第 8 讲：全局页替换算法

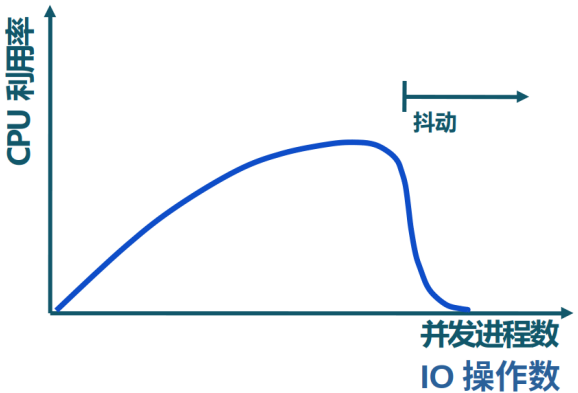
第四节：面向缓存的页替换算法-FBR

向勇、陈渝

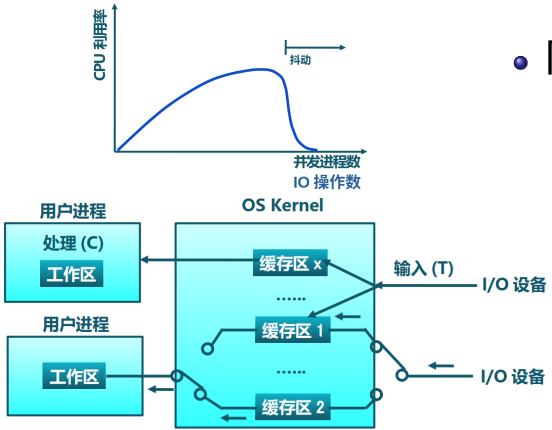
清华大学计算机系

xyong,yuchen@tsinghua.edu.cn

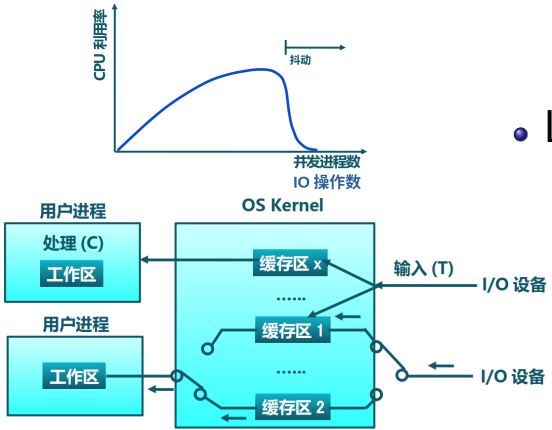
2020 年 4 月 12 日



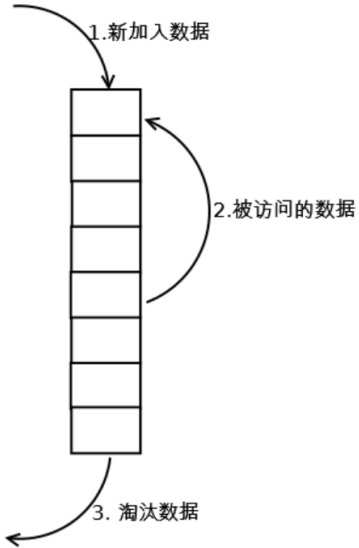
- 内存抖动问题（thrashing）
  - 大量的 IO 访问
  - 内存空间不够用



- 内存抖动问题（thrashing）
  - 全局页面置换算法
  - 工作集算法：换出不在工作集中的页面
  - 缺页率算法：使每个进程的缺页率保持在一个合理范围内
  - LRU 算法：选择最长时间没有被引用的页面进行置换

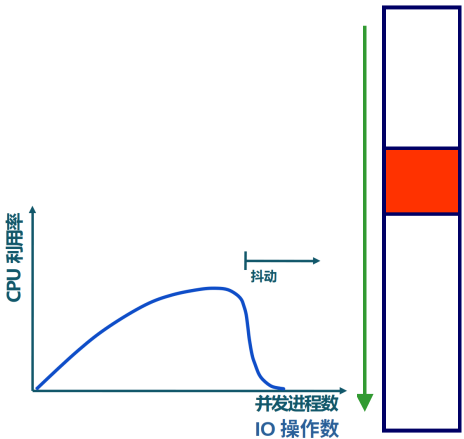


- LRU 算法
  - 选择在内存驻留时间最长的页面进行置换
  - 最优置换算法的一种近似
  - 还有缺点与不足吗?

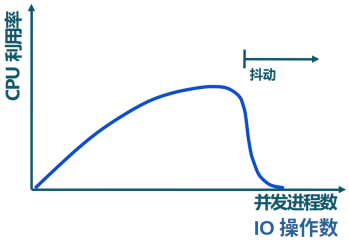


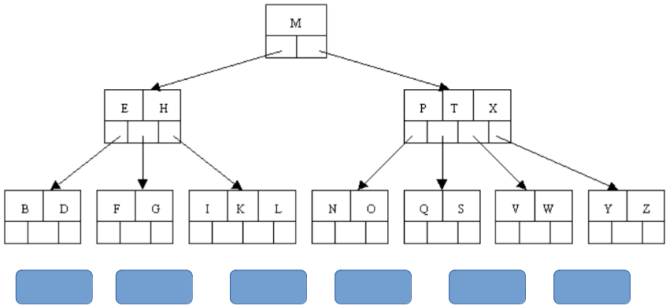
- LRU 算法的命中率
  - 当存在“热点”数据时，LRU 的效率很好
  - 但偶发性的、周期性的批量操作会导致 LRU 命中率急剧下降
  - 缓存污染情况比较严重

缓存污染：是指 OS 将不常用的数据从内存移到缓存，造成常用数据的挤出，降低了缓存效率的现象。

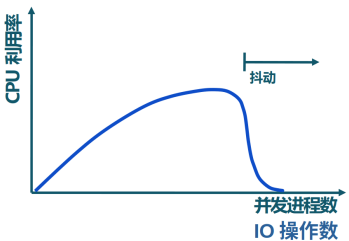


- 多个运行程序访问客户记录的例子：LRU?
  - 1,000,000 个客户关系数据
  - 大多数程序访问其中的一些红色部分 (0.5%)
  - 个别程序批处理方式 (绿线) 访问整个客户关系数据

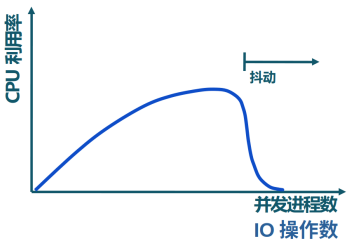




- 一个程序访问客户记录的例子: LRU?
  - 20,000 个客户关系数据
  - B-tree 放置客户 ID，20Bytes/key
  - 客户记录: 2000Bytes/record
  - 100 pages for 客户 ID
  - 10,000 pages for 客户记录
  - 共有 101 pages, 4KBytes/page
  - 访问模式：I1,R1,I2,R2,I3,R3...
  - 访问概率: 客户 ID pages 0.005
  - 访问概率: 客户记录 pages 0.00005



- 为何 LRU 失效?
  - 顺序块访问会把 hot 块替换出 cache
  - 对于索引块和数据块的循环访问时，不会根据访问概率缓存索引块
  - 完全从最近使用的时间角度考虑
  - “最近使用过 1 次”
  - 致命缺陷：就是没有考虑缓存单元的使用频率



- 为何 LRU 失效？那 LFU？
  - LFU（least frequently used）：淘汰使用频率最少的缓存单元
  - 完全从使用频率的角度考虑
  - 对最近的缓存单元的访问情况基本没考虑
  - 对访问模式的改变基本上没有应变的策略

- LRU+LFU?
  - THE LEAST RECENTLY/FREQUENTLY USED (LRFU) POLICY?
  - 在短周期中使用 LRU 算法，而在长周期中使用 LFU 算法?

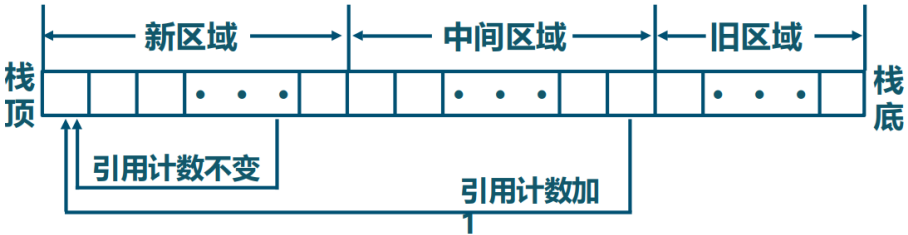
LRFU: A Spectrum of Policies that Subsumes the Least Recently Used and Least Frequently Used Policies ,Donghee Lee，IEEE-TC 2001

- LRU+LFU = FBR 访问频率置换算法 (Frequency-based Replacement)

访问频率置换算法 (Frequency-Based Replacement)

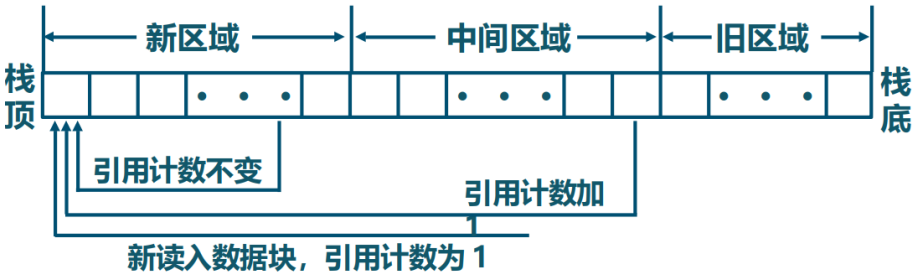
- **把 LRU 算法中的特殊栈分成三部分，并在每个缓存块增加一个引用计数**
  - **新区域 (New Section)**
  - **中间区域 (Middle Section)**
  - **旧区域 (Old Section)**

- LRU+LFU = FBR 访问频率置换算法 (Frequency-based Replacement)



- 栈中缓存块被访问时移到栈顶；如果该块在新区域，引用计数不变；否则，引用计数加 1
  - 在新区域中引用计数不变的目的是避免密集访问对引用计数不利影响
  - 在中间区域和旧区域中引用计数加 1 是为了使用 LFU 算法

- LRU+LFU = FBR 访问频率置换算法 (Frequency-based Replacement)



- 未缓存数据块读入后放在栈顶，引用计数为 1
- 在旧区域中引用计数最小的缓存块被置换
  - 中间区域的定义是为了避免新读入的缓存块在第一次出新区域时马上被置换，有一个过渡期

- FBR 访问频率置换算法的问题
  - 需要可调参数：缓存中三块的大小，Cmax 和 Amax
  - 调整可调参数的时间周期

访问频率置换算法 (Frequency-Based Replacement)

- **把 LRU 算法中的特殊栈分成三部分，并在每个缓存块增加一个引用计数**
  - **新区域 (New Section)**
  - **中间区域 (Middle Section)**
  - **旧区域 (Old Section)**

第 8 讲：全局页替换算法

第四节：面向缓存的页替换算法 – LRU-K 2Q

向勇、陈渝

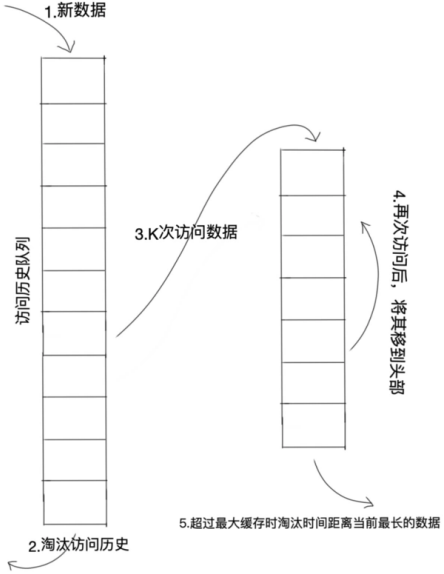
清华大学计算机系

xyong,yuchen@tsinghua.edu.cn

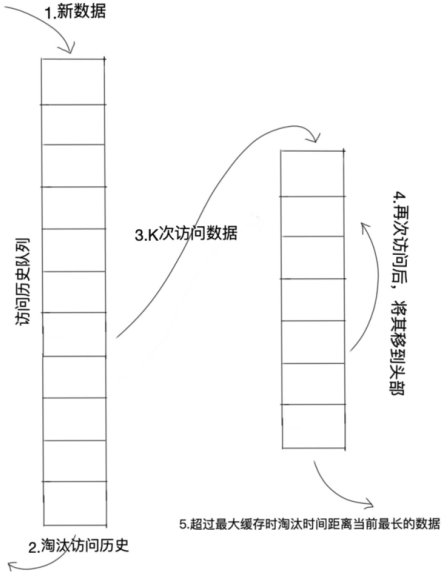
2020 年 4 月 12 日

- 有更简洁的 LRU+LFU 的设计思路吗？能够应对：
  - 顺序访问：所有的块一个接一个被访问，不存在重访问
  - 循环访问：所有块都按照一定的间隔重复访问
  - 时间密集访问：最近被访问的块是将来最有可能被访问的
  - 概率访问：所有块都有固定的访问概率，所有块都互相独立地根据概率被访问

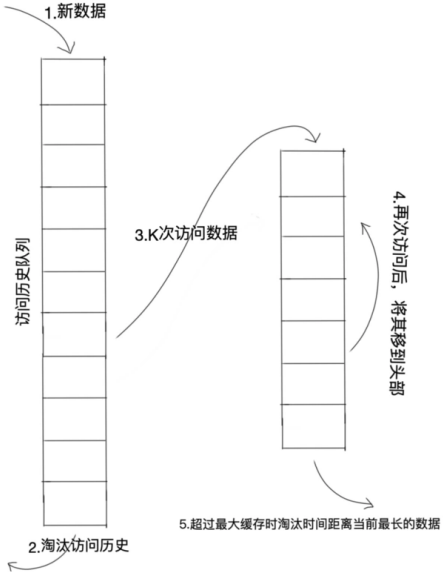
最近使用过 K 次？



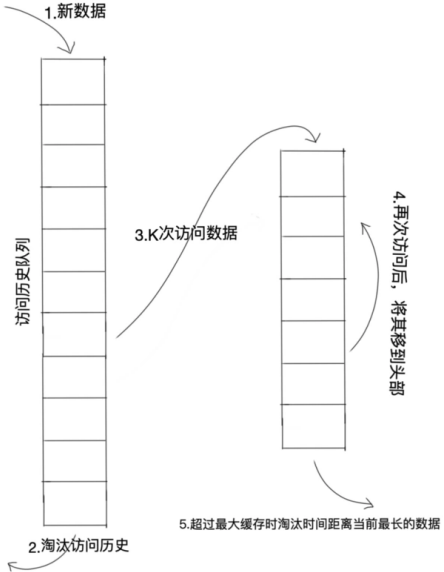
- LRU-K
  - LRU-K 是基于 LRU 算法的扩展，其中 K 代表最近访问的次数，从某种意义上，LRU 可以看作是 LRU-1 算法，引入 K 的意义是为了解决缓存污染问题。
  - 其核心理念是从“数据最近被访问过 1 次”蜕变成“数据最近被访问过 K 次，那么将来被访问的概率会更高”。



- LRU-K
  - LRU-K 与 LRU 区别：LRU-K 多了一个数据访问历史记录队列（需要注意的是，访问历史记录队列并不是缓存队列，所以是不保存数据本身的，只是保存对数据的访问记录），访问历史记录队列中维护着数据被访问的次数以及时间戳。
  - 只有当这个数据被访问的次数大于等于 K 值时，才会从历史记录队列中删除，然后把数据加入到缓存队列中去。

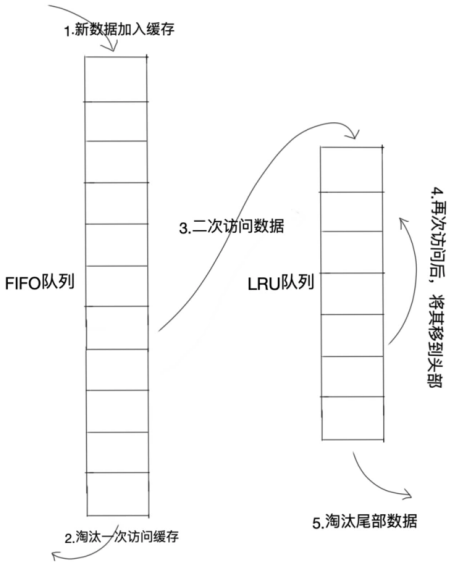


- LRU-K
  - 数据第一次被访问时，加入到历史访问记录队列中，访问次数为 1，初始化访问时间戳；
  - 如果数据访问次数没有达到 K 次，则访问次数 +1，更新时间戳。当队列满了时，按照某种规则（LRU 或者 FIFO）将历史记录淘汰；
  - 当访问历史队列中的数据访问次数达到 K 次后，将数据索引从历史队列删除，将数据移到缓存队列中，缓存队列重新按照时间排序；
  - 缓存数据队列中被再次访问后，重新排序
  - 需要淘汰数据时，淘汰缓存队列中排在末尾的数据，即：淘汰“倒数第 K 次访问离现在最久”的数据

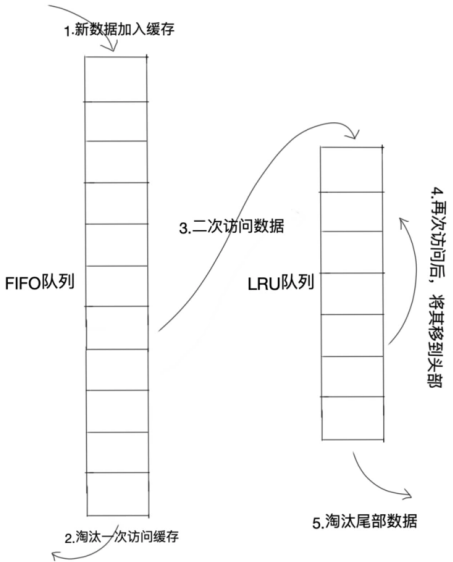


- 评价 LRU-K
  - LRU-K 降低了“缓存污染”带来的问题，命中率比 LRU 要高。
  - 在实际应用中,LRU-2 是综合各种因素后最优的选择。
  - LRU-3 或更大的 K 值命中率会高，但适应性差，一旦访问模式发生变化，需要大量的新数据访问才能将历史热点访问记录清除掉。
  - LRU-K 数据缓存队列一般是一个优先级队列。排序操作需要额外的 O(logN) 的时间复杂度，N 为数据缓存队列的大小。

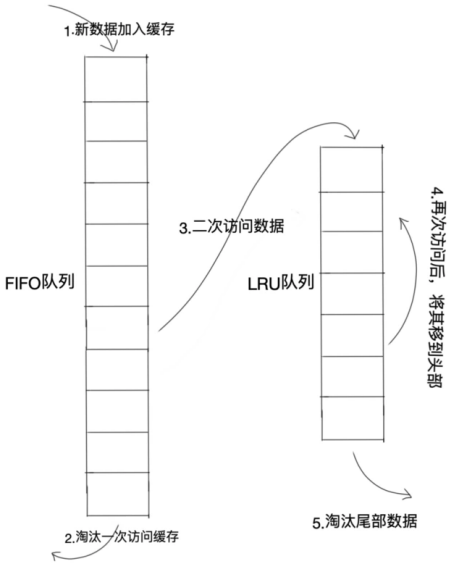
从能否进一步改进?



- 2Q
  - 2Q 算法类似于 LRU-2，不同点在于 2Q 将 LRU-2 算法中的访问历史队列（注意这不是缓存数据的）改为一个 FIFO 缓存队列，即：2Q 算法有两个缓存队列，一个是 FIFO 队列（First in First out，先进先出），一个是 LRU 队列。
  - 当数据第一次访问时，2Q 算法将数据缓存在 FIFO 队列里面，当数据第二次被访问时，则将数据从 FIFO 队列移到 LRU 队列里面，两个队列各自按照自己的方法淘汰数据。



- 2Q
  - 新访问的数据先插入到 FIFO 队列中；
  - 如果数据在 FIFO 队列中一直没有被再次访问，则最终按照 FIFO 规则淘汰；
  - 如果数据在 FIFO 队列中被再次访问，则将数据从 FIFO 删除，加入到 LRU 队列头部；
  - 如果数据在 LRU 队列再次被访问，则将数据移到 LRU 队列头部；
  - LRU 队列淘汰末尾的数据。



- 评价 2Q
  - 计算开销小于 LRU-2
  - 命中率与 LRU-2 类似，命中率要高于 LRU
  - 需要维护两个队列，代价是 FIFO 和 LRU 代价之和
  - 仍然需要配置参数

从能否进一步改进?

第 8 讲：全局页替换算法

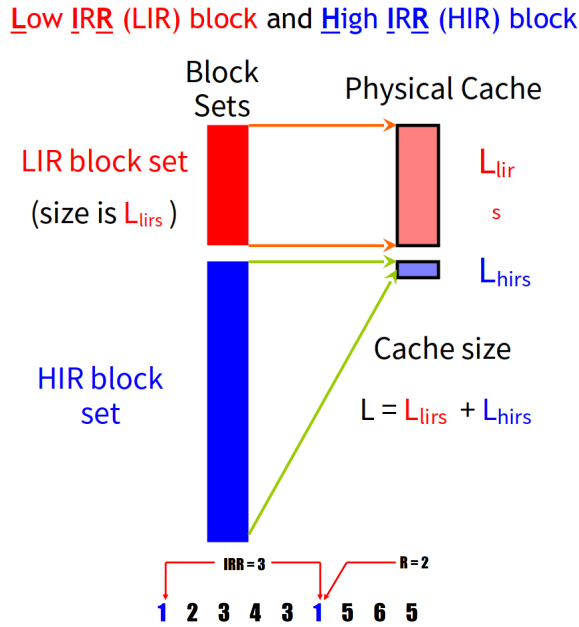
第四节：面向缓存的页替换算法 – LIRS

向勇、陈渝

清华大学计算机系

xyong,yuchen@tsinghua.edu.cn

2020 年 4 月 12 日



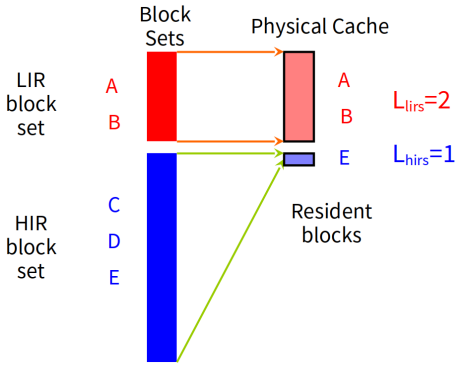
LIRS (Low Inter-reference Recency Set)

- Recency：最近被访问的不重复块数(表示时间属性)。
- IRR(Inter-Reference Recency)：同一块连续两次访问期间中间访问过的不重复块数。
- LIRS 算法动态区分低 IRR(LIR) 和高 IRR(HIR) 的块，LIR 块一般会常驻 cache，HIR 块则会较快被替换出 cache。
- 基本思路：如果块的 IRR 值高，那么它的下一次 IRR 值也会很高，所以要替换 IRR 值高的块。

LIRS: An Efficient Low Inter-reference Recency Set Replacement Policy to Improve Buffer Cache Performance, Song Jiang, Xiaodong Zhang, SIGMETRICS, 2002

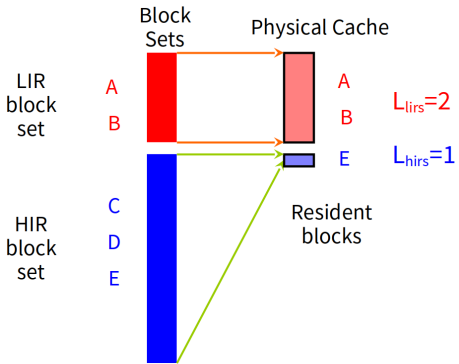
LIRS 例子

L_{lirs}=2, L_{hirs}=1



V time / Blocks	1	2	3	4	5	6	7	8	9	10	R	IRR
A	X					X		X			1	1
B			X		X						3	1
C				X							4	inf
D		X					X				2	3
E									X		0	inf

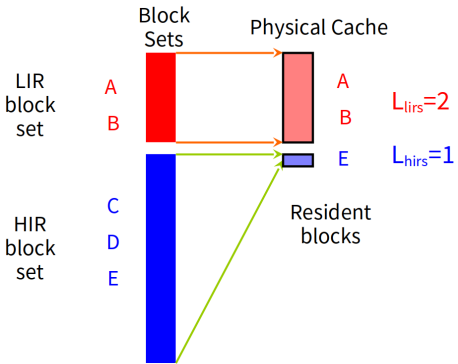
LIR block set = {A, B}, HIR block set = {C, D, E}



LIRS 例子：D 在时刻 10 被访问，替换哪个块?

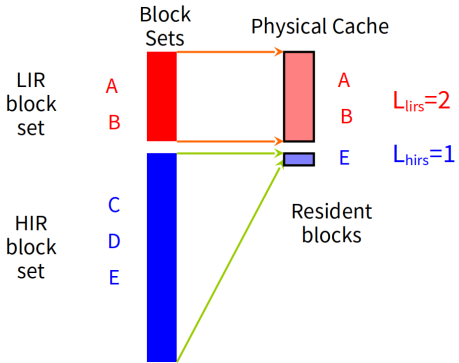
V time / Blocks	1	2	3	4	5	6	7	8	9	10	R	IRR
A	X					X		X			1	1
B			X		X						3	1
C				X							4	inf
D		X					X			X	0	3
E									X		1	Inf

替换 HIR 块 E



LIRS 例子：LIR Set 如何更新?

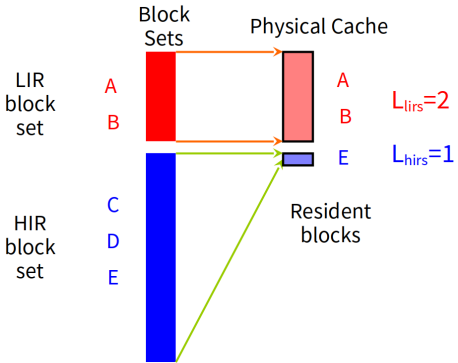
V time / Blocks	1	2	3	4	5	6	7	8	9	10	R	IRR
A	X					X		X			2	1
B			X		X						3	1
C				X							4	inf
D		X					X			X	0	2
E									X		1	Inf



LIRS 例子：LIR Set 如何更新?

V time / Blocks	1	2	3	4	5	6	7	8	9	10	R	IRR
A	X					X		X			2	1
B			X		X						3	1
C				X							4	inf
D		X					X			X	0	2
E									X		1	Inf

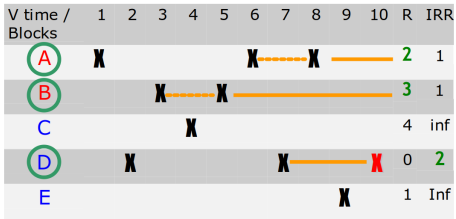
E 被替换，D 进入 LIR Set，B 进入 HIR Set



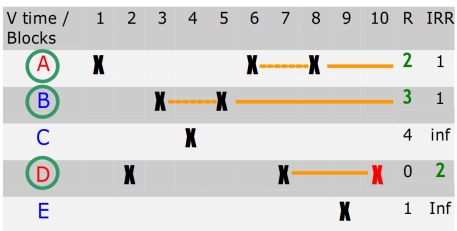
LIRS 例子：C 在时刻 10 被访问，替换哪个块？如何更新 LIR/HIR Set?

V time / Blocks	1	2	3	4	5	6	7	8	9	10	R	IRR
A	X					X		X			2	1
B			X		X						4	1
C				X						X	0	4
D		X					X				3	3
E									X		1	Inf

E 被替换，C 不能进入 LIR 块集合

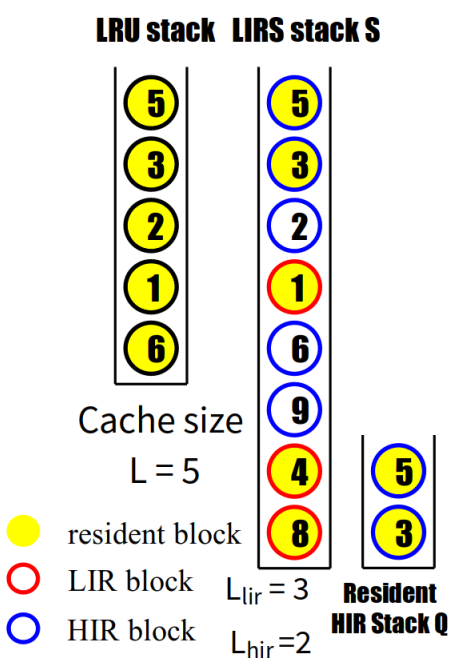


E 被换掉，D 与 B 的状态进行了交换



LIRS 替换策略

- LIR 块一般会常驻 cache，HIR 块则会较快被替换出 cache。通常假设 LIR 占 99% 的 cache 大小，HIR 占 1% 即可。
- 如果块的 IRR 值高，那么它的下一次 IRR 值也会很高，所以要替换 IRR 值高的块。
- 要保证所有 LIR 块都能缓存，当 LIR 块的 Recency 超过 IRR 最大值，且 HIR 块在一个更小的 Recency 中被访问，两者的状态就会交换。

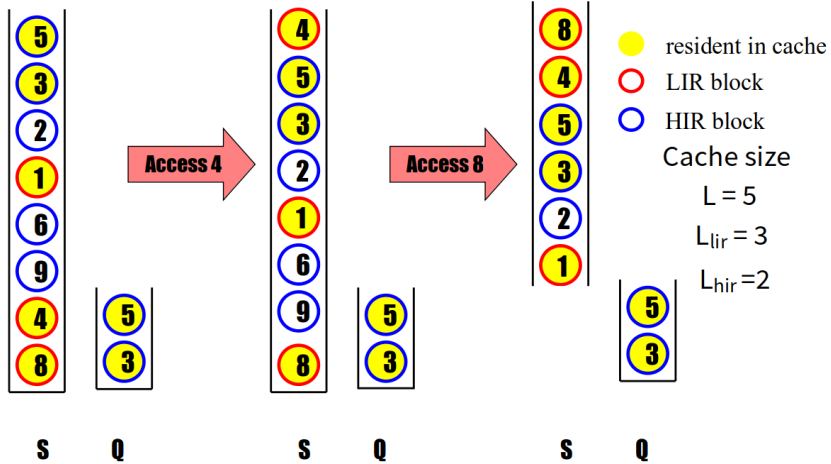


LIRS 替换算法

- Stack S: 包括 LIR 块、少于 LIR 块最大 recency 的 HIR 块
- Stack Q: HIR 块 FIFO 缓存队列 (加快 HIR 块缓存的搜索)
- “栈裁剪”操作，栈 S 的底部 LIR 块被删除，则一直删除底部块直到遇到另一个 LIR 块。这样做的目的是因为如果底部存在 HIR 块，则这些 HIR 块必定大于 LIR 块的最大 recency，这样它们肯定不能转变为 LIR 块。

LIRS 替换算法 (LIR hit)

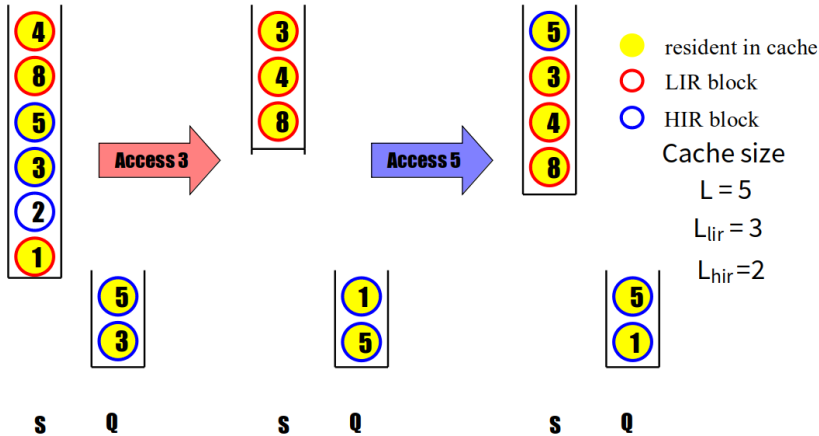
- 访问栈 S 中的 LIR 块 X：LIR 块必定驻 cache 中，所以必定命中缓存。然后把块 X 移动到栈 S 的头部，如果块 X 之前是在栈 S 的底部，则执行“栈裁剪”操作。



从

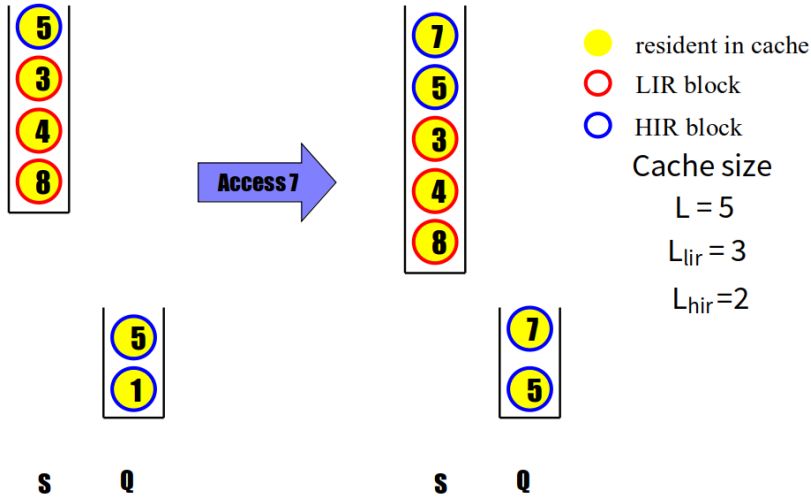
LIRS 替换算法 (HIR hit)

- 访问驻 cache 中的 HIR 块 X，把 X 移动到栈 S 头部，有两种情况：
- 在 S 中，X 状态转换为 LIR，删除 Q 中 X 的 cache，把 S 底部的 LIR 块转为 HIR 块，移动到 Q 中，最后“栈裁剪”。
- 不在 S 中，则 X 状态保持 HIR，从队列 Q 中移动到 Q 头部。



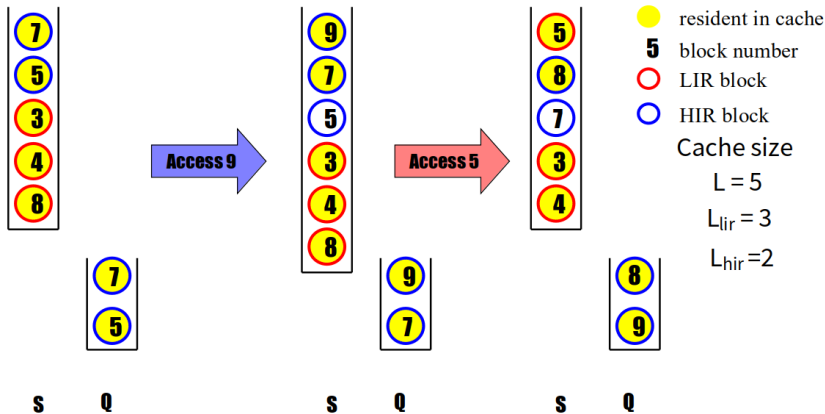
LIRS 替换算法 (Missing)

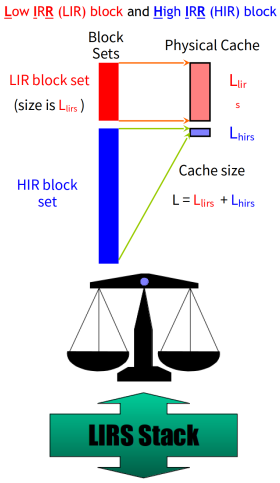
- 访问非驻留的 HIR 块 X。删除 Q 尾部的 HIR 块，如果该块在栈 S，则变为非驻留状态，加载块 X，把 X 移动到栈 S 的顶部。有两种情况：
  - 块 X 不在栈 S 中，则状态为 HIR，并放到队列 Q 的头部。



LIRS 替换算法 (Missing)

- 访问非驻留的 HIR 块 X。删除 Q 尾部的 HIR 块，如果该块在栈 S，则变为非驻留状态，加载块 X，把 X 移动到栈 S 的顶部。有两种情况：
  - 块 X 在栈 S 中，改变状态为 LIR，并同时改变栈底部的 LIR 块为 HIR 块，并移动到队列 Q 的头部，然后“栈裁剪”。





- 评价 LIRS
  - LIRS 能够快速适应上述 4 种访问模式。
  - 特别对于循环访问，LIRS 能够固定开始的 LIR 块驻 cache 中，保证了 cache 命中率。
  - LIRS 不像 2Q 等需要设置过多参数。
  - 实现的复杂度类似 LRU。
  - 从性能角度来看，LIRS>2Q>LRU-K>LRU

从能否进一步改进？(Your Work)

第 1 讲：操作系统概述

第一节：课程概述

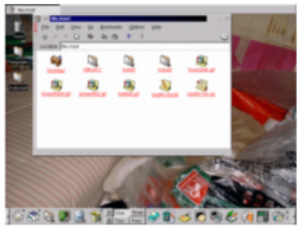
向勇、陈渝

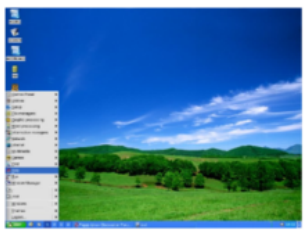
清华大学计算机系

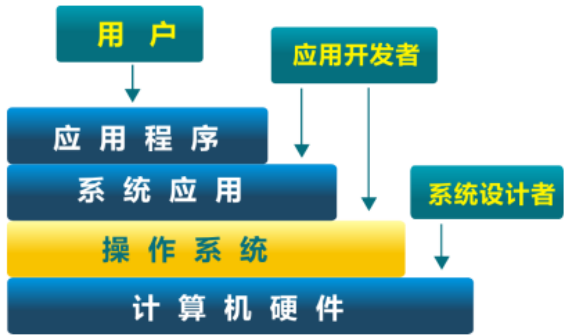
xyong,yuchen@tsinghua.edu.cn

2020 年 4 月 12 日

- Apple







- Apple
  - RedApple

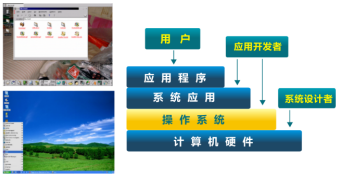
- Apple
  - RedApple

- Apple

- Apple
  - RedApple

- Apple
  - BlueApple

- Apple
  - RedApple



操作系统定义

没有公认的精确定义

操作系统定义

操作系统是管理硬件资源、控制程序运行、改善人机界面和为应用软件提供支持的一种系统软件。[计算机百科全书]

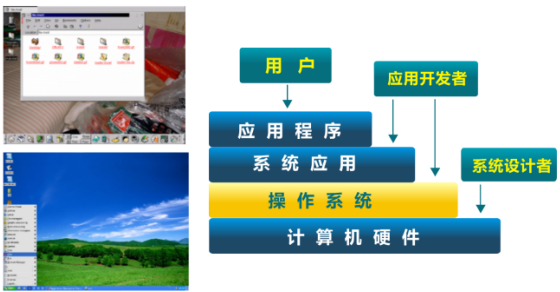


图: 承上启下的操作系统

- 第一章：独立式可执行程序
- 第六章：内核线程

This is slide number 12345.

- Apple
- Peach
- Plum
- Orange

- This is important.
- We want to highlight this and this.
- What is the matrix?

An Algorithm For Finding Primes Numbers.

```
^^Iint main (void)
^^I{
^^Istd::vector<bool> is_prime (100, true);
^^Ifor (int i = 2; i < 100; i++)
^^Iif (is_prime[i])
^^I{
^^Istd::cout << i << " ";
^^Ifor (int j = i; j < 100; is_prime [j] = false, j+=i);
^^I}
^^Ireturn 0;
^^I}
^^I
```

Note the use of `std::`.

An Algorithm For Finding Primes Numbers.

```
^^I^^Iint main (void)
^^I^^I{
^^I^^I
^^I^^I^^Istd::vector<bool> is_prime (100, true);
^^I^^I
^^I^^I^^Ifor (int i = 2; i < 100; i++)
^^I^^I
^^I^^I^^Iif (is_prime[i])
^^I^^I
^^I^^I^^I{
^^I^^I
^^I^^I^^Istd::cout << i << " ";
^^I^^I
^^I^^I^^Ifor (int j = i; j < 100;
```

What's Still To Do?

Answered Questions

How many primes are there?

Open Questions

Is every even number the sum of two primes?

What's Still To Do?

Answered Questions

How many primes are there?

Open Questions

Is every even number the sum of two primes?

What Are Prime Numbers?

定义

A prime number is a number that has exactly two divisors.

课程信息

- First point.

<.-> Second point.

- Third point.

课程信息

- First point.
- Second point.
- Third point.

课程信息

- 主讲教师：向勇、陈渝
- 助教:
  - 王润基、贾越凯、戴臻旸、刘丰源
  - 吴一凡、潘庆霖、张译仁、陈嘉杰
- First point.
- Second point.
- Third point.

预备知识

- 程序设计语言（汇编、C 和 Rust）
  - :(不是开发应用程序
  - :) 而是开发系统程序
- 数据结构
  - :) 理解基本数据结构即可
- 计算机组成原理
  - :(康总的 x86/mips 原理
  - :) Patterson 的 RISC-V 原理
- 编译原理
  - :) 没学过影响不大
  - :(但还是要了解高级语言 <–>RISC-V 汇编语言

课程信息

- 课程 **WIKI**
  - 所有课程信息的入口，也是最新课程信息的官方发布网站
- 课程视频
  - 学堂在线：操作系统 (RISC-V)
- 作业
  - 网络学堂 – 向老师
  - 网络学堂 – 陈老师
- 实验楼的在线实验环境
  - ucore os 在线实验环境
  - rcore os 在线实验环境
- 讨论区
  - Piazza 交流论坛
  - 微信：OS2020