Customised Dynamic File System

#include<stdio.h>

1. #include<stdlib.h>
2. #include<string.h>
3. #include<unistd.h>
4. #include<iostream>
5. //#include<io.h> 7.

8. #define MAXINODE 50 9.

1. #define READ 1
2. #define WRITE 2
3. 12.
4. #define MAXFILESIZE 2048
5. 14.
6. #define REGULAR 1
7. #define SPECIAL 2
8. 17.
9. #define START 0
10. #define CURRENT 1
11. #define END 2
12. 21.
13. typedef struct superblock 23.{
14. int TotalInodes;
15. int FreeInode; 26.}SUPERBLOCK, \*PSUPERBLOCK; 27.

28.typedef struct inode 29.{

1. char FileName[50];
2. int InodeNumber;
3. int FileSize;
4. int FileActualSize;
5. int FileType;
6. char \*Buffer;
7. int LinkCount;
8. int ReferenceCount;
9. int permission; // 1 23
10. struct inode \*next; 40.}INODE,\*PINODE,\*\*PPINODE; 41.

42.typedef struct filetable 43.{

1. int readoffset;
2. int writeoffset;
3. int count;
4. int mode; // 1 2 3
5. PINODE ptrinode; 49.}FILETABLE,\*PFILETABLE; 50.

51.typedef struct ufdt 52.{

53. PFILETABLE ptrfiletable; 54.}UFDT;

55.

1. UFDT UFDTArr[50];
2. SUPERBLOCK SUPERBLOCKobj;
3. PINODE head = NULL;
4. 59.
5. void man(char \*name) 61.{

62. if(name == NULL) return; 63.

64. if(strcmp(name,"create") == 0)

65. {

1. printf("Description : Used to create new regular file\n");
2. printf("Usage : create File\_name Permission\n");

68. }

69. else if(strcmp(name,"read") == 0)

70. {

1. printf("Description : Used to read data from regular file\n");
2. printf("Usage : read File\_name No\_Of\_Bytes\_To\_Read\n");

73. }

74. else if(strcmp(name,"write") == 0)

75. {

1. printf("Description : Used to write into regular file\n");
2. 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. {

1. printf("Description : Used to list all information of files\n");
2. printf("Usage : ls\n");

83. }

84. else if(strcmp(name,"stat") == 0)

85. {

1. printf("Description : Used to display information of file\n");
2. printf("Usage : stat File\_name\n");

88. }

89. else if(strcmp(name,"fstat") == 0)

90. {

1. printf("Description : Used to display information of file\n");
2. printf("Usage : stat File\_Descriptor\n");

93. }

94. else if(strcmp(name,"truncate") == 0)

95. {

1. printf("Description : Used to remove data from file\n");
2. printf("Usage : truncate File\_name\n");

98. }

99. else if(strcmp(name,"open") == 0)

100. {

1. printf("Description : Used to open existing file\n");
2. printf("Usage : open File\_name mode\n");

103. }

104. else if(strcmp(name,"close") == 0)

105. {

1. printf("Description : Used to close opened file\n");
2. printf("Usage : close File\_name\n");

108. }

109. else if(strcmp(name,"closeall") == 0)

110. {

1. printf("Description : Used to close all opened file\n");
2. printf("Usage : closeall\n");

113. }

114. else if(strcmp(name,"lseek") == 0)

115. {

1. printf("Description : Used to change file offset\n");
2. printf("Usage : lseek File\_Name ChangeInOffset StartPoint\n");

118. }

119. else if(strcmp(name,"rm") == 0)

120. {

1. printf("Description : Used to delete the file\n");
2. 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.{

1. printf("ls : To List out all files\n");
2. printf("clear : To clear console\n");
3. printf("open : To open the file\n");
4. printf("close : To close the file\n");
5. printf("closeall : To close all opened files\n");
6. printf("read : To Read the contents from file\n");
7. printf("write :To Write contents into file\n");
8. printf("exit : To Terminate file system\n");
9. printf("stat : To Display information of file using name\n");
10. printf("fstat :To Display information of file using file descriptor\n");
11. printf("truncate : To Remove all data from file\n");
12. 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. {

1. if(UFDTArr[i].ptrfiletable != NULL)
2. if(strcmp((UFDTArr[i].ptrfiletable->ptrinode->FileName),name)==0)
3. break;

155. i++;

156. } 157.

1. if(i == 50) return -1;
2. else return i;

160.}

161.

162.PINODE Get\_Inode(char \* name) 163.{

1. PINODE temp = head;
2. int i = 0; 166.
3. if(name == NULL)
4. return NULL; 169.

170. while(temp!= NULL)

171. {

1. if(strcmp(name,temp->FileName) == 0)
2. break;
3. temp = temp->next;

175. }

176. return temp;

177.}

178.

179.void CreateDILB() 180.{

1. int i = 1;
2. PINODE newn = NULL;
3. PINODE temp = head; 184.

185. while(i<= MAXINODE)

186. {

187. newn = (PINODE)malloc(sizeof(INODE)); 188.

1. newn->LinkCount =0;
2. newn->ReferenceCount = 0;
3. newn->FileType = 0;
4. newn->FileSize = 0; 193.
5. newn->Buffer = NULL;
6. newn->next = NULL; 196.

197. newn->InodeNumber = i; 198.

199. if(temp == NULL)

200. {

1. head = newn;
2. temp = head;

203. }

204. else

205. {

1. temp->next = newn;
2. temp = temp->next;

208. }

209. i++;

210. }

211. printf("DILB created successfully\n"); 212.}

213.

214.void InitialiseSuperBlock() 215.{

1. int i = 0;
2. while(i< MAXINODE)

218. {

219. UFDTArr[i].ptrfiletable = NULL;

220. i++;

221. } 222.

1. SUPERBLOCKobj.TotalInodes = MAXINODE;
2. SUPERBLOCKobj.FreeInode = MAXINODE; 225.}

226.

227.int CreateFile(char \*name,int permission) 228.{

1. int i = 0;
2. PINODE temp = head; 231.
3. if((name == NULL) || (permission == 0) || (permission > 3))
4. return -1; 234.
5. if(SUPERBLOCKobj.FreeInode == 0)
6. return -2; 237.

238. (SUPERBLOCKobj.FreeInode)--; 239.

1. if(Get\_Inode(name) != NULL)
2. return -3; 242.

243. while(temp!= NULL)

244. {

1. if(temp->FileType == 0)
2. break;
3. temp=temp->next;

248. } 249.

250. while(i<50)

251. {

1. if(UFDTArr[i].ptrfiletable == NULL)
2. break;

254. i++;

255. } 256.

257. UFDTArr[i].ptrfiletable = (PFILETABLE)malloc(sizeof(FILETABLE)); 258.

1. UFDTArr[i].ptrfiletable->count = 1;
2. UFDTArr[i].ptrfiletable->mode = permission;
3. UFDTArr[i].ptrfiletable->readoffset = 0;
4. UFDTArr[i].ptrfiletable->writeoffset = 0; 263.

264. UFDTArr[i].ptrfiletable->ptrinode = temp; 265.

1. strcpy(UFDTArr[i].ptrfiletable->ptrinode->FileName,name);
2. UFDTArr[i].ptrfiletable->ptrinode->FileType = REGULAR;
3. UFDTArr[i].ptrfiletable->ptrinode->ReferenceCount = 1;
4. UFDTArr[i].ptrfiletable->ptrinode->LinkCount = 1;
5. UFDTArr[i].ptrfiletable->ptrinode->FileSize = MAXFILESIZE;
6. UFDTArr[i].ptrfiletable->ptrinode->FileActualSize = 0;
7. UFDTArr[i].ptrfiletable->ptrinode->permission = permission;
8. 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. {

1. UFDTArr[fd].ptrfiletable->ptrinode->FileType = 0;
2. //free(UFDTArr[fd].ptrfiletable->ptrinode->Buffer);
3. free(UFDTArr[fd].ptrfiletable);

294. } 295.

1. UFDTArr[fd].ptrfiletable = NULL;
2. (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.

1. read\_size = (UFDTArr[fd].ptrfiletable->ptrinode->FileActualSize) - (UFDTArr[fd].ptrfiletable-

>readoffset);

1. 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.{

1. int i = 0;
2. PINODE temp = NULL; 354.
3. if(name == NULL || mode <= 0)
4. return -1; 357.
5. temp = Get\_Inode(name);
6. if(temp == NULL)
7. return -2; 361.
8. if(temp->permission < mode)
9. return -3; 364.

365. while(i<50)

366. {

1. if(UFDTArr[i].ptrfiletable == NULL)
2. break;

369. i++;

370. } 371.

1. UFDTArr[i].ptrfiletable = (PFILETABLE)malloc(sizeof(FILETABLE));
2. if(UFDTArr[i].ptrfiletable == NULL) return -1;
3. UFDTArr[i].ptrfiletable->count = 1;
4. UFDTArr[i].ptrfiletable->mode = mode;
5. if(mode == READ + WRITE)

377. {

1. UFDTArr[i].ptrfiletable->readoffset = 0;
2. UFDTArr[i].ptrfiletable->writeoffset = 0;

380. }

381. else if(mode == READ)

382. {

383. UFDTArr[i].ptrfiletable->readoffset = 0;

384. }

385. else if(mode == WRITE)

386. {

387. UFDTArr[i].ptrfiletable->writeoffset = 0;

388. }

1. UFDTArr[i].ptrfiletable->ptrinode = temp;
2. (UFDTArr[i].ptrfiletable->ptrinode->ReferenceCount)++; 391.

392. return i; 393.}

394.

395.void CloseFileByName(int fd) 396.{

1. UFDTArr[fd].ptrfiletable->readoffset = 0;
2. UFDTArr[fd].ptrfiletable->writeoffset = 0;
3. (UFDTArr[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->ptrinode->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->ptrinode->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-

>ptrinode->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->ptrinode->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->ptrinode->FileActualSize) + size > MAXFILESIZE) return -1;

454. if(((UFDTArr[fd].ptrfiletable->readoffset) + size) < 0) return -1;

455. (UFDTArr[fd].ptrfiletable->readoffset) = (UFDTArr[fd].ptrfiletable->ptrinode-

>FileActualSize) + size;

456. }

457. }

458. else if(UFDTArr[fd].ptrfiletable->mode == WRITE)

459. {

460. if(from == CURRENT)

461. {

1. if(((UFDTArr[fd].ptrfiletable->writeoffset) + size) > MAXFILESIZE) return -1;
2. if(((UFDTArr[fd].ptrfiletable->writeoffset) + size) < 0) return -1;
3. if(((UFDTArr[fd].ptrfiletable->writeoffset) + size) > (UFDTArr[fd].ptrfiletable-

>ptrinode->FileActualSize))

1. (UFDTArr[fd].ptrfiletable->ptrinode->FileActualSize) = (UFDTArr[fd].ptrfiletable->writeoffset) + size;
2. (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(UFDTArr[fd].ptrfiletable == NULL) return -2; 517.

518. temp = UFDTArr[fd].ptrfiletable->ptrinode; 519.

1. printf("\n---------Statistical Information about file \n");
2. printf("File name : %s\n",temp->FileName);
3. printf("Inode Number %d\n",temp->InodeNumber);
4. printf("File size : %d\n",temp->FileSize);
5. printf("Actual File size : %d\n",temp->FileActualSize);
6. printf("Link count : %d\n",temp->LinkCount);
7. printf("Reference count : %d\n",temp->ReferenceCount); 527.
8. if(temp->permission == 1)
9. printf("File Permission : Read only\n");
10. else if(temp->permission == 2)
11. printf("File Permission : Write\n");
12. else if(temp->permission == 3)
13. printf("File Permission : Read & Write\n");
14. 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.

1. printf("\n---------Statistical Information about file \n");
2. printf("File name : %s\n",temp->FileName);
3. printf("Inode Number %d\n",temp->InodeNumber);
4. printf("File size : %d\n",temp->FileSize);
5. printf("Actual File size : %d\n",temp->FileActualSize);
6. printf("Link count : %d\n",temp->LinkCount);
7. printf("Reference count : %d\n",temp->ReferenceCount); 562.
8. if(temp->permission == 1)
9. printf("File Permission : Read only\n");
10. else if(temp->permission == 2)
11. printf("File Permission : Write\n");
12. else if(temp->permission == 3)
13. printf("File Permission : Read & Write\n");
14. 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(UFDTArr[fd].ptrfiletable->ptrinode->Buffer,0,1024);

581. UFDTArr[fd].ptrfiletable->readoffset = 0;

582. UFDTArr[fd].ptrfiletable->writeoffset = 0;

583. UFDTArr[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("%[^\n]",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); | | |

**Marvellous Infosystems : Logic Building & Industrial Project Development**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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.  810.} | return | 0; | | |