reading schedule and notes

lightjames

2017 年 6 月 1 日

目录

1 STARTED 并行算法设计与性能优化

1.1 STARTED chapter01 绪论

1.1.1 TODO 1.1 并行和向量化的作用

-

2 STARTED Advanced Programming in the Unix Environment, Third Edition

2.1 STARTED chapter 04 文件和目录

2.1.1 DONE 4.10 Sticky Bit

- 程序的正文部分 ==> 机器指令

2.1.2 DONE 4.11 chown, fchown, fchownat and lchown Functions

- results: change a file's user ID and group ID.
- when the **referenced file is a symbolic link**, lchown and fchownat (with $\text{AT}_{\text{SYMLINKNOFOLLOW}}$ flag set) only change the owners of the symbolic link itself, not the file pointed to by the symbolic link.
- what if $\_\text{POSIX}_{\text{CHOWNRESTRICTED}}$ is in effect?

2.1.3 DONE 4.12 File Size

- for a symbolic link, the file size is **the number of bits in the filename**.

- **holes in a file**

2.1.4 DONE 4.13 File Truncation

- a special case of truncation: open a file with the O_{TRUNC} flag to empty a file.
- functions: truncate() and ftruncate().

2.1.5 DONE 4.14 File Systems

- i-nodes: the i-nodes are fixed-length entries that contain most of the information about a file.
- notice:
  1. each i-node has a link count that contains the number of directory entries that point to it. **only when the link count goes to 0 can the file be deleted.** these types of links are called **hard links**.
  2. the other type of link is called a **symbolic link**.
  3. only two items of interest are stored in the directory entry: the filename and the i-node number.
  4. because the i-node number in the directory entry points to an i-node in the same file system, **a directory entry can't refer to an i-node in a different file system**.
  5. when **renaming a file** without changing file system, the actual contents of the file needn't be moved ==> all that need to be done is to **add a new directory entry that points to the existing i-node** and then **unlink the old directory entry**.
- what about the link count field of a directory?

1. any *leaf directory*(a directory that does not contain any other directories) always has a link count of 2: the directory entry that names te directory and the entry for dot in that directory.
2. each subdirectory in a parent directory causes the parent directory's link count to be increased by 1. (.. entry added)

2.1.6 DONE 4.15 link, linkat, unlink, unlinkat, and remove Functions

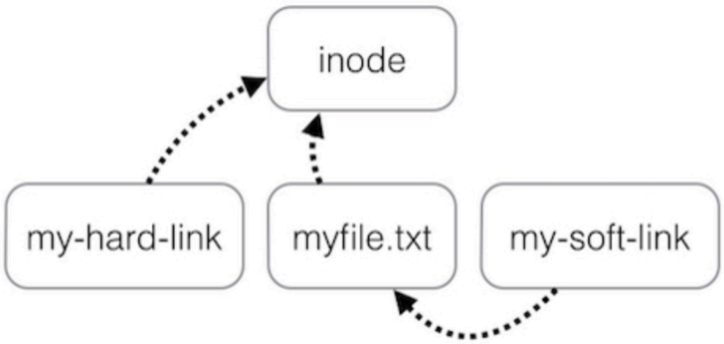
- link and linkat
  - only the last component of the *newpath* is created, **the rest of the path must already exist**.
  - when the **existing file is a symbolic link**, the *flag* argument controls whether the linkat function creates a link to the symbolic link or to the file to which the symbolic link points.
  - the **creation** of the new directory entry and the **increment** of the link count must be an **atomic** operation.
  - if supports **the creation of hard links to directories**, it may **cause loops** in the file system. many file system implementations disallow hard links to directories for this reason.
- unlink and unlinkat
  - conditions that prevents the contents of the file from being deleted:
    - link count > 0
    - some process has the file open

2.1.7 DONE 4.16 rename and renameat Functions

- depending on whether *oldname* refers to a file, a directory, or a symbolic link, and what if *newname* already exist.

2.1.8 DONE 4.17 Symbolic Links

- differences between symbolic links and hard links:



  a symbolic link is an **indirect pointer** to a file, unlike the hard links which **point directly to the i-node** of the file.

- when using functions that refer to a file by name, we always need to know **whether the function follows a symbolic link**.

- notice: using symbolic links may **introduce loops into the file system**.

```
$ mkdir foo                    make a new directory
$ touch foo/a                  create a 0-length file
$ ln -s ../foo foo/testdir     create a symbolic link
$ ls -l foo
total 0
-rw-r-----  1 sar           0 Jan 22 00:16 a
lrwxrwxrwx  1 sar           6 Jan 22 00:16 testdir -> ../foo
```

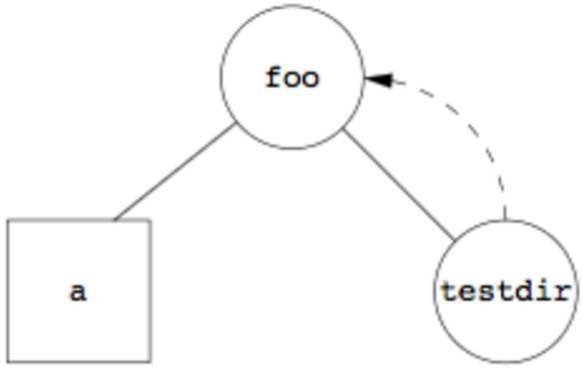


Figure 4.18 Symbolic link `testdir` that creates a loop

2.1.9 DONE 4.18 Creating and Reading Symbolic Links

- It's not required that *actualpath* exist when the symbolic link is created.
- Also, *actualpath* and *sympath* **needn't reside in the file system**.

2.1.10 DONE 4.19 File Times

- the difference between the modification time(st_{mtim}) and the changed-status time(st_{ctim}):
  - the **modification time** indicates when **the contents** fo the file were last modified.
  - the **changed-status time** indicates when **the i-node** of the file was last modified.
- Note that the system doesn't maintain the last-access time for an i-node.
- by default, the *ls* command displays or sorts using the modification time of the files.

2.1.11 DONE 4.20 futimens, utimensat, and utimes, Functions

- the three functions above only change the access time(st_{atim}) and the modification time(st_{mtim}), the changed-status time(st_{ctim}) is automaticlly updated.

2.1.12 DONE 4.21 mkdir, mkdirat, and rmdir Functions

- if the **link count** of the directory becomes 0 with a *rmdir* call, and if no other process has the directory open, then the **space** occupied by the directory is **freed**.

2.1.13 DONE 4.22 Reading Directories

- to preserve file system sanity(ok), only the kernel can write to a directory.

2.1.14 TODO 4.23 chdir, fchdir, and getcwd Functions

3 TODO Understanding the linux kernel, Third Edition (kernel version: 2.6.11)

4 TODO Linux Kernel Development, Third Edition (kernel version: 2.6.34)

5 STARTED Coding Interview Guide

5.1 DONE 第 1 章栈和队列

5.1.1 设计一个有 getMin 功能的栈

- 使用 2 个栈
- 一个保存 data，一个保存最小值

5.1.2 由两个栈组成的队列

- 如果 stackPush 要往 stackPop 中压入数据，必须一次性把 stackPush 中的数据全部压入。
- 如果 stackPop 不为空，stackPush 不能向 stackPop 中压入数据。

5.1.3 如何仅用递归函数和栈操作逆序一个栈

- 递归函数 1：将栈底元素返回并移除。
- 递归函数 2：逆序一个栈。

5.1.4 猫狗队列

- 不能改变用户原有的类
- 方法：将不同的实例盖上时间戳

5.1.5 用一个栈实现另一个栈的排序

- see page 14

5.1.6 用栈来求解汉诺塔问题

- recursive method, see page 15
- using stack, see page 17

5.1.7 生成窗口最大值数组

- 要求时间复杂度 O(n)
- 利用双端队列实现窗口最大值的更新，see page 20

5.1.8 构造数组的 MaxTree

- using stack and map, see page 23-25

5.1.9 求最大子矩阵的大小

- DP, using stack for each line, see page 27

5.1.10 最大值减去最小值小于或等于 num 的子数组数量

- using two deque: qmin and qmax

5.2 DONE 第 2 章链表问题

5.2.1 打印两个有序链表的公共部分

- easy

5.2.2 在单链表和双链表中删除倒数第 K 个节点

- see page 35

5.2.3 删除链表的中间节点和 a/b 处的节点

- 中间节点采用 two pointers, 一个 slow，一个 fast
- a/b：先找到该删除第 n 个节点（需要先遍历链表求表长）

5.2.4 反转单向和双向链表

- easy

5.2.5 反转部分单向链表

- 先找到反转部分的前一个节点和后一个节点，然后反转需要反转的部分，最后与前一个节点连接起来

5.2.6 环形单链表的约瑟夫问题

- 约瑟夫问题：用数学归纳法分析问题，寻求复杂度更低的解

5.2.7 判断一个链表是否为回文结构

- using stack, space O(n)
- reverse the right part, space O(1)

5.2.8 将单向链表按某值划分成左边小、中间相等、右边大的形式

- 方法 1：将所有 nodes 放入数组，对数组进行 partition 排序，connect
- 方法 2：将节点依次连入 small、equal、big 三个链表中，connect

5.2.9 复制含有随机指针节点的链表

- 方法 1：using map
- 方法 2：1->2->3 ==> 1->1'->2->2'->3->3'

5.2.10 两个单链表生成相加链表

- using stack
- 先 reverse，再相加

5.2.11 两个单链表相交的一系列问题

- 问题 1：判断链表是否有环
- 问题 2：判断两个无环链表是否相交
- 问题 3：判断两个有环链表是否相交

5.2.12 将单链表的每 K 个节点之间逆序

- method 1: using stack
- method 2: just using pointers

5.2.13 删除无序单链表中值重复出现的节点

- method 1: using $\text{unordered}_{\text{set}}$, time O(n), space O(n)
- method 2: selection sort, time $O(n^2)$, space O(1)

5.2.14 在单链表中删除指定值的节点

- method 1: using stack
- method 2: just using pointers

5.2.15 将搜索二叉树转换成双向链表

- method 1: using queue
- method 2: recursive method(kind of special)

5.2.16 单链表的选择排序

- $O(n^2)$

5.2.17 一种怪异的节点删除方式

- unsafe, unstable, not awlays correct

5.2.18 向有序的环形单链表中插入新节点

- 注意极端情况：新值大于所有值或小于所有值

5.2.19 合并两个有序的链表

- easy

5.2.20 按照左右半区的方式重新组合单链表

- easy

5.3 DONE 第 3 章二叉树问题

5.3.1 分别用递归和非递归方式实现二叉树先序、中序和后序遍历

- 非递归的后序遍历比较难

5.3.2 打印二叉树的边界节点

- getHeight ==> get height of a node
- get edges for each level and print(mind the sequence) ==> edgeMap
- print leaf not in edgeMap

5.3.3 如何较为直观地打印二叉树

- 反中序：右中左

5.3.4 二叉树的序列化和反序列化

- 先序遍历 + queue
- 层次遍历

5.3.5 遍历二叉树的神级方法

- Morris traversal
- hard

5.3.6 在二叉树中找到累加和为指定值得最长路径长度

- record posible sums of each level using map

5.3.7 找到二叉树中的最大搜索二叉子树

- 后序遍历
- 动态规划

5.3.8 找到二叉树中符合搜索二叉树条件的最大拓扑结构

- method 1: using queue, O(n^2)
- method 2: 拓扑贡献记录

5.3.9 二叉树的按层打印与 ZigZag 打印

- 关键是知道何时换行

5.3.10 调整搜索二叉树中两个错误的节点

- 先找到这两个节点 ==> 中序遍历，降序处
- 找到这两个节点的父节点 ==> 中序遍历
- 交换这两个节点 ==> 情况很复杂!

5.3.11 判断 t1 树是否包含 t2 树全部的拓扑结构

- 遍历

5.3.12 判断 t1 树中是否有与 t2 树拓扑结构完全相同的子树

- using KMP, O(N+M)

5.3.13 判断二叉树是否为平衡二叉树

- 后序遍历
- 在 getHeight 的过程中 check 左右子树的高度差

5.3.14 根据后序数组重建搜索二叉树

- 后序遍历的特点：头节点在最后，比头节点小的在左半部分，比头节点大的在右半部分

5.3.15 判断一颗二叉树是否为搜索二叉树和完全二叉树

- 中序遍历

5.3.16 通过有序数组生成平衡搜索二叉树

- 递归用数组中间的值生成头节点

5.3.17 在二叉树中找到一个节点的后继节点

- 分情况讨论

5.3.18 在二叉树中找到两个节点的最近公共祖先

- 后序遍历
- 记录

5.3.19 Tarjan 算法与并查集解决二叉树节点间最近公共祖先的批量查询问题

- hard

5.3.20 二叉树节点间的最大距问题

- 后序遍历

5.3.21 先序、中序和后序数组两两结合重构二叉树

- 给定先序和后序不一定能重构：如果一颗二叉树除叶节点之外，其他所有节点都有左孩子和右孩子，只有这样的树才可以被先序和后序数组重构出来

5.3.22 通过先序和中序数组生成后序数组

- 划分

5.3.23 统计和生成所有不同的二叉树

- DP

5.3.24 统计完全二叉树的节点数

- 利用完全二叉树的性质

5.4 TODO 第 4 章递归和动态规划

5.4.1 斐波那契系列问题的递归和动态规划

- method 1: 暴力递归，O(2^n)
- method 2: 顺序计算，O(n)
- method 3: n 阶递推数列 ==> 矩阵乘法，O(log n)

5.4.2 矩阵的最小路径和

- dp
  - space: O(m*n)
  - space: O(min(m, n))

5.4.3 换钱的最少货币数

- dp

5.4.4 换钱的方法数

- 面试中的一般优化轨迹：暴力递归 ==> 记忆搜索 ==> 动态规划

6 STARTED Beauty of Programming

6.1 DONE 面试杂谈

- 做题时要注意陷阱，而且面试者会不断深化这个问题
- 尽量弥补信息不对称，了解公司的最新科技、业务方向等细节很有帮助
- 考虑问题的全面程度和逻辑分析能力
- 每个人都是独立的个体，要有自己的想法，对自己的未来有规划
- 专业技巧:
  - 程序设计思路
  - 编程风格
  - 对细节的考虑
  - 内存泄漏
  - 最优算法
  - 修改程序以满足新的需求
  - 举一反三
- 职业技巧:
  - 交流能力

  - 合作能力
  - 自我评价和期望
  - 抗压能力
  - 追求卓越

- tips:
  - 知己知彼
    - 知己，就是要了解自己的能力、兴趣、职业发展方向
    - 知彼，就是要了解公司的文化、战略方向和择才标准
  - 笔试就是考察基础，用扎实的理解和考虑完备的解答征服阅卷者
  - 面试就是探讨，缜密的代码和严密的分析，思考问题的过程比结果更重要
  - 纸上得来终觉浅，绝知此事要躬行

6.2 TODO 第 1 章游戏之乐——游戏中碰到的题目

6.3 第 2 章数字之魅——数字中的技巧

6.4 第 3 章结构之法——字符串及链表的探索

6.5 第 4 章数字之趣——数字游戏的乐趣

7 C++ Primer (5e)

7.1 第一章开始

7.1.1 编译、运行程序

- C/C++ 程序编译流程：预处理 -> 编译 -> 汇编 -> 链接
  - 预处理：展开宏定义、处理条件编译指令、处理 #include、删除注释、添加行号和文件名标识等等
  - 编译：词法分析、语法分析、语义分析、优化，产生汇编代码

  - 汇编：将汇编代码翻译成机器指令，生成目标文件
  - 链接：将目标文件（或库文件）链接在一起生成可执行文件

- 打印上一个程序的返回值：

```
echo $?
```

7.1.2 初识输入输出

- endl: 结束当前行，将与设备相关的缓冲区中的内容刷新到设备中，等价于

```
os.put('\n');
os.flush();
```

7.1.3 控制流

- for or while?
  - 循环次数已知时，使用 for 更简洁；否则 while 更适合
- 从键盘键入文件结束符：
  - in Unix: Ctrl+D
  - in Window: Ctrl+Z

7.2 第二章变量和基本类型

7.2.1 基本内置类型

- 字和字节
  - 字节：可寻址的最小内存块
  - 字：存储的基本单元
- 选择数据类型：

  - 算术表达式中不用 char：char 在某些机器上是有符号的，而在另一些机器上是无符号的
  - 浮点运算用 double：float 精度不够，double 和 float 的计算代价相差无几

- 类型转换：
  - 右值超出范围时，有符号数和无符号数的处理方式不同：
    - 无符号数：初始值对无符号类型表示数值总数取模后的余数
    - 有符号数：结果是未定义的 (**undefined**)
  - 不要混用有符号数和无符号数：
    - 当有符号数为负时，会自动转换为无符号数，导致异常结果

- 指定字面值的类型（see p37)

7.2.2 变量