# CS232 Operating Systems
# Assignment 4

Name: Nasir
ID: aa04377

December 15, 2020

## 1 Pointers

Directory sizes are equivalent to the the total size of all of contents in them. Eg. If root directory has a sub directory home and directory home has a file then, the size of home = size of file and size of root = 1024(size of a direcotry eg. home) + size of home(size of file).
Size of empty directory is 0.
LL lists only directory/file names that are in use and their sizes.

## 2 Codes

### 2.1 filesystem.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>


/*
 *    ___  ___  ___  ___  ___  ___  ___  ___  ___  ___  ___
 *   |   |   |   |   |                                 |   |
 *   | 0 | 1 | 2 | 3 |      .....                      |127|
 *   |___|___|___|___|_________________________________|___|
 *   |    \     <------  data blocks ------>
 *   |      \
 *   |        \
 *   |          \
 *   |            \
 *   |              \
 *   |                \
 *   |                  \
 *   |                    \
 *   |                      \
 *   |                        \
 *   |                          \
 *   |                            \
 *   |                              \
 *   |                                \
 *   |                                  \
 *   |                                    \
 *   |                                      \
 *   |                                        \
 *   |      <--- super block --->                \
 *   |-------------------------------------------|
 *   |                |      |      |        |      |
```

```
 *  |        free    |      |      |         |        |
 *  |       block    |inode0|inode1|    ....  |inode15|
 *  |        list    |      |      |         |        |
 *  |----------------|------|------|---------|--------|
 *
 *
 */

#define FILENAME_MAXLEN 8  // including the NULL char
#define NO_BLOCKS 128
#define NO_INODES 16
#define FREE_LST_LEN 128

// int fd;


/*
 * inode
 */
typedef struct
{
    int  dir;  // boolean value. 1 if it's a directory.
    char name[FILENAME_MAXLEN];
    int  size;  // actual file/directory size in bytes.
    int  blockptrs [8];  // direct pointers to blocks containing file's content.
    int  used;  // boolean value. 1 if the entry is in use.
    int  rsvd;  // reserved for future use
}
inode;

/*
 * directory entry
 */
typedef struct
{
    char name[FILENAME_MAXLEN];
    int  namelen;  // length of entry name
    int  inode;  // this entry inode index
}
dirent;

// super block
typedef struct
{
    char free_block_lst[FREE_LST_LEN];
    inode inode_table[NO_INODES];
}
super;

// block
typedef union
{
  //if this block is super block
    super super_block_ptr;

  //if this block is directory
    dirent directory_table[64];

  //if this block is file;
  // char file_data[1024];
}
block;


/*
 * functions
 */
// initialize disk
void init_disk(block*);
```

```
106
107 // create file
108 void CR(block*, char*);
109
110 // Delete file
111 void DL(block*, char*);
112
113 // Copy/move File
114 void CP(block*, char*, int);
115
116 // create directory
117 void CD(block*, char*);
118
119 // delete directory
120 void DD(block*, char*);
121
122 // list file info
123 void LL(block*, int);
124
125 //check if path exist.
126 void verify_path(block*, char**, int, int*);
127 void verify_path_2(block*, char**, int, int*);
128
129
130 /*
131  * main
132  */
133 int main(int argc, char* argv[])
134 {
135   //create and init disk.
136     block disk_block_ptr[128];
137   init_disk(disk_block_ptr);
138
139   FILE* myfs = fopen("myfs", "w");
140
141   //open file.
142     FILE * stream = fopen(argv[1], "r");
143     if(stream == NULL)
144     {
145         fprintf(stderr, "Couldn't read file.\n");
146         exit(1);
147     }
148
149     char line[1000];
150   char func[3];
151   char str[1000];
152
153     //read file.
154     while (fgets(line, 1000, stream) != NULL)
155     {
156         line[strcspn(line, "\n")] = '\0';
157
158     strcpy(func, strtok(line, " "));
159     if(strcmp(func, "LL"))
160     {
161       strcpy(str, strtok(NULL, "\0"));
162
163       //call create file
164       if(!strcmp(func, "CR"))
165       {
166         CR(disk_block_ptr, str);
167       }
168
169       //call delete file
170       if(!strcmp(func, "DL"))
171       {
172         DL(disk_block_ptr, str);
173       }
174
```

```
175       //call copy file
176       if(!strcmp(func, "CP"))
177       {
178         CP(disk_block_ptr, str, 0);
179       }
180
181       //call copy file then delete src.
182       if(!strcmp(func, "MV"))
183       {
184         CP(disk_block_ptr, str, 1);
185       }
186
187       //call create directory
188       if(!strcmp(func, "CD"))
189       {
190         CD(disk_block_ptr, str);
191       }
192
193       //call delete directory
194       if(!strcmp(func, "DD"))
195       {
196         DD(disk_block_ptr, str);
197       }
198     }
199     //call list info
200     else
201     {
202       strcpy(str, "\0");
203
204       LL(disk_block_ptr, 1);
205     }
206
207     //write to myfs.
208     for (int i = 0; i < NO_BLOCKS; i++)
209     {
210       fwrite((disk_block_ptr + i), 1024, 1, myfs);
211     }
212     }
213
214   fclose(myfs);
215     fclose(stream);
216   return 0;
217 }
218
219 void init_disk(block* block_lst_ptr)
220 {
221   FILE* stream = fopen("myfs", "r");
222
223   if (stream == NULL)
224   {
225     printf("Creating myfs.\n");
226   //init inode table in super block.
227   for(int i = 0; i < NO_INODES; i++)
228   {
229     //init first inode for root,
230     if (i == 0)
231     {
232       strcpy(((block_lst_ptr + 0) -> super_block_ptr).inode_table[i].name, "/");
233       ((block_lst_ptr + 0) -> super_block_ptr).inode_table[i].dir = 1;
234       ((block_lst_ptr + 0) -> super_block_ptr).inode_table[i].size = 0;
235       ((block_lst_ptr + 0) -> super_block_ptr).inode_table[i].used = 1;
236       ((block_lst_ptr + 0) -> super_block_ptr).inode_table[i].rsvd = 1;
237
238       //init first direct block pointers for root inode to 1.
239       ((block_lst_ptr + 0) -> super_block_ptr).inode_table[i].blockptrs[0] = 1;
240
241       //init remaining t0 -1.
242       for(int j = 1; j < 8; j++)
243       {
```

```
244         ((block_lst_ptr + 0) -> super_block_ptr).inode_table[i].blockptrs[j] =
    -1;
245       }
246     }
247
248     //init remaining inodes.
249     else
250     {
251       strcpy(((block_lst_ptr + 0) -> super_block_ptr).inode_table[i].name, " ");
252       ((block_lst_ptr + 0) -> super_block_ptr).inode_table[i].dir = 0;
253       ((block_lst_ptr + 0) -> super_block_ptr).inode_table[i].size = 0;
254       ((block_lst_ptr + 0) -> super_block_ptr).inode_table[i].used = 0;
255       ((block_lst_ptr + 0) -> super_block_ptr).inode_table[i].rsvd = 0;
256
257
258       //init direct block pointers in remaining inodes.
259       for(int j = 0; j < 8; j++)
260       {
261         ((block_lst_ptr + 0) -> super_block_ptr).inode_table[i].blockptrs[j] =
    -1;
262       }
263     }
264   }
265
266   //init free block list in super block.
267   //superblock
268   ((block_lst_ptr + 0) -> super_block_ptr).free_block_lst[0] = '0';
269   //root
270   ((block_lst_ptr + 0) -> super_block_ptr).free_block_lst[1] = '0';
271   //remaining blocks
272   for(int i = 2; i < FREE_LST_LEN; i++)
273   {
274     ((block_lst_ptr + 0) -> super_block_ptr).free_block_lst[i] = '1';
275   }
276
277   //init rest of the blocks.
278   for(int i = 1; i < NO_BLOCKS; i++)
279   {
280     for(int j = 0; j < (NO_INODES - 1); j++)
281     {
282       (block_lst_ptr + i) -> directory_table[j].inode = -1;
283       strcpy((block_lst_ptr + i) -> directory_table[j].name, " ");
284       (block_lst_ptr + i) -> directory_table[j].namelen = 0;
285     }
286   }
287   }
288
289   else
290   {
291     printf("Reading myfs.\n");
292
293     // fread(&((block_lst_ptr + 0) -> super_block_ptr), 1024, 1, stream);
294
295     for (int i = 0; i < NO_BLOCKS; i++)
296     {
297       fread((block_lst_ptr + i), 1024, 1, stream);
298     }
299
300     fclose(stream);
301   }
302
303   return;
304 }
305
306 void verify_path(block* disk_block_ptr, char** arr, int last_index, int*
    inode_arr)
307 {
308   //init parent inode array.
309   for(int i = 0; i < last_index; i++)
```

```
310   {
311     inode_arr[i] = -1;
312   }
313
314   //if new file/dir in root.
315   if (last_index == 1)
316   {
317     for (int i = 0; i < NO_INODES - 1; i++)
318     {
319       if ((disk_block_ptr + 1) -> directory_table[i].inode != -1 && !strcmp((
      disk_block_ptr + 1) -> directory_table[i].name, arr[last_index - 1]))
320       {
321         printf("the ");
322
323         if (((disk_block_ptr + 0) -> super_block_ptr).inode_table[(disk_block_ptr
       + 1) -> directory_table[i].inode].dir == 1)
324         {
325           printf("directory ");
326         }
327         else
328         {
329           printf("file ");
330         }
331
332         printf("already exists\n");
333         inode_arr[0] = -1;
334         break;
335       }
336
337       else
338       {
339         inode_arr[0] = 0;
340       }
341     }
342   }
343
344   //check path if path not in root.
345   else
346   {
347     int next_block = 1;
348     int next_inode = -1;
349     int root_block = ((disk_block_ptr + 0) -> super_block_ptr).inode_table[0].
      blockptrs[0];
350
351     for(int i = 0; i < (NO_INODES - 1); i++)
352     {
353       if(((disk_block_ptr + root_block) -> directory_table[i].inode != -1) && !
      strcmp((disk_block_ptr + root_block) -> directory_table[i].name, arr[0]))
354       {
355         next_inode = (disk_block_ptr + root_block) -> directory_table[i].inode;
356         inode_arr[0] = 0;
357         inode_arr[1] = next_inode;
358         break;
359       }
360     }
361
362     if(next_inode == -1)
363     {
364       printf("The directory %s in the given path does not exit.\n", arr[0]);
365       return;
366     }
367     for (int i = 1; i < last_index - 1; ++i)
368     {
369       next_block = ((disk_block_ptr + 0) -> super_block_ptr).inode_table[
      next_inode].blockptrs[0];
370       next_inode = -1;
371
372       for(int j = 0; j < (NO_INODES - 1); j++)
373       {
```

```
374         if(((disk_block_ptr + next_block) -> directory_table[j].inode != -1) && !
    strcmp((disk_block_ptr + next_block) -> directory_table[j].name, arr[i]))
375         {
376           next_inode = (disk_block_ptr + next_block) -> directory_table[j].inode;
377           inode_arr[i+1] = next_inode;
378           break;
379         }
380       }
381
382       if (next_inode == -1)
383       {
384         printf("The directory %s in the given path does not exit.\n", arr[i]);
385         inode_arr[0] = -1;
386         inode_arr[1] = -1;
387         break;
388       }
389     }
390
391     if (next_inode != -1)
392     {
393       next_block = ((disk_block_ptr + 0) -> super_block_ptr).inode_table[
    next_inode].blockptrs[0];
394
395       for(int j = 0; j < NO_INODES -1; j++)
396       {
397         if(((disk_block_ptr + next_block) -> directory_table[j].inode != -1) && !
    strcmp((disk_block_ptr + next_block) -> directory_table[j].name, arr[
    last_index-1]))
398         {
399           printf("the ");
400
401           if (((disk_block_ptr + 0) -> super_block_ptr).inode_table[(
    disk_block_ptr + 1) -> directory_table[j].inode].dir == 1)
402           {
403             printf("directory ");
404           }
405           else
406           {
407             printf("file ");
408           }
409
410           printf("already exists\n");
411
412           inode_arr[0] = -1;
413           inode_arr[1] = -1;
414           break;
415         }
416       }
417     }
418   }
419
420   return;
421 }
422
423 void verify_path_2(block* disk_block_ptr, char** arr, int last_index, int*
    inode_arr)
424 {
425   //init parent inode array.
426   for(int i = 0; i < last_index; i++)
427   {
428     inode_arr[i] = -1;
429   }
430
431   //if new file/dir in root.
432   if (last_index == 1)
433   {
434     for (int i = 0; i < NO_INODES - 1; i++)
435     {
```

```
436       if ((disk_block_ptr + 1) -> directory_table[i].inode != -1 && !strcmp((
      disk_block_ptr + 1) -> directory_table[i].name, arr[last_index - 1]))
437       {
438         inode_arr[0] = 0;
439         break;
440       }
441     }
442
443     if (inode_arr[0] == -1)
444     {
445       printf("the file does not exist\n");
446       return;
447     }
448   }
449
450   else
451   {
452     int next_block = 1;
453     int next_inode = -1;
454     // go to root.
455     int root_block = ((disk_block_ptr + 0) -> super_block_ptr).inode_table[0].
      blockptrs[0];
456
457     for(int i = 0; i < (NO_INODES - 1); i++)
458     {
459       if(((disk_block_ptr + root_block) -> directory_table[i].inode != -1) && !
      strcmp((disk_block_ptr + root_block) -> directory_table[i].name, arr[0]))
460       {
461         next_inode = (disk_block_ptr + root_block) -> directory_table[i].inode;
462         inode_arr[0] = 0;
463         inode_arr[1] = next_inode;
464         break;
465       }
466     }
467
468     if(next_inode == -1)
469     {
470       printf("The directory %s in the given path does not exit.\n", arr[0]);
471       return;
472     }
473
474     //check path.
475     for (int i = 1; i < last_index - 1; ++i)
476     {
477       next_block = ((disk_block_ptr + 0) -> super_block_ptr).inode_table[
      next_inode].blockptrs[0];
478       next_inode = -1;
479
480       for(int j = 0; j < (NO_INODES - 1); j++)
481       {
482         //dir not exists in path.
483         if(((disk_block_ptr + next_block) -> directory_table[j].inode != -1) && !
      strcmp((disk_block_ptr + next_block) -> directory_table[j].name, arr[i]))
484         {
485           next_inode = (disk_block_ptr + next_block) -> directory_table[j].inode;
486           inode_arr[i+1] = next_inode;
487           break;
488         }
489       }
490
491       if (next_inode == -1)
492       {
493         printf("The directory %s in the given path does not exit.\n", arr[i]);
494         inode_arr[0] = -1;
495         inode_arr[1] = -1;
496         break;
497       }
498     }
499   }
```

```
500
501   return;
502 }
503
504 void CR(block* disk_block_ptr, char* str)
505 {
506   printf("CR called, str:");
507   printf("%s\n", str);
508
509   char path[256];
510   int file_size = 0;
511
512   char* token = strtok(str, " ");
513   strcpy(path, token);
514
515   token = strtok(NULL, " ");
516   file_size = atoi(token);
517
518   //find no of blocks to allocate.
519   int no_blocks = (file_size/1024);
520   no_blocks++;
521
522   if (no_blocks > 8)
523   {
524     printf("not enough space.\n");
525     return;
526   }
527
528   int data_blocks[no_blocks];
529   data_blocks[0] = -1;
530
531   int path_length = 0;
532
533   //find path length.
534   for (int i = 0; path[i]; i++)
535   {
536     if(path[i] == '/')
537     {
538       path_length++;
539     }
540   }
541
542   //store directories in path in an array.
543   int i = 0;
544   char* arr[path_length];
545   token = strtok(path, "/");
546   while (token != NULL)
547   {
548     arr[i++] = token;
549     token = strtok(NULL, "/");
550   }
551
552   //new file name
553   char* file_name = arr[path_length - 1];
554
555   // array to store inode of parents.
556   int parent_inode_arr[path_length];
557
558   verify_path(disk_block_ptr, arr, path_length, parent_inode_arr);
559
560
561   //do only if path was ok and file does not already exist.
562   if(parent_inode_arr[0] == 0)
563   {
564     //find free block(s).
565     int count = 0;
566     for(int i = 0; i < NO_BLOCKS; i++)
567     {
568       if(((disk_block_ptr + 0) -> super_block_ptr).free_block_lst[i] == '1')
```

```
569       {
570         data_blocks[count] = i;
571         count++;
572         ((disk_block_ptr + 0) -> super_block_ptr).free_block_lst[i] = '0';
573 
574         //required number of free blocks found.
575         if(count == no_blocks)
576         {
577           break;
578         }
579       }
580     }
581 
582     //if no data block assigned.
583     if (data_blocks[0] == -1)
584     {
585       printf("not enough space.\n");
586       return;
587     }
588 
589     else
590     {
591       //if not enough data blocks availabe. reclaim alloted data blocks.
592       if (count < no_blocks)
593       {
594         printf("not enough space.\n");
595 
596         //reclaim allocated data blocks.
597         for (int i = 0; i < count; i++)
598         {
599           ((disk_block_ptr + 0) -> super_block_ptr).free_block_lst[data_blocks[i
    ]] = '1';
600           data_blocks[i] = -1;
601         }
602       }
603 
604       else
605       {
606         //find free inode
607         int inode = -1;
608         for(int i = 0; i < NO_INODES; i++)
609         {
610 
611           if(((disk_block_ptr + 0) -> super_block_ptr).inode_table[i].used == 0)
612           {
613             inode = i;
614             strcpy(((disk_block_ptr + 0) -> super_block_ptr).inode_table[inode].
    name, file_name);
615             ((disk_block_ptr + 0) -> super_block_ptr).inode_table[inode].dir = 0;
616             ((disk_block_ptr + 0) -> super_block_ptr).inode_table[inode].used =
    1;
617             ((disk_block_ptr + 0) -> super_block_ptr).inode_table[inode].size =
    file_size;
618             for (int j = 0; j < no_blocks; j++)
619             {
620               ((disk_block_ptr + 0) -> super_block_ptr).inode_table[inode].
    blockptrs[j] = data_blocks[j];
621             }
622             ((disk_block_ptr + 0) -> super_block_ptr).inode_table[inode].rsvd =
    0;
623             break;
624           }
625         }
626 
627         //free inode not found reclaim allocated data blocks.
628         if (inode == -1)
629         {
630           printf("not enough space.\n");
631 
```

```
632           for (int i = 0; i < count; i++)
633           {
634             ((disk_block_ptr + 0) -> super_block_ptr).free_block_lst[data_blocks[
    i]] = '1';
635             data_blocks[i] = -1;
636           }
637           return;
638         }
639
640         //assign inode in parent directory.
641         else
642         {
643           int assigned = 0;
644           int parent_block = ((disk_block_ptr + 0) -> super_block_ptr).
    inode_table[parent_inode_arr[path_length - 1]].blockptrs[0];
645
646           for (int i = 0; i < NO_INODES - 1; i++)
647           {
648             if((disk_block_ptr + parent_block) -> directory_table[i].inode == -1)
649             {
650               strcpy((disk_block_ptr + parent_block) -> directory_table[i].name,
    file_name);
651               (disk_block_ptr + parent_block) -> directory_table[i].inode = inode
    ;
652               (disk_block_ptr + parent_block) -> directory_table[i].namelen =
    strlen(file_name);
653
654               assigned = 1;
655               break;
656             }
657           }
658
659           //not enough space in directory
660           if(assigned == 0)
661           {
662             printf("not enough space\n");
663
664             //reclaim allocated inode;
665             ((disk_block_ptr + 0) -> super_block_ptr).inode_table[inode].used =
    0;
666
667             //reclaim allocated blocks
668             for (int i = 0; i < count; i++)
669             {
670               ((disk_block_ptr + 0) -> super_block_ptr).free_block_lst[
    data_blocks[i]] = '1';
671               data_blocks[i] = -1;
672             }
673             return;
674           }
675
676           //update sizes of all parent directories.
677           else
678           {
679             for(int i = 0; i < path_length; i++)
680             {
681               ((disk_block_ptr + 0) -> super_block_ptr).inode_table[
    parent_inode_arr[i]].size += file_size;
682             }
683           }
684         }
685       }
686     }
687   }
688   // printf("exiting\n");
689 }
690
691 void DL(block* disk_block_ptr, char* path)
692 {
```

```
693   printf("DL called, str:");
694   printf("%s\n", path);
695
696   int path_length = 0;
697
698   //find path length.
699   for (int i = 0; path[i]; i++)
700   {
701     if(path[i] == '/')
702     {
703       path_length++;
704     }
705   }
706
707   //store directories in path in an array.
708   int i = 0;
709   char* arr[path_length];
710   char* token = strtok(path, "/");
711   while (token != NULL)
712   {
713     arr[i++] = token;
714     token = strtok(NULL, "/");
715   }
716
717   //to delete file name
718   char* file_name = arr[path_length - 1];
719
720   // array to store inode of parents.
721   int parent_inode_arr[path_length];
722
723   verify_path_2(disk_block_ptr, arr, path_length, parent_inode_arr);
724
725   if (parent_inode_arr[0] == 0)
726   {
727     int inode_del = -1;
728
729     //go to parents data block.
730     int parent_inode = parent_inode_arr[path_length - 1];
731     int parent_block = ((disk_block_ptr + 0) -> super_block_ptr).inode_table[
      parent_inode].blockptrs[0];
732
733     //find file to delete in parent directory.
734     for (int i = 0; i < NO_INODES - 1; i++)
735     {
736       if ((disk_block_ptr + parent_block) -> directory_table[i].inode != -1 && !
      strcmp((disk_block_ptr + parent_block) -> directory_table[i].name, file_name)
      )
737       {
738         inode_del = (disk_block_ptr + parent_block) -> directory_table[i].inode;
739
740         //file found
741         if (((disk_block_ptr + 0) -> super_block_ptr).inode_table[inode_del].dir
      == 0)
742         {
743           (disk_block_ptr + parent_block) -> directory_table[i].inode = -1;
744           break;
745         }
746         else
747         {
748           inode_del = -1;
749         }
750       }
751     }
752
753     //if no such file exists in parent directory.
754     if (inode_del == -1)
755     {
756       printf("the file does not exist.\n");
757     }
```

```
758
759     else
760     {
761       //update all parent size;
762       int file_size = ((disk_block_ptr + 0) -> super_block_ptr).inode_table[
    inode_del].size;
763       for(int i = 0; i < path_length; i++)
764       {
765         ((disk_block_ptr + 0) -> super_block_ptr).inode_table[parent_inode_arr[i
    ]].size -= file_size;
766       }
767
768       //recliam allocated data blocks.
769       for (int i = 0; i < 8; i++)
770       {
771         if (((disk_block_ptr + 0) -> super_block_ptr).inode_table[inode_del].
    blockptrs[i] != -1)
772         {
773           // strcpy(((disk_block_ptr + ((disk_block_ptr + 0) -> super_block_ptr).
    inode_table[inode_del].blockptrs[i]) -> file_data), " ");
774
775           ((disk_block_ptr + 0) -> super_block_ptr).free_block_lst[((
    disk_block_ptr + 0) -> super_block_ptr).inode_table[inode_del].blockptrs[i]]
    = '1';
776
777           ((disk_block_ptr + 0) -> super_block_ptr).inode_table[inode_del].
    blockptrs[i] = -1;
778         }
779       }
780
781       //reclaim inode
782       ((disk_block_ptr + 0) -> super_block_ptr).inode_table[inode_del].used = 0;
783     }
784   }
785
786   //printf("exiting.\n");
787 }
788
789 void CP(block* disk_block_ptr, char* str, int mv)
790 {
791   printf("CP called, str:");
792   printf("%s\n", str);
793
794   char src[256];
795   char src_2[256];
796
797   char dst[256];
798   char dst_2[256];
799   char dst_3[256];
800
801   char* token = strtok(str, " ");
802   strcpy(src, token);
803   strcpy(src_2, token);
804
805   token = strtok(NULL, " ");
806   strcpy(dst, token);
807   strcpy(dst_2, token);
808   strcpy(dst_3, token);
809
810
811   if (!strcmp(src, dst))
812   {
813     return;
814   }
815
816   //find src path length.
817   int src_path_length = 0;
818   for (int i = 0; src[i]; i++)
819   {
```

```
820       if(src[i] == '/')
821       {
822         src_path_length++;
823       }
824     }
825
826     //find dst path length.
827     int dst_path_length = 0;
828     for (int i = 0; dst[i]; i++)
829     {
830       if(dst[i] == '/')
831       {
832         dst_path_length++;
833       }
834     }
835
836     //store directories of src path in an array.
837     int i = 0;
838     char* src_arr[src_path_length];
839     token = strtok(src, "/");
840     while (token != NULL)
841     {
842       src_arr[i++] = token;
843       token = strtok(NULL, "/");
844     }
845
846     //store directories of dst path in an array.
847     i = 0;
848     char* dst_arr[dst_path_length];
849     token = strtok(dst, "/");
850     while (token != NULL)
851     {
852       dst_arr[i++] = token;
853       token = strtok(NULL, "/");
854     }
855
856     char* src_file = src_arr[src_path_length - 1];
857     char* dst_file = dst_arr[dst_path_length - 1];
858
859     // array to store inode of parents for src and dst.
860     int src_parent_inode_arr[src_path_length];
861     int dst_parent_inode_arr[dst_path_length];
862
863     verify_path_2(disk_block_ptr, src_arr, src_path_length, src_parent_inode_arr);
864
865     if (src_parent_inode_arr[0] == -1)
866     {
867       return;
868     }
869     //if source path ok
870     else
871     {
872       //check if src file exists or is a dir.
873       int src_exists = 0;
874       int src_size = 0;
875       int src_parent_inode = src_parent_inode_arr[src_path_length - 1];
876       int src_parent_block = ((disk_block_ptr + 0) -> super_block_ptr).inode_table[
      src_parent_inode].blockptrs[0];
877
878       for (int i = 0; i < NO_INODES - 1; i++)
879       {
880         if((disk_block_ptr + src_parent_block) -> directory_table[i].inode != -1 &&
       !strcmp((disk_block_ptr + src_parent_block) -> directory_table[i].name,
      src_file))
881         {
882           //check if src is dir.
883           if(((disk_block_ptr + 0) -> super_block_ptr).inode_table[(disk_block_ptr
     + src_parent_block) -> directory_table[i].inode].dir == 1)
884           {
```

```
885           printf("can't handle directories.\n");
886           return;
887         }
888         else
889         {
890           src_exists = 1;
891           src_size = ((disk_block_ptr + 0) -> super_block_ptr).inode_table[(
    disk_block_ptr + src_parent_block) -> directory_table[i].inode].size;
892         }
893 
894         break;
895       }
896     }
897 
898     if(src_exists == 1)
899     {
900       verify_path_2(disk_block_ptr, dst_arr, dst_path_length,
    dst_parent_inode_arr);
901 
902       if (dst_parent_inode_arr[0] == -1)
903       {
904         return;
905       }
906       //if dst path ok
907       else
908       {
909         //check if dst file exits or is a dir.
910         int dst_exists = 0;
911         int dst_parent_inode = dst_parent_inode_arr[dst_path_length - 1];
912         int dst_parent_block = ((disk_block_ptr + 0) -> super_block_ptr).
    inode_table[dst_parent_inode].blockptrs[0];
913 
914         for (int i = 0; i < NO_INODES - 1; i++)
915         {
916           if ((disk_block_ptr + dst_parent_block) -> directory_table[i].inode !=
    -1 && !strcmp((disk_block_ptr + dst_parent_block) -> directory_table[i].name,
     dst_file))
917           {
918             //check if dst is dir.
919             if(((disk_block_ptr + 0) -> super_block_ptr).inode_table[(
    disk_block_ptr + dst_parent_block) -> directory_table[i].inode].dir == 1)
920             {
921               printf("can't handle directories.\n");
922               return;
923             }
924             else
925             {
926               dst_exists = 1;
927             }
928 
929             break;
930           }
931         }
932 
933         //overwrite
934         if (dst_exists == 1)
935         {
936           DL(disk_block_ptr, dst_3);
937         }
938 
939         char buffer[20];
940         sprintf(buffer, "%d", src_size);
941 
942         strcat(dst_2, " ");
943         strcat(dst_2, buffer);
944 
945         CR(disk_block_ptr, dst_2);
946 
947         //if move file called. DL src file.
```

```
948         if (mv == 1)
949         {
950           DL(disk_block_ptr, src_2);
951         }
952       }
953     }
954     else
955     {
956       printf("source file does not exist.\n");
957     }
958   }
959 }
960
961 void CD(block* disk_block_ptr, char* path)
962 {
963   printf("CD called, str:");
964   printf("%s\n", path);
965
966   int path_length = 0;
967
968   //find path length.
969   for (int i = 0; path[i]; i++)
970   {
971     if(path[i] == '/')
972     {
973       path_length++;
974     }
975   }
976
977   //store directories in path in an array.
978   int i = 0;
979   char* arr[path_length];
980   char* token = strtok(path, "/");
981   while (token != NULL)
982   {
983     arr[i++] = token;
984     token = strtok(NULL, "/");
985   }
986
987   //new dir name
988   char* dir_name = arr[path_length - 1];
989
990   // array to store inode of parents.
991   int parent_inode_arr[path_length];
992
993   verify_path(disk_block_ptr, arr, path_length, parent_inode_arr);
994
995   //do only if path was ok and dir does not already exist.
996   if(parent_inode_arr[0] == 0)
997   {
998     //find free block(s).
999     int data_block = -1;
1000    for(int i = 0; i < NO_BLOCKS; i++)
1001    {
1002      if(((disk_block_ptr + 0) -> super_block_ptr).free_block_lst[i] == '1')
1003      {
1004        data_block = i;
1005        ((disk_block_ptr + 0) -> super_block_ptr).free_block_lst[data_block] = '0
     ';
1006        break;
1007      }
1008    }
1009
1010    //if not enough data blocks availabe.
1011    if (data_block == -1)
1012    {
1013      printf("not enough space.\n");
1014      return;
1015    }
```

```
1016      else
1017      {
1018        //find free inode
1019        int inode = -1;
1020        for(int i = 0; i < NO_INODES; i++)
1021        {
1022          if(((disk_block_ptr + 0) -> super_block_ptr).inode_table[i].used == 0)
1023          {
1024            inode = i;
1025            strcpy(((disk_block_ptr + 0) -> super_block_ptr).inode_table[inode].
    name, dir_name);
1026            ((disk_block_ptr + 0) -> super_block_ptr).inode_table[inode].dir = 1;
1027            ((disk_block_ptr + 0) -> super_block_ptr).inode_table[inode].used = 1;
1028            ((disk_block_ptr + 0) -> super_block_ptr).inode_table[inode].size = 0;
1029            ((disk_block_ptr + 0) -> super_block_ptr).inode_table[inode].blockptrs
    [0] = data_block;
1030            ((disk_block_ptr + 0) -> super_block_ptr).inode_table[inode].rsvd = 0;
1031            break;
1032          }
1033        }
1034
1035        //free inode not found reclaim allocated data block.
1036        if (inode == -1)
1037        {
1038          printf("not enough space.\n");
1039
1040          ((disk_block_ptr + 0) -> super_block_ptr).free_block_lst[data_block] = '1
    ';
1041        }
1042
1043        //assign inode in parent directory.
1044        else
1045        {
1046          int assigned = 0;
1047          int parent_block = ((disk_block_ptr + 0) -> super_block_ptr).inode_table[
    parent_inode_arr[path_length - 1]].blockptrs[0];
1048
1049          for (int i = 0; i < NO_INODES - 1; i++)
1050          {
1051            if((disk_block_ptr + parent_block) -> directory_table[i].inode == -1)
1052            {
1053              strcpy((disk_block_ptr + parent_block) -> directory_table[i].name,
    dir_name);
1054              (disk_block_ptr + parent_block) -> directory_table[i].inode = inode;
1055              (disk_block_ptr + parent_block) -> directory_table[i].namelen =
    strlen(dir_name);
1056
1057              assigned = 1;
1058              break;
1059            }
1060          }
1061
1062          //not enough space in parent directory
1063          if(assigned == 0)
1064          {
1065            printf("not enough space\n");
1066
1067            //reclaim allocated inode;
1068            ((disk_block_ptr + 0) -> super_block_ptr).inode_table[inode].used = 0;
1069
1070            //reclaim allocated block
1071            ((disk_block_ptr + 0) -> super_block_ptr).free_block_lst[data_block] =
    '1';
1072          }
1073
1074          //update sizes of all parent directories.
1075          else
1076          {
```

```
1077            //update size loop doesn't run if new file in root.(e.g. pathlenght =
    1)
1078            for(int i = 0; i < path_length; i++)
1079            {
1080              ((disk_block_ptr + 0) -> super_block_ptr).inode_table[
    parent_inode_arr[i]].size += 1024;
1081            }
1082          }
1083        }
1084      }
1085    }
1086
1087    // printf("exiting\n");
1088  }
1089
1090  void DD(block* disk_block_ptr, char* path)
1091  {
1092    printf("DD called, str:");
1093    printf("%s\n", path);
1094
1095    char path_save[256];
1096    char path_send[256];
1097
1098    //save path for recursion.
1099    strcpy(path_save, path);
1100
1101    int path_length = 0;
1102
1103    //find path length.
1104    for (int i = 0; path[i]; i++)
1105    {
1106      if(path[i] == '/')
1107      {
1108        path_length++;
1109      }
1110    }
1111
1112    //store directories in path in an array.
1113    int i = 0;
1114    char* arr[path_length];
1115    char* token = strtok(path, "/");
1116    while (token != NULL)
1117    {
1118      arr[i++] = token;
1119      token = strtok(NULL, "/");
1120    }
1121
1122    //to delete dir name
1123    char* dir_name = arr[path_length - 1];
1124
1125    // array to store inode of parents.
1126    int parent_inode_arr[path_length];
1127
1128    verify_path_2(disk_block_ptr, arr, path_length, parent_inode_arr);
1129
1130    if (parent_inode_arr[0] == 0)
1131    {
1132      //go to parents data block.
1133      int parent_inode = parent_inode_arr[path_length - 1];
1134      int parent_block = ((disk_block_ptr + 0) -> super_block_ptr).inode_table[
    parent_inode].blockptrs[0];
1135
1136      int del_inode = -1;
1137
1138      int un_link = -1;
1139
1140      //find inode of dir to delete.
1141      for (int i = 0; i < NO_INODES -1; i++)
1142      {
```

```
1143        if((disk_block_ptr + parent_block) -> directory_table[i].inode != -1 && !
    strcmp((disk_block_ptr + parent_block) -> directory_table[i].name, dir_name))
1144        {
1145          del_inode = (disk_block_ptr + parent_block) -> directory_table[i].inode;
1146          un_link = i;
1147
1148          //dir inode found
1149          if(((disk_block_ptr + 0) -> super_block_ptr).inode_table[del_inode].dir
    == 1)
1150          {
1151            break;
1152          }
1153          //dir inode not found
1154          else
1155          {
1156            del_inode = -1;
1157          }
1158        }
1159      }
1160      //dir does not exit
1161      if (del_inode == -1)
1162      {
1163        printf("the directory does not exist.\n");
1164        return;
1165      }
1166
1167      else
1168      {
1169        //go to dir block
1170        int dir_block = ((disk_block_ptr + 0) -> super_block_ptr).inode_table[
    del_inode].blockptrs[0];
1171        int child_del_inode = -1;
1172        //traverse over dir contents
1173        for (int i = 0; i < NO_INODES -1; i++)
1174        {
1175          if ((disk_block_ptr + dir_block) -> directory_table[i].inode != -1)
1176          {
1177            child_del_inode = (disk_block_ptr + dir_block) -> directory_table[i].
    inode;
1178
1179            //create path
1180            strcpy(path_send, path_save);
1181            strcat(path_send, "/");
1182            strcat(path_send, ((disk_block_ptr + 0) -> super_block_ptr).inode_table
    [child_del_inode].name);
1183
1184            //delete file.
1185            if (((disk_block_ptr + 0) -> super_block_ptr).inode_table[
    child_del_inode].dir == 0)
1186            {
1187              DL(disk_block_ptr, path_send);
1188            }
1189
1190            //delete dir
1191            else
1192            {
1193              DD(disk_block_ptr, path_send);
1194            }
1195
1196            (disk_block_ptr + dir_block) -> directory_table[i].inode = -1;
1197            child_del_inode = -1;
1198          }
1199        }
1200
1201        // printf("del_inode:%d\n", del_inode);
1202
1203        //unlink dir from parrent.
1204        (disk_block_ptr + parent_block) -> directory_table[un_link].inode = -1;
1205
```

```
1206        //free dir block.
1207        ((disk_block_ptr + 0) -> super_block_ptr).free_block_lst[dir_block] = '1';
1208
1209        //free dir inode
1210        ((disk_block_ptr + 0) -> super_block_ptr).inode_table[del_inode].used = 0;
1211
1212        //update all parent sizes.
1213        for (int i = 0; i < path_length; i++)
1214        {
1215          ((disk_block_ptr + 0) -> super_block_ptr).inode_table[parent_inode_arr[i
      ]].size -= 1024;
1216        }
1217      }
1218    }
1219  }
1220
1221  void LL(block* disk_block_ptr, int block)
1222  {
1223    printf("LL called\n");
1224
1225    //traverse inodes in use in superblock.
1226    for (int i = 0; i < NO_INODES; i++)
1227    {
1228      if(((disk_block_ptr + 0) -> super_block_ptr).inode_table[i].used == 1)
1229      {
1230        printf("%s ", ((disk_block_ptr + 0) -> super_block_ptr).inode_table[i].name
      );
1231        printf("%d\n", ((disk_block_ptr + 0) -> super_block_ptr).inode_table[i].
      size);
1232      }
1233    }
1234  }
```

Listing 1: filesystem.c