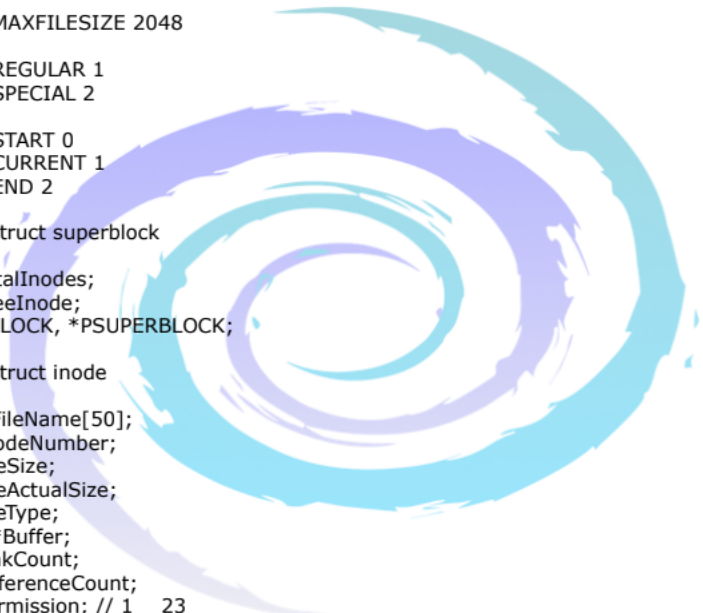


Customised Dynamic File System

```
1. #include<stdio.h>
2. #include<stdlib.h>
3. #include<string.h>
4. #include<unistd.h>
5. #include<iostream>
6. // #include<io.h>
7.
8. #define MAXINODE 50
9.
10. #define READ 1
11. #define WRITE 2
12.
13. #define MAXFILESIZE 2048
14.
15. #define REGULAR 1
16. #define SPECIAL 2
17.
18. #define START 0
19. #define CURRENT 1
20. #define END 2
21.
22. typedef struct superblock
23. {
24.     int TotalInodes;
25.     int FreeInode;
26. }SUPERBLOCK, *PSUPERBLOCK;
27.
28. typedef struct inode
29. {
30.     char FileName[50];
31.     int InodeNumber;
32.     int FileSize;
33.     int FileActualSize;
34.     int FileType;
35.     char *Buffer;
36.     int LinkCount;
37.     int ReferenceCount;
38.     int permission; // 1 2 3
39.     struct inode *next;
40. }INODE, *PINODE, **PPINODE;
41.
42. typedef struct filetable
43. {
44.     int readoffset;
45.     int writeoffset;
46.     int count;
47.     int mode; // 1 2 3
48.     PINODE ptrinode;
49. }FILETABLE, *PFILETABLE;
50.
51. typedef struct ufdt
52. {
53.     PFILETABLE ptrfiletable;
54. }UFDT;
55.
56. UFDT UFDTArr[50];
```



```

57.SUPERBLOCK SUPERBLOCKobj;
58.PINODE head = NULL;

59.
60.void man(char *name)
61.{
62.    if(name == NULL) return;
63.
64.    if(strcmp(name,"create") == 0)
65.    {
66.        printf("Description : Used to create new regular file\n");
67.        printf("Usage : create File_name Permission\n");
68.    }
69.    else if(strcmp(name,"read") == 0)
70.    {
71.        printf("Description : Used to read data from regular file\n");
72.        printf("Usage : read File_name No_Of_Bytes_To_Read\n");
73.    }
74.    else if(strcmp(name,"write") == 0)
75.    {
76.        printf("Description : Used to write into regular file\n");
77.        printf("Usage : write File_name\n After this enter the data that we want to write\n");
78.    }
79.    else if(strcmp(name,"ls") == 0)
80.    {
81.        printf("Description : Used to list all information of files\n");
82.        printf("Usage : ls\n");
83.    }
84.    else if(strcmp(name,"stat") == 0)
85.    {
86.        printf("Description : Used to display information of file\n");
87.        printf("Usage : stat File_name\n");
88.    }
89.    else if(strcmp(name,"fstat") == 0)
90.    {
91.        printf("Description : Used to display information of file\n");
92.        printf("Usage : stat File_Descriptor\n");
93.    }
94.    else if(strcmp(name,"truncate") == 0)
95.    {
96.        printf("Description : Used to remove data from file\n");
97.        printf("Usage : truncate File_name\n");
98.    }
99.    else if(strcmp(name,"open") == 0)
100.    {
101.        printf("Description : Used to open existing file\n");
102.        printf("Usage : open File_name mode\n");
103.    }
104.    else if(strcmp(name,"close") == 0)
105.    {
106.        printf("Description : Used to close opened file\n");
107.        printf("Usage : close File_name\n");
108.    }
109.    else if(strcmp(name,"closeall") == 0)
110.    {
111.        printf("Description : Used to close all opened file\n");
112.        printf("Usage : closeall\n");
113.    }
114.    else if(strcmp(name,"lseek") == 0)
115.    {

```

```

116.         printf("Description : Used to change file offset\n");
117.         printf("Usage : lseek File_Name ChangeInOffset StartPoint\n");
118.     }
119. else if(strcmp(name,"rm") == 0)
120. {
121.     printf("Description : Used to delete the file\n");
122.     printf("Usage : rm File_Name\n");
123. }
124. else
125. {
126.     printf("ERROR : No manual entry available.\n");
127. }
128.}
129.
130.void DisplayHelp()
131.{
132.    printf("ls : To List out all files\n");
133.    printf("clear : To clear console\n");
134.    printf("open : To open the file\n");
135.    printf("close : To close the file\n");
136.    printf("closeall : To close all opened files\n");
137.    printf("read : To Read the contents from file\n");
138.    printf("write :To Write contents into file\n");
139.    printf("exit : To Terminate file system\n");
140.    printf("stat : To Display information of file using name\n");
141.    printf("fstat :To Display information of file using file descriptor\n");
142.    printf("truncate : To Remove all data from file\n");
143.    printf("rm : To Delet the file\n");
144.}
145.
146.int GetFDFromName(char *name)
147.{
148.    int i = 0;
149.
150.    while(i<50)
151.    {
152.        if(UFDtArr[i].ptrfiletable != NULL)
153.            if(strcmp((UFDtArr[i].ptrfiletable->ptrinode->FileName),name)==0)
154.                break;
155.        i++;
156.    }
157.
158.    if(i == 50)        return -1;
159.    else                return i;
160.}
161.
162.PINODE Get_Inode(char * name)
163.{
164.    PINODE temp = head;
165.    int i = 0;
166.
167.    if(name == NULL)
168.        return NULL;
169.
170.    while(temp!= NULL)
171.    {
172.        if(strcmp(name,temp->FileName) == 0)
173.            break;
174.        temp = temp->next;
175.    }

```

```

176. return temp;
177.}
178.
179.void CreateDILB()
180.{
181.    int i = 1;
182.    PINODE newn = NULL;
183.    PINODE temp = head;
184.
185.    while(i<= MAXINODE)
186.    {
187.        newn = (PINODE)malloc(sizeof(INODE));
188.
189.        newn->LinkCount = 0;
190.        newn->ReferenceCount = 0;
191.        newn->FileType = 0;
192.        newn->FileSize = 0;
193.
194.        newn->Buffer = NULL;
195.        newn->next = NULL;
196.
197.        newn->InodeNumber = i;
198.
199.        if(temp == NULL)
200.        {
201.            head = newn;
202.            temp = head;
203.        }
204.        else
205.        {
206.            temp->next = newn;
207.            temp = temp->next;
208.        }
209.        i++;
210.    }
211.    printf("DILB created successfully\n");
212.}
213.
214.void InitialiseSuperBlock()
215.{
216.    int i = 0;
217.    while(i< MAXINODE)
218.    {
219.        UFDTArr[i].ptrfiletable = NULL;
220.        i++;
221.    }
222.
223.    SUPERBLOCKObj.TotalInodes = MAXINODE;
224.    SUPERBLOCKObj.FreeInode = MAXINODE;
225.}
226.
227.int CreateFile(char *name,int permission)
228.{
229.    int i = 0;
230.    PINODE temp = head;
231.
232.    if((name == NULL) || (permission == 0) || (permission > 3))
233.        return -1;
234.
235.    if(SUPERBLOCKObj.FreeInode == 0)

```

```

236.         return -2;
237.
238.     (SUPERBLOCKObj.FreeInode)--;
239.
240.     if(Get_Inode(name) != NULL)
241.         return -3;
242.
243.     while(temp!= NULL)
244.     {
245.         if(temp->FileType == 0)
246.             break;
247.         temp=temp->next;
248.     }
249.
250.     while(i<50)
251.     {
252.         if(UFDTArr[i].ptrfiletable == NULL)
253.             break;
254.         i++;
255.     }
256.
257.     UFDTArr[i].ptrfiletable = (PFILETABLE)malloc(sizeof(FILETABLE));
258.
259.     UFDTArr[i].ptrfiletable->count = 1;
260.     UFDTArr[i].ptrfiletable->mode = permission;
261.     UFDTArr[i].ptrfiletable->readoffset = 0;
262.     UFDTArr[i].ptrfiletable->writeoffset = 0;
263.
264.     UFDTArr[i].ptrfiletable->ptrinode = temp;
265.
266.     strcpy(UFDTArr[i].ptrfiletable->ptrinode->FileName,name);
267.     UFDTArr[i].ptrfiletable->ptrinode->FileType = REGULAR;
268.     UFDTArr[i].ptrfiletable->ptrinode->ReferenceCount = 1;
269.     UFDTArr[i].ptrfiletable->ptrinode->LinkCount = 1;
270.     UFDTArr[i].ptrfiletable->ptrinode->FileSize = MAXFILESIZE;
271.     UFDTArr[i].ptrfiletable->ptrinode->FileActualSize = 0;
272.     UFDTArr[i].ptrfiletable->ptrinode->permission = permission;
273.     UFDTArr[i].ptrfiletable->ptrinode->Buffer = (char *)malloc(MAXFILESIZE);
274.
275.     return i;
276. }
277.
278. // rm_File("Demo.txt")
279. int rm_File(char * name)
280. {
281.     int fd = 0;
282.
283.     fd = GetFDFromName(name);
284.     if(fd == -1)
285.         return -1;
286.
287.     (UFDTArr[fd].ptrfiletable->ptrinode->LinkCount)--;
288.
289.     if(UFDTArr[fd].ptrfiletable->ptrinode->LinkCount == 0)
290.     {
291.         UFDTArr[fd].ptrfiletable->ptrinode->FileType = 0;
292.         //free(UFDTArr[fd].ptrfiletable->ptrinode->Buffer);
293.         free(UFDTArr[fd].ptrfiletable);
294.     }
295.

```

```

296. UFDTArr[fd].ptrfiletable = NULL;
297. (SUPERBLOCKobj.FreeInode)++;
298.}
299.
300.int ReadFile(int fd, char *arr, int isize)
301.{
302. int read_size = 0;
303.
304. if(UFDTArr[fd].ptrfiletable == NULL) return -1;
305.
306. if(UFDTArr[fd].ptrfiletable->mode !=READ && UFDTArr[fd].ptrfiletable->mode !=READ+WRITE)
    return -2;
307.
308. if(UFDTArr[fd].ptrfiletable->ptrinode->permission != READ && UFDTArr[fd].ptrfiletable-
    >ptrinode->permission != READ+WRITE) return -2;
309.
310. if(UFDTArr[fd].ptrfiletable->readoffset == UFDTArr[fd].ptrfiletable->ptrinode->FileActualSize)
    return -3;
311.
312. if(UFDTArr[fd].ptrfiletable->ptrinode->FileType != REGULAR) return -4;
313.
314. read_size = (UFDTArr[fd].ptrfiletable->ptrinode->FileActualSize) - (UFDTArr[fd].ptrfiletable-
    >readoffset);
315. if(read_size < isize)
316. {
317.     strncpy(arr,(UFDTArr[fd].ptrfiletable->ptrinode->Buffer) + (UFDTArr[fd].ptrfiletable-
        >readoffset),read_size);
318.
319.     UFDTArr[fd].ptrfiletable->readoffset = UFDTArr[fd].ptrfiletable->readoffset + read_size;
320. }
321. else
322. {
323.     strncpy(arr,(UFDTArr[fd].ptrfiletable->ptrinode->Buffer) + (UFDTArr[fd].ptrfiletable-
        >readoffset),isize);
324.
325.     (UFDTArr[fd].ptrfiletable->readoffset) = (UFDTArr[fd].ptrfiletable->readoffset) + isize;
326. }
327.
328. return isize;
329.}
330.
331.int WriteFile(int fd, char *arr, int isize)
332.{
333. if(((UFDTArr[fd].ptrfiletable->mode) !=WRITE) && ((UFDTArr[fd].ptrfiletable->mode) !
    =READ+WRITE))return -1;
334.
335. if(((UFDTArr[fd].ptrfiletable->ptrinode->permission) !=WRITE) && ((UFDTArr[fd].ptrfiletable-
    >ptrinode->permission) != READ+WRITE)) return -1;
336.
337. if((UFDTArr[fd].ptrfiletable->writeoffset) == MAXFILESIZE) return -2;
338.
339. if((UFDTArr[fd].ptrfiletable->ptrinode->FileType) != REGULAR) return -3;
340.
341. strncpy((UFDTArr[fd].ptrfiletable->ptrinode->Buffer) + (UFDTArr[fd].ptrfiletable-
    >writeoffset),arr,isize);
342.
343. (UFDTArr[fd].ptrfiletable->writeoffset) = (UFDTArr[fd].ptrfiletable->writeoffset) + isize;
344.
345. (UFDTArr[fd].ptrfiletable->ptrinode->FileActualSize) = (UFDTArr[fd].ptrfiletable->ptrinode-
    >FileActualSize) + isize;

```



```

346.
347. return isize;
348.}
349.
350.int OpenFile(char *name, int mode)
351.{
352.    int i = 0;
353.    PINODE temp = NULL;
354.
355.    if(name == NULL || mode <= 0)
356.        return -1;
357.
358.    temp = Get_Inode(name);
359.    if(temp == NULL)
360.        return -2;
361.
362.    if(temp->permission < mode)
363.        return -3;
364.
365.    while(i<50)
366.    {
367.        if(UFDTErr[i].ptrfiletable == NULL)
368.            break;
369.        i++;
370.    }
371.
372.    UFDTErr[i].ptrfiletable = (PFILETABLE)malloc(sizeof(FILETABLE));
373.    if(UFDTErr[i].ptrfiletable == NULL) return -1;
374.    UFDTErr[i].ptrfiletable->count = 1;
375.    UFDTErr[i].ptrfiletable->mode = mode;
376.    if(mode == READ + WRITE)
377.    {
378.        UFDTErr[i].ptrfiletable->readoffset = 0;
379.        UFDTErr[i].ptrfiletable->writeoffset = 0;
380.    }
381.    else if(mode == READ)
382.    {
383.        UFDTErr[i].ptrfiletable->readoffset = 0;
384.    }
385.    else if(mode == WRITE)
386.    {
387.        UFDTErr[i].ptrfiletable->writeoffset = 0;
388.    }
389.    UFDTErr[i].ptrfiletable->ptrinode = temp;
390.    (UFDTErr[i].ptrfiletable->ptrinode->ReferenceCount)++;
391.
392.    return i;
393.}
394.
395.void CloseFileByName(int fd)
396.{
397.    UFDTErr[fd].ptrfiletable->readoffset = 0;
398.    UFDTErr[fd].ptrfiletable->writeoffset = 0;
399.    (UFDTErr[fd].ptrfiletable->ptrinode->ReferenceCount)--;
400.}
401.
402.int CloseFileByName(char *name)
403.{
404.    int i = 0;
405.    i = GetFDFromName(name);

```

```

406. if(i == -1)
407.     return -1;
408.
409. UFDTArr[i].ptrfiletable->readoffset = 0;
410. UFDTArr[i].ptrfiletable->writeoffset = 0;
411. (UFDTArr[i].ptrfiletable->ptrnode->ReferenceCount)--;
412.
413. return 0;
414.}
415.
416.void CloseAllFile()
417.{
418.    int i = 0;
419.    while(i<50)
420.    {
421.        if(UFDTArr[i].ptrfiletable != NULL)
422.        {
423.            UFDTArr[i].ptrfiletable->readoffset = 0;
424.            UFDTArr[i].ptrfiletable->writeoffset = 0;
425.            (UFDTArr[i].ptrfiletable->ptrnode->ReferenceCount)--;
426.            break;
427.        }
428.        i++;
429.    }
430.}
431.
432.int LseekFile(int fd, int size, int from)
433.{
434.    if((fd<0) || (from > 2)) return -1;
435.    if(UFDTArr[fd].ptrfiletable == NULL) return -1;
436.
437.    if((UFDTArr[fd].ptrfiletable->mode == READ) || (UFDTArr[fd].ptrfiletable->mode ==
        READ+WRITE))
438.    {
439.        if(from == CURRENT)
440.        {
441.            if(((UFDTArr[fd].ptrfiletable->readoffset) + size) > UFDTArr[fd].ptrfiletable-
                >ptrnode->FileActualSize) return -1;
442.            if(((UFDTArr[fd].ptrfiletable->readoffset) + size) < 0) return -1;
443.            (UFDTArr[fd].ptrfiletable->readoffset) = (UFDTArr[fd].ptrfiletable->readoffset) +
                size;
444.        }
445.        else if(from == START)
446.        {
447.            if(size > (UFDTArr[fd].ptrfiletable->ptrnode->FileActualSize)) return -1;
448.            if(size < 0) return -1;
449.            (UFDTArr[fd].ptrfiletable->readoffset) = size;
450.        }
451.        else if(from == END)
452.        {
453.            if((UFDTArr[fd].ptrfiletable->ptrnode->FileActualSize) + size > MAXFILESIZE)
                return -1;
454.            if(((UFDTArr[fd].ptrfiletable->readoffset) + size) < 0) return -1;
455.            (UFDTArr[fd].ptrfiletable->readoffset) = (UFDTArr[fd].ptrfiletable->ptrnode-
                >FileActualSize) + size;
456.        }
457.    }
458.    else if(UFDTArr[fd].ptrfiletable->mode == WRITE)
459.    {
460.        if(from == CURRENT)

```



```

461.     {
462.         if(((UFDTArr[fd].ptrfiletable->writeoffset) + size) > MAXFILESIZE) return -1;
463.         if(((UFDTArr[fd].ptrfiletable->writeoffset) + size) < 0) return -1;
464.         if(((UFDTArr[fd].ptrfiletable->writeoffset) + size) > (UFDTArr[fd].ptrfiletable-
>ptrinode->FileActualSize))
465.             (UFDTArr[fd].ptrfiletable->ptrinode->FileActualSize) =
(UFDTArr[fd].ptrfiletable->writeoffset) + size;
466.         (UFDTArr[fd].ptrfiletable->writeoffset) = (UFDTArr[fd].ptrfiletable->writeoffset) +
size;
467.     }
468.     else if(from == START)
469.     {
470.         if(size > MAXFILESIZE) return -1;
471.         if(size < 0) return -1;
472.         if(size > (UFDTArr[fd].ptrfiletable->ptrinode->FileActualSize))
473.             (UFDTArr[fd].ptrfiletable->ptrinode->FileActualSize) = size;
474.         (UFDTArr[fd].ptrfiletable->writeoffset) = size;
475.     }
476.     else if(from == END)
477.     {
478.         if(((UFDTArr[fd].ptrfiletable->ptrinode->FileActualSize) + size > MAXFILESIZE)
return -1;
479.         if(((UFDTArr[fd].ptrfiletable->writeoffset) + size) < 0) return -1;
480.         (UFDTArr[fd].ptrfiletable->writeoffset) = (UFDTArr[fd].ptrfiletable->ptrinode-
>FileActualSize) + size;
481.     }
482. }
483.}
484.
485.void ls_file()
486.{
487.    int i = 0;
488.    PINODE temp = head;
489.
490.    if(SUPERBLOCKObj.FreeInode == MAXINODE)
491.    {
492.        printf("Error : There are no files\n");
493.        return;
494.    }
495.
496.    printf("\nFile Name\tInode number\tFile size\tLink count\n");
497.    printf("-----\n");
498.    while(temp != NULL)
499.    {
500.        if(temp->FileType != 0)
501.        {
502.            printf("%s\t\t%d\t\t%d\t\t%d\n",temp->FileName,temp-
>InodeNumber,temp->FileActualSize,temp->LinkCount);
503.        }
504.        temp = temp->next;
505.    }
506.    printf("-----\n");
507.}
508.
509.int fstat_file(int fd)
510.{
511.    PINODE temp = head;
512.    int i = 0;
513.
514.    if(fd < 0) return -1;

```

```

515.
516. if(UFDTErr[fd].ptrfiletable == NULL) return -2;
517.
518. temp = UFDTErr[fd].ptrfiletable->ptrinode;
519.
520. printf("\n-----Statistical Information about file-----\n");
521. printf("File name : %s\n",temp->FileName);
522. printf("Inode Number %d\n",temp->InodeNumber);
523. printf("File size : %d\n",temp->FileSize);
524. printf("Actual File size : %d\n",temp->FileActualSize);
525. printf("Link count : %d\n",temp->LinkCount);
526. printf("Reference count : %d\n",temp->ReferenceCount);
527.
528. if(temp->permission == 1)
529.     printf("File Permission : Read only\n");
530. else if(temp->permission == 2)
531.     printf("File Permission : Write\n");
532. else if(temp->permission == 3)
533.     printf("File Permission : Read & Write\n");
534. printf("-----\n\n");
535.
536. return 0;
537.}
538.
539.int stat_file(char *name)
540.{
541. PINODE temp = head;
542. int i = 0;
543.
544. if(name == NULL) return -1;
545.
546. while(temp!= NULL)
547. {
548.     if(strcmp(name,temp->FileName) == 0)
549.         break;
550.     temp = temp->next;
551. }
552.
553. if(temp == NULL) return -2;
554.
555. printf("\n-----Statistical Information about file-----\n");
556. printf("File name : %s\n",temp->FileName);
557. printf("Inode Number %d\n",temp->InodeNumber);
558. printf("File size : %d\n",temp->FileSize);
559. printf("Actual File size : %d\n",temp->FileActualSize);
560. printf("Link count : %d\n",temp->LinkCount);
561. printf("Reference count : %d\n",temp->ReferenceCount);
562.
563. if(temp->permission == 1)
564.     printf("File Permission : Read only\n");
565. else if(temp->permission == 2)
566.     printf("File Permission : Write\n");
567. else if(temp->permission == 3)
568.     printf("File Permission : Read & Write\n");
569. printf("-----\n\n");
570.
571. return 0;
572.}
573.
574.int truncate_File(char *name)

```

```

575.{
576. int fd = GetFDFromName(name);
577. if(fd == -1)
578.     return -1;
579.
580. memset(UFDTErr[fd].ptrfiletable->ptrinode->Buffer,0,1024);
581. UFDTErr[fd].ptrfiletable->readoffset = 0;
582. UFDTErr[fd].ptrfiletable->writeoffset = 0;
583. UFDTErr[fd].ptrfiletable->ptrinode->FileActualSize = 0;
584.}
585.
586.int main()
587.{
588. char *ptr = NULL;
589. int ret = 0, fd = 0, count = 0;
590. char command[4][80], str[80], arr[1024];
591.
592. InitialiseSuperBlock();
593. CreateDILB();
594.
595. while(1)
596. {
597.     fflush(stdin);
598.     strcpy(str,"");
599.
600.     printf("\nMarvellous VFS : > ");
601.
602.     fgets(str,80,stdin); // scanf("%[^'\n']s",str);
603.
604.     count = sscanf(str,"%s %s %s %s",command[0],command[1],command[2],command[3]);
605.
606.     if(count == 1)
607.     {
608.         if(strcmp(command[0],"ls") == 0)
609.         {
610.             ls_file();
611.         }
612.         else if(strcmp(command[0],"closeall") == 0)
613.         {
614.             CloseAllFile();
615.             printf("All files closed successfully\n");
616.             continue;
617.         }
618.         else if(strcmp(command[0],"clear") == 0)
619.         {
620.             system("cls");
621.             continue;
622.         }
623.         else if(strcmp(command[0],"help") == 0)
624.         {
625.             DisplayHelp();
626.             continue;
627.         }
628.         else if(strcmp(command[0],"exit") == 0)
629.         {
630.             printf("Terminating the Marvellous Virtual File System\n");
631.             break;
632.         }
633.         else
634.         {

```

```

635.         printf("\nERROR : Command not found !!!\n");
636.         continue;
637.     }
638. }
639. else if(count == 2)
640. {
641.     if(strcmp(command[0],"stat") == 0)
642.     {
643.         ret = stat_file(command[1]);
644.         if(ret == -1)
645.             printf("ERROR : Incorrect parameters\n");
646.         if(ret == -2)
647.             printf("ERROR : There is no such file\n");
648.         continue;
649.     }
650.     else if(strcmp(command[0],"fstat") == 0)
651.     {
652.         ret = fstat_file(atoi(command[1]));
653.         if(ret == -1)
654.             printf("ERROR : Incorrect parameters\n");
655.         if(ret == -2)
656.             printf("ERROR : There is no such file\n");
657.         continue;
658.     }
659.     else if(strcmp(command[0],"close") == 0)
660.     {
661.         ret = CloseFileByName(command[1]);
662.         if(ret == -1)
663.             printf("ERROR : There is no such file\n");
664.         continue;
665.     }
666.     else if(strcmp(command[0],"rm") == 0)
667.     {
668.         ret = rm_File(command[1]);
669.         if(ret == -1)
670.             printf("ERROR : There is no such file\n");
671.         continue;
672.     }
673.     else if(strcmp(command[0],"man") == 0)
674.     {
675.         man(command[1]);
676.     }
677.     else if(strcmp(command[0],"write") == 0)
678.     {
679.         fd = GetFDFromName(command[1]);
680.         if(fd == -1)
681.         {
682.             printf("Error : Incorrect parameter\n");
683.             continue;
684.         }
685.         printf("Enter the data : \n");
686.         scanf("%s",arr);
687.
688.         ret = strlen(arr);
689.         if(ret == 0)
690.         {
691.             printf("Error : Incorrect parameter\n");
692.             continue;
693.         }

```

```

694.         ret = WriteFile(fd,arr,ret);
695.         if(ret == -1)
696.             printf("ERROR : Permission denied\n");
697.         if(ret == -2)
698.             printf("ERROR : There is no sufficient memory to write\n");
699.         if(ret == -3)
700.             printf("ERROR : It is not regular file\n");
701.     }
702.     else if(strcmp(command[0],"truncate") == 0)
703.     {
704.         ret = truncate_File(command[1]);
705.         if(ret == -1)
706.             printf("Error : Incorrect parameter\n");
707.     }
708.     else
709.     {
710.         printf("\nERROR : Command not found !!!\n");
711.         continue;
712.     }
713. }
714. else if(count == 3)
715. {
716.     if(strcmp(command[0],"create") == 0)
717.     {
718.         ret = CreateFile(command[1],atoi(command[2]));
719.         if(ret >= 0)
720.             printf("File is successfully created with file descriptor : %d\n",ret);
721.         if(ret == -1)
722.             printf("ERROR : Incorrect parameters\n");
723.         if(ret == -2)
724.             printf("ERROR : There is no inodes\n");
725.         if(ret == -3)
726.             printf("ERROR : File already exists\n");
727.         if(ret == -4)
728.             printf("ERROR : Memory allocation failure\n");
729.         continue;
730.     }
731.     else if(strcmp(command[0],"open") == 0)
732.     {
733.         ret = OpenFile(command[1],atoi(command[2]));
734.         if(ret >= 0)
735.             printf("File is successfully opened with file descriptor : %d\n",ret);
736.         if(ret == -1)
737.             printf("ERROR : Incorrect parameters\n");
738.         if(ret == -2)
739.             printf("ERROR : File not present\n");
740.         if(ret == -3)
741.             printf("ERROR : Permission denied\n");
742.         continue;
743.     }
744.     else if(strcmp(command[0],"read") == 0)
745.     {
746.         fd = GetFDFromName(command[1]);
747.         if(fd == -1)
748.         {
749.             printf("Error : Incorrect parameter\n");
750.             continue;
751.         }
752.         ptr = (char *)malloc(sizeof(atoi(command[2]))+1);

```

```

753.         if(ptr == NULL)
754.         {
755.             printf("Error : Memory allocation failure\n");
756.             continue;
757.         }
758.         ret = ReadFile(fd,ptr,atoi(command[2]));
759.         if(ret == -1)
760.             printf("ERROR : File not existing\n");
761.         if(ret == -2)
762.             printf("ERROR : Permission denied\n");
763.         if(ret == -3)
764.             printf("ERROR : Reached at end of file\n");
765.         if(ret == -4)
766.             printf("ERROR : It is not regular file\n");
767.         if(ret == 0)
768.             printf("ERROR : File empty\n");
769.         if(ret > 0)
770.         {
771.             write(2,ptr,ret);
772.         }
773.         continue;
774.     }
775.     else
776.     {
777.         printf("\nERROR : Command not found !!!\n");
778.         continue;
779.     }
780. }
781. else if(count == 4)
782. {
783.     if(strcmp(command[0],"lseek") == 0)
784.     {
785.         fd = GetFDFromName(command[1]);
786.         if(fd == -1)
787.         {
788.             printf("Error : Incorrect parameter\n");
789.             continue;
790.         }
791.         ret = LseekFile(fd,atoi(command[2]),atoi(command[3]));
792.         if(ret == -1)
793.         {
794.             printf("ERROR : Unable to perform lseek\n");
795.         }
796.     }
797.     else
798.     {
799.         printf("\nERROR : Command not found !!!\n");
800.         continue;
801.     }
802. }
803. else
804. {
805.     printf("\nERROR : Command not found !!!\n");
806.     continue;
807. }
808. }
809. return 0;
810. }

```