

**操作系统实验报告**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **姓名** | **胡聿鑫** | **专业班级** | | **计科3班** | | **指导老师** | **王小妹** |
| **学号** | **2220192813** | | **实验日期** | | **2021.11.26** | | |
| **实验**  **名称** | **文件系统扩展实验** | | | | | | |
| **实验**  **目的** | * 掌握Linux系统下文件管理的基本原理和实现方式 * 掌握文件和目录操作之间系统调用的用户接口 * 能够通过系统调用输出指定目录下文件的大小和修改时间 * 能够用BFS和DFS的方法遍历目录并输出有关信息 | | | | | | |
| **实**  **验**  **内**  **容** | * 在运行程序（.out文件）后输入目录的参数名，用DFS和BFS的方法遍历该目录，并输出目录下所有文件的大小和修改时间，若目录下还有子目录，也要将子目录添加到路径后再输出该子目录下所有文件信息。实验的基本思路为：首先对输入参数进行判断，确保已输入文件名；然后判断文件类型，若stat调用出错，则输出错误信息，若是普通文件则输出其大小和最后修改时间，若是目录文件则递归读取目录下的文件，直至出错或者全部读取完毕。 * DFS和BFS的区别在于：DFS会先遍历所有层级的目录，将最高层目录下的文件优先输出，而BFS则是从低到高逐层目录遍历，并输出其下输出文件。 * 运用DFS方式时，实验流程图如下：      * 运用BFS方式时，实验流程图如下： | | | | | | |
| **程**  **序**  **5\_9**  **(深度)** | #include <stdio.h>  #include <stdlib.h>  #include <sys/stat.h>  #include <sys/types.h>  #include <fcntl.h>  #include <errno.h>  #include <unistd.h>  #include <time.h>  #include <ctype.h>  #include <dirent.h>  #include <string.h>  static int get\_file\_size\_time(const char\* s)  {  struct stat statbuf;  *struct dirent\* direntp;*  *char dirt[1024];*  *char temp[1024];*  *DIR\* p;*  *strcpy(temp,s);*  if(stat(temp, &statbuf)==-1)  {  printf("Get stat on %s Error %s\n", temp, strerror(errno));  return(-1);  }  if(S\_ISDIR(statbuf.st\_mode))  {  *if((p=opendir(temp))==NULL)*  *{*  *printf("Open Directory %s Error: %s\n", temp, strerror(errno));*  exit(1);  }  *char iterbuf[512];*  *while((direntp = readdir(p))!=NULL)*  *{*  *if (strcmp(direntp->d\_name, "..") == 0 ||*  *strcmp(direntp->d\_name, ".") == 0)*  *continue;*  *memset(iterbuf,0,sizeof(iterbuf));*  *strcpy(iterbuf,temp);*  *strcat(iterbuf,"/");*  *strcat(iterbuf,direntp->d\_name);*  *memset(dirt,0,sizeof(dirt));*  *strcpy(dirt,iterbuf);*  *if (get\_file\_size\_time(dirt) == -1)*  *break;*  *}*  return(1);  }  if(S\_ISREG(statbuf.st\_mode))  {  printf("%s size:%ldbtytes\tmodified at%s", temp, statbuf.st\_size,ctime(&statbuf.st\_mtime));  return(0);  }  }  int main(int argc, char\*\* argv)  {  DIR\* dirp;  struct dirent\* direntp;  *struct stat statbuf;*  if(argc!=2)  {  printf("Usage:%s filename\n\a",argv[0]);  exit(1);  }  *if(stat(argv[1],&statbuf)==-1)*  *{*  *printf("Get stat on %s Error %s\n", argv[1], strerror(errno));*  *}*  *if(S\_ISREG(statbuf.st\_mode))*  *printf("%ssize:%ldbtytes\tmodified at%s",argv[1], statbuf.st\_size, ctime(&statbuf.st\_mtime));*  *if((dirp=opendir(argv[1]))==NULL)*  *{*  *printf("Open Directory %s Error:%s\n", argv[1], strerror(errno));*  *exit(1);*  *}*  *get\_file\_size\_time(argv[1]);*  closedir(dirp);  } | | | | | | |
| **程**  **序**  **5\_9**  **(广度)** | #include <unistd.h>  #include <stdio.h>  #include <stdlib.h>  #include <errno.h>  #include <sys/types.h>  #include <sys/stat.h>  #include <dirent.h>  #include <time.h>  #include <string.h>  #include <ctype.h>  #include"malloc.h"  *typedef struct node*  *{//队列数据结构定义*  *char data[1024];//数据域*  *struct node \*next;//指针域*  *}QN;*  *typedef struct*  *{*  *QN \*front;//队首*  *QN \*rear;//队尾*  *}Queue,\*QS;*  *void InitQueue(QS Q)*  *{//初始化*  *Q->front = Q->rear = NULL;*  *}*  *int IsEmpty(QS Q)*  *{//判断队列是否为空*  *return(Q->front == NULL && Q->rear == NULL);*  *}*  *void EnQueue(QS Q, char \*d)*  *{//入队*  *QN \*p;*  *p = (QN \*)malloc(sizeof(QN));*  *strcpy(p->data , d);*  *p->next = NULL;*  *if(IsEmpty(Q)) Q->front = Q->rear = p;*  *else*  *{*  *Q->rear->next = p;*  *Q->rear = p;*  *}*  *}*  *void DeQueue(QS Q,char \*n)*  *{//出队*  *QN \*s;*  *if(IsEmpty(Q))*  *{*  *printf("Error,the Queue is empty!");*  *return;*  *}*  *s = Q->front;*  *strcpy(n,s->data);*  *if(Q->front == Q->rear)*  *Q->front = Q->rear = NULL;*  *else*  *Q->front = s->next;*  *free(s);*  *}*  static int get\_file\_size\_time(const char \*filename)  {  struct stat statbuf;  if(stat(filename,&statbuf) == -1)  {  printf("Get start on %s Error:%s\n",filename,strerror(errno));  return(-1);  }  if(S\_ISDIR(statbuf.st\_mode))//判断是否是目录  {  return(1);  }  if(S\_ISREG(statbuf.st\_mode))  printf("%s size:%ld bytes\tmodified at %s",filename,statbuf.st\_size,ctime(&statbuf.st\_mtime));  return(0);//若是普通文件，则输出文件路径，大小和修改时间  }  *int traversal(QS Q,char \*argv)//遍历队列中的目录*  *{*  *DIR \*dirp;*  *struct dirent \*direntp;*  *int stats;*  *char Path[1024];*  *char temp[1024];*  *if(((stats = get\_file\_size\_time(argv))==0)||(stats==-1))*  *exit(1);*  *if((dirp=opendir(argv))==NULL)*  *{*  *printf("Open Directory %s Error:%s\n",argv,strerror(errno));*  *exit(1);*  *}*  *while((direntp = readdir(dirp))!=NULL)*  *{//遍历当前目录下所有文件*  *strcpy(Path,argv);*  *strcat(Path,"/");*  *strcat(Path,direntp->d\_name);*  *if(get\_file\_size\_time(Path)==-1)*  *break;*  *if(get\_file\_size\_time(Path)==1&&Path[strlen(Path)-1]!='.')*  *{*  *EnQueue(Q,Path);*  *}//目录下还有目录，则添加到输出路径*  *}*  *if(IsEmpty(Q) == 0)*  *{//继续向下一级目录遍历*  *char all[1024];*  *memset(all,0,1024);*  *DeQueue(Q,temp);*  *strcpy(all,temp);*  *traversal(Q,all);//递归调用，遍历目录下所有文件*  *}*  *closedir(dirp);*  *return 0;*  *}*  int main(int argc,char \*\*argv)  {  DIR \*dirp;  struct dirent \*direntp;  int stats;  *char path[1024];*  *char temp[1024];*  *Queue qst;*  *QS Q=&qst;*  *InitQueue(Q);*  if(argc!=2)//判断输入是否为两个参数  {  printf("Usage:%s filename\n\a",argv[0]);  exit(1);  }  if(((stats=get\_file\_size\_time(argv[1]))==0)||(stats==-1))  exit(1);//若 argv[1]为文件（输出文件大小和最后修改时间）或者 stats 错误，都退出  if((dirp=opendir(argv[1]))==NULL)  {  printf("Open Directory %s Error:%s\n",argv[1],strerror(errno));  exit(1);  }  while((direntp = readdir(dirp))!=NULL)  {  memset(path,0,1024);  strcpy(path,argv[1]);  strcat(path,"/");  strcat(path,direntp->d\_name);  if(get\_file\_size\_time(path)==-1)//读取 dirp 目录下文件，直到出错或结束，退出  break;  *if(get\_file\_size\_time(path)==1&&path[strlen(path)-1]!='.')*  *EnQueue(Q,path);*  }  *if(!IsEmpty(Q))*  *{*  *char all[1024];*  *memset(all,0,1024);*  *DeQueue(Q,temp);*  *strcpy(all,temp);*  *traversal(Q,all);*  *}*  closedir(dirp);  exit(1);  } | | | | | | |
| **运**  **行**  **结**  **果** | 深度遍历运行结果：    广度遍历运行结果：    目录文件分布情况如下： | | | | | | |
| **实**  **验**  **总**  **结** | 1. 最开始实验时，并没有头绪，不知道该如何修改，因此在网上查阅了有关资料，并画出了两种方式的流程图，才开始逐步地修改代码。**DFS**的关键点在于**递归调用**部分的代码，在原程序基础上，先遍历所有的目录直到没有子目录时，再输出当前目录下的文件，然后**指针回溯**指向上一级目录，输出该目录下的文件，循环执行直到指针指向主目录，输出结束。**BFS**的关键点在于充分利用**队列先进先出**的特点，先遍历**最先入队的目录**，输出其下所有的文件信息，然后再让**该目录下的子目录**入队，输出子目录下的文件信息，循环执行上述步骤直到结束。 2. 深度遍历在访问目录时，得到的结果和预期不一致：若主目录下没有其他文件，则会先访问子目录再继续访问子目录中的子目录直到遇到文件才会输出相关信息，但是若目录下除了子目录外还有其他的文件，则会先将这些文件的信息输出，然后才会继续遍历子目录。 3. 广度遍历访问目录时，直接在主函数中对目录调用出入队函数时运行失败：对此我反复进行改进，发现除了要编写基本的队列初始化和入队、出队函数外，还要有一个用于遍历队列中目录的函数（即程序中的traversal函数），该函数可遍历输出当前目录下的所有普通文件后才会访问下一层目录。 | | | | | | |