1. 本文概要

一个文件，如果需要拷贝，使用一个进程在单位时间内由调度器分配得到的时间片应该会少于两个进程。但是一般情况下，开启一个进程消耗的资源要多于开启一个线程。所以使用线程拷贝数据很划算。那么多线程拷贝一个文件，linux初学者应该怎样去尝试实现呢？

拷贝数据涉及三个要素：源 - 目标 - 长度

1. 涉及知识
2. 示例展示（含注释，编译gcc filename -lpthread -lm）

/\*

\* 拷贝文件的教学案例

\* 涉及知识：

\* @函数封装

\* @链式队列

\* @线程同步之屏障

\* 注意：

\* 该cp功能并不完善，

\* 仅仅实现一个文件的拷贝，而且需要指定名称,作者眼花手拙，如有错误请您斧正！

\* \*/

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <errno.h>

#include <math.h>

#include <pthread.h>

//支持的最大线程开启数量

#define PRC\_NUM 16

//拷贝文件时，需要的文件信息

typedef struct copy\_info{

char\* src;/\*源文件路径\*/

char\* dts;/\*目标文件路径\*/

off\_t len;/\*源文件文件长度（lseek测得长度 - 1）\*/

off\_t start;/\*拷贝的起始位置\*/

off\_t end;/\*拷贝的终止位置\*/

int thread\_num;/\*当前拷贝文件使用的线程数量\*/

void\* data;/\*自定义数据\*/

struct copy\_info\* next;

}linknode\_t,\*linklistp\_t;

linklistp\_t create\_linklist\_head(const char\* src, const char\* dts, off\_t len, int thread\_num);

int add\_new\_cpinfo\_node(linklistp\_t hp, off\_t start, off\_t end, int thread\_num);

int display\_node\_info(linklistp\_t p);

/\* 向队列中添加拷贝文件的信息结构体

\* 参数：

\* hp: 队列的头节点指针

\* start:拷贝数据在源文件和目标文件中的开始位置

\* end:拷贝文件在源文件和目标文件中的结束位置

\* thread\_num：当前拷贝文件使用的线程数量。

\* \*/

int add\_new\_cpinfo\_node(linklistp\_t hp, off\_t start, off\_t end, int thread\_num){

linklistp\_t p = NULL;

if(NULL == (p = malloc(sizeof(linknode\_t)))){

perror("malloc failed");

return -1;

}

p->next = hp->next;

hp->next = p;

p->src = hp->src;

p->dts = hp->dts;

p->len = hp->len;

p->start = start;

p->end = end;

p->thread\_num = thread\_num;

return 0;

}

/\*

\* 创建队列头，填充源文件/目标文件的信息以及处理的线程数量

\*

\*/

linklistp\_t create\_linklist\_head(const char\* src, const char\* dts, off\_t len, int thread\_num){

linklistp\_t hp = NULL;

if(NULL == (hp = malloc(sizeof(linknode\_t)))){

perror("malloc failed");

return NULL;

}

hp->next = NULL;

hp->src = (char\*)src;

hp->dts = (char\*)dts;

hp->len = len;

hp->thread\_num = thread\_num;

return hp;

}

//释放整个链表

void free\_linklist(linklistp\_t hp){

linklistp\_t p = hp->next;

while(p){

hp->next = p->next;

free(p);

p = hp->next;

}

free(hp);

hp = NULL;

}

int display\_node\_info(linklistp\_t p){

printf("###\n");

printf("p->thread\_num:%d, p->data:%p\n",p->thread\_num, p->data);

printf("p:%p, p->next:%p\n",p, p->next);

printf("p->src:%s, p->dts:%s, p->start:%ld, p->end:%ld, p->len:%ld\n",p->src, p->dts, p->start, p->end, p->len);

return 0;

}

/\*

\* 线程处理函数，每个线程处理一个节点的数据

\* \*/

void\* pthread\_handler(void\* args){

linklistp\_t p = args;

display\_node\_info(p);

int ret = 0;

int fd\_src = -1, fd\_dts = -1;

char ch = '\0';

if(0 > (fd\_src = open(p->src, O\_RDWR))){

perror("open failed");

goto pthread\_err0;

}

if(0 > (fd\_dts = open(p->dts, O\_RDWR))){

perror("open failed ");

goto pthread\_err1;

}

if(0 > (ret = lseek(fd\_src, p->start, SEEK\_SET))){

perror("lseek failed");

goto pthread\_err2;

}

if(0 > (ret = lseek(fd\_dts, p->start, SEEK\_SET))){

perror("lseek failed");

goto pthread\_err2;

}

while(p->end >= lseek(fd\_src, 0, SEEK\_CUR)){

if(0 > read(fd\_src, &ch, 1)){

perror("read failed");

goto pthread\_err3;

}

if(0 > write(fd\_dts, &ch, 1)){

perror("write failed");

goto pthread\_err3;

}

}

close(fd\_dts);

close(fd\_src);

return args;

pthread\_err3:

pthread\_err2:

close(fd\_dts);

pthread\_err1:

close(fd\_src);

pthread\_err0:

return NULL;

}

int main(int argc, const char \*argv[])

{

int fd\_src = -1;

int fd\_dts = -1;

off\_t offn = -1;

off\_t len\_temp = -1, start\_temp = -1, end\_temp = -1;

int ret = -1;

int thread\_num = 0, i = 0;

char dts[32] = {0};

pthread\_t thread[PRC\_NUM] = {0};

if(argc < 3){

printf("Wrong format!\n eg: ./a.out + src\_filename + dts\_filename\n");

exit(0);

}

//打开源文件

if(0 > (fd\_src = open(argv[1],O\_RDWR))){

perror("fd\_src open failed");

goto main\_err0;

}

//创建新文件

if(0 > (fd\_dts = creat(argv[2],S\_IRWXU))){

if(EEXIST == errno){

puts("same name");

goto main\_err1;

}

perror("can't create new dts\_filename\n");

goto main\_err1;

}

//得到新文件的长度，

//注意要减1，因为lseek要偏移到文章的结尾EOF处，所以文章的实际长度要比lseek测得的小一个字符

if(0 > (offn = lseek(fd\_src, 0, SEEK\_END) - 1)){

perror("src lseek failed");

goto main\_err2;

}

printf("offn:%ld\n",offn);

if(0 > lseek(fd\_src, 0, SEEK\_SET)){

perror("src lseek failed");

goto main\_err2;

}

//根据文件的长度来推算需要几个处理的线程, 0 < text < 2^10用1个线程，2^10 < text < 2^20用2个线程，小于2^30用3个线程，

thread\_num = (int)((log(offn)/log(2)))/10 + 1;

printf("thread\_num:%d\n",thread\_num);

//最大线程处理数量的限定，此处没有实际作用

if(thread\_num > PRC\_NUM)thread\_num = PRC\_NUM;

//创建队列头，并填充必要信息

linklistp\_t hp = NULL;

if(NULL == (hp = create\_linklist\_head(argv[1], argv[2], offn, thread\_num))){

perror("create\_linklist\_head failed");

goto main\_err3;

}

printf("debug:%s, %d\n",\_\_func\_\_,\_\_LINE\_\_);

start\_temp = 0;

end\_temp = 0;

len\_temp = offn/thread\_num;

for(i=0;i<thread\_num;i++){

end\_temp = start\_temp + len\_temp;

if(thread\_num - 1 == i){

end\_temp = offn;

}

if(0 > add\_new\_cpinfo\_node(hp, start\_temp, end\_temp, thread\_num)){

perror("add\_new\_cpinfo\_node failed");

goto main\_err3;

}

start\_temp = end\_temp;

}

//根据前面推算数来的线程数量来创建线程，并且每次都把新节点的信息传递进去

linklistp\_t p = NULL;

p = hp->next;

for(i = 0;i < thread\_num;i++){

if(0 > (ret = pthread\_create(&thread[i], NULL, pthread\_handler, (void\*)p))){

perror("pthread\_create failed");

goto main\_err4;

}

if(i < (thread\_num - 1)){

p = p->next;

}

}

//此处应该等待子线程结束，如果主线程立即退出会导致整个进程推出，所以用pthread\_join阻塞等待所有进程退出

for(i=0; i<thread\_num; i++){

pthread\_join(thread[i], NULL);

}

//手动释放malloc申请的空间

free\_linklist(hp);

return 0;

main\_err4:

//rm dts file

//pthread\_join

main\_err3:

main\_err2:

close(fd\_dts);

main\_err1:

close(fd\_src);

main\_err0:

return -1;

}