```
/*****************************************************************************
                    Operating & Systems Administration
Author: Jonathan Ambrose
Date: 11/12/08
Version: 253
Description: A shell that will use inbuilt commands for:
        a. pwd
        b. cd
        c. ls
        d. ps
        e. kill
        If the command does not exist in the shell then it will execute
        it to the system i.e. Opening emacs.

Compile: gcc -pedantic -Wall insomnia.c -o insomnia
Run Code: ./insomnia
*****************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <strings.h>
#include "graphic.c"
#include <errno.h>
#include <linux/limits.h>
#include <dirent.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <pwd.h> /*provides a definition for struct passwd*/
#include <grp.h> /*provides structure for group*/
#include <time.h> /*provides structure for time*/
#include <limits.h>/*implementation-defined constants*/
#include <ctype.h>
#include <unistd.h>
#include <signal.h>

/*****************************Prototypes************************************/
int cmpstr(const void *vs1, const void *vs2);
int IScd(char *args);
int ISls(char *args);
int ISpwd(void);
int ISps(char *args);
int ISkill(char *args);
int IScommands(char *args);
```

```
/*****************************************************************************
 * Function name : main
 * returns : return void
 * Created by : Jon Ambrose
 * Date reviewed : 12/12/2008
 * Description : compares to files and then sorts them in A-Z format.
 *****************************************************************************/

int main(int argc, char *argv[])
{
  /* this is the list of the inbuilt commands into Insomnia  */
  char *command[] =
    {
      "cd",
      "commands",
      "exit",
      "kill",
      "ls",
      "ps",
      "pwd",
    };

  size_t numcommands = sizeof command / sizeof command[0];
  char line[4096] = {0};
  char safeline[4096] = {0};
  char *s = line;
  char **t = NULL;
  char *prompt = "jambrose@I.S>";
  char *args = NULL;
  char *nl = NULL;
  char *array;
  char **array2;
  int c;
  int done = 0;
  int i = 0;
  char *y, *r;

  graphic(); /*Include the graphic welcome screen from the file graphic.c*/

  while(!done)
    {
      printf("%s", prompt); /*print the prompt*/
      fflush(stdout);
      scanf("%s",array);
      if(NULL == fgets(line, sizeof line, stdin))
    {
      t = NULL;
      done = 1;
    }
      else
    {
      nl = strchr(line, '\n');
      if(nl != NULL)
        {
          *nl = '\0';
          strcpy(safeline, line);
```

```
        }
      else
        {
          int ch;
          printf("Line too long! Ignored.\n");
          while((ch = getchar()) != '\n' && ch != EOF) /*while getting characters and not enter key is pressed AND
not end of file*/
        {
          continue;
        }
          if(ch == EOF)
        {
          done = 1;
        }
        }
      args = strchr(line, ' ');
      if(args != NULL) /*contine if not NULL*/
        {
          *args++ = '\0'; /*increment args*/
        }
      t = bsearch(&s,command,numcommands,sizeof command[0],cmpstr);
    }

      if(!done && t !=NULL)

    {
      i = (int)(t - command);

      switch(i)
        {
        case 0: IScd(args); break;
        case 1: IScommands(args);break;
        case 2: return 1;break;
        case 3: ISkill(args); break;
        case 4: ISls(args);break;
        case 5: ISps(args); break;
        case 6: ISpwd(); break;
        default: break;
        }
    }
      else
    {
      printf("%s is not a insomnia shell command. "
         "Trying to execute it instead.\n", line);
      system(safeline);
    }
    }

  return 0;
}
```

```
/*****************************************************************************
* Function name : IScd
* returns : return 1
* Created by : Jon Ambrose
* Student Number: 07503425
* Date created : 7/11/08
* Description : change working directory
* Notes : if no path is set then it returns to the home varible.
*         If no home varible is set it will display error.
*         CD will allow you to change to a directory test/ or test/test.
*****************************************************************************/

int IScd(char *args)
{

  char *workingdirectory;
  char *newdir;

  if(args == NULL)
    { /*If no home varible is set it will display error.*/
      if ((newdir = getenv("HOME")) == NULL)
    {
      printf("Environment variable 'HOME' is not set\n");
      return 0; /*exit program*/
    }
    }
  else
    {
      newdir = args; /*read argument from command line to change directory to */
    }
  /*gets current directory*/
  workingdirectory = malloc((PATH_MAX * sizeof(char))+1);
  /* if it cannot get working directory - display error*/
  if (!(getcwd(workingdirectory,PATH_MAX)))
    {
      perror("pwd");
      free(workingdirectory);
      return 0;
    }
  /*print the current working directory*/
  printf("** Insomnia Shell is in %s\n",workingdirectory);
  /*if program cannot change directory to the new directory display error and quit.*/
  if ((chdir(newdir)) == -1)
    {
      perror(newdir);
      return 0;
    }
  /* if it cannot get working directory - display error*/
  if (!(getcwd(workingdirectory,PATH_MAX)))
    {
      perror("pwd");
      free(workingdirectory);
      return 0;
    }

  printf("** You have changed to %s\n",workingdirectory);
```

```
  return 1;
}

/*****************************************************************************
* Function name : ISkill
* returns : return 0
* Created by : Jon Ambrose
* Version: 45
* Description : kills applications
* Notes : Kill -l displays a whole list of kill commands
*         kill [[-signum] pid]
*****************************************************************************/

int ISkill(char *args)
{
  int isnumber(char *str);
  int pid,sig = 9;
  struct args_temp;
  char newvariable[100];
  int i;
  char *argv1;
  char *argv2;
  char *argv3;

  argv1 = args;
  argv2 = argv1;


  if(args == NULL)
    /*print error message if no argument has been yet*/
    {
      printf("usage: kill [[-signum] pid] | -l\n");
      return 0;
    }
  if(args !=NULL)
    {
      printf("before:%s\n",args);
      if(!strcmp(args,"-l"))
    {
      printf(" 1) SIGHUP\n");
      printf(" 2) SIGINT\n");
      printf(" 3) SIGQUIT\n");
      printf(" 4) SIGILL\n");
      printf(" 5) SIGTRAP\n");
      printf(" 6) SIGABRT\n");
      printf(" 8) SIGFPE\n");
      printf(" 9) SIGKILL\n");
      printf("11) SIGSEGV\n");
      printf("13) SIGPIPE\n");
      printf("14) SIGALRM\n");
      printf("15) SIGTERM\n");

      return 1;

    }

      char *p[10];
```

```
    int pc;
    p[0] = args;
    for(pc=1; pc<10;pc++)
  {
    p[pc] = p[pc-1];

    while ((*p[pc] == ' ')&&(*p[pc] != '\0'))
      {
        p[pc]++;
      }

    if (*p[pc] == '\0') break;

    p[pc-1] = p[pc];

    while ((*p[pc] != ' ')&&(*p[pc] != '\0'))
      {
        p[pc]++;
      }

    if (*p[pc] == '\0') break;

    *p[pc] = '\0';

    p[pc]++;
  }
    if (p[0][0] == '-') /*if args starts with - do the if statement*/
  {
    //printf("yay\n");
    if (!strcasecmp(&(p[0][1]),"SIGHUP")) sig = 1;
    else if (!strcasecmp(&(p[0][1]),"SIGINT")) sig = 2;
    else if (!strcasecmp(&(p[0][1]),"SIGQUIT")) sig = 3;
    else if (!strcasecmp(&(p[0][1]),"SIGILL")) sig = 4;
    else if (!strcasecmp(&(p[0][1]),"SIGTRAP")) sig = 5;
    else if (!strcasecmp(&(p[0][1]),"SIGABRT")) sig = 6;
    else if (!strcasecmp(&(p[0][1]),"SIGFPE")) sig = 8;
    else if (!strcasecmp(&(p[0][1]),"SIGKILL")) sig = 9;
    else if (!strcasecmp(&(p[0][1]),"SIGSEGV")) sig = 11;
    else if (!strcasecmp(&(p[0][1]),"SIGPIPE")) sig = 13;
    else if (!strcasecmp(&(p[0][1]),"SIGALRM")) sig = 14;
    else if (!strcasecmp(&(p[0][1]),"SIGTERM")) sig = 15;

    else sig = atoi(&(p[0][1]));

    //printf("sig:%d\n",sig);

    pid = atoi(p[1]);
  }

    else
  {
    pid = atoi(p[0]);
  }

    //printf("pid:%d\n",pid);

    if (kill(pid,sig) == -1) perror("kill");
```

```
    }
  return 0;


}
/*****************************************************************************
 * Function name : ISls
 * returns : return 0
 * Created by : Jon Ambrose
 * Student Number: 07503425
 * Date created : 10/12/2008
 * Version: 142
 * Flags that can be used are: -a, -l, -la, -g, -G
 * Description : print name of current/working directory
 * Notes : Links don't work
 *****************************************************************************/
int ISls(char *args)
{

  if((args == NULL) || (!strcmp(args,"-G")))
    {

      char dir[100];
      char *sort[100];
      int i;
      int sortc;
      int compls (const void *i, const void *j);
      struct stat get_info;
      struct dirent *read_dir;
      DIR *direct_;
      getcwd(dir,100);          /*gets current directory*/
      direct_ = opendir(dir);
      sortc=0;

      if(direct_ == NULL)
    {     /*if error opening the directory print the details of the error and exit*/
      perror("Error opening directory: ");
      return 1;
    }

      while ((read_dir = readdir(direct_)) != NULL)
    {

      if((read_dir-> d_name[0]) != '.')
        {
          sort[sortc++] = read_dir-> d_name;
        }
    }

      /*call quick sort */
      qsort(sort, sortc, sizeof(char *), compls);

      for(i = 0; i< sortc; i++)
    {
      if( sort[i] != NULL)
        {
```

```
          stat(sort[i],&get_info);
          // if (get_info.st_mode & S_ISLNK) {
          // if (access(this_item->mypath,F_OK) == 0) {
          //  printf("%c[96;1m%s%c[0m", 27, get_info.st_mode & S_ISLNK, 27);
          // } else {
          //   printf("%c[40;91;1m%s%c[0m", 27, get_info.st_mode & S_ISLNK, 27); /*broken link*/
          // }


          if(get_info.st_mode & S_IFDIR) printf("%c[94;1m%s%c[0m", 27, sort [i], 27);
          else if(((get_info.st_mode & S_IXUSR)|||(get_info.st_mode & S_IXGRP)||
               (get_info.st_mode & S_IXOTH))&&(!(get_info.st_mode & S_IFDIR)))

        printf("%c[92;1m%s%c[0m", 27, sort[i], 27);

          //else if (get_info.st_mode & S_ISFIFO) printf("%c[40;33m%s%c[0m"  , 27, sort[i], 27);
          //else if (get_info.st_mode & S_ISSOCK) printf("%c[95m%s%c[0m"  , 27, sort[i], 27);
          //else if (get_info.st_mode &S_ISBLK)  printf("%c[40;93;1m%s%c[0m" , 27, sort[i], 27);
          //else if (get_info.st_mode &S_ISCHR)  printf("%c[40;93;1m%s%c[0m" , 27, sort[i], 27);

          else printf("%s",sort[i]);



          //printf("%s", sort [ i ]);

          /*Added*/
          if(((get_info.st_mode & S_IXUSR)|||(get_info.st_mode & S_IXGRP)|||(get_info.st_mode &
S_IXOTH))&&(!(get_info.st_mode & S_IFDIR))) printf("*  "); /*executable file*/
          else if(get_info.st_mode & S_IFDIR) printf("/  "); /*it is a directory*/
          //else if (get_info.st_mode & S_ISLNK)printf("@ ");
          else printf("  ");
        }
    }


      printf("\n");


      closedir(direct_);

      return (0);
    }


  if(args != NULL)
    {
      if((!strcmp(args,"-la")) || (!strcmp(args,"-al")))

    {

      char dir[100],expansion[10],*time_formated;
      char buf[PATH_MAX];
      char *sort[100];
      int i;
      int sortc;
      int add_files=0;
```

```
    int compls (const void *i, const void *j);
    struct dirent *read_dir;
    struct stat get_info;
    struct passwd *user_info;
    struct group *group_buf;
    DIR *direct_;
    getcwd(dir,100);          /*gets current directory*/
    direct_ = opendir(dir);
    sortc=0;

    if(direct_ == NULL)
      {     /*if error opening the directory print the details of the error and exit*/
        perror("Error opening directory: ");
        return 1;
      }

    while ((read_dir = readdir(direct_)) != NULL)

      {
        sort[sortc++] = read_dir-> d_name;

      }

    /*call quick sort */
    qsort(sort, sortc, sizeof(char *), compls);

    for(i = 0; i< sortc; i++)
      {
        if( sort[i] != NULL)
      {
        stat(sort[i],&get_info);
        user_info = getpwuid(get_info.st_uid);
        group_buf = getgrgid(get_info.st_gid);
        time_formated = ctime(&get_info.st_mtime);
        time_formated[24] = ' ';

        if(get_info.st_mode & S_IFDIR) printf("d"); else printf("-");
        if(get_info.st_mode & S_IRUSR) printf("r"); else printf("-");
        if(get_info.st_mode & S_IWUSR) printf("w"); else printf("-");
        if(get_info.st_mode & S_IXUSR) printf("x"); else printf("-");
        if(get_info.st_mode & S_IRGRP) printf("r"); else printf("-");
        if(get_info.st_mode & S_IWGRP) printf("w"); else printf("-");
        if(get_info.st_mode & S_IXGRP) printf("x"); else printf("-");
        if(get_info.st_mode & S_IROTH) printf("r"); else printf("-");
        if(get_info.st_mode & S_IWOTH) printf("w"); else printf("-");
        if(get_info.st_mode & S_IXOTH) printf("x"); else printf("-");
        add_files = add_files + get_info.st_blocks;
        printf("  %d"  ,get_info.st_nlink); /*number of links*/
        printf("  %s"  ,user_info->pw_name);/*name of owner*/
        printf("  %s\t"  ,group_buf->gr_name);/*name of group*/
        printf("%ld\t" ,get_info.st_size);  /*size*/
        printf("%s"  ,time_formated);      /*time when created*/

        if (get_info.st_mode & S_IFDIR) printf("%c[94;1m%s%c[0m", 27, sort [i], 27);
        else if(((get_info.st_mode & S_IXUSR)||(get_info.st_mode & S_IXGRP)||
             (get_info.st_mode & S_IXOTH))&&(!(get_info.st_mode & S_IFDIR))) printf(
"%c[92;1m%s%c[0m", 27, sort[i], 27);
```

```
          //else if (get_info.st_mode & S_ISFIFO) printf("%c[40;33m%s%c[0m"  , 27, sort[i], 27);
          //else if (get_info.st_mode & S_ISSOCK) printf("%c[95m%s%c[0m"  , 27, sort[i], 27);
          //else if (get_info.st_mode &S_ISBLK)  printf("%c[40;93;1m%s%c[0m" , 27, sort[i], 27);
          //else if (get_info.st_mode &S_ISCHR)  printf("%c[40;93;1m%s%c[0m" , 27, sort[i], 27);

          else printf("%s",sort[i]);

          //printf("%s", sort [ i ]);

          /*Added*/
          if(((get_info.st_mode & S_IXUSR)|||(get_info.st_mode & S_IXGRP)|||(get_info.st_mode &
S_IXOTH))&&(!(get_info.st_mode & S_IFDIR))) printf("*  \n"); /*executable file*/
          else if(get_info.st_mode & S_IFDIR) printf("/  \n"); /*it is a directory*/
          else printf("  \n");

          printf("Total %d\n",add_files/2);add_files=0; /*size in kb of blocks*/
        }

        }
      printf("\n");

      return (0);
    }

      /*********************************************|s -a*************************************************/

      else if(!strcmp(args,"-a"))
    {

      char dir[100];
      char buf[PATH_MAX];
      char *sort[100];
      int i;
      int sortc;
      int compls (const void *i, const void *j);
      struct stat get_info;
      struct dirent *read_dir;

      DIR *direct_;
      getcwd(dir,100);          /*gets current directory*/
      direct_ = opendir(dir);
      sortc=0;

      if(direct_ == NULL){     /*if error opening the directory print the details of the error and exit*/
        perror("Error opening directory: ");
        return 1;
      }

      while ((read_dir = readdir(direct_)) != NULL)
        {
          sort[sortc++] = read_dir-> d_name;
        }
      /*call quick sort */
      qsort(sort, sortc, sizeof(char *), compls);

      for(i = 0; i< sortc; i++)
        {
```

```
          if( sort[i] != NULL)
        {
          stat(sort[i],&get_info);
          if (get_info.st_mode & S_IFDIR) printf("%c[94;1m%s%c[0m", 27, sort [i], 27);
          else if(((get_info.st_mode & S_IXUSR)||(get_info.st_mode & S_IXGRP)||
               (get_info.st_mode & S_IXOTH))&&(!(get_info.st_mode & S_IFDIR))) printf(
"%c[92;1m%s%c[0m", 27, sort[i], 27);
          //else if (get_info.st_mode & S_ISFIFO) printf("%c[40;33m%s%c[0m"  , 27, sort[i], 27);
          //else if (get_info.st_mode & S_ISSOCK) printf("%c[95m%s%c[0m"  , 27, sort[i], 27);
          //else if (get_info.st_mode &S_ISBLK)  printf("%c[40;93;1m%s%c[0m" , 27, sort[i], 27);
          //else if (get_info.st_mode &S_ISCHR)  printf("%c[40;93;1m%s%c[0m" , 27, sort[i], 27);

          else printf("%s",sort[i]);
          //printf("%s", sort [ i ]);

          /*Added*/
          if(((get_info.st_mode & S_IXUSR)||(get_info.st_mode & S_IXGRP)||(get_info.st_mode &
 S_IXOTH))&&(!(get_info.st_mode & S_IFDIR))) printf("*  "); /*executable file*/
          else if(get_info.st_mode & S_IFDIR) printf("/  "); /*it is a directory*/
          else printf("  ");
        }

        }
      printf("\n");
      return (0);
    }


    /*****************************************************************************************/

    else if((!strcmp(args,"-l")) || (!strcmp(args,"-g")))
    {
      char dir[100],expansion[10],*time_formated;
      char *sort[100];
      int i;
      int sortc;
      int add_files=0;
      int compls (const void *i, const void *j);
      struct dirent *read_dir;
      struct stat get_info;
      struct passwd *user_info;
      struct group *group_buf;

      DIR *direct_;
      getcwd(dir,100);          /*gets current directory*/
      direct_ = opendir(dir);
      sortc=0;
      if(direct_ == NULL)
        {     /*if error opening the directory print the details of the error and exit*/
          perror("Error opening directory: ");
          return 1;
        }
      while ((read_dir = readdir(direct_)) != NULL)

        if((read_dir-> d_name[0]) != '.')
          {
        sort[sortc++] = read_dir-> d_name;
```

```
        add_files = add_files + get_info.st_blocks;
          }

      /*call quick sort */
      qsort(sort, sortc, sizeof(char *), compls);
      printf("Total %d\n",add_files/2);add_files=0; /*size in kb of blocks*/
      for(i = 0; i< sortc; i++)
        {
          if( sort[i] != NULL)
        {
          stat(sort[i],&get_info);
          user_info = getpwuid(get_info.st_uid);
          group_buf = getgrgid(get_info.st_gid);
          time_formated = ctime(&get_info.st_mtime);
          time_formated[24] = ' ';

          if(get_info.st_mode & S_IFDIR) printf("d"); else printf("-");
          if(get_info.st_mode & S_IRUSR) printf("r"); else printf("-");
          if(get_info.st_mode & S_IWUSR) printf("w"); else printf("-");
          if(get_info.st_mode & S_IXUSR) printf("x"); else printf("-");
          if(get_info.st_mode & S_IRGRP) printf("r"); else printf("-");
          if(get_info.st_mode & S_IWGRP) printf("w"); else printf("-");
          if(get_info.st_mode & S_IXGRP) printf("x"); else printf("-");
          if(get_info.st_mode & S_IROTH) printf("r"); else printf("-");
          if(get_info.st_mode & S_IWOTH) printf("w"); else printf("-");
          if(get_info.st_mode & S_IXOTH) printf("x"); else printf("-");

          printf("  %d"  ,get_info.st_nlink); /*number of links*/
          if(!strcmp(args,"-l"))
            {
              printf("  %s\t"  ,user_info->pw_name);/*name of owner*/
            }
          printf("  %s\t"  ,group_buf->gr_name);/*name of group*/
          printf("%ld\t" ,get_info.st_size);  /*size*/
          printf("%s"  ,time_formated);      /*time when created*/

          if (get_info.st_mode & S_IFDIR) printf("%c[94;1m%s%c[0m", 27, sort [i], 27);
          else if(((get_info.st_mode & S_IXUSR)|||(get_info.st_mode & S_IXGRP)||
               (get_info.st_mode & S_IXOTH))&&(!(get_info.st_mode & S_IFDIR))) printf(
"%c[92;1m%s%c[0m", 27, sort[i], 27);
          //else if (get_info.st_mode & S_ISFIFO) printf("%c[40;33m%s%c[0m"  , 27, sort[i], 27);
          //else if (get_info.st_mode & S_ISSOCK) printf("%c[95m%s%c[0m"  , 27, sort[i], 27);
          //else if (get_info.st_mode &S_ISBLK)  printf("%c[40;93;1m%s%c[0m" , 27, sort[i], 27);
          //else if (get_info.st_mode &S_ISCHR)  printf("%c[40;93;1m%s%c[0m" , 27, sort[i], 27);

          else printf("%s",sort[i]);

          //printf("%s", sort [ i ]);

          /*Added*/
          if(((get_info.st_mode & S_IXUSR)|||(get_info.st_mode & S_IXGRP)|||(get_info.st_mode &
 S_IXOTH))&&(!(get_info.st_mode & S_IFDIR))) printf("*  \n"); /*executable file*/
          else if(get_info.st_mode & S_IFDIR) printf("/  \n"); /*it is a directory*/
          else printf("  \n");
        }
        }
      printf("\n");
```

```
      return (0);
    }

      else /*If Flag doesn't exist then display the following error*/
    {
      perror("flag does not exist\n");
    }

    }
  return (0);
}



/*****************************************************************************
* Function name : ISPWD
* returns : return 0
* Created by : Jon Ambrose
* Student Number: 07503425
* Date created : 30/10/08
* Version:v3
* Description : print name of current/working directory
* Notes : if you put arguments after it - it will still continue as pwd does not read them
*****************************************************************************/


int ISpwd(void)
{

  char *mydirectory;
  /*allocate unused space for path specified*/
  mydirectory = malloc( (PATH_MAX * sizeof(char)) + 1);

  if (!(getcwd(mydirectory,PATH_MAX)))
    {    /*if it cannot get the working directory perror  it and free memory*/
      perror("pwd");
      free(mydirectory);
      return 0;
    }
  /*print working directory and free memory*/
  printf("%s\n",mydirectory);
  free(mydirectory);
  return (0);
}

/*****************************************************************************
* Function name : ISps
* returns : return void
* Created by : Jon Ambrose
* Date created : 05/12/2008
* Version: V50
* Description : ps gives a snapshot of the current processes owned by the user.
* Flags: -A as a argument and list all current processes and their process Id's
*****************************************************************************/
int ISps(char *args)
{
```

```
  char dir[50] = "/proc";
  DIR *direct_;

  direct_ = opendir(dir);                          /*try to open directory*/

  if(direct_ == NULL)
    {
      perror("Opening file: ");
      exit (1);
    }


  if(args != NULL)
    {
      if(!strcmp(args,"-A"))
    {

      {
        char *cmd,temp_c,tty_string[6];
        int i,check_folder,pid,tty,temp_d,hours,minutes,seconds,ppid;
        long int temp_ld,temp_lu,stime,utime,cmd_time;
        unsigned int temp_u;
        struct dirent *read_dir;
        struct stat get_info;
        FILE *fp;

        printf("  PID TTY          TIME CMD\n");

        while((read_dir = readdir(direct_)) != NULL)
          {
        check_folder = 0;
        stat(read_dir->d_name,&get_info);          /*get information of the file*/

        if(get_info.st_mode & S_IFDIR)
          {
            strcat(dir,"/");
            strcat(dir,read_dir->d_name);            /*  get info/proc/NUMBER */
            for(i=0;i<strlen(read_dir->d_name);i++)
              {
            if(!isdigit(read_dir->d_name[i])) check_folder++;
              }
            strcat(dir,"/stat");                     /*  go to  /proc/[NUMBER]/stat */
            if(!check_folder)
              {
            fp = fopen(dir,"r");

            fscanf(fp,"%d %as %c %d %d %d %d %d %u %lu %lu %lu %lu %lu %lu %ld ",&pid,&cmd,&
temp_c,&ppid,&temp_d,&temp_d,&tty,&temp_d,&temp_u,&temp_lu,&temp_lu,&temp_lu,&temp_lu,&utime
,&stime,&temp_ld);
            cmd_time = (utime+stime)/100;
            hours   = cmd_time / 360; /*1 hour  = 360 seconds*/
            minutes = cmd_time / 60 ; /*1 minute =  60 seconds*/
            seconds = cmd_time % 60 ;

            if (major(tty) == 4) sprintf(tty_string,"tty%d  ",minor(tty));
            else if (major(tty) == 136) sprintf(tty_string,"pts/%d ",minor(tty));
```

```
            else strcpy(tty_string,"?     ");

            cmd[0] = ' ';
            cmd[strlen(cmd)-1] = ' ';


            printf("%5d",pid);printf(" %s",tty_string);
            for(i=0;i>(6-strlen(tty_string));i++)printf(" ");
            printf("   %02d:%02d:%02d",hours,minutes,seconds);
            printf("%s\n",cmd);
            fclose(fp);
              }
          }
        strcpy(dir,"/proc");
          }
      }
    }

      else /*If Flag doesn't exist then display the following error*/
    {
      perror("PS flag does not exist in Insomnia\n");
    }
      return 1;
    }

  if(args == NULL)
    {
      char *cmd,temp_c,tty_string[6];
      int i,check_folder,pid,tty,temp_d,hours,minutes,seconds,ppid,parent_id;
      long int temp_ld,temp_lu,stime,utime,cmd_time;
      unsigned int temp_u;
      struct dirent *read_dir;
      struct stat get_info;

      FILE *fp;

      printf("  PID TTY          TIME  CMD\n");

      parent_id = getppid();        /*get the parent ID*/

      while((read_dir = readdir(direct_)) != NULL)
    { /*go through every file/folder in /proc*/
      check_folder = 0;                                 /*it's used later to check if the folder is a numeric folder*/
      stat(read_dir->d_name,&get_info);                 /*get information of the file*/

      if(get_info.st_mode & S_IFDIR)
        {            /*if it is a directory*/
          strcat(dir,"/");
          strcat(dir,read_dir->d_name);                   /*  /proc/NUMBER */
          for(i=0;i<strlen(read_dir->d_name);i++)
        {
          if(!isdigit(read_dir->d_name[i])) check_folder++;   /*if it has a character in it, check_folder is
incremented*/
        }
          strcat(dir,"/stat");                            /*  /proc/[NUMBER]/stat */
          if(!check_folder)
        {                 /*check_folder = 0 means that it's a numeric folder*/
```

```
          fp = fopen(dir,"r");                /*open the specified folder*/

          fscanf(fp,"%d %as %c %d %d %d %d %d %u %lu %lu %lu %lu %lu %lu %ld ",&pid,&cmd,&
temp_c,&ppid,&temp_d,&temp_d,&tty,&temp_d,&temp_u,&temp_lu,&temp_lu,&temp_lu,&temp_lu,&utime
,&stime,&temp_ld);
          /*get the important values from the /stat*/
          if((parent_id == ppid)||(parent_id == pid))
            {
              cmd_time = (utime+stime)/100;      /*get the CMD in giffies*/
              hours   = cmd_time / 360;               /*1 hour  = 360 seconds*/
              minutes = cmd_time / 60 ;               /*1 minute =  60 seconds*/
              seconds = cmd_time % 60 ;

              if (major(tty) == 4) sprintf(tty_string,"tty%d  ",minor(tty));
              else if (major(tty) == 136) sprintf(tty_string,"pts/%d ",minor(tty));
              else strcpy(tty_string,"?     ");

              /*remove [] in the cmd*/
              cmd[0] = ' ';
              cmd[strlen(cmd)-1] = ' ';


              printf("%5d",pid);printf(" %s",tty_string);
              /*align string*/
              for(i=0;i>(6-strlen(tty_string));i++)printf(" ");
              printf("   %02d:%02d:%02d",hours,minutes,seconds);
              printf("%s\n",cmd);
              fclose(fp);
            }
        }
        }
       /*get to /proc again to go to the next folder*/
       strcpy(dir,"/proc");
    }
    }

  return (0);
}



/*****************************************************************************
* Function name : IScommands
* returns : return void
* Created by : Jon Ambrose
* Description : help gives a snapshot of the features availble in the Insomnia Shell.
* Flags: ----------maybe separte HELP for EACH COMMAND
*****************************************************************************/

int IScommands(char *args)
{
  if(args != NULL)
    {
      if(!strcmp(args,"-pwd"))
    {
      printf("help pwd\n");
    }
```

```
      else if(!strcmp(args,"-cd"))
    {
      printf("**************CD Help**********************\n\n");
      printf("CD allows you to change the directory\n");
      printf("-If no directory is specified to change directory to then it will go to the
users HOME environment varible\n");
      printf("-If environment varible is not set then it will display a error.\n");
      printf("-CD will allow you to change to a directory test/ or test/test\n");

    }
      else if(!strcmp(args,"-ls"))
    {
      printf("**************LS Help**********************\n\n");
      printf("LS allows you to list the contents of your directory\n");
      printf("-ls -a - shows all the files including hidden files and folders\n");
      printf("-ls -l - shows along list of files and folder and more details about them\n");
      printf("-ls -al - shows hidden long list format.\n");
    }
      else if(!strcmp(args,"-ps"))
    {
      printf("**************PS Help**********************\n\n");
      printf("PS allows you to list the processes\n");
      printf("-PS -A - shows all the processes\n");
    }
      else if(!strcmp(args,"-kill"))
    {
      printf("**************KILL Help**********************\n\n");
      printf("KILL allows you to kill a process using default signal SIGTERM\n");
      printf("KILL -A - shows all the processes\n");
    }
      else /*If Flag doesn't exist then display the following error*/
    {
      perror("Command does not exist in Insomnia\n");
    }
    }

  else if(args == NULL)
    {
      printf("Insomnia shell will use inbuilt custom commands for:\n");
      printf("a. pwd\n");
      printf("b. cd\n");
      printf("c. ls\n");
      printf("d. ps\n");
      printf("e. kill\n");
      printf("commands need to be run from the insomnia shell by typing ""commands
-(command)""\n");
      printf("If the command does not exist in the shell then it will execute\n");
      printf("it to the system i.e. Opening emacs.t\n");

    }
  return (0);
}
```

```
/*****************************************************************************
 * Function name : compls
 * returns : return strcmp
 * Created by : Jon Ambrose
 * Description : compares two files and then sorts them in A-Z format.
 *****************************************************************************/

/* sorting function - A - Z */
int compls(const void *a, const void * b)
{
  char **x, **y;
  x = (char **) a;
  y = (char **) b;
  return strcmp(*x,*y);
}



/*****************************************************************************
 * Function name : cmpstr
 * returns : returns the strings
 * Created by : Jon Ambrose
 * Description : compares two strings to see if they contain the same characters.
 *****************************************************************************/

int cmpstr(const void *vs1, const void *vs2)
{
  char * const *s1 = vs1;
  char * const *s2 = vs2;
  return strcasecmp(*s1, *s2);
}
```