 ***DEPARTMENT OF COMPUTER ENGINEERING***

Experiment No.

|  |  |
| --- | --- |
| Semester | S.E. Semester IV – Computer Engineering |
| Subject | Operating System |
| Subject Professor In-charge | Snehal Andhare |
| Assisting Teachers | Ms. Rasika Ransing |
| Laboratory | M310B – Computer Engineering Laboratory |

|  |  |  |
| --- | --- | --- |
| Student Name | Akshay Prabhu | |
| Roll Number | 18102A0058 | |
| Grade and Subject Teacher’s Signature |  |  |

|  |  |  |
| --- | --- | --- |
| Experiment Number |  | |
| Experiment Title | Implement basic commands of linux like ls, cp, mv and others using kernel APIs. | |
| Resources / Apparatus Required | Hardware: | Software: |
| Objectives  (Skill Set / Knowledge Tested / Imparted) |  | |
| Theory: | **mv:**  **mv** (short for move) is a Unix command that moves one or more files or directories from one place to another. If both filenames are on the same filesystem, this results in a simple file rename; otherwise the file content is copied to the new location and the old file is removed. Using mv requires the user to have write permission for the directories the file will move between. This is because mv changes the content of both directories (i.e., the source and the target) involved in the move. When using the mv command on files located on the same filesystem, the file's timestamp is not updated.  **cp:**  In computing, **cp** is a command in various Unix and Unix-like operating systems for copying files and directories. The command has three principal modes of operation, expressed by the types of arguments presented to the program for copying a file to another file, one or more files to a directory, or for copying entire directories to another directory.  **ls:**  In computing, **ls** is a command to list computer files in Unix and Unix-like operating systems. ls is specified by POSIX and the Single UNIX Specification. When invoked without any arguments, ls lists the files in the current working directory. The command is also available in the EFI shell. In other environments, such as DOS, OS/2, and Microsoft Windows, similar functionality is provided by the dir command. The numerical computing environments MATLAB and GNU Octave include an ls function with similar functionality. | |
| Code: | **ls:**  #include <stdio.h>  #include <stdlib.h>  #include <unistd.h>  #include <fcntl.h>  int main(int argc, char \*\*argv)  {  char buffer[1024];  int files[2];  ssize\_t count;  /\* Check for insufficient parameters \*/  if (argc < 3)  return -1;  files[0] = open(argv[1], O\_RDONLY);  if (files[0] == -1) /\* Check if file opened \*/  return -1;  files[1] = open(argv[2], O\_WRONLY | O\_CREAT | S\_IRUSR | S\_IWUSR);  if (files[1] == -1) /\* Check if file opened (permissions problems ...) \*/  {  close(files[0]);  return -1;  }  while ((count = read(files[0], buffer, sizeof(buffer))) != 0)  write(files[1], buffer, count);  return 0;  }  **mv:**  #include <stdio.h>  #include <stdlib.h>  #include <string.h>  #include <sys/stat.h>  #include <unistd.h>  #include <errno.h>  // check if it is the same inode on the same device  #define SAME\_INODE(a, b) ((a).st\_ino == (b).st\_ino && (a).st\_dev == (b).st\_dev)  // ALL CHECKS OMMITTED!  int main(int argc, char \*\*argv)  {  struct stat statbuf\_src, statbuf\_dest;  char \*src, \*dest, \*new\_src, \*new\_dest;  char \*current\_directory;  if (argc != 3) {  fprintf(stderr, "usage: %s src dest\n", argv[0]);  exit(EXIT\_FAILURE);  }  // work on copy  src = malloc(strlen(argv[1]) + 1);  dest = malloc(strlen(argv[2]) + 1);  strcpy(src, argv[1]);  strcpy(dest, argv[2]);  stat(src, &statbuf\_src);  stat(dest, &statbuf\_dest);  // there are many more, of course  printf("\"%s\" is a ", src);  if (S\_ISREG(statbuf\_src.st\_mode)) {  puts("a regular file");  }  if (S\_ISDIR(statbuf\_src.st\_mode)) {  puts("a directory");  }  printf("\"%s\" is a ", dest);  if (S\_ISREG(statbuf\_dest.st\_mode)) {  puts("a regular file");  }  if (S\_ISDIR(statbuf\_dest.st\_mode)) {  puts("a directory");  }  if (SAME\_INODE(statbuf\_dest, statbuf\_src)) {  printf("%s and %s are the identical\n", src, dest);  }  // if that is not set you have to do it by hand:  // climb up the tree, concatenating names until the inodes are the same  current\_directory = getenv("PWD");  printf("current directory is \"%s\"\n", current\_directory);  // I'm pretty sure it can be done in a much more elegant way  new\_src = malloc(strlen(src) + 1 + strlen(current\_directory) + 1);  strcpy(new\_src,current\_directory);  strcat(new\_src,"/");  strcat(new\_src,src);  printf("new\_src = %s\n",new\_src);  new\_dest = malloc(strlen(dest) + 1 + strlen(current\_directory) + 1 + strlen(src) + 1);  strcpy(new\_dest,current\_directory);  strcat(new\_dest,"/");  strcat(new\_dest,dest);  strcat(new\_dest,"/");  strcat(new\_dest,src);  printf("new\_dest = %s\n",new\_dest);  if(rename(new\_src,new\_dest) != 0){  fprintf(stderr,"rename failed with error %s\n",strerror(errno));  }  free(new\_src);  free(new\_dest);  free(src);  free(dest);  exit(EXIT\_SUCCESS);  }  **cp:**  #include <stdio.h>  #include <stdlib.h>  #include <fcntl.h>  #include <errno.h>    #define BUFF\_SIZE 1024    int main(int argc, char\* argv[])  {  int srcFD,destFD,nbread,nbwrite;  char \*buff[BUFF\_SIZE];    /\*Check if both src & dest files are received or --help is received to get usage\*/  if(argc != 3 || argv[1] == "--help")  {  printf("\nUsage: cpcmd source\_file destination\_file\n");  exit(EXIT\_FAILURE);  }    /\*Open source file\*/  srcFD = open(argv[1],O\_RDONLY);    if(srcFD == -1)  {  printf("\nError opening file %s errno = %d\n",argv[1],errno);  exit(EXIT\_FAILURE);  }    /\*Open destination file with respective flags & modes  O\_CREAT & O\_TRUNC is to truncate existing file or create a new file  S\_IXXXX are file permissions for the user,groups & others\*/  destFD = open(argv[2],O\_WRONLY | O\_CREAT | O\_TRUNC, S\_IRUSR | S\_IWUSR | S\_IRGRP | S\_IWGRP | S\_IROTH | S\_IWOTH);    if(destFD == -1)  {  printf("\nError opening file %s errno = %d\n",argv[2],errno);  exit(EXIT\_FAILURE);  }    /\*Start data transfer from src file to dest file till it reaches EOF\*/  while((nbread = read(srcFD,buff,BUFF\_SIZE)) > 0)  {  if(write(destFD,buff,nbread) != nbread)  printf("\nError in writing data to %s\n",argv[2]);  }    if(nbread == -1)  printf("\nError in reading data from %s\n",argv[1]);    if(close(srcFD) == -1)  printf("\nError in closing file %s\n",argv[1]);    if(close(destFD) == -1)  printf("\nError in closing file %s\n",argv[2]);    exit(EXIT\_SUCCESS);  } | |
| Output: |  | |