

期末作业：编写一个线程池应用

- 编写一个线程池，实现对海量数据排序
- 输入
 - 给定一个目录，其中包含大量的普通文件
 - 每个文件中又包含了大量的数据
 - 每个数据都是**64**位的有符号数
 - 文件/数据格式可自由设定，数据随机生成
 - 典型设置：
 - 所有文件包含的总的数据量达到**128GB**
 - 每个文件包含的数据个数并不平均，其空间占用量
 - 少的可能仅几十**KB**，多的可能上**GB**
 - 文件个数可以达到**10万个**

期末作业：编写一个线程池应用

■ 输出

- 按照从小到大的顺序，将排好序的数据写入到指定文件中，文件格式自定义

■ 限制

- 用于缓存文件数据、中间排序结果的内存远小于**128GB**，比如只能使用**64MB**缓存

要求

- 建立一个包含若干线程的线程池
- 可以按照归并排序思路，在多线程间完成排序
- 线程间交换数据按照粗粒度锁、免锁、无锁实现方式，从低到高综合评分
- 要尽可能减少数据复制操作

期末作业：编写一个线程池应用

■ 线程间排序的一种参考思路

- 假设有3个相同大小的数据文件需要排序
- 线程池有两个线程
- 线程1和2，分别对文件1和2完成排序
- 然后选择线程1对两个排序结果进行归并
- 与此同时，线程2对文件3进行排序
- 处理完后将结果交由线程1完成最终的归并
- 实现时要考虑各个线程负载均衡的问题

■ 线程间通信加锁问题

- 免锁算法CAS操作，C11标准、gcc自带的
__sync_bool_compare_and_swap函数
- 单读单写的模式，是否需要锁？

作业1

- 任务：编写一个带缓存的文件操作类
- 从执行体程序库中的CLogger类可知，通过缓存要写入文件中的数据，能够提高读写磁盘的性能
- 其要求如下：
 - 需要提供open/read/write/lseek/close等函数的封装函数
 - 该类要提供数据缓存服务。
 - 调用该类的写操作接口时，数据要首先写到缓存，然后再根据策略写到文件中。
 - 调用该类的读操作接口时，该类能根据策略缓存读出的数据

作业2

- 自学stat等函数，获取文件元数据信息
 - 有关说明见后
- 实现“ls -l”的基本功能
- 至少能输出：
 - 文件类型
 - 9个权限位信息
 - 文件大小
 - 文件名称

stat函数

- 用于获取有关文件的信息结构
- 函数原型

- `int stat(const char* restrict pathname, struct stat* restrict buf);`

参数与返回值

- 第一个参数pathname: 文件名，需要获取该文件的信息
 - 第二个参数buf: stat函数将pathname对应的文件信息，填入buf指向的stat结构中
 - 返回值: 0成功; -1出错

stat结构体

```
struct stat {
```

```
.....
```

ino_t	st_ino;	/* inode number*/
mode_t	st_mode;	/* file type & mode */
nlink_t	st_nlink;	/* number of hard links

```
*/
```

uid_t	st_uid;	/* user ID of owner
-------	---------	---------------------

```
*/
```

gid_t	st_gid;	/* group ID of
-------	---------	----------------

```
owner */
```

off_t	st_size;	/* total size, in bytes */
-------	----------	----------------------------

unsigned long	st_blksize;	/* blocksize */
---------------	-------------	-----------------

unsigned long	st_blocks;	/* number of blocks
---------------	------------	---------------------

```
allocated
```

time_t	st_atime;	/* time of last access */
--------	-----------	---------------------------

time_t	st_mtime;	/* time of last
--------	-----------	-----------------

```
modification */
```

time_t	st_ctime;	/* time of inode last
--------	-----------	-----------------------

fstat、lstat函数

- 用于获取有关文件的信息结构
- 函数原型

```
int stat(const char* restrict pathname,  
         struct stat* restrict buf);  
int fstat(int filedes, struct stat *buf);  
int lstat(const char* restrict pathname,  
         struct stat* restrict buf);
```

*lstat*返回符号链接本身的信息
*stat*返回符号链接所引用的文件信息

fstat、lstat函数

- 用于获取有关文件的信息结构
- 函数原型

```
int stat(const char* restrict pathname,  
         struct stat* restrict buf);  
int fstat(int filedes, struct stat *buf);  
int lstat(const char* restrict pathname,  
         struct stat* restrict buf);
```

*Stat*通过文件名返回文件的信息
*fstat*通过文件描述符

stat结构体

```
struct stat {
```

```
.....
```

```
    ino_t      st_ino;          /* inode number*/
```

```
    mode_t     st_mode;        /* file type & mode */
```

```
    nlink_t    st_nlink;       /* number of hard links
```

```
*/
```

```
    uid_t      st_uid;         /* user ID of owner
```

```
*/
```

```
    gid_t      st_gid;         /* group ID of
```

```
owner */
```

```
    off_t      st_size;        /* total size, in bytes
```

```
*/
```

```
    unsigned long st_blksize;  /* preferred block size */
```

```
    unsigned long st_blocks;   /* number of blocks
```

```
allocated
```

```
    time_t      st_atime;       /* time of last access */
```

```
    time_t      st_mtime;       /* time of last
```

```
modification */
```

包含了文件
类型信息

文件类型的判定

- 使用如下的宏，判断文件类型
 - 普通文件 S_ISREG()
 - 目录文件 S_ISDIR()
 - 字符特殊文件 S_ISCHR()
 - 块特殊文件 S_ISBLK()
 - FIFO文件 S_ISFIFO()
 - 套接口文件 S_ISSOCK()
 - 符号连接 S_ISLNK()

文件类型

- 示例代码:

```
struct stat buf;  
lstat( filename, &buf);  
if (S_ISDIR(buf.st_mode))  
    cout << "directory" << endl;
```

- 注意，此处必须用lstat获取文件信息，而不用stat， 为什么？

stat结构体

```
struct stat {
```

```
.....
```

```
    ino_t      st_ino;          /* inode number*/
```

```
    mode_t    st_mode;        /* file type & mode */
```

```
    nlink_t    st_nlink;      /* number of hard links
```

```
*/
```

```
    uid_t      st_uid;        /* user ID of owner
```

```
*/
```

```
    gid_t      st_gid;        /* group ID of
```

```
owner */
```

```
    off_t      st_size;       /* total size, in bytes */
```

```
    unsigned long st_blksize; /* preferred block size */
```

```
    unsigned long st_blocks;  /* number of blocks
```

```
allocated
```

```
    time_t     st_atime;      /* last access */
```

```
    time_t     st_mtime;      /* time of last
```

```
modification */
```

```
    time_t     st_ctime;      /* time of inode last
```

包含了文件
访问权限位

st_mode字段

st_mode的低11bit



st_mode字段

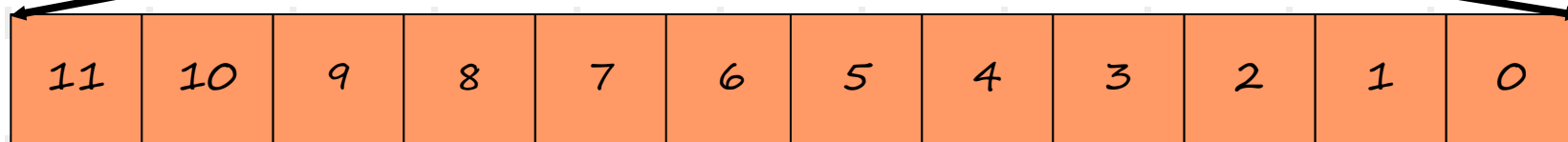
st_mode的低11bit



针对文件所有者的访问权限

st_mode字段

st_mode的低11bit



指示文件所有者是否可执行

指示文件所有者是否可写

指示文件所有者是否可读

st_mode字段

st_mode的低11bit



针对与文件所有者
同组的用户的访问
权限

st_mode字段

st_mode的低11bit



指示组用户是否可执行

指示组用户是否可写

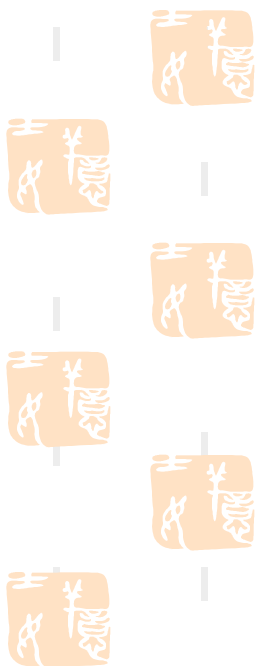
指示组用户是否可读

st_mode字段

st_mode的低11bit

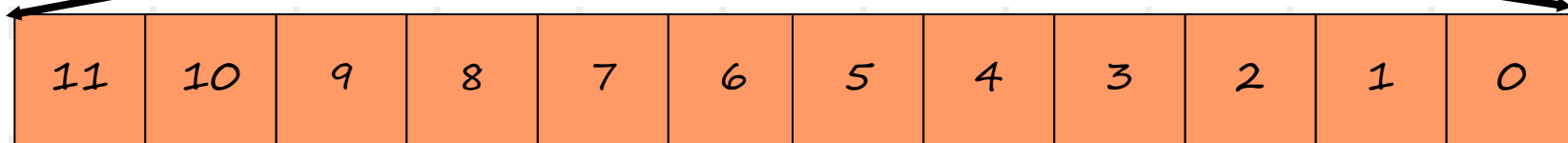


针对其他用户的访问
权限



st_mode字段

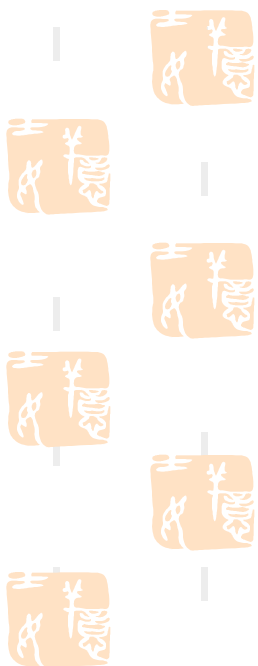
st_mode的低11bit



指示其他用户
是否可读

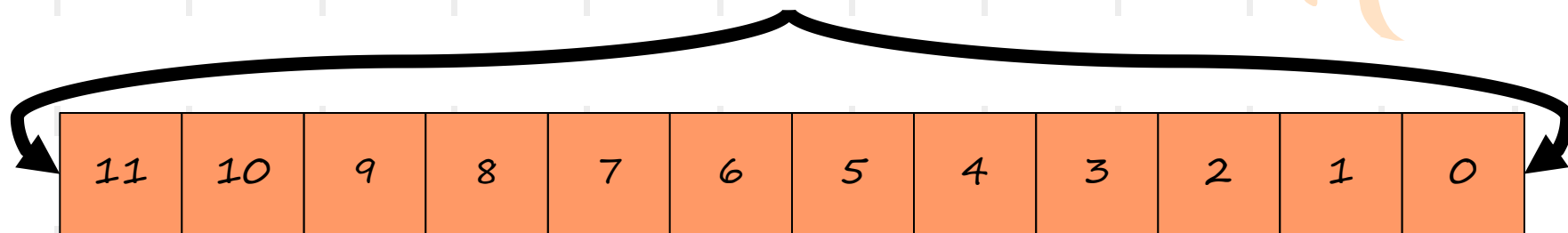
指示其他用户
是否可写

指示其他用户
是否可执行



st_mode字段

st_mode的低11bit



9个文件访问权限位



判定文件访问权限的屏蔽字

<code>st_mode</code> 屏蔽	意义
<code>S_IRUSR</code>	用户—读
<code>S_IWUSR</code>	<pre>if(buf.st_mode & S_IRUSR) { 用户可读 }</pre> <p><code>open</code>的第三个参数 <code>creat</code>的第二个参数</p>
<code>S_IXUSR</code>	
<code>S_IRGRP</code>	
<code>S_IWGRP</code>	
<code>S_IXGRP</code>	组—执行
<code>S_IROTH</code>	其他—读
<code>S_IWOTH</code>	其他—写
<code>S_IXOTH</code>	其他—执行

读目录的基本操作

- 打开目录 (opendir)
- 逐一读出目录项 (readdir、rewinddir)



关闭目录 (closedir)



opendir函数

- 用于打开目录

- 函数原型:

- `DIR* opendir(const char* pathname);`

- 返回值和参数

- 返回值: 返回打开目录的索引结构, 出错返回 NULL

- `pathname`: 要打开的目录名

readdir函数

- 用于读取目录项

- 函数原型:

- `struct dirent *readdir(DIR *dp);`

- 参数与返回值

- `dp`: 由`opendir`返回的

- 返回值: `dp`对应的目录中包含的一个目录项

readdir函数

- dirent结构

- struct dirent {

- ino_t d_ino; //索引节点号

- char d_name[NAME_MAX + 1]; //文件名

-

- }

rewinddir函数

- 用来设置目录流目前的读取位置为原来开头的读取位置
- 函数原型

- `void rewinddir(DIR *dp);`

参数

- `dp`: 由`opendir`返回

closedir函数

- 用于关闭目录

- 函数原型:


- `int closedir(DIR *dp);`

- 参数与返回值

- `dp`: 由`opendir`返回

- 返回值: 成功返回0, 出错返回-1

获得目录下的所有文件

- `DIR *dir;`
 - `struct dirent *ptr;`
 - `dir=opendir("/etc/rc.d");`
 - `while((ptr=readdir(dir))!=NULL)`
- 
- `{`
 - `printf("d_name: %s\n", ptr->d_name);`



- `}`

