进程调度-介绍

邰颖
南京大学
智能科学与技术学院

什么构成操作系统？内核 + 其他组件

- 一个**特权/内核进程**在内核模式下运行（由硬件控制）
- 在启动时由引导加载程序（GRUB或自定义）加载
- 一直运行直到设备关闭
- 使得**用户进程**能够以**受限的直接执行**方式通过**上下文切换**在CPU上运行
- 通过**系统调用**（通过**陷阱指令**调用）代表用户进程执行特权操作

操作系统信息

```
[direct-execution]$ neofetch
                    .-/+oossssoo+/-.               ty@os-linux
                `:+ssssssssssssssssss+:`           -----------
            -+ssssssssssssssssssyyssss+-           OS: Ubuntu 24.10 aarch64
          .ossssssssssssssssssdMMMNysssso.         Host: QEMU Virtual Machine virt-9.1
         /ssssssssssshdmmNNmmyNMMMMhssssss/        Kernel: 6.11.0-14-generic
        +ssssssssshmydMMMMMMMNddddyssssssss+       Uptime: 3 hours, 36 mins
       /sssssssshNMMMyhhyyyyhmNMMMNhssssssss/      Packages: 829 (dpkg)
      .ssssssssdMMMNhsssssssssshNMMMdssssssss.     Shell: bash 5.2.32
      +ssssshhhyNMMNyssssssssssssyNMMMysssssss+    Resolution: 1280x800
      ossyNMMMNyMMhsssssssssssssshmmmhssssssso     Terminal: tmux
      ossyNMMMNyMMhsssssssssssssshmmmhssssssso     CPU: (2)
      +ssssshhhyNMMNyssssssssssssyNMMMysssssss+    GPU: 00:02.0 Red Hat, Inc. Virtio 1.
      .ssssssssdMMMNhsssssssssshNMMMdssssssss.     Memory: 203MiB / 3389MiB
       /sssssssshNMMMyhhyyyyhdNMMMNhssssssss/
        +ssssssssdmydMMMMMMMNddddyssssssss+
         /ssssssssssshdmNNNNmyNMMMMhssssss/
          .ossssssssssssssssssdMMMNysssso.
            -+sssssssssssssssssyyyssss+-
              `:+ssssssssssssssssss+:`
                  .-/+oossssoo+/-.
```

```
[direct-execution]$ uname -v -r
6.11.0-14-generic #15-Ubuntu SMP PREEMPT_DYNAMIC
[direct-execution]$ lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 24.10
Release:        24.10
Codename:       oracular
```

制定调度策略

- 我们如何选择哪些用户进程在CPU上运行？
- 关键假设是什么？
  - 进程有不同的需求：计算密集，I/O密集
  - CPU资源有限
  - 优先级概念
- 关于工作负载等。哪些指标是重要的？
  - 吞吐量
  - 响应时间
  - 公平性
  - CPU利用率
- 一些历史上的经典方法有哪些？

https://www.jollibee.com.ph/

当前的工作负载假设（简化版，不现实）

- 假设1: 每个进程运行相同的时间
- 假设2: 所有进程同时到达
- 假设3: 一旦启动，每个进程会运行直到完成
- 假设4: 所有进程仅使用CPU（即它们不执行I/O操作）
- 假设5: 每个进程的运行时间是已知的

我们将在后续过程中逐渐放宽这些假设。

调度指标（用于比较策略）

- **周转时间** – 性能指标
  - 周转时间是指：**进程完成的时间减去进程到达系统的时间**
  - 假设所有进程同时到达，方便计算，到达时间设为 $T_{arrival}=0$

$$T_{turnaround}=T_{completion}-T_{arrival}$$

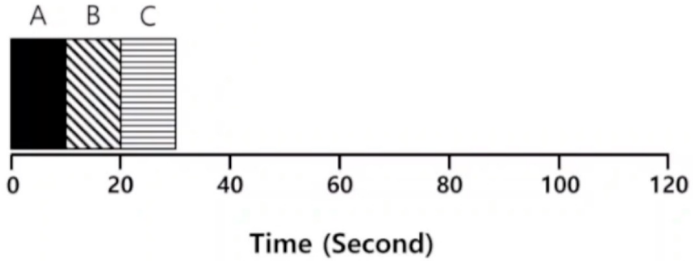
  - 我们通常关注的是给定一组进程的**平均周转时间（Average Turnaround Time, ATT）**

- **公平性**
  - 每个进程都有“公平的机会”在CPU上运行
  - 性能和公平性在调度中往往是相互矛盾的

策略 #1: First come first served (FCFS)

- 先到先服务 (也称为First in first out, FIFO)
  - 非常简单且易于实现

- 示例：
  - A在B之前到达，B在C之前到达。
  - 每个任务运行10秒。



$$Average\ turnaround\ time = \frac{10 + 20 + 30}{3} = 20\ sec$$

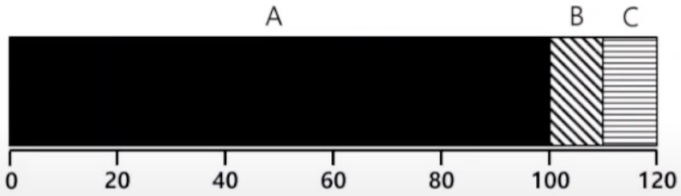
图示显示了A、B、C的执行顺序，A先执行10秒，然后是B和C

南京大学 NANJING UNIVERSITY

策略 #1: First come first served (FCFS)

- 放宽假设1：每个进程现在不再运行相同的时间。

- **FIFO存在护航效应（Convoy Effect）：**
  - 短时间运行的进程必须等待长时间运行的进程，严重影响了**平均周转时间（ATT）**
- **示例：**
  - A在B之前到达，B在C之前到达。
  - A运行100秒，B和C分别运行10秒。



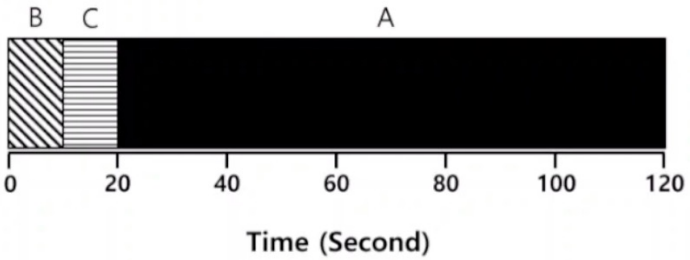
$$Average\ turnaround\ time = \frac{100 + 110 + 120}{3} = 110\ sec$$

图示显示A、B和C的执行顺序，A先执行100秒，之后是B和C各自运行10秒

策略 #2: 最短作业优先 (Shortest job first, SJF)

- **先执行最短的进程，然后是下一个最短的，以此类推**
  - 这是一种**非抢占式调度**方法——即进程执行完毕才会中断（与之相对的是**抢占式调度**，现代系统通常采用这种方式）。

- **示例1：**
  - A在B之前到达，B在C之前到达。
  - A运行100秒，B和C分别运行10秒。

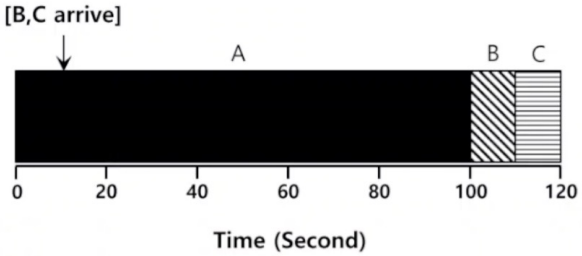


$$Average\ turnaround\ time = \frac{10 + 20 + 120}{3} = 50\ sec$$

图示显示了A、B、C的执行顺序，B和C先执行各自10秒，之后是A执行100秒

策略 #2: 最短作业优先 (Shortest job first, SJF)

- 我们放宽了假设2：进程现在可以在任何时间到达。
- 由于非抢占式调度，导致了“车队效应”。
- 示例（带有延迟到达的SJF）：
  - A在 t=0 到达，并需要运行100秒。
  - B和C在 t=10 到达，每个需要运行10秒。



$$Average\ turnaround\ time = \frac{100 + (110 - 10) + (120 - 10)}{3} = 103.33\ sec$$

图中显示了任务A从0秒开始运行，任务B和C在10秒时到达并开始执行，任务A在执行完成后才开始B和C的处理。

策略 #3:最短完成时间优先（STCF）

- 放宽假设3：进程现在不需要运行到完成。
  - 为SJF增加了**抢占**机制。
  - 也称为**抢占式最短作业优先**（PSJF）。

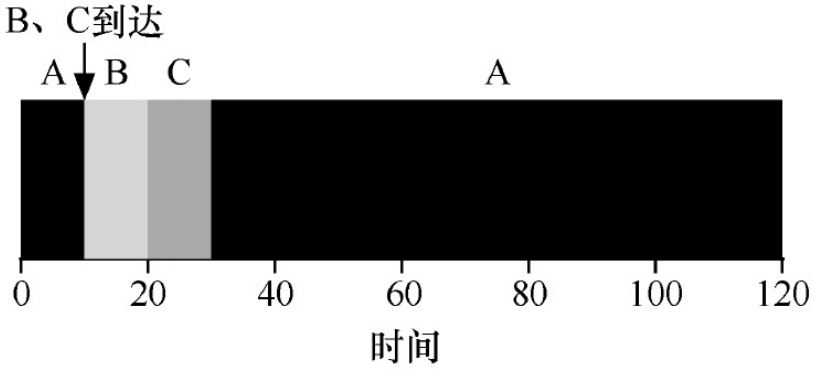
- 一个新的进程到达系统时：
  - 确定现有进程和新进程的**剩余时间**。
  - 调度剩余时间最短的进程。

- 注意：决策点发生在**新进程到达**或**时钟中断**时。

策略 #3:最短完成时间优先（STCF）

•示例：

- A在 t=0 到达，并需要运行100秒。
- B和C在 t=10 到达，每个需要运行10秒。



$$Average\ turnaround\ time = \frac{(120-0)+(20-10)+(30-10)}{3} = 50\ sec$$

图中显示了任务A、B和C的执行情况。任务A在0秒到达并开始运行。

但由于STCF的抢占性，当B和C到达时，它们优先执行（因为它们的剩余时间更短）。

在B和C运行完之后，A继续执行。

一种新的调度度量

- **响应时间**——引入到**分时系统**和**交互式系统**中
  - 之前描述的策略主要适用于**批处理系统**
- 从**进程到达**的时间到**首次调度**（或产生首次输出）的时间

$$T_{response} = T_{firstrun} - T_{arrival}$$

- 我们再次关注的是给定一组进程时的**平均响应时间（ART）**
- STCF及相关策略关注周转时间，对于响应时间不太有效。

如何构建一个对响应时间敏感的调度器？

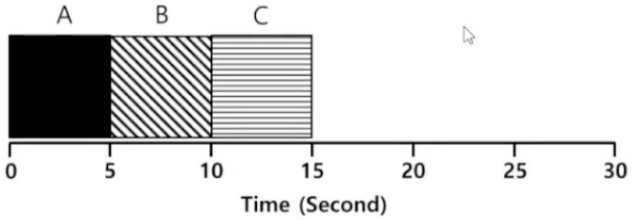
轮转（Round Robin, RR）

- 也称为**时间片调度**（Time-Slicing Scheduling）
  - 运行一个进程一段时间片（有时称为调度量子，记作 q），然后切换到运行队列中的下一个进程。
  - 反复执行，直到所有任务完成。
  - 时间片必须是时钟中断周期的**倍数**。
    - 例如：如果时钟每10毫秒中断一次，那么时间片必须是10毫秒、20毫秒等。

轮转法（RR）是公平的，但在周转时间等度量上表现较差。

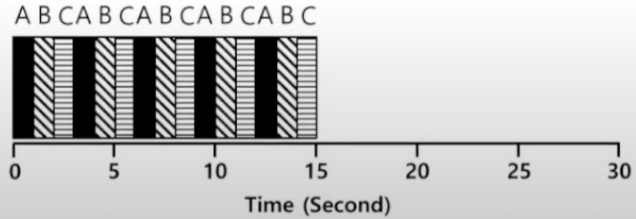
轮转法（Round Robin, RR）

- 例子：
  - A、B 和 C 同时到达
  - 它们每个希望运行 5 秒钟



SJF (Bad for Response Time)

$$T_{average\ response} = \frac{0+5+10}{3} = 5sec$$



RR with a time slice of 1sec (Good for Response Time)

$$T_{average\ response} = \frac{0+1+2}{3} = 1sec$$

轮转法（Round Robin, RR）

- 时间片的长度至关重要

- **较短的时间片**
  - 更好的响应时间
  - 上下文切换的成本将主导整体性能

- **较长的时间片**
  - 会加剧响应时间
  - 平摊上下文切换的成本——减少上下文切换

- **注意：** 上下文切换的成本不仅仅来自于**寄存器的复制**，还包括**其他架构方面**，如缓存、转换后备页表（TLB）等。

决定时间片的长度对系统设计者来说是一种权衡。
那么在轮转法中，ATT（平均周转时间）怎么样？

融合I/O

- 放宽假设4：现在所有进程都可以执行I/O操作

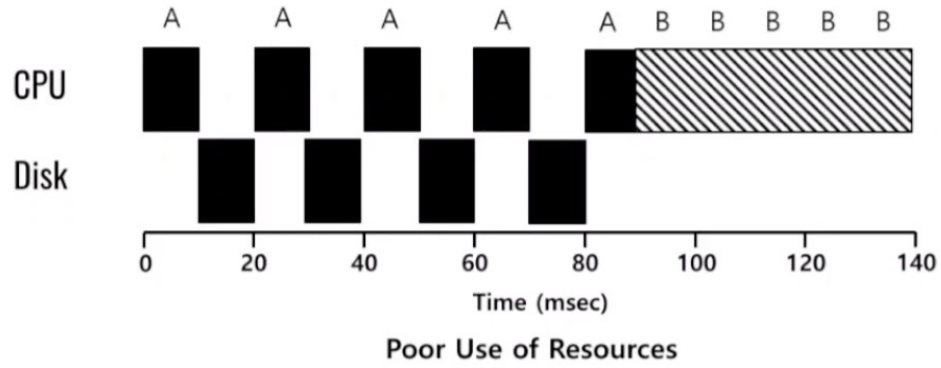
- **当一个进程发起I/O请求时：**
  - 进程状态被设置为**阻塞/等待**（进程在等待I/O完成）
  - 调度程序应当在CPU上调度另一个进程，否则CPU将处于空闲状态

- **当I/O完成时：**
  - 产生一个中断，控制权转交给内核
  - 内核将请求I/O的进程的状态从**等待**改为**就绪**状态
  - 调度程序**可以**将该进程调度到CPU上

融合I/O

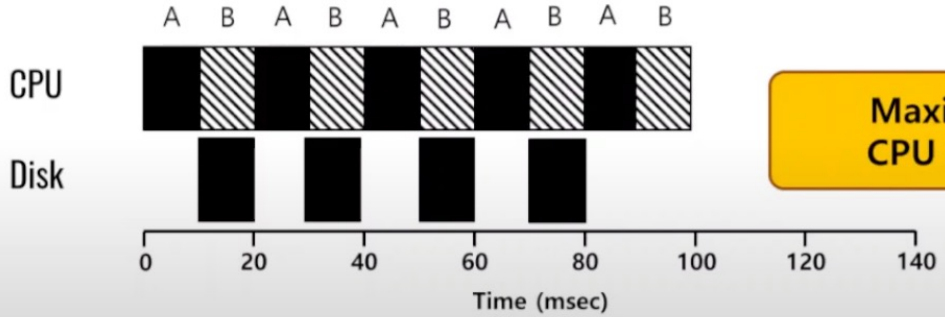
- 例子：
  - A 和 B 每个都需要 50 毫秒的 CPU 时间
  - A 运行 10 毫秒后发出 I/O 请求
    - 每个 I/O 操作需要 10 毫秒
  - B 简单地使用 CPU 50 毫秒，不执行 I/O 操作
  - 调度程序先运行 A，然后运行 B

融合I/O



Poor Use of Resources

资源使用不充分的情况



Maximizes the CPU utilization

Overlap Allows Better Use of Resources

资源利用更优的情况

不再有预知功能

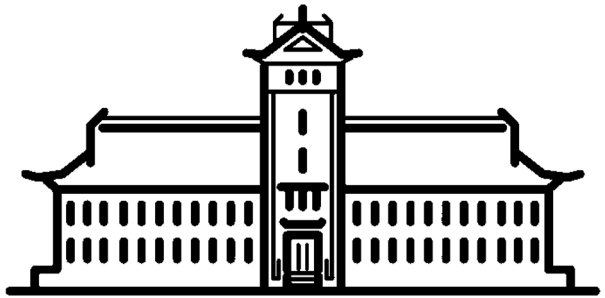
- 放宽假设 5：现在调度器**不知道**进程的运行时间

- 这种情况更加现实，但如何实现呢？——**多级反馈队列（MLFQ）**

小结

- 介绍了调度的基本策略
  - 先到先服务（FCFS）
  - 最短作业优先 (SJF)
  - 最短完成时间优先（STCF）
  - 轮转（RR）
  - 融合I/O

谢谢



南京大學
NANJING UNIVERSITY