memmove() function:

The C library function void \*memmove(void \*str1, const void \*str2, size\_t n) copies n characters from str2 to str1, but for overlapping memory blocks, memmove() is a safer approach than memcpy().

Declaration

Following is the declaration for memmove() function.

void \*memmove(void \*str1, const void \*str2, size\_t n)

Parameters

str1 − This is a pointer to the destination array where the content is to be copied, type-casted to a pointer of type void\*.

str2 − This is a pointer to the source of data to be copied, type-casted to a pointer of type void\*.

n − This is the number of bytes to be copied.

Return Value

This function returns a pointer to the destination, which is str1.

Example

The following example shows the usage of memmove() function.

#include <stdio.h>

#include <string.h>

int main () {

char dest[] = "oldstring";

const char src[] = "newstring";

printf("Before memmove dest = %s, src = %s\n", dest, src);

memmove(dest, src, 9);

printf("After memmove dest = %s, src = %s\n", dest, src);

return(0);

}

Memset () function:

Description

The C library function void \*memset(void \*str, int c, size\_t n) copies the character c (an unsigned char) to the first n characters of the string pointed to, by the argument str.

Declaration

Following is the declaration for memset() function.

void \*memset(void \*str, int c, size\_t n)

Parameters

str − This is a pointer to the block of memory to fill.

c − This is the value to be set. The value is passed as an int, but the function fills the block of memory using the unsigned char conversion of this value.

n − This is the number of bytes to be set to the value.

Return Value

This function returns a pointer to the memory area str.

Example

The following example shows the usage of memset() function.

#include <stdio.h>

#include <string.h>

int main () {

char str[50];

strcpy(str,"This is string.h library function");

puts(str);

memset(str,'$',7);

puts(str);

return(0);

}

Source: https://www.tutorialspoint.com/c\_standard\_library/c\_function\_memset.htm

**xv6 customization myls.c**

/\*program to list files and directories implementation of ls command \*/

#include "types.h"

#include "stat.h"

#include "user.h"

#include "fs.h"

int lo=0; int dot=0; int help=0;

char\* fmtname(char \*path)

{

static char buf[DIRSIZ+1];

char \*p;

// Find first character after last slash.

for(p=path+strlen(path); p >= path && \*p != '/'; p--);

p++;

// Return blank-padded name.

if(strlen(p) >= DIRSIZ)

return p;

memmove(buf, p, strlen(p));

memset(buf+strlen(p), ' ', DIRSIZ-strlen(p));

return buf;

}

Void ls(char \*path)

{

char buf[512], \*p;

int fd;

struct dirent de;

struct stat st;

if((fd = open(path, 0)) < 0) {

printf(2, "ls: cannot open %s\n", path);

return;

}

if(fstat(fd, &st) < 0){

printf(2, "ls: cannot stat %s\n", path);

close(fd);

return;

}

switch(st.type){

case T\_FILE:

printf(1, "%s %d %d %d\n", fmtname(path), st.type, st.ino, st.size);

break;

case T\_DIR:

if(strlen(path) + 1 + DIRSIZ + 1 > sizeof buf){

printf(1, "ls: path too long\n");

break;

}

strcpy(buf, path);

p = buf+strlen(buf);

\*p++ = '/';

while(read(fd, &de, sizeof(de)) == sizeof(de)){

if(de.inum == 0)

continue;

if(dot==0 && de.name[0]=='.')

continue;

memmove(p, de.name, DIRSIZ);

p[DIRSIZ] = 0;

if(stat(buf, &st) < 0){

printf(1, "ls: cannot stat %s\n", buf);

continue;

}

if(lo==1)

{

if(st.type==T\_DIR) printf(1, "\033[1m\x1B[34m%s\x1B[0m %d %d %d\n", fmtname(buf), st.type, st.ino, st.size);

else if(st.type==T\_DEV) printf(1, "\033[1m\x1B[31m%s\x1B[0m %d %d %d\n", fmtname(buf), st.type, st.ino, st.size);

else printf(1, "%s %d %d %d\n", fmtname(buf), st.type, st.ino, st.size);

}

else

{

if(st.type==T\_DIR) printf(1, "\033[1m\x1B[34m%s\x1B[0m\n", fmtname(buf));

else if(st.type==T\_DEV) printf(1, "\033[1m\x1B[31m%s\x1B[0m\n", fmtname(buf));

else printf(1, "%s\n", fmtname(buf));

}

}

break;

}

close(fd);

}

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

{

int i;

for(i=1;i<argc;i++)

{

if(argv[i][0]=='-')

{

if(strcmp(argv[i],"-l")==0) lo=1;

else if(strcmp(argv[i],"-a")==0) dot=1;

else if(strcmp(argv[i],"--help")==0) help=1;

else printf(1,"invalid OPTIONS try 'ls --help' for more information ");

}

}

if(help)

{

printf(1,"Usage : \033[1mls\x1B[0m [OPTION]... [FILE]...\n");

printf(1,"List information about the FILEs (the current directory by default).\n");

printf(1,"OPTION:\n");

printf(1,"\t\033[1m-a\x1B[0m do not ignore entries starting with .\n");

printf(1,"\t\033[1m-l\x1B[0m use a long listing format\n");

}

if(argc < 2)

{

ls(".");

exit();

}

else if((lo==1 || dot==1 || help==1) && argc < 3)

{

ls(".");

exit();

}

else if((lo==1 || dot==1 || help==1) && argc < 4)

{

ls(argv[2]);

exit();

}

else

{

ls(argv[1]);

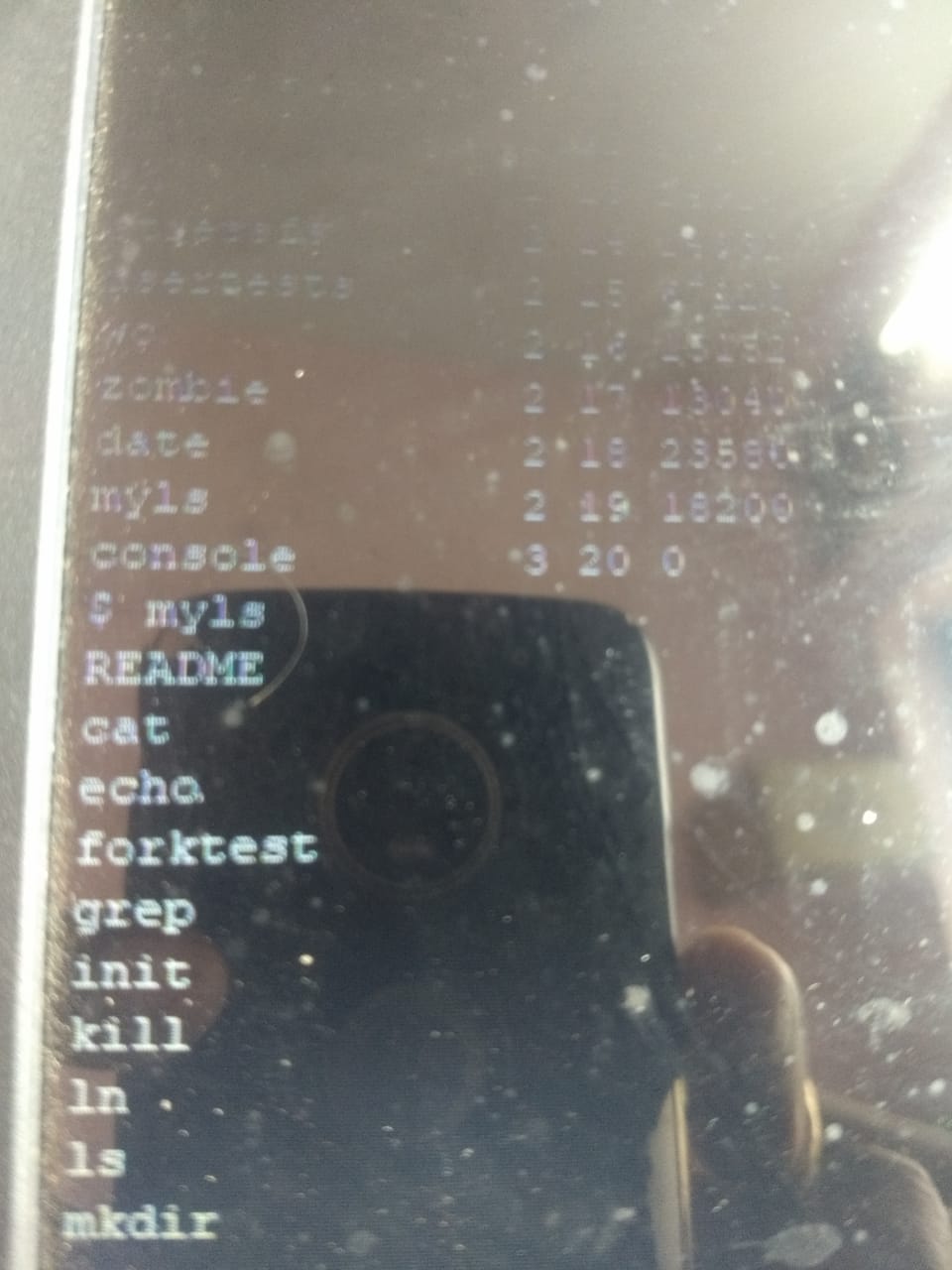
exit();

}

exit();

}

OUTPUT :



**ls.c source code xv6**

#include "types.h"

#include "stat.h"

#include "user.h"

#include "fs.h"

char\* fmtname(char \*path)

{

static char buf[DIRSIZ+1];

char \*p;

// Find first character after last slash.

for(p=path+strlen(path); p >= path && \*p != '/'; p--)

;

p++;

// Return blank-padded name.

if(strlen(p) >= DIRSIZ)

return p;

memmove(buf, p, strlen(p));

memset(buf+strlen(p), ' ', DIRSIZ-strlen(p));

return buf;

}

void

ls(char \*path)

{

char buf[512], \*p;

int fd;

struct dirent de;

struct stat st;

if((fd = open(path, 0)) < 0){

printf(2, "ls: cannot open %s\n", path);

return;

}

if(fstat(fd, &st) < 0){

printf(2, "ls: cannot stat %s\n", path);

close(fd);

return;

}

switch(st.type){

case T\_FILE:

printf(1, "%s %d %d %d\n", fmtname(path), st.type, st.ino, st.size);

break;

case T\_DIR:

if(strlen(path) + 1 + DIRSIZ + 1 > sizeof buf){

printf(1, "ls: path too long\n");

break;

}

strcpy(buf, path);

p = buf+strlen(buf);

\*p++ = '/';

while(read(fd, &de, sizeof(de)) == sizeof(de)){

if(de.inum == 0)

continue;

memmove(p, de.name, DIRSIZ);

p[DIRSIZ] = 0;

if(stat(buf, &st) < 0){

printf(1, "ls: cannot stat %s\n", buf);

continue;

}

printf(1, "%s %d %d %d\n", fmtname(buf), st.type, st.ino, st.size);

}

break;

}

close(fd);

}

int

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

{

int i;

if(argc < 2){

ls(".");

exit();

}

for(i=1; i<argc; i++)

ls(argv[i]);

exit();

}