

CS232L Operating Systems Lab

Assignment 04 : A simply file system

Name: Muhammad Munawwar Anwar
ID: ma04289

December 8, 2020

1 filesystem.c

1.1 my_list.c

```
1 #include <stdio.h>
2 #include <string.h>
3 #include <unistd.h>
4 #include <fcntl.h>
5 #include <stdlib.h>
6 #include <stdbool.h>
7 /*
8  *
9  * 0 | 1 | 2 | 3 | ..... | 127 |
10  * -----
11  *      <----- data blocks ----->
12  *
13  *
14  *
15  *
16  *
17  *
18  *
19  *
20  *
21  *
22  *
23  *
24  *
25  *
26  *
27  *
28  *
29  *
30  *
31  *
32  *
33  * <----- super block ----->
34  *
35  *
36  * free
37  * block
38  * list | inode0 | inode1 | .... | inode15 |
39  * -----
40  *
41  *
42  */
43
44 #define FILENAME_MAXLEN 8 // including the NULL char
45 #define BLOCK_SIZE 1024
46 #define MAX_BLOCK 128
```

```

48 int myfs;
49
50 /*
51  * inode
52  */
53
54 typedef struct inode {
55     int dir; // boolean value. 1 if it's a directory.
56     char name[FILENAME_MAXLEN];
57     int size; // actual file/directory size in bytes.
58     int blockptrs [8]; // direct pointers to blocks containing file's content.
59     int used; // boolean value. 1 if the entry is in use.
60     int rsvd; // reserved for future use
61 } inode;
62
63
64 /*
65  * directory entry
66  */
67
68 typedef struct dirent {
69     char name[FILENAME_MAXLEN];
70     int namelen; // length of entry name
71     int inode; // this entry inode index
72 } dirent;
73
74 /*
75  * Data Block
76  */
77
78 typedef struct DataBlock {
79     char Data[1024];
80 } block;
81
82 /*
83  * Directory Block
84  */
85
86 typedef struct DirectoryBlock{
87     struct dirent DirectoryTable[17];
88     // One for Parent Directory 0th Position —> Empty in case of Root Directory
89     // One for Current Directory 1th Position
90     // 15 Directories at Max (2-16)th Position
91 } DirectoryBlock;
92
93
94
95 /*
96  * functions
97  */
98 // create file
99 // copy file
100 // remove/delete file
101 // move a file
102 // list file info
103 // create directory
104 // remove a directory
105
106
107 /*
108  * Initialize File System
109  *
110  */
111
112 int initiliaze()
113 {
114     myfs = open("myfs", O_CREAT | ORDWR, 0222);
115     bool FreeBlockList[128]; // Free Block List
116     struct inode InodeTable [16]; // Inode Table

```

```

117 char Data [BLOCK_SIZE];
118 int i = 0;
119
120 for (i = 0; i<128;i++) // Free Block Initialize
121 {
122     FreeBlockList[i] = false;
123 }
124 for (i = 0; i<16;i++) // Inode Table Initialize
125 {
126
127     strcpy(InodeTable[i].name, "");
128     InodeTable[i].used = 0;
129     InodeTable[i].rsvd = 0;
130     InodeTable[i].size = 0;
131     InodeTable[i].dir = 0;
132     for (int j = 0; j < 8;j++){
133         InodeTable[i].blockptrs[j] = -1;
134     }
135 }
136
137 // Root Inode Initialize
138 FreeBlockList[0] = true;
139 FreeBlockList[1] = true;
140 strcpy(InodeTable[0].name, "/");
141 InodeTable[0].size = sizeof(DirectoryBlock);
142 InodeTable[0].dir = 1;
143 InodeTable[0].used = 1;
144 InodeTable[0].blockptrs[0] = 1;
145
146 // Root Directory Block Initialize
147 struct DirectoryBlock RootBlock;
148 strcpy(RootBlock.DirectoryTable[0].name, "NA");
149 RootBlock.DirectoryTable[0].namelen = 2;
150 RootBlock.DirectoryTable[0].inode = -2;
151 strcpy(RootBlock.DirectoryTable[1].name, ".");
152 RootBlock.DirectoryTable[1].namelen = 1;
153 RootBlock.DirectoryTable[1].inode = 0;
154 for (int i = 2; i < 17;i++)
155 {
156     RootBlock.DirectoryTable[i].inode = -1;
157     RootBlock.DirectoryTable[i].namelen = 0;
158     strcpy(RootBlock.DirectoryTable[i].name, "");
159 }
160
161
162 write(myfs, (char*)&FreeBlockList, 128);
163 write(myfs, (char*)&InodeTable, 16*56);
164 write(myfs, (char*)&RootBlock, BLOCK_SIZE);
165
166 // Initialize the Data Region
167 for (i = 1; i < 127; i++){
168     write(myfs, (char*)&Data, BLOCK_SIZE);
169 }
170 return myfs;
171 }
172
173 int getFreeInode(int flag, char *name)
174 {
175     struct inode ReadTable[16];
176     int i = 1;
177     lseek(myfs, 128, SEEK_SET);
178     read(myfs, (char*)&ReadTable, 16*56);
179
180     if (flag == 1)
181     {
182         for (i = 1; i < 16;i++)
183         {
184             if (strcmp(ReadTable[i].name, name) == 0)
185                 {

```

```

186         return -1;
187     }
188 }
189 }
190 for (i = 1; i < 16; i++)
191 {
192     if (ReadTable[i].used == 0)
193     {
194         return i;
195     }
196 }
197
198 return -2;
199 }
200 // Scan the inode table. Return the corresponding inode. If inode is not found,
    then return -1
201 int getInode(char*name){
202     struct inode ReadTable[16];
203     lseek(myfs,128,SEEK_SET);
204     read(myfs,(char*)&ReadTable,16*56);
205     int i = 0;
206     for (i = 0 ; i < 16; i++ )
207     {
208         if (strcmp(ReadTable[i].name,name) == 0 && ReadTable[i].used == 1){
209             return i;
210         }
211     }
212     return -1;
213 }
214 // Scan the bit map and return the first free block
215 int getFreeBlock()
216 {
217     lseek(myfs,0,SEEK.SET);
218     bool ReadFreeBlockList[128];
219     read(myfs,(char*)&ReadFreeBlockList,128);
220     int i = 0;
221     for (i = 0 ; i < 128; i++)
222     {
223         if (ReadFreeBlockList[i] == false)
224         {
225             return i;
226         }
227     }
228     return -1;
229 }
230
231 // CREATE A FILE
232
233 /*
234 Start from the Root Directory
235 Traverse Path while keeping directory inode.
236 Check if the directory in the file path exists
237 Else return -1
238 If inode available
239 Check if the file does not exist
240 Write in directory
241 Create directory entry
242
243 */
244 int CR(char* filename, int size)
245 {
246     if (size > 1024 * 8)
247     {
248         printf("error: File size exceeding maximum limit .\n");
249         return -1;
250     }
251     if (strcmp(filename,"/") == 0 )
252     {
253         printf("error: Root directory is an invalid filename .\n");

```

```

254     return -1;
255 }
256 char * token = strtok(filename, "/");
257 char * ChildDirectory = token;
258 char * ParentDirectory = "/";
259
260 // Get Parent Inode & Parent Directory
261 int address = getInode(ParentDirectory);
262 struct inode ParentInode;
263 struct DirectoryBlock ParentDirectoryBlock;
264
265 int i = 0;
266 lseek(myfs, (128 + (address * 56)), SEEK_SET);
267 read(myfs, (char*)&ParentInode, 56);
268 int value = ParentInode.blockptrs[0];
269 lseek(myfs, value * 1024, SEEK_SET);
270 read(myfs, (char*)&ParentDirectoryBlock, sizeof(DirectoryBlock));
271 while(token != NULL)
272 {
273     token = strtok(NULL, "/");
274     if (token != NULL)
275     {
276         // Scan the child inode's entry in the Parent's directory table. If not
277         // found then return -1.
278         // If found then the child is the new parent.
279         int flag = 0;
280         for (i = 2; i < 17; i++)
281         {
282             if(strcmp(ParentDirectoryBlock.DirectoryTable[i].name, ChildDirectory) ==
283             0){
284                 address = ParentDirectoryBlock.DirectoryTable[i].inode;
285                 flag = 1;
286                 break;
287             }
288         }
289         if (flag == 0)
290         {
291             printf("error: The %s directory in the given path does not exist \n",
292             ChildDirectory);
293             return -1;
294         }
295         // Get the new parent's inode and directory
296         lseek(myfs, (128 + (address * 56)), SEEK_SET);
297         read(myfs, (char*)&ParentInode, 56);
298         value = ParentInode.blockptrs[0];
299         lseek(myfs, value * 1024, SEEK_SET);
300         read(myfs, (char*)&ParentDirectoryBlock, sizeof(DirectoryBlock));
301         ParentDirectory = ChildDirectory;
302         ChildDirectory = token;
303     }
304 }
305 // Check if free inodes are available
306 int freeInode = getFreeInode(0, ChildDirectory);
307 if (freeInode == -2)
308 {
309     printf("error: All inodes are occupied\n");
310     return -1;
311 }
312 // Check if the file exists already
313 for(int i = 2; i < 17; i++)
314 {
315     if (ParentDirectoryBlock.DirectoryTable[i].inode != -1 && strcmp(
316     ParentDirectoryBlock.DirectoryTable[i].name, ChildDirectory) == 0 ){
317
318         printf("error: The file %s already exists .\n", ChildDirectory);
319         return -1;
320     }
321 }

```

```

319 }
320 // Get Free Data Blocks & Fill them according to the size specified
321 struct inode ChildNode;
322 struct DataBlock ChildData[8];
323 int DataPtr[8];
324 for (int i = 0 ; i < 8 ; i++)
325 {
326     DataPtr[i] = -1;
327 }
328 char Buffer[1024];
329 char NewBuffer[1024];
330 int fullblocks;
331 int partialblock;
332 fullblocks = (size)/1024;
333 partialblock = (size%1024);
334 if (size > 1024)
335 {
336     if(partialblock != 0)
337     {
338         fullblocks++;
339     }
340 }
341 int j = 0 ;
342 bool lflag = true;
343 for (i = 0 ; i < fullblocks; i++)
344 {
345     DataPtr[i] = getFreeBlock();
346     lseek(myfs,DataPtr[i],SEEK.SET);
347     write(myfs,(char*)&lflag,1);
348     for (j = 0 ; j < 1024; j++)
349     {
350         Buffer[j] = (char) (97 + j % 26);
351     }
352     strcpy(ChildData[i].Data, Buffer);
353 }
354
355 if (partialblock>0)
356 {
357     DataPtr[fullblocks] = getFreeBlock();
358     lseek(myfs,DataPtr[fullblocks],SEEK.SET);
359     write(myfs,(char*)&lflag,1);
360     for (i = 0; i < partialblock; i++)
361     {
362         NewBuffer[i] = (char) (97 + i % 26);
363     }
364     strcpy(ChildData[fullblocks].Data, NewBuffer);
365 }
366 // Create File's entry in the inode table
367 lseek(myfs,128 + (56*freeInode),SEEK.SET);
368 read(myfs,(char*)&ChildNode,sizeof(inode));
369 ChildNode.dir = 0;
370 strcpy(ChildNode.name,ChildDirectory);
371 ChildNode.size = size;
372 ChildNode.used = 1;
373 ChildNode.rsvd = 0;
374 // Update's parents directory
375 for (int i = 0 ; i < 8; i++)
376 {
377     ChildNode.blockptrs[i] = DataPtr[i];
378 }
379 for ( i = 2; i < 17; i++)
380 {
381     if (ParentDirectoryBlock.DirectoryTable[i].inode == -1)
382     {
383         ParentDirectoryBlock.DirectoryTable[i].inode = freeInode;
384         ParentDirectoryBlock.DirectoryTable[i].namelen = strlen(ChildDirectory);
385         strcpy(ParentDirectoryBlock.DirectoryTable[i].name,ChildDirectory);
386         break;
387     }

```

```

388 }
389 lseek(myfs,128 + (56*freeInode),SEEK_SET);
390 write(myfs,(char*)&ChildNode,sizeof(inode));
391 lseek(myfs,value*1024,SEEK_SET);
392 write(myfs,(char*)&ParentDirectoryBlock,sizeof(DirectoryBlock));
393 for (i = 0 ; i < 8;i++)
394 {
395     if (ChildNode.blockptrs[i] != -1)
396     {
397         lseek(myfs,1024 * ChildNode.blockptrs[i],SEEK_SET);
398         write(myfs,(char*)&ChildData[i],1024);
399     }
400 }
401 return 0;
402 }
403
404 // DELETE A FILE
405
406 /*
407  Start from the Root Directory
408  Traverse Path while keeping directory inode.
409  Check if the directory in the file path exists
410  Else return -1
411  Check if the file does exist
412  Delete directory entry
413 */
414 int DL(char* filename)
415 {
416
417     if (strcmp(filename,"/") == 0 )
418     {
419         printf("error: Cannot delete root directory \n");
420         return -1;
421     }
422     char * token = strtok(filename,"/");
423     char * ChildDirectory = token;
424     char * ParentDirectory = "/";
425     // Get Parent Inode & Parent Directory
426     int address = getInode(ParentDirectory);
427     struct inode ParentInode;
428     struct DirectoryBlock ParentDirectoryBlock;
429
430     int i = 0;
431
432     lseek(myfs,(128+(address*56)),SEEK_SET);
433     read(myfs,(char*)&ParentInode,56);
434     int value = ParentInode.blockptrs[0];
435     lseek(myfs,value*1024,SEEK_SET);
436     read(myfs,(char*)&ParentDirectoryBlock,sizeof(DirectoryBlock));
437     while(token!=NULL)
438     {
439         token = strtok(NULL,"/");
440         if (token !=NULL)
441         {
442             // Scan the child inode's entry in the Parent's directory table. If not
443             // found then return -1.
444             // If found then the child is the new parent.
445             int flag = 0;
446             for (i = 2; i < 17;i++)
447             {
448                 if(strcmp(ParentDirectoryBlock.DirectoryTable[i].name,ChildDirectory) ==
449                 0){
450                     address = ParentDirectoryBlock.DirectoryTable[i].inode;
451                     flag = 1;
452                     break;
453                 }
454             }
455             if (flag == 0)
456             {
457                 printf("error: The directory %s in the given path does not exist \n",

```

```

ChildDirectory);
455     return -1;
456 }
457 // Get the new parent's inode and directory
458 lseek(myfs,(128+(address*56)),SEEK_SET);
459 read(myfs,(char*)&ParentInode,56);
460 value = ParentInode.blockptrs[0];
461 lseek(myfs,value*1024,SEEK_SET);
462 read(myfs,(char*)&ParentDirectoryBlock,sizeof(DirectoryBlock));
463 ParentDirectory = ChildDirectory;
464 ChildDirectory = token;
465 }
466 } // Check if the file exists already
467 int DeleteInode = -1;
468 for (i = 2; i < 17;i++)
469 {
470     if(strcmp(ParentDirectoryBlock.DirectoryTable[i].name,ChildDirectory) == 0)
471     {
472         DeleteInode = ParentDirectoryBlock.DirectoryTable[i].inode;
473         ParentDirectoryBlock.DirectoryTable[i].inode = -1;
474         ParentDirectoryBlock.DirectoryTable[i].namelen = 0;
475         strcpy(ParentDirectoryBlock.DirectoryTable[i].name,"");
476         break;
477     }
478 }
479 }
480 if (DeleteInode == -1)
481 {
482     printf("error: The file %s does not exist.\n",ChildDirectory);
483     return -1;
484 }
485 // Release all the blocks pointed by the file
486 struct inode DelInode;
487 lseek(myfs,128 + (56 * DeleteInode),SEEK_SET);
488 read(myfs,(char*)&DelInode,sizeof(inode));
489
490 bool lflag = false;
491 for (i = 0 ; i < 8 ; i++)
492 {
493     if (DelInode.blockptrs[i]!=-1)
494     {
495         DelInode.blockptrs[i] = -1;
496         lseek(myfs,DelInode.blockptrs[i],SEEK_SET);
497         write(myfs,(char*)&lflag,1);
498     }
499 }
500 // Release the inode
501 strcpy(DelInode.name,"");
502 DelInode.used = 0;
503 DelInode.rsvd = 0;
504 DelInode.size = 0;
505 DelInode.dir = 0;
506
507 lseek(myfs,128 + (56 * DeleteInode),SEEK_SET);
508 write(myfs,(char*)&DelInode,sizeof(inode));
509
510 lseek(myfs,value*1024,SEEK_SET);
511 write(myfs,(char*)&ParentDirectoryBlock,sizeof(DirectoryBlock));
512 return 0;
513 }
514 /*
515 Start from the Root Directory
516 Traverse both paths while keeping track of directory inode.
517 Check if the directory in the file path exists
518 Else return -1
519 Check if the source file exists in the destination path
520 If exists then overwrite contents
521 Copy contents in directory
522 Create directory entry

```



```

523 */
524 int CP(char* srcname, char * dstname )
525 {
526     if(strcmp(srcname,"/") == 0)
527     {
528         printf("error: The Root Directory is an invalid source \n");
529         return -1;
530     }
531     if(strcmp(dstname,"/") == 0)
532     {
533         printf("error: The Root Directory is an invalid source \n");
534         return -1;
535     }
536     char * token = strtok(srcname, "/");
537     char * srcChildDirectory = token;
538     char * srcParentDirectory = "/";
539     // Get Source Parent Inode & Parent Directory
540     int srcaddress = getInode(srcParentDirectory);
541
542     struct inode srcParentInode;
543     struct DirectoryBlock srcParentDirectoryBlock;
544     lseek(myfs, (128 + (srcaddress * 56)), SEEK_SET);
545     read(myfs, (char*)&srcParentInode, 56);
546     int i = 0;
547     int value = srcParentInode.blockptrs[0];
548     lseek(myfs, value * 1024, SEEK_SET);
549     read(myfs, (char*)&srcParentDirectoryBlock, sizeof(DirectoryBlock));
550     while(token != NULL)
551     {
552         token = strtok(NULL, "/");
553         if (token != NULL)
554         {
555             // Scan the child inode's entry in the Parent's directory table. If not
556             // found then return -1.
557             // If found then the child is the new parent.
558             int flag = 0;
559             for (i = 2; i < 17; i++)
560             {
561                 if(strcmp(srcParentDirectoryBlock.DirectoryTable[i].name,
562                 srcChildDirectory) == 0){
563                     srcaddress = srcParentDirectoryBlock.DirectoryTable[i].inode;
564                     flag = 1;
565                     break;
566                 }
567             }
568             if (flag == 0)
569             {
570                 printf("error: The %s directory in the given path does not exist \n",
571                 srcChildDirectory);
572                 return -1;
573             }
574             // Get the new parent's inode and directory
575             lseek(myfs, (128 + (srcaddress * 56)), SEEK_SET);
576             read(myfs, (char*)&srcParentInode, 56);
577             value = srcParentInode.blockptrs[0];
578             lseek(myfs, value * 1024, SEEK_SET);
579             read(myfs, (char*)&srcParentDirectoryBlock, sizeof(DirectoryBlock));
580             srcParentDirectory = srcChildDirectory;
581             srcChildDirectory = token;
582         }
583     }
584     token = strtok(dstname, "/");
585     char * ChildDirectory = token;
586     char * ParentDirectory = "/";
587     // Destination Parent Inode & Parent Directory
588     int address = getInode(ParentDirectory);
589
590     struct inode ParentInode;
591     struct DirectoryBlock ParentDirectoryBlock;

```

```

589 lseek(myfs,(128+(address*56)),SEEK.SET);
590 read(myfs,(char*)&ParentInode,56);
591 i = 0;
592 int value1 = ParentInode.blockptrs[0];
593 lseek(myfs,value1*1024,SEEK.SET);
594 read(myfs,(char*)&ParentDirectoryBlock,sizeof(DirectoryBlock));
595
596 while(token!=NULL)
597 {
598     token = strtok(NULL,"/");
599     if (token !=NULL)
600     {
601         // Scan the child inode's entry in the Parent's directory table. If not
602         found then return -1.
603         // If found then the child is the new parent.
604         int flag = 0;
605         for (i = 2; i < 17;i++)
606         {
607             if(strcmp(ParentDirectoryBlock.DirectoryTable[i].name,ChildDirectory) ==
608 0){
609                 address = ParentDirectoryBlock.DirectoryTable[i].inode;
610                 flag = 1;
611                 break;
612             }
613             if (flag == 0)
614             {
615                 printf("error: The %s directory in the given path does not exist \n",
616 ChildDirectory);
617                 return -1;
618             }
619             // Get the new parent's inode and directory
620             lseek(myfs,(128+(address*56)),SEEK.SET);
621             read(myfs,(char*)&ParentInode,56);
622             value1 = ParentInode.blockptrs[0];
623             lseek(myfs,value1*1024,SEEK.SET);
624             read(myfs,(char*)&ParentDirectoryBlock,sizeof(DirectoryBlock));
625             ParentDirectory = ChildDirectory;
626             ChildDirectory = token;
627         }
628     }
629     struct inode SrcChildNode;
630     struct inode DstChildNode;
631     struct DataBlock SrcChildData[8];
632     struct DataBlock DstChildData[8];
633
634     srcaddress = getInode(srcChildDirectory);
635     lseek(myfs,128 + srcaddress * 56,SEEK.SET);
636     read(myfs,(char*)&SrcChildNode,sizeof(inode));
637     // Check if the src path is a directory
638     if (SrcChildNode.dir == 1)
639     {
640         printf("error: cannot handle directories .\n");
641         return -1;
642     }
643
644     // Check if the file already exists in the dst directory
645     bool alreadypresent = false;
646     bool lflag = true;
647     int foundat = -1;
648     int FreeInode;
649     for ( i = 2; i < 17; i++)
650     {
651         if (ParentDirectoryBlock.DirectoryTable[i].inode != -1 && strcmp(
652 ParentDirectoryBlock.DirectoryTable[i].name,ChildDirectory) == 0)
653         {
654             foundat = i;

```

```

654     alreadypresent = true;
655 }
656 }
657
658 if (alreadypresent == true)
659 {
660     // Check if the dst path is not a directory path . If yes then overwrite
        content
661     FreeInode = ParentDirectoryBlock.DirectoryTable[foundat].inode;
662     lseek(myfs, 128 + 56 * FreeInode, SEEK_SET);
663     read(myfs, (char*)&DstChildNode, sizeof(inode));
664
665     if (DstChildNode.dir == 1)
666     {
667         printf("error: cannot handle directories \n");
668         return -1;
669     }
670     bool tempflag = false;
671     for (i = 0 ; i < 8 ; i++)
672     {
673         if (DstChildNode.blockptrs[i] != -1)
674         {
675             lseek(myfs, DstChildNode.blockptrs[i], SEEK_SET);
676             write(myfs, (char*)&tempflag, 1);
677         }
678     }
679 }
680 else
681 {
682     // If not present then look for a free inode
683     FreeInode = getFreeInode(0, "");
684     if (FreeInode == -2){
685         printf("error: All inodes are occupied\n");
686         return -1;
687     }
688     lseek(myfs, 128 + 56 * FreeInode, SEEK_SET);
689     read(myfs, (char*)&DstChildNode, sizeof(inode));
690 }
691 // Get Data from the Data region
692 for (i = 0 ; i < 8 ; i++)
693 {
694     if (SrcChildNode.blockptrs[i] != -1)
695     {
696         lseek(myfs, SrcChildNode.blockptrs[i]*1024, SEEK_SET);
697         read(myfs, (char*)&SrcChildData[i], 1024);
698     }
699 }
700 // Update Destination Child Inode
701 strcpy(DstChildNode.name, ChildDirectory);
702 DstChildNode.used = 1;
703 DstChildNode.size = SrcChildNode.size;
704 DstChildNode.dir = SrcChildNode.dir;
705 for(int i = 0 ; i < 8 ; i++)
706 {
707     if (SrcChildNode.blockptrs[i] != -1){
708         strcpy(DstChildData[i].Data, SrcChildData[i].Data);
709         DstChildNode.blockptrs[i] = getFreeBlock();
710         lseek(myfs, DstChildNode.blockptrs[i], SEEK_SET);
711         write(myfs, (char*)&lflag, 1);
712     }
713 }
714 // Update the entry in the Parent's directory
715 if (alreadypresent == false)
716 {
717     for (i = 2; i < 17; i++)
718     {
719         if (ParentDirectoryBlock.DirectoryTable[i].inode == -1)
720         {
721             ParentDirectoryBlock.DirectoryTable[i].inode = FreeInode;

```

```

722     ParentDirectoryBlock.DirectoryTable[i].namelen = strlen(ChildDirectory)
723     ;
724     strcpy(ParentDirectoryBlock.DirectoryTable[i].name, ChildDirectory);
725     break;
726 }
727 }
728 // Create new entry in the Parent's directory
729 else
730 {
731     ParentDirectoryBlock.DirectoryTable[foundat].inode = FreeInode;
732     ParentDirectoryBlock.DirectoryTable[foundat].namelen = strlen(
ChildDirectory);
733     strcpy(ParentDirectoryBlock.DirectoryTable[foundat].name, ChildDirectory);
734 }
735
736 lseek(myfs, 128 + (56*FreeInode), SEEK_SET);
737 write(myfs, (char*)&DstChildNode, sizeof(inode));
738 lseek(myfs, value1*1024, SEEK_SET);
739 write(myfs, (char*)&ParentDirectoryBlock, sizeof(DirectoryBlock));
740 for (i = 0 ; i < 8; i++)
741 {
742     if (DstChildNode.blockptrs[i] != -1)
743     {
744         lseek(myfs, 1024 * DstChildNode.blockptrs[i], SEEK_SET);
745     }
746 }
747 return 0 ;
748 }
749 /*
750 MOVE A FILE
751 Start from the Root Directory
752 Traverse both paths while keeping track of directory inode.
753 Check if the directory in the file path exists
754 Else return -1
755 Check if the source file exists in the destination path
756 If exists then overwrite contents
757 Else Write in directory
758 Create directory entry in the destination directory
759 Delete directory entry in the source directory
760 */
761 int MV(char* srcname, char * dstname)
762 {
763     if(strcmp(srcname, "/")==0)
764     {
765         printf("error: Root directory is an invalid source name");
766         return -1;
767     }
768     if(strcmp(dstname, "/")==0)
769     {
770         printf("error: Root directory is an invalid destinatio name");
771         return -1;
772     }
773     char * token = strtok(srcname, "/");
774     char * srcChildDirectory = token;
775     char * srcParentDirectory = "/";
776     // Get Source Parent Inode & Parent Directory
777     int srcaddress = getInode(srcParentDirectory);
778
779     struct inode srcParentInode;
780     struct DirectoryBlock srcParentDirectoryBlock;
781
782
783     lseek(myfs, (128+(srcaddress*56)), SEEK_SET);
784     read(myfs, (char*)&srcParentInode, 56);
785     int i = 0;
786     int value = srcParentInode.blockptrs[0];
787     lseek(myfs, value*1024, SEEK_SET);
788     read(myfs, (char*)&srcParentDirectoryBlock, sizeof(DirectoryBlock));

```

```

789 while(token!=NULL)
790 {
791     token = strtok(NULL, "/");
792     if (token !=NULL)
793     { // Scan the child inode's entry in the Parent's directory table. If not
794       found then return -1.
795       // If found then the child is the new parent.
796       int flag = 0;
797       for (i = 2; i < 17;i++)
798       {
799         if(strcmp(srcParentDirectoryBlock.DirectoryTable[i].name,
800 srcChildDirectory) == 0){
801           srcaddress = srcParentDirectoryBlock.DirectoryTable[i].inode;
802           flag = 1;
803           break;
804         }
805       }
806       if (flag == 0)
807       {
808         printf("error: The %s directory in the given path does not exist \n",
809 srcChildDirectory);
810         return -1;
811       }
812       // Get the new parent's inode and directory
813       lseek(myfs,(128+(srcaddress*56)),SEEK_SET);
814       read(myfs,(char*)&srcParentInode,56);
815       value = srcParentInode.blockptrs[0];
816       lseek(myfs,value*1024,SEEK_SET);
817       read(myfs,(char*)&srcParentDirectoryBlock,sizeof(DirectoryBlock));
818       srcParentDirectory = srcChildDirectory;
819       srcChildDirectory = token;
820     }
821 }
822 token = strtok(dstname, "/");
823 char * ChildDirectory = token;
824 char * ParentDirectory = "/";
825 // Destination Parent Inode & Parent Directory
826 int address = getInode(ParentDirectory);
827
828 struct inode ParentInode;
829 struct DirectoryBlock ParentDirectoryBlock;
830 lseek(myfs,(128+(address*56)),SEEK_SET);
831 read(myfs,(char*)&ParentInode,56);
832 i = 0;
833 int value1 = ParentInode.blockptrs[0];
834 lseek(myfs,value1*1024,SEEK_SET);
835 read(myfs,(char*)&ParentDirectoryBlock,sizeof(DirectoryBlock));
836
837 while(token!=NULL)
838 {
839     token = strtok(NULL, "/");
840     if (token !=NULL)
841     { // Scan the child inode's entry in the Parent's directory table. If not
842       found then return -1.
843       // If found then the child is the new parent.
844       int flag = 0;
845       for (i = 2; i < 17;i++)
846       {
847         if(strcmp(ParentDirectoryBlock.DirectoryTable[i].name, ChildDirectory) ==
848 0){
849           address = ParentDirectoryBlock.DirectoryTable[i].inode;
850           flag = 1;
851           break;
852         }
853       }
854       if (flag == 0)
855       {
856         printf("error: The %s directory in the given path does not exist \n",

```

```

ChildDirectory);
853     return -1;
854 }
855 // Get the new parent's inode and directory
856 lseek(myfs,(128+(address*56)),SEEK.SET);
857 read(myfs,(char*)&ParentInode,56);
858
859 value1 = ParentInode.blockptrs[0];
860 lseek(myfs,value1*1024,SEEK.SET);
861 read(myfs,(char*)&ParentDirectoryBlock,sizeof(DirectoryBlock));
862 ParentDirectory = ChildDirectory;
863 ChildDirectory = token;
864
865 }
866 }
867
868 bool isFoundSrc = false;
869 int foundatSrc = -1;
870 for ( i = 2; i < 17; i++)
871 {
872     if (srcParentDirectoryBlock.DirectoryTable[i].inode != -1 && strcmp(
srcParentDirectoryBlock.DirectoryTable[i].name,srcChildDirectory) == 0)
873     {
874         foundatSrc = i;
875         isFoundSrc = true;
876         break;
877     }
878 }
879 if (isFoundSrc == false)
880 {
881     printf("error : file or directory with this name does not exist \n");
882     return -1;
883 }
884 struct inode SrcChildNode;
885 srcaddress = srcParentDirectoryBlock.DirectoryTable[foundatSrc].inode;
886 lseek(myfs,128 + 56 * srcaddress , SEEK.SET);
887 read(myfs,(char*)&SrcChildNode,sizeof(inode));
888 // Check if the src path is a directory
889 if (SrcChildNode.dir == 1)
890 {
891     printf("error : does not handle directories \n");
892     return -1;
893 }
894 bool isFound = false;
895 int foundat = -1;
896 for ( i = 2; i < 17; i++)
897 {
898     if (ParentDirectoryBlock.DirectoryTable[i].inode != -1 && strcmp(
ParentDirectoryBlock.DirectoryTable[i].name,ChildDirectory) == 0)
899     {
900         foundat = i;
901         isFound = true;
902         break;
903     }
904 }
905 struct inode DstChildNode;
906 if (isFound == true)
907 {
908     // If already exists , then overwrite old entry. The inode is same but the
blk pts our updated
909     address = ParentDirectoryBlock.DirectoryTable[foundat].inode;
910     lseek(myfs,128 + 56 * address , SEEK.SET);
911     read(myfs,(char*)&DstChildNode,sizeof(inode));
912     if (DstChildNode.dir == 1)
913     {
914         printf("error : does not handle directories \n");
915         return -1;
916     }
917     for ( i = 0 ; i < 8 ; i++)

```

```

918     {
919         DstChildNode.blockptrs[i] = SrcChildNode.blockptrs[i];
920         SrcChildNode.blockptrs[i] = -1;
921     }
922     DstChildNode.size = SrcChildNode.size;
923     SrcChildNode.dir = 0;
924     SrcChildNode.rsvd = 0;
925     SrcChildNode.size = 0;
926     SrcChildNode.used = 0;
927     strcpy(SrcChildNode.name, "");
928
929     srcParentDirectoryBlock.DirectoryTable[foundatSrc].inode = -1;
930     srcParentDirectoryBlock.DirectoryTable[foundatSrc].namelen = 0;
931     strcpy(srcParentDirectoryBlock.DirectoryTable[foundatSrc].name, "");
932
933     lseek(myfs, 128 + 56 * srcaddress, SEEK_SET);
934     write(myfs, (char*)&SrcChildNode, sizeof(inode));
935     lseek(myfs, 128 + 56 * address, SEEK_SET);
936     write(myfs, (char*)&DstChildNode, sizeof(inode));
937     lseek(myfs, value*1024, SEEK_SET);
938     write(myfs, (char*)&srcParentDirectoryBlock, sizeof(DirectoryBlock));
939 }
940 else
941 {
942     // Delete the entry in src Parent's directory table and add it in the dst
943     // Parent's directory table
944     for (i = 2; i < 17; i++)
945     {
946         if (ParentDirectoryBlock.DirectoryTable[i].inode == -1 )
947         {
948             ParentDirectoryBlock.DirectoryTable[i].inode = srcaddress;
949             strcpy(ParentDirectoryBlock.DirectoryTable[i].name, ChildDirectory);
950             ParentDirectoryBlock.DirectoryTable[i].namelen = strlen(ChildDirectory);
951             break;
952         }
953     }
954     srcParentDirectoryBlock.DirectoryTable[foundatSrc].inode = -1;
955     srcParentDirectoryBlock.DirectoryTable[foundatSrc].namelen = 0;
956     strcpy(srcParentDirectoryBlock.DirectoryTable[foundatSrc].name, "");
957     lseek(myfs, value*1024, SEEK_SET);
958     write(myfs, (char*)&srcParentDirectoryBlock, sizeof(DirectoryBlock));
959     lseek(myfs, value1*1024, SEEK_SET);
960     write(myfs, (char*)&ParentDirectoryBlock, sizeof(DirectoryBlock));
961
962 }
963
964
965
966
967
968     return 0;
969 }
970 /*
971 Start from the Root Directory
972 Traverse Path while keeping directory inode.
973 Check if the directory in the file path exists
974 Else return -1
975 If inode available
976 Check if the director does not exist
977 Create an empty directory
978 Create directory entry
979 */
980 // CREATE A DIRECTORY
981 int CD(char* dirname)
982 {
983
984     if (strcmp(dirname, "/") == 0)
985     {

```

```

986     printf("error : Root directory already exists \n");
987     return -1;
988 }
989 char * token = strtok(dirname, "/");
990 char * ChildDirectory = token;
991 char * ParentDirectory = "/";
992
993 int address = getInode(ParentDirectory);
994 // Get Parent Inode & Parent Directory
995 struct inode ParentInode;
996 struct DirectoryBlock ParentDirectoryBlock;
997 lseek(myfs, (128 + (address * 56)), SEEK_SET);
998 read(myfs, (char*)&ParentInode, 56);
999 int i = 0;
1000 int value = ParentInode.blockptrs[0];
1001 lseek(myfs, value * 1024, SEEK_SET);
1002 read(myfs, (char*)&ParentDirectoryBlock, sizeof(DirectoryBlock));
1003
1004 while(token != NULL)
1005 {
1006     token = strtok(NULL, "/");
1007     if (token != NULL)
1008     { // Scan the child inode's entry in the Parent's directory table. If not
1009         // found then return -1.
1010         // If found then the child is the new parent.
1011         int flag = 0;
1012         for (i = 2; i < 17; i++)
1013         {
1014             if (strcmp(ParentDirectoryBlock.DirectoryTable[i].name, ChildDirectory) ==
1015             0) {
1016                 address = ParentDirectoryBlock.DirectoryTable[i].inode;
1017                 flag = 1;
1018                 break;
1019             }
1020             if (flag == 0)
1021             {
1022                 printf("error: The %s directory in the given path does not exist \n",
1023                 ChildDirectory);
1024                 return -1;
1025             }
1026             // Get the new parent's inode and directory
1027             lseek(myfs, (128 + (address * 56)), SEEK_SET);
1028             read(myfs, (char*)&ParentInode, 56);
1029             value = ParentInode.blockptrs[0];
1030             lseek(myfs, value * 1024, SEEK_SET);
1031             read(myfs, (char*)&ParentDirectoryBlock, sizeof(DirectoryBlock));
1032             ParentDirectory = ChildDirectory;
1033             ChildDirectory = token;
1034         }
1035     }
1036     struct inode ChildInode;
1037     struct DirectoryBlock ChildDirectoryBlock;
1038     // Check if free inodes are available & Check if the file exists already
1039
1040     int freeInode = getFreeInode(1, ChildDirectory);
1041     if (freeInode == -1)
1042     {
1043         printf("error : file or directory with this name already exists \n");
1044         return -1;
1045     }
1046     if (freeInode == -2)
1047     {
1048         printf("error: No free Inodes are available \n");
1049         return -1;
1050     }
1051     int FreeBlock = getFreeBlock();

```



```

1052 // Update's parents directory & Create Directory entry in the inode table
1053 lseek(myfs,128 + (freeInode * 56),SEEK_SET);
1054 read(myfs,(&ChildInode,sizeof(inode));
1055 strcpy(ChildInode.name,ChildDirectory);
1056 ChildInode.used = 1;
1057 ChildInode.size = sizeof(DirectoryBlock);
1058 ChildInode.dir = 1;
1059 ChildInode.rsvd = 0;
1060 ChildInode.blockptrs[0] = FreeBlock;
1061 // Initilize the directory table for the new directory
1062 strcpy(ChildDirectoryBlock.DirectoryTable[0].name, ".");
1063 ChildDirectoryBlock.DirectoryTable[0].namelen = 2;
1064 ChildDirectoryBlock.DirectoryTable[0].inode = address;
1065
1066 strcpy(ChildDirectoryBlock.DirectoryTable[1].name, ".");
1067 ChildDirectoryBlock.DirectoryTable[1].namelen = 1;
1068 ChildDirectoryBlock.DirectoryTable[1].inode = freeInode;
1069
1070 for ( i = 2; i <17;i++)
1071 {
1072     ChildDirectoryBlock.DirectoryTable[i].namelen = 0;
1073     ChildDirectoryBlock.DirectoryTable[i].inode = -1;
1074 }
1075
1076 for (int i = 2; i < 17;i++)
1077 {
1078     if (ParentDirectoryBlock.DirectoryTable[i].inode == -1)
1079     {
1080         strcpy(ParentDirectoryBlock.DirectoryTable[i].name,ChildDirectory);
1081         ParentDirectoryBlock.DirectoryTable[i].namelen = strlen(ChildDirectory);
1082         ParentDirectoryBlock.DirectoryTable[i].inode = freeInode;
1083         break;
1084     }
1085 }
1086
1087 lseek(myfs,128+(freeInode*56),SEEK_SET);
1088 write(myfs,(&ChildInode,sizeof(ChildInode));
1089 lseek(myfs,1024 * FreeBlock,SEEK_SET);
1090 write(myfs,(&ChildDirectoryBlock,sizeof(DirectoryBlock));
1091 lseek(myfs,value*1024,SEEK_SET);
1092 write(myfs,(&ParentDirectoryBlock,sizeof(DirectoryBlock));
1093 bool temp = true;
1094 lseek(myfs,FreeBlock,SEEK_SET);
1095 write(myfs,(&temp,1);
1096 return 0;
1097 }
1098
1099 void Recurse(int inodeptr)
1100 {
1101     struct inode currentnode;
1102     int i = 0;
1103     bool Delete = false;
1104     lseek(myfs,128 + inodeptr * 56,SEEK_SET);
1105     read(myfs,(&currentnode,sizeof(inode));
1106
1107     currentnode.used = 0;
1108     currentnode.rsvd = 0;
1109     currentnode.size = 0;
1110     strcpy(currentnode.name,"");
1111
1112     lseek(myfs,128 + inodeptr * 56,SEEK_SET);
1113     write(myfs,(&currentnode,sizeof(inode));
1114 // File , So release all blocks
1115 if (currentnode.dir == 0)
1116 {
1117     for (i = 0 ; i < 8; i++)
1118     {

```

```

1121         if (currentnode.blockptrs[i] != -1)
1122         {
1123             lseek(myfs, currentnode.blockptrs[i], SEEK_SET);
1124             write(myfs, (char*)&Delete, 1);
1125         }
1126     }
1127
1128 }
1129 else
1130 {
1131     // Directory so recursive call
1132     struct DirectoryBlock currentnodedirectory;
1133     lseek(myfs, currentnode.blockptrs[0]*1024, SEEK_SET);
1134     read(myfs, (char*)&currentnodedirectory, sizeof(DirectoryBlock));
1135     for (i = 2; i < 17; i++)
1136     {
1137         if (currentnodedirectory.DirectoryTable[i].inode != -1)
1138         {
1139             Recurse(currentnodedirectory.DirectoryTable[i].inode);
1140         }
1141     }
1142     lseek(myfs, currentnode.blockptrs[0], SEEK_SET);
1143     write(myfs, (char*)&Delete, 1);
1144 }
1145
1146 }
1147
1148
1149 /*
1150 Start from the Root Directory
1151 Traverse Path while keeping directory inode.
1152 Check if the directory in the file path exists
1153 Else return -1
1154 If inode available
1155 Check if the directory exist
1156 Recursively delete the content of the directory
1157 */
1158 // DELETE A DIRECTORY
1159 int DD(char* dirname)
1160 {
1161     if (strcmp(dirname, "/") == 0)
1162     {
1163         printf("error: cannot delete root directory \n");
1164         return -1;
1165     }
1166     char * token = strtok(dirname, "/");
1167     char * ChildDirectory = token;
1168     char * ParentDirectory = "/";
1169
1170     int address = getInode(ParentDirectory);
1171
1172     int i = 0;
1173     struct inode ParentInode;
1174     struct DirectoryBlock ParentDirectoryBlock;
1175
1176
1177     lseek(myfs, (128 + (address * 56)), SEEK_SET);
1178     read(myfs, (char*)&ParentInode, 56);
1179
1180     int value = ParentInode.blockptrs[0];
1181     lseek(myfs, value * 1024, SEEK_SET);
1182     read(myfs, (char*)&ParentDirectoryBlock, sizeof(DirectoryBlock));
1183
1184     while(token != NULL)
1185     {
1186         token = strtok(NULL, "/");
1187         if (token != NULL)
1188         {
1189             int flag = 0;

```

```

1190     for (i = 2; i < 17; i++)
1191     {
1192         if (strcmp(ParentDirectoryBlock.DirectoryTable[i].name, ChildDirectory) ==
0){
1193             address = ParentDirectoryBlock.DirectoryTable[i].inode;
1194             flag = 1;
1195             break;
1196         }
1197     }
1198     if (flag == 0)
1199     {
1200         printf("error: The %s directory in the given path does not exist \n",
ChildDirectory);
1201         return -1;
1202     }
1203
1204     lseek(myfs, (128 + (address * 56)), SEEK_SET);
1205     read(myfs, (char*)&ParentInode, 56);
1206     value = ParentInode.blockptrs[0];
1207     lseek(myfs, value * 1024, SEEK_SET);
1208     read(myfs, (char*)&ParentDirectoryBlock, sizeof(DirectoryBlock));
1209     ParentDirectory = ChildDirectory;
1210     ChildDirectory = token;
1211
1212 }
1213 }
1214
1215 int ChildInodeAddress = getNode(ChildDirectory);
1216 if (ChildInodeAddress == -1)
1217 {
1218     printf("error: the directory does not exist \n");
1219     return -1;
1220 }
1221 Recurse(ChildInodeAddress);
1222 return 0;
1223 }
1224 /*
1225 Traverse the Inode Table
1226 Print all those inodes which are currently being used.
1227 */
1228 // LIST ALL FILES
1229 void LL()
1230 {
1231     struct inode Table[16];
1232     lseek(myfs, 128, SEEK_SET);
1233     read(myfs, (char*)&Table, (16 * 56));
1234     int i = 0;
1235     for (i = 0; i < 16; i++)
1236     {
1237         if (Table[i].used == 1)
1238         {
1239             printf("Name : %s , Dir : %d , Size : %d \n", Table[i].name, Table[i].dir,
Table[i].size);
1240         }
1241     }
1242 }
1243 }
1244 }
1245 /*
1246 * main
1247 *
1248 */
1249 void printInodeTable() {
1250     struct inode Table[16];
1251     lseek(myfs, 128, SEEK_SET);
1252     read(myfs, (char*)&Table, (16 * 56));
1253     int i = 0;
1254     for (i = 0; i < 16; i++)

```

```

1256 {
1257     if (Table[i].used == 1)
1258     {
1259         printf("Index : %d , Name : %s , Dir : %d , Size : %d , Used : %d \n", i,
1260             Table[i].name, Table[i].dir, Table[i].size, Table[i].used);
1261         printf("BlkPtr[0] : %d BlkPtr[1] : %d BlkPtr[2] : %d BlkPtr[3] : %d BlkPtr[4]
1262             : %d BlkPtr[5] : %d BlkPtr[6] : %d BlkPtr[7] : %d \n",
1263             Table[i].blockptrs[0], Table[i].blockptrs[1], Table[i].blockptrs[2], Table[i].
1264             blockptrs[3], Table[i].blockptrs[4], Table[i].blockptrs[5],
1265             Table[i].blockptrs[6], Table[i].blockptrs[7]);
1266     }
1267 }
1268
1269 int main (int argc, char* argv[]) {
1270     // while not EOF
1271     // read command
1272     // parse command
1273
1274     // call appropriate function
1275     myfs = open("myfs", ORDWR); // read-write enabled
1276     if (myfs == -1)
1277     {
1278         printf("myfs does not exist \n");
1279         initiliaze();
1280     }
1281
1282     FILE * stream = fopen(argv[1], "r");
1283     char *Line = NULL;
1284     char Command[3];
1285     size_t len = 0;
1286     while (getline(&Line, &len, stream) != -1)
1287     {
1288
1289         sscanf(Line, "%[^ \n] %[^ \n]", Command, Line);
1290         // Create File
1291         if (strcmp(Command, "CR") == 0)
1292         {
1293             char * FileName = strtok(Line, " ");
1294             int Size = atoi(strtok(NULL, " "));
1295             CR(FileName, Size);
1296         }
1297         // Delete File
1298         else if (strcmp(Command, "DL") == 0)
1299         {
1300             DL(Line);
1301         }
1302         // Copy File
1303         else if (strcmp(Command, "CP") == 0)
1304         {
1305             char * srcname = strtok(Line, " ");
1306             char * dstname = strtok(NULL, " ");
1307             CP(srcname, dstname);
1308         }
1309
1310         // Move a File
1311         else if (strcmp(Command, "MV") == 0)
1312         {
1313             printf("MV \n");
1314             char * srcname = strtok(Line, " ");
1315             char * dstname = strtok(NULL, " ");
1316             MV(srcname, dstname);
1317         }
1318
1319         // Create Directory
1320         else if (strcmp(Command, "CD") == 0)
1321

```

```

1322     {
1323         Line = strtok(Line, "\n");
1324         CD(Line);
1325     }
1326     // Remove Directory
1327     else if (strcmp(Command, "DD") == 0)
1328     {
1329         DD(Line);
1330     }
1331     // List all files
1332     else if (strcmp(Command, "LL") == 0)
1333     {
1334         LL();
1335     }
1336 }
1337 close(myfs);
1338 fclose(stream);
1339 free(Line);
1340
1341
1342
1343
1344
1345
1346
1347 return 0;
1348 }

```

Listing 1: filesystem.c