguments into a message and sends it to the appropriate server process. The receiving stub decodes it, identifies the procedure, extracts the arguments and calls the procedure locally. The results are then packed into a message and sent back in a reply which is passed back by the stub to the client as a procedure return.
  工作过程：客户端进程调用所需的程序并提供参数。客户端 RPC 存根机制将参数打包到消息中并将其发送到适当的服务器进程。接收存根对其进行解码，识别程序，提取参数并本地调用程序。然后将结果打包到消息中，并作为回复发送回，存根将其作为过程返回传递回客户端。
- Server Location: Although RPC stub mechanisms understand the rules of how to communicate with a service interface, they may not know where the particular service exists on the network. The server's socket is located by first communicating with a directory service with which the server has registered and then establishing a connection over which RPC style exchanges can take place.
  服务器定位：尽管 RPC 存根机制了解如何与服务接口通信的规则，但它们可能不知道特定服务在网络上的位置。通过首先与服务器已注册的目录服务通信，然后建立连接以进行 RPC 风格的交换，可以找到服务器的套接字。
- Challenges: Some implementation problems make it difficult to make the remote procedure call mechanism completely transparent. Arguments are passed by value, as in the remote address space, pointer references cannot be evaluated correctly. This makes it difficult to pass memory pointer-based data structures or object references for example.
  挑战：一些实现问题使远程过程调用机制难以完全透明。参数按值传递，在远程地址空间中，指针引用无法正确评估。例如，这使得传递基于内存指针的数据结构或对象引用变得困难。
- Data Format Translation: Another difficulty may result if the server is running on a different machine architecture to the client or was written using a different programming language. It will be necessary for the RPC mechanism to find an agreeable external data representation format for representing the information exchanged so that it is interpreted correctly by both parties.
  数据格式转换：如果服务器在与客户端不同的机器架构上运行，或者使用不同的编程语言编写，则可能会导致另一个困难。RPC 机制需要找到一个双方都认可的外部数据表示格式，以便正确解释交换的信息。
- Failure Semantics: Different fault handling execution semantics may apply...
  故障语义：可能适用不同的故障处理执行语义...

Unix Interprocess Communication Mechanisms

Message Queues

- Definition: Message queues allow processes to exchange data in the form of whole messages asynchronously. No rendezvous required. Messages have an associated priority and are queued in priority order.
  定义：消息队列允许进程以完整消息的形式异步交换数据。不需要会合。消息具有相关优先级，并按优先级顺序排队。
- Message Queue Manager: The operating system maintains the message queue persistently and independently of user processes until it is explicitly unlinked (destroyed) by a user process or the system is shutdown. Communicating processes must have access permission to a particular message queue and the message queue API.

消息队列管理器：操作系统持续维护消息队列，并独立于用户进程，直到用户进程显式解除链接（销毁）或系统关闭。通信进程必须对特定消息队列和消息队列 API 具有访问权限。

- Comparison with Pipes: Pipes have no internal structure, just a byte stream and cannot distinguish between content of different writers. Message queues have internal structure and geometry set by user, separate messages are distinguishable, and queue is sorted on message priority.
  与管道比较：管道没有内部结构，只是一个字节流，无法区分不同写入者的内容。消息队列具有用户设置的内部结构和几何形状，可以区分单独的消息，并按消息优先级排序队列。

Summary of Unix Interprocess Communication Mechanisms

- Pipes: Fast memory based, stream oriented, between related processes.
  管道：基于快速内存，流导向，在相关进程之间。
- Named Pipes: Like pipes but uses shared files, usable by unrelated processes.
  命名管道：像管道但使用共享文件，不相关进程可以使用。
- Sockets: Low-level internet communication abstraction generally over TCP/IP between unrelated processes.
  套接字：不相关进程之间的低级互联网通信抽象，通常基于 TCP/IP。
- Remote Procedure Call: Simplifies client/server programming.
  远程过程调用：简化客户端/服务器编程。
- Message Queues: Asynchronous whole message communication with priority ordering.
  消息队列：具有优先级排序的异步完整消息通信。