浙江理工大学学报(自然科学版),第35卷,第3期,2016年5月
Journal of Zhejiang Sci-Tech University (Natural Sciences)
Vol. 35, No. 3, May. 2016

DOI:10.3969/j.issn.1673-3851.2016.05.015

Linux文件搜索命令解析以及locate命令查询优化

王维维[1],李仁旺[1],赵亚平[2],王海周[1],宋圣涛[1]
(1.浙江理工大学机械与自动控制学院,杭州 310018;2.南京师范大学生命科学学院,南京 210046)

摘　要: 随着linux系统逐渐被广泛使用,系统使用者一般使用命令行的方式操作系统包括文件查找,特别是将linux系统作为服务器时,在数量众多的文件中找到需要的文件需要消耗很多时间,为了更快找到所需要的文件,针对现有的两种linux文件搜索命令的实现原理以及实现过程进行剖析,以及对著名的Boyer- Moore串匹配算法进行分析后,使用改进的BM算法对其中一种搜索命令即locate命令中字符串匹配算法进行优化,对改进算法的复杂度进行分析发现,在文件名的字符串匹配过程中与原来KMP算法相比较具有更高的效率,查找的速度更快。

关键词: linux文件搜索;find命令;locate命令;BM算法;updatedb命令
中图分类号: TP3-05　　**文献标志码:** A　**文章编号:** 1673-3851 (2016) 03-0409-05　**引用页码:** 050603

0 引　言

linux是一个开源软件,用户可以自由地获取程序和源代码,能够自由地修改和使用它们,随着网络的发展,linux爱好者以及众多技术人员已对其研究和完善。linux系统有着广泛的应用前景,特别是在嵌入式领域,移动通信设施、便携计算机、平板电脑、消费电子领域甚至军事领域都或多或少的存在嵌入式系统的影子[1]。在linux系统越来越普遍的情况下,如何使用命令行来查询系统中用户需要的文件以及采用哪种方式可以提高效率是用户特别是程序开发人员关心的问题。目前在linux系统中主要的查询命令有find和locate两种。find命令提供了很多的查找条件,包括文件名称、时间、文件类型、用户名称、文件大小等,功能比较强大,但是查询需要时间比较多,平时个人电脑中使用这个命令感觉不到,如果在文件很多或者服务器中搜索需要很长的时间。locate命令查询的文件名存储在一个数据库文件中,但是数据库中只有目录时间和文件名称。如果要查询某一文件需要先运行updatedb命令对数据库进行更新,否则无法检测到新增的文件,但是linux系统会定期执行这个命令。locate命令搜索时间快,局限性是只能根据文件的名称进行搜索。locate命令就是对用户输入的字符串和文件的路径名采用KMP算法的一个匹配的过程。本文在linux查找命令详细分析的基础上,用改进的BM算法对原来的KMP字符串模式匹配算法进行优化。

用户输入的字符串称为模式串,系统中所有文件的绝对路径称为文本串。模式匹配的问题可描述为:

$$\text{文本串:}T = T_0T_1\cdots T_{n-1};$$
$$\text{模式串:}P = P_0P_1\cdots P_{m-1}.$$

其中:$n > m$。要求在T文本串中找到等于模式串P的子串,如果T中存在P的子串则匹配成功,否则匹配失败,相当于在众多路径中选择出含有用户输入的字符串的路径。

1 find命令

find命令是通过find(char * arg)函数读取并且选择出符合用户要求的文件名称。这个函数用FTS数据结构来记录一个文件层次的句柄,它由函数fts_open()返回,代表文件层次本身也相当于一个目录,另一个是FTSENT代表文件层次里面的文件,fts_read()函数都会返回每个文件的一个

收稿日期:2015-07-26
基金项目:国家自然科学基金项目(51475434)
作者简介:王维维(1991-),男,安徽广德县人,硕士研究生,主要从事机械制造及其自动化方面的研究。
通信作者:李仁旺,E-mail:renwangli@zstu.edu.cn

FTSENT 结构。FTSENT 数据结构里面含有该文件的访问路径、根路径、文件名以及索引节点信息。consider_visiting()函数会根据该结构里面的内容判断该路径是否是用户需要查找的结果，find 函数的部分代码如下：

```
find (char *arg){
    …
    FTS *p;
    FTSENT *ent;
    …
    p = fts_open (arglist, ftsoptions, NULL);
    …
    while ((ent=fts_read(p)) != NULL)
    {
        consider_visiting(p, ent);
    }
}
```

该函数参数是用户输入的字符串，包括查询条件以及要查询的文件名称等，首先会用 fts_open()返回用户输入的目录参数的一个句柄，也就是文件层次，然后遍历文件层次里面的文件判断是否符合条件。对于文件层次、文件名称、索引节点之间的关系则由文件系统组织结构决定。

linux 文件系统都相应的被划分至少 4 个部分，其中引导块是介质上的第一个块，索引节点则取决于磁盘大小等参数，这些参数都存储在超级块当中[2]。linux 文件系统由引导快和分区或者记录块组组成，分区的数量取决于设备的容量和分区大小，如图 1 所示。

索引节点(*i* 节点) 用于存储文件的元数据的一个数据结构。文件的元数据，也就是文件的相关信息，和文件本身是两个不同的概念。它包含文件的大小，创建时间，修改时间，拥有者磁盘位置等和文件相关的信息。当访问文件的内容必须要通过 *i* 节点。

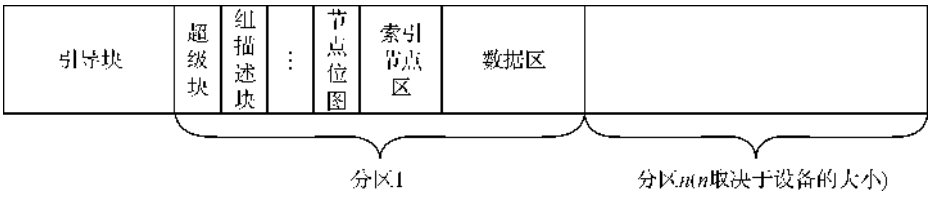

图 1 linux 系统文件格式

但是文件名不包含其中，当知道一个文件名称时候想知道这个文件相关的信息，这就是目录树的作用，目录树中包含目录名、索引节点以及数据部分，数据部分也就是各个目录项[2]。该目录相对于电脑上面的文件夹，目录项相当于文件夹里面的文件或者子目录以及其他一些信息。从专业角度说，目录是对文件管理的有效信息集合，也就是说看到的文件夹也就是目录的形式化表示[3]，目录树的结构图如图 2 所示。所以该函数用 FST 结构获取当前目录的文件层次也就是该目录，然后遍历该文件层次获取文件的 FTSENT 的数据结构，如果用户值用 find 命令在根目录下查找文件名为 find. c 的文件：find / －name find. c，则不需要使用索引节点，只需要判断用户名是否符合用户要求。如果需要根据文件的权限、所有者、类型等去查找则需要去使用索引节。例如 find / －perm 777。find 命令查询的范围比较广，能满足用户的各种查询需求，但是每次都需要遍历文件层次以及文件和索引节点信息，消耗的时间比较多。

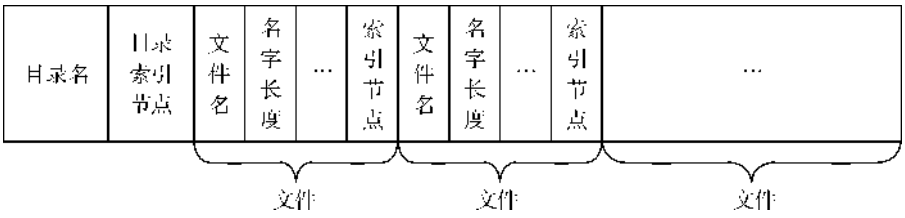

图 2 目录树结构

2 updatedb 命令

updatedb 用来建立或者更新数据库，该数据库主要包含文件的名称以及文件的修改时间。一般位于/var/lib/mlocate/mlocate. db 目录下，这个数据库已经存在它将读取未改变的部分以及将发生改变的部分重新写入其中。updatedb 是由 cron 命令每天每隔一段时间更新数据库。updatedb 的实现原

理是将文件名或者目录名以及每个目录的时间以一定的格式写入 mlocate. db 数据库。

它在执行之前,需要读取名称/etc/updated. conf 的文件检索配置信息,这个配置信息会过滤一些不需要检索的文件,提高检索速度。updatedb. conf 内容如下:

PRUNE_BIND_MOUNTS="yes":是否扫描 mount 过来的文件系统的文件,yes:1, no:0。

PRUNENAMES=". git . bzr . hg . svn":表示对哪些后缀的文件不采取检索,也就是列在这里面的后缀。

PRUNEPATHS="/tmp /var/spool/media":表示不检索的路径,即列出的路径下的文件和子文件夹均跳过不进行检索。

mlocate. db 是有一定格式的数据库文件,它是由含有 16 字节的头部,配置文件信息,路径为"/"的根目录和数据区组成,该数据区记录在一定时间某个路径下面所含有的文件名称和类型,它含有一个结束,符标志该数据区结束,其内容结构如图 3 所示。

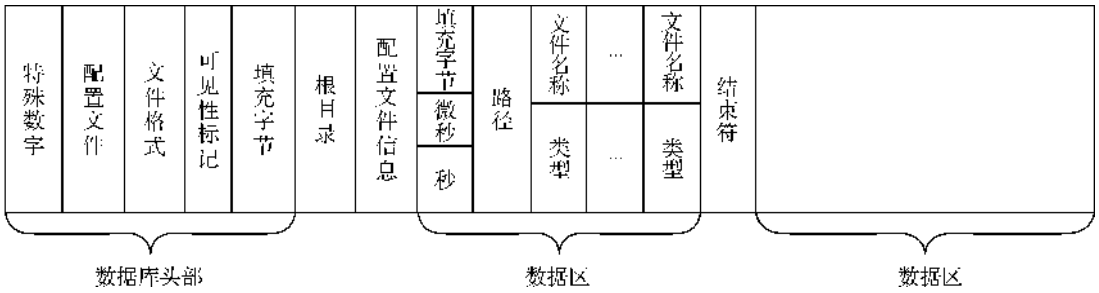

图 3　mlocate. db 数据库内容结构

这些功能都由 scan 函数实现,主要实现过程如下:

```
static int scan()
{
…//根据文件配置信息检查文件名是否符合目录要求
if(old_dir. path…) //路径和时间符合要求
{
    copy_old_dir(&dir)//从旧的数据库中读取信息
    goto have_dir
}
scan_cwd(&dir) //时间不符合要求重新扫描该目录
have_dir:write_directory(&dir) //写入数据库
if (have_subdir ! = false) { //含有与子文件夹
    scan_subdirs(&dir, &st) //继续调用 scan 递归扫描该目录的子文件夹
    }…
}
```

当数据库未建立的时候,updatedb 先写入头部信息,以及配置文件信息,然后主要是从根目录扫描文件系统,获取文件名字以及文件类型,并且根据文件的名称排序。写入数据库的过程中先写入该文件夹上次修改对应的时间(*s*,ms),以及该文件夹的路径,接下来按排好后的顺序依次写入文件类型(文件或者目录)、文件名称,最后写入结束符。然后对已经写入数据库的并且是文件目录的,执行递归操作,直到所有文件都已经访问并写入数据库。

当数据库已经建立之后,updatedb 从原先数据库读取头部,配置信息写入一个新的数据库,然后开始读取数据区,它依照原有数据库中依次读取记录,每当读取一个数据区或者文件夹后,它用数据库中时间与现在文件夹中最近一次更新的时间进行对比,如果时间没有改变那么直接写入新数据库,否则重新扫描该目录下的所有文件然后写入,直到结束。

3　locate 命令

locate 命令能够快速查找文件从 mlocate . db 数据库中,当前新创建的文件最好需要先运行 updatedb 命令,否则最新的文件可能不在数据库中。locate 实现原理是按照一定的格式从 mlocate . db 读取数据区的数据,这个格式与 updatedb 命令写入的格式相同。然后对路径加上文件名之后的字符串相当于正文与需要查找的字符串相当于模式串之间匹配,代码中 mbsstr 函数用的匹配算法是 KMP 算法。

3.1 KMP 算法

KMP 算法[4]的思想是利用已经匹配的结果将模式串 P 向右移动一定的距离[5]。首先将模式串 P 和文本 T 左端对其进行比较，比较 P_i 和 T_j 时，若相等则继续向右比较；若不等则正文中 j 位置不变，模式中 i 指向 $next[i]$所致的位置，$next[i]$表示第 i 个字符与主串中相应的字符“失配”时，在模式中需重新和主串中的该字符进行比较的字符的位置。这个位置与模式本身有关与正文无关。

KMP 算法的搜索阶段在最坏的情况下的复杂度为 $O(m\times n)$，但是在一般情况下的时间复杂度为 $O(m+n)$，其中 $next[i]$的时间复杂度为 $O(m)$[6]。

3.2 BM 算法

BM(Boyer-Moore) 算法[7]是一种效率较高的算法，在实际应用中采用较多，很多种文本编辑器的“查找”功能(Ctrl+F)采用 Boyer－Moore 算法[8]。BM 算法提出了一个巧妙的方法，从字符串的尾部开始比较来解决匹配问题。它是最早的跳跃型算法，通过两个启发规则来确定扫描窗口的跳跃距离[9]。

首先，模式串与文本串头部对齐，从尾部 P_{m-1} 开始比较。因为如果尾部字符不匹配，那么只要一次比较，就可以判断文本串的前 m 个字符肯定不是要找的结果。当模式串和文本串从右边开始 $P_i\neq T_j$ 开始不匹配时，该不匹配字符就被称为“坏字符”。则对应的移动距离为坏字符出现的位置减去搜索词中上一次出现的位置。如果坏字符不在字符中，则上次搜索的位置为－1。

已经匹配了的字符串称为“好后缀”，好后缀的位置以最后一个字符为准，如果好后缀在搜索词中只出现一次，则它的上一次出现位置为 －1，如果好后缀有多个，则除了最长的那个好后缀，其他好后缀的上一次出现位置必须在头部。每一次完成之后它会根据坏字符移动表和好后 缀移动表来进行位置后移。假设 i 为坏字符，$P[i+1,m-1]$ 为好后缀。

对于坏字符：$distanceChar=\{i-k\mid (P[i]\neq T[j])\ \&\&\ (P[i]=P[k](k<j)\ ||\ k=-1))\}$

对已好后缀则有 $distanceFix=\begin{cases}t,\ P[t,m-1]=P[0,m-t-1],i<t<m\\ \quad\quad =m-1\end{cases}$

移位结果为：$dist[i]=\max(distanceChar,distanceFix)$。

BM 算法使用好后缀移动与坏字符移动计算得到的移动中的最大值来向后移动尝试位置。KMP 算法与 BM 算法在最坏的情况下均具有线性的搜索时间，但是在实际的应用中 BM 算法往往比 KMP 算法快上 3 ～ 5 倍，比较次数只有文本串长度的 20% ～ 30%[10]。

3.3 改进的 BM 算法

3.3.1 改进的 BM 算法的思想

对于 BM 算法都是根据好后缀和坏字符算法，都是从尾部开始比较，但是有一定的局限性。例如从字符串的末尾开始匹配且匹配了后面的字符串的时候发现首字符不匹配，则此次匹配是失败的，除首字符之外的所有匹配都无用。所以一个字符串不仅尾部需要匹配而且还需要匹配字符串的开头部分，具有好后缀的时候还应该具有一个好开头，好开头就是模式串的第一个字符。因此本文提出一种 BM 的改进算法。算法的思想是：将首字符 $P[0]$ 与 $T[j]$ 对齐，末尾字符 $P[m-1]$ 和 $T[j+m]$ 对齐。当末尾符为坏字符即 $P[m-1]\neq T[j+m]$ 时按照普通 BM 算法向右移动 $dist$ 个字符的距离。当末尾字符相同且首字符不同，即 $P[0]\neq T[j]$，此时向右移动一位以减少字符串的回溯比较。当末尾字符和首字符都不相同时，则使用 BM 算法进行 $dist$ 个字符位置移动。对模式串 $P=$“$abcdc$”和文本串 $T=$“ebcdcabfbccdcabcdc”，改进的 BM 的算法的匹配过程如表 1 所示。

表 1 改进的 BM 的算法的匹配过程

顺序	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
文本串	e	b	c	d	c	a	b	f	b	c	c	d	c	a	b	c	d	c
第一趟	a	b	c	d	c													
第二趟		a	b	c	d	c												
第三趟						a	b	c	d	c								
第四趟											a	b	c	d	c			
第五趟														a	b	c	d	c

比较过程：第一趟比较首字符不匹配末尾字符匹配，此时向右移动一位。第二趟末尾字符出现了不匹配的按照 BM 算法向右移动四位。第三趟末尾字符和首字符都匹配，继续向右移动距离为 5。第四趟末尾字符不匹配，则为坏字符向右移动 3 位。第五趟匹配成功结束。

3.3.2　改进的 BM 算法描述

改进的 BM 算法对 BM 算法添加了一个开头的概念,考虑到从末尾开始匹配时匹配了很多字符然而开始字符却没有匹配,就造成了无用匹配。改算法执行的步骤如下描述:

a) 将模式串 P 和文本串 T 对齐,比较末尾字符和首字符即 $P[m-1]$ 和 $T[j+m]$,$P[0]$ 和 $T[j]$。

b) 当 $P[m-1]=T[j+m]$,$P[0]\neq T[j]$ 时,模式串 P 向右移动 1 位。

c) 当 $P[0]=T[j]$ 时,不论 $P[m-1]$ 等于或者不等于 $T[j+m]$,自右向左开始匹配,如果出现坏字符与 $P[i]$ 不匹配,向右移动 $dist[i]$ 个距离。匹配成功则退出。

设 $slide$ 为改进的 BM 算法滑动距离函数,它的单位是一个字符长度,如下所示:

$$slide=\begin{cases}1 & P[m-1]=T[j+m]\&\&P[0]\neq T[j]\\ dist[i] & P[0]=T[j]\&\&P[i]\neq T[j+i]\end{cases}$$

改进的 BM 算法减少了对已知该目标字符串不匹配时的回溯比较次数,从而使总的比较次数减少,以提高匹配效率。例如在表 1 中 BM 算法的字符的比较次数为 14 次,改进后的比较次数为 10 次。第一趟过程中当首字符出现了不匹配,即前 5 个字符已经匹配失效了,但是按照 BM 算法从后向前比较需要比较 5 次,因为后面几个字符都相同。所以这里后面这几个字符的 4 次比较就是多余的。改进后只需要匹配 1 次即可,对于同时比较首字符和末尾字符这样可以根据原来的 BM 算法算出移动距离,减少匹配次数。

4　结　语

改进的 BM 算法综合 3 传统的 BM 算法高效率的优点,解决了只有好后缀没有好开头的情况,减少了匹配次数,提高了匹配效率。由于在 linux 文件查找的过程中文件的绝对路径名称并不是很长,所以比较尾部同时又比较开头,使得 linux 的 locate 命令在文件查找的时候速度更快,提高用户体验。

参考文献:

[1]尹泉. 浅谈嵌入式系统设计及发展趋势[J]. 计算机光盘软件以及应用,2014,17(2):253－254.

[2]毛德操,胡希明. LINUX 内核源代码情景分[M]. 杭州:浙江大学出版社,2001:530－534.

[3]段海梦. Linux 文件系统解析与模拟[J]. 网络安全技术与应用,2014(8): 30-32.

[4]GOSCINSKI A. Two algorithms for mutual exclusion in real- time distributed computer systems[J]. Journal of Parallel and Distributed Computing,1990,9(1):77-82.

[5]严蔚敏,吴伟明. 数据结构:C 语言版[M]. 北京:清华大学出版社,1999:60-81.

[6]闵联营,赵婷婷. BM 算法的研究与改进[J]. 武汉理工大学学报,2006,30(3):529-530.

[7] BOYER R S, MOORE J S. A fast string searching algorithm[J]. Communications of the ACM, 1977, 20(10):762-772.

[8]阮一峰. 字符串匹配的 Boyer-Moore 算法[EB/OL]. (2013-05-03)[2015-07-26]. http://www. ruanyifeng. com/blog/2013/05/boyer-moore _ string _ search _ algorithm. html. 2013-05-03.

[9]韩光辉,曾诚. Boyer-Moore 串匹配算法的改进[J]. 计算机应用,2014,34(3):865-868.

[10]郝爽. BM 算法中好后缀规则的研究[J]. 科技信息,2007(36):539-540.

Command Parsing of File Searching and Query Optimization of Locate Command in Linux

WANG Weiwei, LI Renwang, ZHAO Yaping, WANG Haizhou, SONG Shengtao

(1. Faculty of Mechanical Engineering & Automation, Zhejiang Sci-Tech University, Hangzhou 310018, China; 2. College of life science, Nanjing Normal University, Nanjing 210046, China)

Abstract: Linux system has been widely used recently. Users often use command line to operate the system as well as find the files; especially, when the linux system is used as a server, one often spends much time in finding the file needed in a large number of files. In order to find the file needed quickly, we analyze the implementation principles and process of two existing linux file search commands; besides, after analyzing the famous Boyer Moore - string matching algorithm, we used the improved algorithm of BM to optimize the string matching algorithm in locate command, a kind of search command. Based on the analysis on complexity of the improved algorithm, it has higher efficiency and a faster query speed in the string matching process of filename, compared with the former KMP algorithm.

Key words: Linux file search; find command; locate command; BM algorithm; updatedb command

(责任编辑:陈和榜)