多处理器调度

邰颖
南京大学
智能科学与技术学院

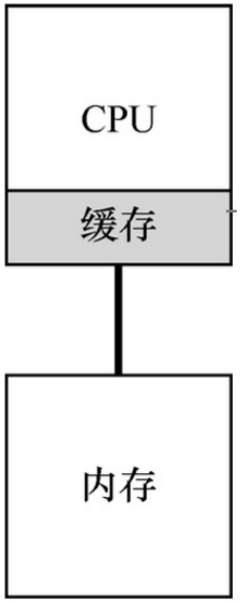
多处理器调度

- 多核处理器（multicore processor）的兴起推动了多处理器调度的普及。
  - **多核（Multicore）：** 多个CPU核心封装在同一个芯片上

- 增加更多的核心**并不会**让单个应用程序自动运行得更快。
  - 如果希望充分利用多核架构，需要重写应用程序，使其能够并行运行，即使用**线程（threads）**

如何在多个核心上调度任务？

单核CPU与缓存（Single CPU with Cache）

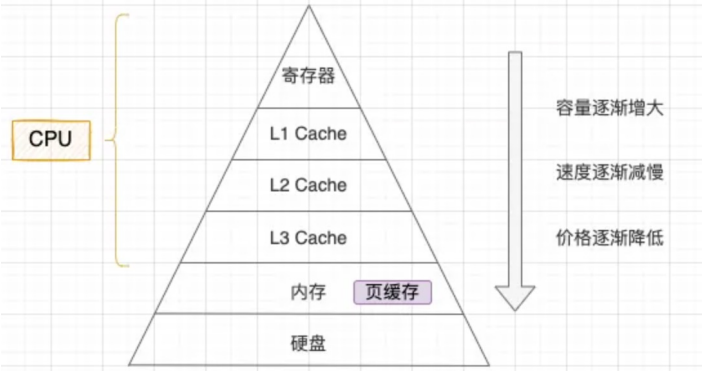


缓存（Cache）

- 小型且高速的存储器。
- 存储主存（内存）中**常用**数据的副本。
- 利用**时间局部性（Temporal Locality）**和**空间局部性（Spatial Locality）**提高访问效率。

主存（Main Memory）

- 存储所有数据。
- 访问主存的速度比缓存慢。



通过缓存数据，系统可以让对主存的慢速访问**看起来更快**

lscpu

- lscpu指令

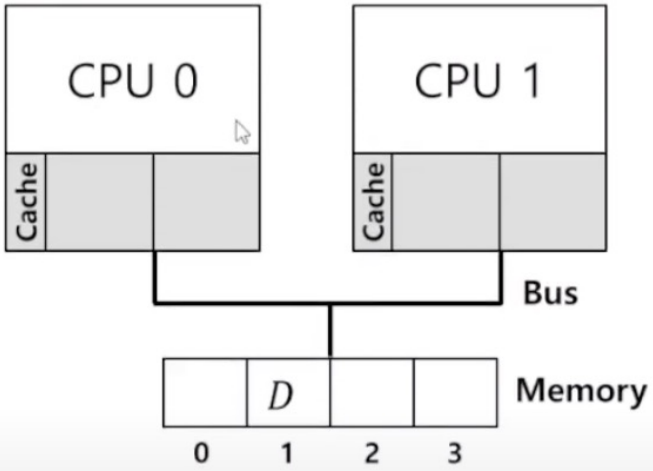
```
Architecture:                    aarch64
CPU op-mode(s):                  64-bit
Byte Order:                      Little Endian
CPU(s):                          2
On-line CPU(s) list:             0,1
Vendor ID:                       0x00
Model name:                      -
Model:                           0
Thread(s) per core:              1
Core(s) per socket:              2
Socket(s):                       1
Stepping:                        0x0
BogoMIPS:                        48.00
Flags:                           fp asimd evtstrm aes pmull sha1 sha2 crc32
 atomics fphp asimdhp cpuid asimdrdm jscvt fcma lrcpc dcpop sha3 asimddp sha512
asimdfhm dit uscat ilrcpc flagm ssbs sb paca pacg dcpodp flagm2 frint
NUMA node(s):                    1
NUMA node0 CPU(s):               0,1
Vulnerability Gather data sampling:   Not affected
Vulnerability Itlb multihit:          Not affected
Vulnerability L1tf:                   Not affected
Vulnerability Mds:                    Not affected
Vulnerability Meltdown:               Not affected
```

问题#1：缓存一致性（cache coherence）

- 多个缓存中存储的共享资源数据需要保持一致性

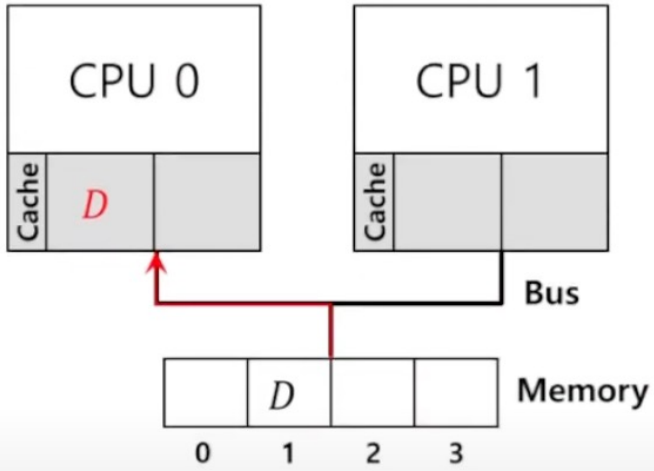
0. 两颗CPU共享同一片内存

•CPU0 和 CPU1 都有各自的缓存，并通过总线访问主存。



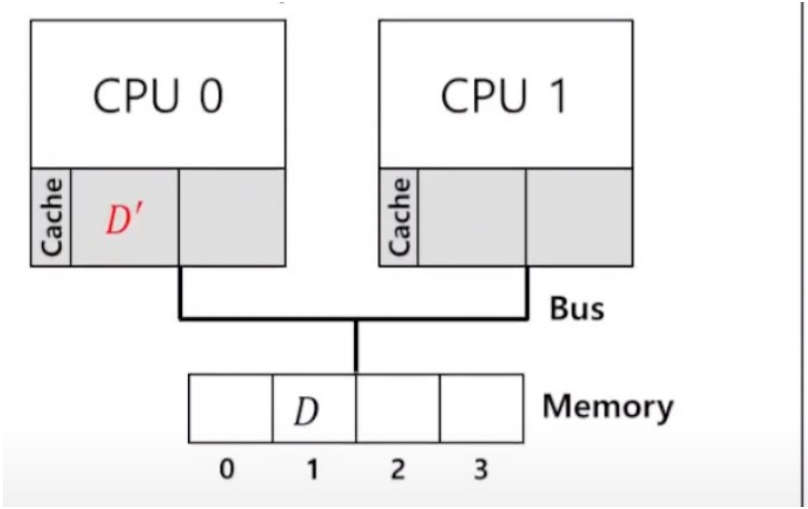
1. CPU0 读取地址 1 处的数据

•CPU0 从主存地址 1 读取数据 D，并将其存入自己的缓存。

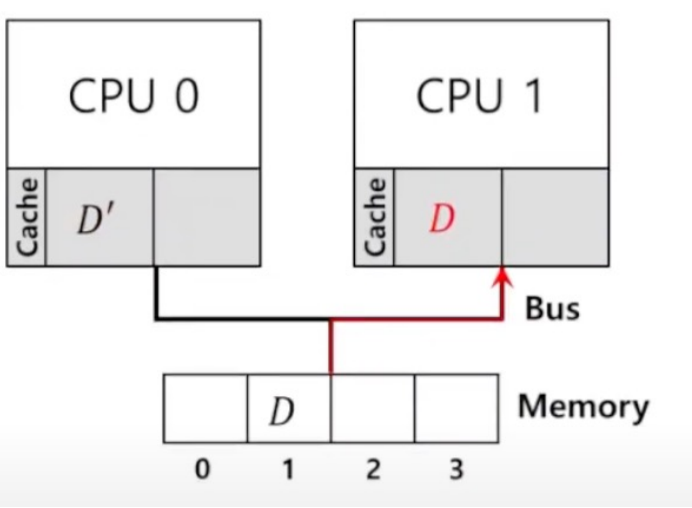


问题#1：缓存一致性（cache coherence）

- 数据更新与缓存不一致
  - CPU0 更新了内存地址1的值 **D → D'**（但仅存于CPU0的缓存）。
  - CPU1 被调度执行。



- **CPU1 重新读取地址1的数据**
  - CPU1 读取的仍然是**旧数据 D**，而不是正确的**D'**。



由于缓存一致性问题，CPU1 获取的是**错误的旧值 D，而不是更新后的正确值 D'**

问题#1：缓存一致性（cache coherence）

解决方案：总线嗅探（Bus Snooping）

- 每个缓存通过**监视总线（observing the bus）**来关注内存的更新情况。

- 当某个CPU发现**缓存中存储的数据项被更新**，它会检测到这一变化，并**使缓存副本失效（invalidate）或进行更新（update）**。

问题#2：不要忘记同步（Synchronization）

- 在多核系统中，多个 CPU 可能会 同时访问和修改共享数据
- 使用互斥（mutual exclusion）机制来保证正确性（guarantee correctness）

```
1   typedef struct __Node_t {
2       int value;
3       struct __Node_t *next;
4   } Node_t;
5
6   int List_Pop() {
7       Node_t *tmp = head;              // remember old head ...
8       int value = head->value;         // ... and its value
9       head = head->next;               // advance head to next pointer
10      free(tmp);                       // free old head
11      return value;                    // return value at head
12  }
```

代码示例：简单的链表删除操作（Simple List Delete Code with Lock）

问题#2：不要忘记同步（Synchronization）

- 解决方案：采用**互斥锁（mutex）**来确保线程安全。
- 锁的概念：加锁保持数据共享的一致性

```
1   pthread_mutex_t m;
2   typedef struct __Node_t {
3           int value;
4           struct __Node_t *next;
5   } Node_t;
6
7   int List_Pop() {
8           lock(&m);
9           Node_t *tmp = head;          // remember old head ...
10          int value = head->value;     // ... and its value
11          head = head->next;           // advance head to next pointer
12          free(tmp);                   // free old head
13          unlock(&m);
14          return value;                // return value at head
15  }
```

代码示例：带锁的链表删除操作（Simple List Delete Code with Lock）

问题#3：缓存亲和性（Cache Affinity）

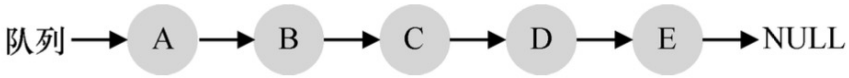
- **尽可能保持进程在相同的 CPU 上运行**
  - 进程在某个 CPU 上运行时，会在该 CPU 的**缓存**（**cache**）中存储部分状态信息。
  - 当进程再次运行时，如果其部分状态仍**存储在该 CPU 的缓存中**，执行速度会更快。

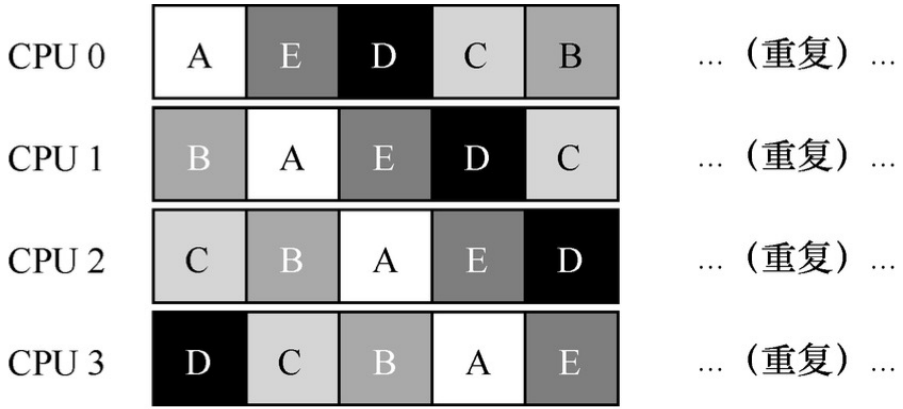
多处理器调度器在做出调度决策时，应考虑缓存亲和性（cache affinity）。

方法#1：单队列多处理器调度（SQMS）

- 所有作业都放入单一队列每个 CPU
  - 从全局共享队列中取出下一个作业执行
  - 优势：简单

- 短板
  - **缺乏可扩展性**：通过加锁保证准确性。系统开销大
  - **缓存亲和性低**：任务可能频繁在不同 CPU 之间切换，导致缓存数据失效

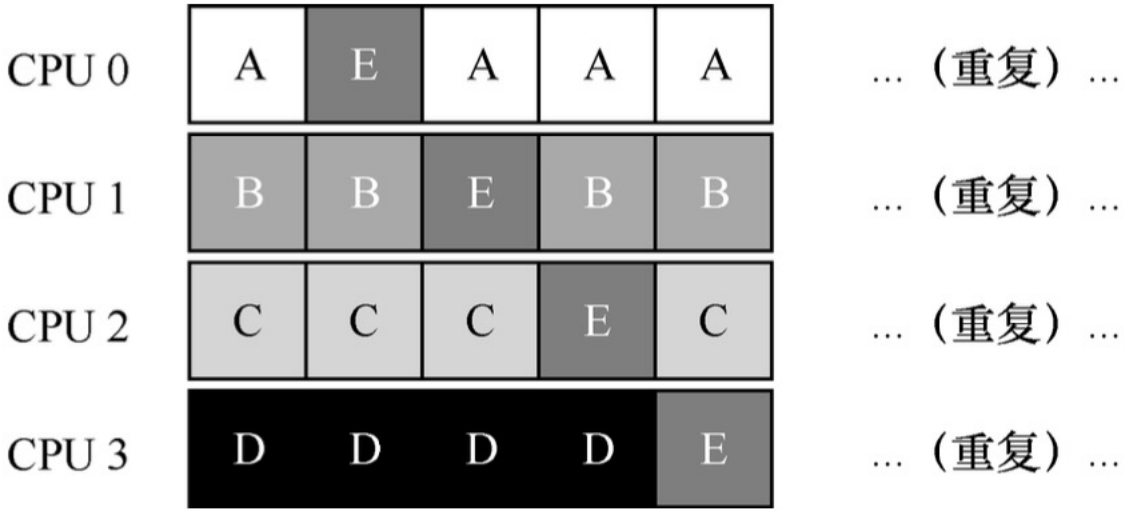




缓存亲和性调度示例

- 解决方案：尽量让同一个任务在同一个 CPU 上执行，减少缓存失效（Cache Miss）
- 尽量保持进程的缓存亲和性，可能需要牺牲其他工作的亲和度来实现负载均衡

- 任务 A 到 D 不会跨 CPU 迁移。
- 只有任务 E 在多个 CPU 之间迁移。
- 实现这种调度方案可能较为复杂。

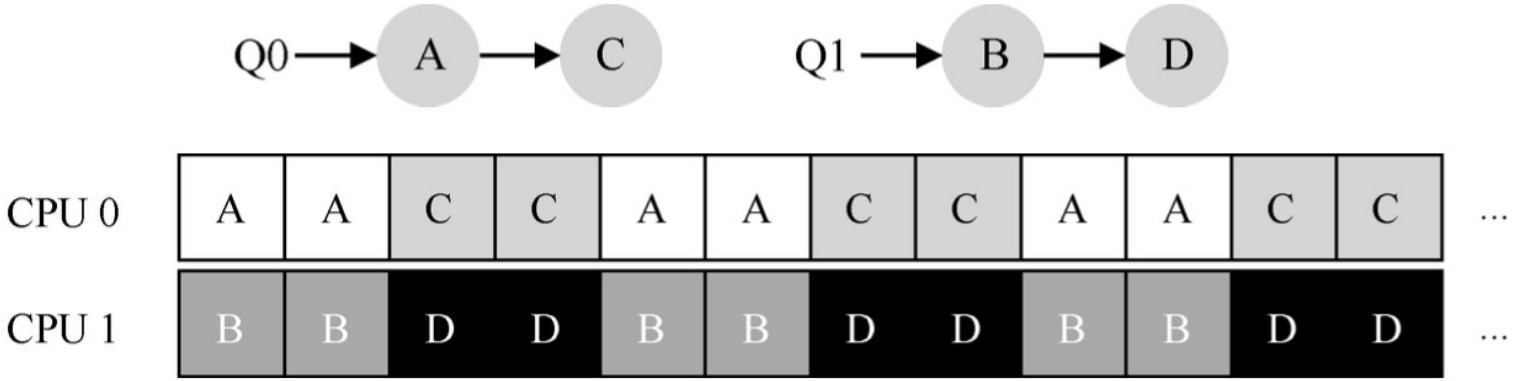


方法#2：多队列多处理器调度（MQMS）

- MQMS 由多个调度队列组成，提高系统可扩展性，并减少同步开销。
  - 每个队列遵循特定的调度策略。
  - 进程进入系统时，**只会被放入一个调度队列**。
- **主要优势：**
  - **减少信息共享和同步开销**，降低多核系统中的锁竞争问题，提高系统扩展性。
  - **提高缓存亲和性**，进程倾向于在固定 CPU 上执行，减少缓存失效（Cache Miss）。
  - **更灵活的调度策略**：可以根据任务类型或优先级，设计不同的调度策略，提高调度效率。

方法#2：多队列多处理器调度（MQMS）

- 采用轮转调度（Round Robin），每个 CPU 在自己的队列中公平地执行任务

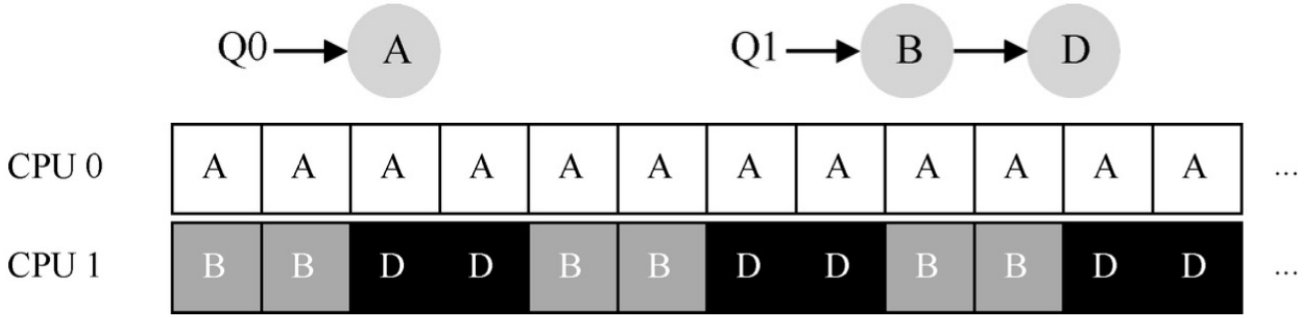


- **提高可扩展性（Scalability）**：避免了单队列调度（SQMS）中的锁竞争问题，提高 CPU 资源利用率。

- **增强缓存亲和性（Cache Affinity）**：任务 A 和 C 始终由 CPU0 处理，任务 B 和 D 始终由 CPU1 处理，减少任务迁移，**提高缓存命中率**，降低内存访问开销。

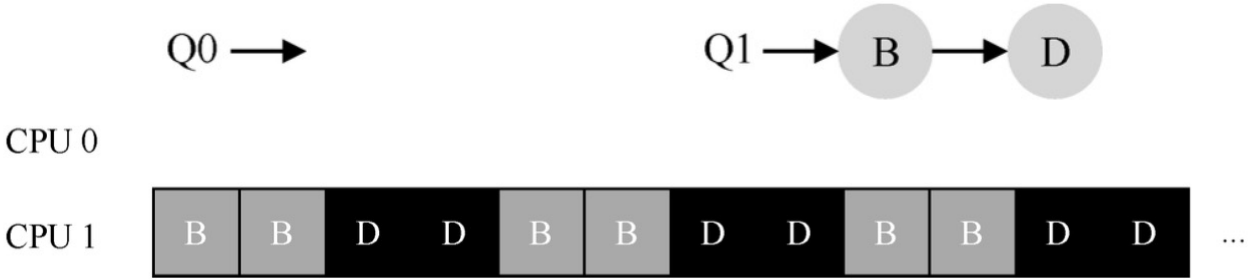
多队列调度的问题：负载不均

- 负载不均：假设某个任务执行完成（如Q0中的C）



A 占用了比 B 和 D 更多的 CPU 资源

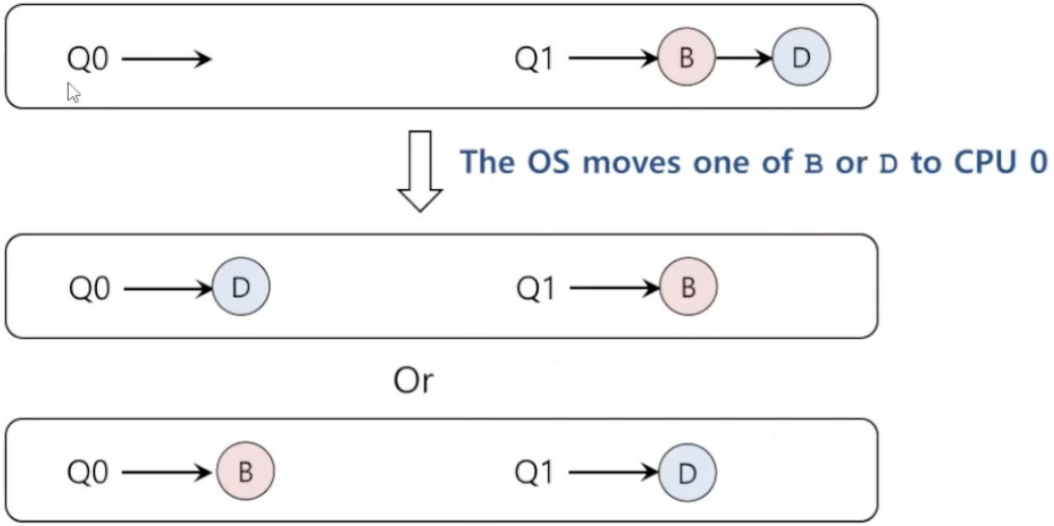
- 负载不均：假设2个任务执行完成（如Q0中的A, C）



CPU0 将处于空闲状态（Idle）

如何解决负载不均衡？

- 解决方案：迁移任务（Migration）。通过任务的跨 CPU 迁移，可以实现动态负载均衡



初始状态：Q0 为空，Q1 包含任务 B 和 D

操作系统（OS）将 B 或 D 迁移到 CPU0

迁移后的两种可能方案：

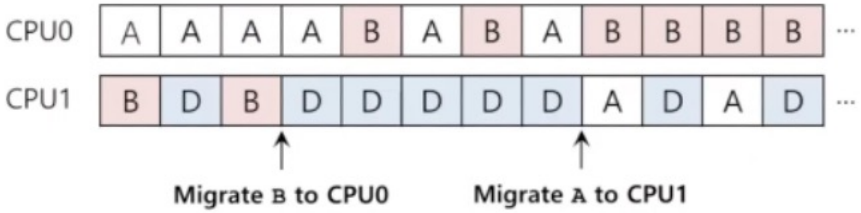
方案 1：D 迁移到 Q0，Q1 只剩 B

方案 2：B 迁移到 Q0，Q1 只剩 D

如何解决负载不均衡？

- 更复杂的情况：单个任务迁移无法解决问题（Q0 只有 A，Q1 仍然包含 B 和 D）

- 可能的迁移模式：
  - 持续交换任务（不断调整任务分配）



初始状态：CPU0 运行 A，CPU1 运行 B 和 D。
迁移 B 到 CPU0，使 CPU0 执行 A 和 B。
迁移 A 到 CPU1，使 CPU1 执行 A 和 D。
形成**动态迁移策略**，持续平衡 CPU 负载。

工作窃取（Work Stealing）

- 跨队列迁移进程（Move processes between queues）
  - 实现方式（Implementation）
    - •选择进程较少的源队列（Source Queue）
    - •源队列偶尔查看另一个目标队列（Target Queue）
    - •如果目标队列的负载高于源队列，源队列将“**窃取**”目标队列中的一个或多个进程
  - 缺点：高开销（High overhead）； 扩展性问题（Scaling trouble）

Linux 多处理器调度器

- **O(1) 调度器**
  - •基于**优先级**的调度算法。
  - •采用**多队列**调度机制。
  - •**动态调整进程优先级**。
  - •**优先调度**高优先级进程。
  - •**专注于交互性（Interactivity）**，优化用户体验。

- **完全公平调度器（CFS）**
  - 采用**确定性的比例公平分配（Proportional-Share Approach）**
  - 通过红黑树（Red-Black Tree）数据结构进行管理
  - **使用多队列**进行调度。

Linux 多处理器调度器

- **BF 调度器（BF Scheduler）**
  - 采用**单队列调度**（Single Queue Approach）
  - **按比例共享（Proportional-Share）**CPU 资源
  - 基于最早可执行虚拟截止时间优先（EEVDF, Earliest Eligible Virtual Deadline First）算法
- 优势：低延迟、高响应
- 缺点：扩展性问题，不适用于高并发场景

相关linux指令

- cat /proc/`pidof cpu`/sched　　%显示特定进程的调度统计信息
- ps –o thcount,nlwp `pidof cpu`　　%显示线程数量
- ps –L –p `pidof cpu`　　%显示线程/轻量级进程（LWP）ID
- ps –mo pid,tid,%cpu,psr `pidof cpu`　　%显示线程在哪个 CPU 核心上运行
- taskset –c 1 ./threads 10000000

cat /proc/`pidof cpu`/sched

./cpu hello

cat /proc/`pidof cpu`/sched

```
cpu (29898, #threads: 1)
-------------------------------------------------------------------
se.exec_start                                :      144607947.826514
se.vruntime                                  :         344942.580388
se.sum_exec_runtime                          :         344708.761098
se.nr_migrations                             :                     0
nr_switches                                  :                  2119
nr_voluntary_switches                        :                    12
nr_involuntary_switches                      :                  2107
se.load.weight                               :               1048576
se.avg.load_sum                              :                 47526
se.avg.runnable_sum                          :              48689026
se.avg.util_sum                              :              48687476
se.avg.load_avg                              :                  1023
se.avg.runnable_avg                          :                  1024
se.avg.util_avg                              :                  1024
se.avg.last_update_time                      :        144607947826176
se.avg.util_est                              :                   503
uclamp.min                                   :                     0
uclamp.max                                   :                  1024
effective uclamp.min                         :                     0
effective uclamp.max                         :                  1024
policy                                       :                     0
```

ps -o thcount,nlwp `pidof cpu `

- thcount（Thread Count）：该进程拥有的线程数。
- nlwp（Number of Light Weight Processes）：该进程的LWP（轻量级进程）数量，也表示线程数

```
[direct-execution]$ ps -o thcount,nlwp `pidof cpu`
THCNT NLWP
    1    1
```

ps –L –p `pidof cpu`

- -p <PID>：指定进程 ID，查询该进程的信息。
- -L：显示该进程的所有 LWP（轻量级进程，即线程）。

```
[direct-execution]$ ps -L -p `pidof cpu`
    PID     LWP TTY          TIME CMD
  29898   29898 pts/0    00:15:53 cpu
```

字段	含义
PID	进程 ID（所有线程共享同一个进程 ID）
LWP	轻量级进程（线程 ID），每个线程有唯一 LWP
TTY	终端信息
TIME	线程的 CPU 占用时间
CMD	进程名

ps –mo pid,tid,%cpu,psr `pidof cpu`

- -m：显示所有线程（LWP） 信息

- -o <字段>：指定要显示的列
  - pid：进程 ID（所有线程共享同一个 PID）。
  - tid：线程 ID（LWP ID），用于区分同一进程的不同线程。
  - %cpu：该线程的 CPU 使用率。
  - psr（Processor）：该线程当前运行在哪个 CPU 核心上。

```
[proc]$ ps -mo pid,tid,%cpu,psr `pidof cpu`
    PID     TID %CPU PSR
  29898       - 99.9   -
      -   29898 99.9   1
```

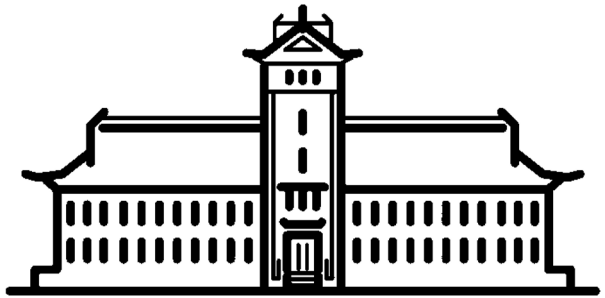
多个线程程序示例：threads

- ./threads-s 100000
- cat /proc/`pidof threads-s`/sched %显示特定进程的调度统计信息
- ps –o thcount,nlwp `pidof threads-s` %显示线程数量
- ps –L –p `pidof threads-s` %显示线程/轻量级进程（LWP）ID
- ps –mo pid,tid,%cpu,psr `pidof threads-s` %显示线程在哪个 CPU 核心上运行
- watch –n.5 “ps –mo pid,tid,%cpu,psr `pidof threads-s`
- htop –p `pidof threads-s`
- taskset –c 1 ./threads-s 10000000

小结

- 介绍了缓存以及相关的3个问题（缓存一致性、同步、亲和性）
- 介绍了多处理器调度的方法（SQMS、MQMS）
- 介绍了如何处理负载不均衡

谢谢



南京大學
NANJING UNIVERSITY