OS Lab Assignment – 1

Implementation of UNIX Commands

**“ls” command:**

*Description*:

ls with no option list files and directories in bare format where we won’t be able to view details like file types, size, modified date and time, permission and links etc.

*Code*:

#include <stdio.h>

#include <dirent.h>

#include <errno.h>

#include <stdlib.h>

void ls(const char \*dir,int op\_a,int op\_l){

  DIR \*dh = opendir(dir);

  if (!dh){

    if (errno = ENOENT){

      perror("Directory doesn't exist");

    }

    else{

      perror("Unable to read directory");

    }

    exit(EXIT\_FAILURE);

  }

  while ((d = readdir(dh)) != NULL){

    if (!op\_a && d->d\_name[0] == '.')

      continue;

    printf("%s  ", d->d\_name);

    if(op\_l) printf("\n");

  }

  if(!op\_l)

  printf("\n");

}

int main(int argc, const char \*argv[]){

  if (argc == 1){

    ls(".",0,0);

  }

  else if (argc == 2){

    if (argv[1][0] == '-'){

      int op\_a = 0, op\_l = 0;

      char \*p = (char\*)(argv[1] + 1);

      while(\*p){

        if(\*p == 'a') op\_a = 1;

        else if(\*p == 'l') op\_l = 1;

        else{

          perror("Option not available");

          exit(EXIT\_FAILURE);

        }

        p++;

      }

      ls(".",op\_a,op\_l);

    }

  }

  return 0;

}

***“rm” command***:

*Description:*

rm stands for remove here. rm command is used to remove objects such as files, directories, symbolic links and so on from the file system like UNIX. To be more precise, rm removes references to objects from the filesystem, where those objects might have had multiple references

*Code:*

#include<stdio.h>

#include<stdlib.h>

int main(int argc, char \*\*argv){

  printf("After completion the mentioned file will be deleted\n");

  int count;

  for(count=1;count<argc;count++){

    printf("The mentioned file '%s' will be deleted\n", argv[count]);

    remove(argv[count]);

  }

  return 0;

}

**“cat” command:**

*Description:*

Cat(concatenate) command is very frequently used in Linux. It reads data from the file and gives their content as output. It helps us to create, view, concatenate files.

*Code:*

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <errno.h>

int main(int argc, char \*argv[]) {

  FILE \*file;

  int chr, count;

  for(count = 1; count < argc; count++) {

    if((file = fopen(argv[count], "r")) == NULL) {

      fprintf(stderr, "%s: %s : %s\n", argv[0], argv[count], strerror(errno));

      continue;

    }

    while((chr = getc(file)) != EOF)

      fprintf(stdout, "%c", chr);

    fclose(file);

  }

  exit(0);

}

**“ps” command**:

*Description*:

A process is an executing instance of a program and carry out different tasks within the operating system. Linux provides us a utility called ps for viewing information related with the processes on a system which stands as abbreviation for “Process Status”. ps command is used to list the currently running processes and their PIDs along with some other information depends on different options. It reads the process information from the virtual files in /proc file-system. /proc contains virtual files, this is the reason it’s referred as a virtual file system.

*Code*:

#include <stdio.h>

#include <string.h>

#include <time.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <ctype.h>

#include <dirent.h>

#include <fcntl.h>

#include <unistd.h>

#define FORMAT "%5s %s\t%8s %s\n"

int main(int argc, char\*\* argv){

  DIR \*dir;

  struct dirent \*ent;

  int i, fd\_self, fd;

  unsigned long time, stime;

  char flag, \*tty;

  char cmd[256], tty\_self[256], path[256], time\_s[256];

  FILE\* file;

  dir = opendir("/proc");

  fd\_self = open("/proc/self/fd/0", O\_RDONLY);

  sprintf(tty\_self, "%s", ttyname(fd\_self));

  printf(FORMAT, "PID", "TTY", "TIME", "CMD");

  while ((ent = readdir(dir)) != NULL){

    flag = 1;

    for (i = 0; ent->d\_name[i]; i++)

    if (!isdigit(ent->d\_name[i])){

      flag = 0;

      break;

    }

    if (flag){

      sprintf(path, "/proc/%s/fd/0", ent->d\_name);

      fd = open(path, O\_RDONLY);

      tty = ttyname(fd);

      if (tty && strcmp(tty, tty\_self) == 0){

        sprintf(path, "/proc/%s/stat", ent->d\_name);

        file = fopen(path, "r");

        fscanf(file, "%d%s%c%c%c", &i, cmd, &flag, &flag, &flag);

        cmd[strlen(cmd) - 1] = '\0';

        for (i = 0; i < 11; i++)

          fscanf(file, "%lu", &time);

        fscanf(file, "%lu", &stime);

        time = (int)((double)(time + stime) / sysconf(\_SC\_CLK\_TCK));

        sprintf(time\_s, "%02lu:%02lu:%02lu",

        (time / 3600) % 3600, (time / 60) % 60, time % 60);

        printf(FORMAT, ent->d\_name, tty + 5, time\_s, cmd + 1);

        fclose(file);

      }

      close(fd);

    }

  }

  close(fd\_self);

  return 0;

}

**“kill” command**:

*Description*:

Kill is a built-in command which is used to terminate processes manually. kill command sends a signal to a process which terminates the process. If the user doesn’t specify any signal which is to be sent along with kill command then default term signal is sent that terminates the process

*Code*:

#include<stdio.h>

#include<stdlib.h>

#include<sys/wait.h>

#include<unistd.h>

#include<signal.h>

int main(void)

{

  int status;

  pid\_t killReturnVal, forkReturnVal;

  forkReturnVal=fork();

  if(forkReturnVal<0)

  {

    printf("Error !!");

    exit(EXIT\_FAILURE);

  }

  if(forkReturnVal==0)

  {

    sleep(100);

    exit(EXIT\_SUCCESS);

  }

  else

  {

    killReturnVal=kill(forkReturnVal,SIGKILL);

    if(killReturnVal)

    {

      printf("ChildProcess Can't be Killed \n");

      waitpid(forkReturnVal,&status,0);

    }

    else

    {

      printf("[+]+ Killed    Child Process got Terminated \n");

    }

  }

  return 0;

}

**“wc” command:**

*Description*:

wc stands for word count. As the name implies, it is mainly used for counting purpose. It is used to find out number of lines, word count, byte and characters count in the files specified in the file arguments.

*Code*:

#include <stdio.h>

#include <string.h>

#include <unistd.h>

int main(int argc, char\*\* argv){

  int bytes = 0;

  int words = 0;

  int newLine = 0;

  char buffer[1];

  enum states { WHITESPACE, WORD };

  int state = WHITESPACE;

  if(argc != 2)printf( "Help: %s filename", argv[0]);

  else{

    FILE \*file = fopen( argv[1], "r");

    if(file == 0)printf("can not find :%s\n",argv[1]);

    else{

      char \*thefile = argv[1];

      char last = ' ';

      while (read(fileno(file),buffer,1) ==1 ){

        bytes++;

        if ( buffer[0]== ' ' || buffer[0] == '\t') state = WHITESPACE;

        else if (buffer[0]=='\n'){

          newLine++;

          state = WHITESPACE;

        }

        else{

          if (state == WHITESPACE) words++;

          state = WORD;

        }

        last = buffer[0];

      }

      printf("New Lines: %d, words: %d, bytes: %d, fileName: %s\n",newLine,words,bytes,thefile);

   }

 }

 return 0;

}