#include <stdio.h>

#include <string.h>

#include <unistd.h>

#include <fcntl.h>

#include <sys/dir.h>

#include <linux/fb.h>

#include <ctype.h>

#include <stdlib.h>

#include <sys/mman.h>

#include <signal.h>

#include <stdlib.h>

#include <sys/ioctl.h>

#include <linux/input.h>

#define NODE\_EXP "/dev/fpga\_char\_lcd"

#define NODE\_EXP1 "/dev/7seg\_led"

#define LCD\_ON 1

#define LCD\_CLEAR 2

#define LCD\_SET\_HOME 3

#define LCD\_CURSOR\_BLINK 4

#define LCD\_DATA\_WRITE 5

#define LCD\_OFF 6

#define SEG\_7\_LED\_ON 1

#define SEG\_7\_LED\_WRITE 2

#define SEG\_7\_LED\_CLEAR 3

#define SEG\_7\_LED\_OFF 4

unsigned short Type;

unsigned long Size; /\* file size in bytes \*/

struct FileHeader {

unsigned short Reserved1; /\* 0 \*/

unsigned short Reserved2; /\* 0 \*/

unsigned long OffBits; /\* offset to bitmap \*/

unsigned long StructSize; /\* size of this struct (40) \*/

unsigned long Width; /\* bmap width in pixels \*/

unsigned long Height; /\* bmap height in pixels \*/

unsigned short Planes; /\* num planes - always 1 \*/

unsigned short BitCount; /\* bits per pixel \*/

unsigned long Compression; /\* compression flag \*/

unsigned long SizeImage; /\* image size in bytes \*/

long XPelsPerMeter; /\* horz resolution \*/

long YPelsPerMeter; /\* vert resolution \*/

unsigned long ClrUsed; /\* 0 -> color table size \*/

unsigned long ClrImportant; /\* important color count \*/

};

struct RGBQUAD {

unsigned char rgbBlue;

unsigned char rgbGreen;

unsigned char rgbRed;

unsigned char rgbReserved;

};

extern int alphasort();

struct fb\_var\_screeninfo vinfo;

struct fb\_fix\_screeninfo finfo;

char \*fbp = 0;

int fbfd = 0;

long int screensize;

char \*fname[100];

int pid;

void ex\_program(int sig)

{

munmap(fbp, screensize);

close(fbfd);

system("pkill -9 -f X");

exit(0);

}

int show\_bmp(int count,int delay)

{

FILE \*fp;

unsigned long int location = 0, BytesPerLine = 0;

unsigned long pixel, p1;

struct FileHeader \*Header;

unsigned int t,x,y;

unsigned long size, bytes\_read;

unsigned char Bmp, dummy, red, blue, green;

int i,hindex,index,j;

struct RGBQUAD Palette[256];

unsigned long \*bgr\_buff;

char buff[50];

do {

j=0;

(void) signal(SIGINT, ex\_program);

while(j < count)

{

fp = fopen(fname[j],"rb");

Header = (struct FileHeader \*) malloc (sizeof(struct FileHeader));

if (!Header) {

perror("Error:");

exit(1);

}

if (!fp) {

printf ("Error opening source file\r\n");

perror ("Error");

exit (1);

}

fread(&Type, sizeof(Type), 1, fp);

bytes\_read = sizeof(Type);

fread(&Size, sizeof(Size), 1, fp);

bytes\_read += sizeof(Size);

if ((fread(Header, sizeof(struct FileHeader), 1, fp)) == -1) {

printf ("Error: Unable to read File header.\r\n");

exit (1);

}

bytes\_read += sizeof(struct FileHeader);

while (bytes\_read < Header->OffBits) {

if (fread(&dummy,sizeof(dummy),1,fp)!=1) {

printf("Error seeking to bitmap data\n");

exit(-1);

}

++bytes\_read;

}

size = Header->Width \* Header->Height;

index=0;

if (Header->BitCount == 24) {

bgr\_buff = (unsigned long \*) malloc (size \* sizeof(unsigned long));

for (i = 0; i < size; i++) {

blue = fgetc(fp);

green = fgetc(fp);

red = fgetc(fp);

p1 = 0;

p1 |= red;

p1 = p1 << 16;

pixel = p1;

p1 = 0;

p1 |= green;

p1 = p1 << 8;

pixel |= p1;

p1 = 0;

p1 |= blue;

pixel |= p1;

bgr\_buff [index] = pixel;

index ++;

}

// At this point bgr\_buff contains the RGB values for the pixels defined by height & width of the BMP file.

hindex=0;

y = Header->Height-1;

while(y > 0) {

for(x=0; x < Header->Width ; x++) {

location = (x+vinfo.xoffset) \* (vinfo.bits\_per\_pixel/8) +(y+vinfo.yoffset) \* finfo.line\_length;

\*((unsigned long \*)(fbp + location)) = bgr\_buff [hindex] ;

hindex++;

}

--y;

}

free(bgr\_buff);

}

free(Header);

printf ("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\r");

j++;

fclose (fp);

sleep(delay);

}

} while(0);

return( 0 );

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int file\_select(const struct direct \*entry)

{

char \*ptr;

if ((strcmp(entry->d\_name, ".")== 0) ||

(strcmp(entry->d\_name, "..") == 0))

return (0);

/\* Check for filename extensions \*/

ptr = rindex(entry->d\_name, '.');

if ((ptr != NULL) &&((strcmp(ptr, ".bmp")==0)))

return (1);

else

return(0);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

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

{

long int o;

unsigned long size, bytes\_read;

struct direct \*\*files;

char \*pathname;

int count=0,delay =0,i;

pid=getpid();

int fd;

char buff[256] = "Name";

int w,rb,xx,yy;

int exp\_dev= 0;

int exp\_dev1= 0;

int res=0,k,v;

int h=0,d,sd,dd,sum=0,q,dmk=0,admk=0,bjp=0,nota=0;

char b[4][4] = {"DMK","BJP","ADMK","NOTA"};

int a[4]={1234,2345,5678,1579};

struct input\_event ev;

struct input\_absinfo ab;

for(k=0;k<10;k++)

{

abc:

printf("\nEnter your Aadhar Number:\n");

scanf("%ld",&o);

printf("\nPlace your Thumb impression\n");

sleep(3);

printf("Verifying...\n");

sleep(3);

for(v=0;v<=3;v++)

{

if(o==a[0])

{

printf("\nPerson verification successful\n\n");

break;

}

else if(o==a[1])

{

printf("\nPerson verification successful\n\n");

break;

}

else if(o==a[2])

{

printf("\nPerson verification successful\n\n");

break;

}

else if(o==a[3])

{

printf("\nPerson verification successful\n\n");

break;

}

else

{

printf("\nIdentification not found \nRe-enter your password \n");

goto abc;

}

}

if (argc <= 2) {

printf ("Usage:./ulk\_photo\_frame <source bitmap directory> <delay>\r\n");

exit (2);

}

pathname=argv[1];

delay=atoi(argv[2]);

size =pathconf(pathname, \_PC\_PATH\_MAX);

if (!(getcwd(pathname,(size\_t)size))) {

printf("Error getting path\n");

return 0;

}

if(!(chdir(pathname)))

count = scandir(pathname, &files, file\_select, alphasort);

if (count <= 0) {

printf("No files in this directory\n");

return 0;

}

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

fname[i-1]=files[i-1]->d\_name;

}

printf("\n");

// Open the file for reading and writing

fbfd = open("/dev/fb0", O\_RDWR);

if (!fbfd) {

printf("Error cannot open framebuffer device.\n");

exit(1);

}

// Get fixed screen information

if (ioctl(fbfd, FBIOGET\_FSCREENINFO, &finfo)) {

printf("Error reading fixed information.\n");

exit(2);

}

// Get variable screen information

if (ioctl(fbfd, FBIOGET\_VSCREENINFO, &vinfo)) {

printf("Error reading variable information.\n");

exit(3);

}

screensize = 0;

// Figure out the size of the screen in bytes

screensize = vinfo.xres \* vinfo.yres \* vinfo.bits\_per\_pixel / 8;

// Map the device to memory

fbp = (char \*)mmap(0, screensize, PROT\_READ | PROT\_WRITE, MAP\_SHARED,fbfd, 0);

if ((int)fbp == -1) {

printf("Error failed to map framebuffer device to memory.\n");

exit(4);

}

show\_bmp(count,delay);

munmap(fbp, screensize);

if ((fd = open("/dev/input/touchscreen0", O\_RDONLY)) < 0)

{

printf ("Error Opening the Device\n");

return 1;

}

ioctl(fd, EVIOCGNAME(sizeof(buff)), buff);

for (w = 0; w < 5; w++)

{

rb=read(fd,&ev,sizeof(struct input\_event));

if ((ev.type == 3) && (ev.code == 0))

{

xx=ev.value;

}

if ((ev.type == 3) && (ev.code == 1))

{

yy=ev.value;

}

}

if((xx<=2000)&&(yy<=2000))

{

q=0;

dmk++;

}

else if((xx>2000)&&(yy<=2000))

{

q=1;

bjp++;

}

else if((xx<=2000)&&(yy>2000))

{

q=2;

admk++;

}

else

{

q=3;

nota++;

}//

exp\_dev = open(NODE\_EXP, O\_RDWR);

if (exp\_dev < 0)

{

fprintf(stderr, "Open error: %s\n", NODE\_EXP);

return 1;

}

if (ioctl(exp\_dev,LCD\_ON, NULL) < 0)

{

printf("Error in Writing the LCD \r\n");

close(exp\_dev);

return 1;

}

if (ioctl(exp\_dev,LCD\_CLEAR, NULL) < 0)

{

printf("Error in Clearing the LCD \r\n");

close(exp\_dev);

return 1;

}

if (ioctl(exp\_dev,LCD\_SET\_HOME, NULL) < 0 )

{

printf("Error in setting the cursor in home\

position \r\n");

close(exp\_dev);

return 1;

}

if (ioctl(exp\_dev,LCD\_CURSOR\_BLINK, NULL) < 0 )

{

printf("Error in Curosr blinking \r\n");

close(exp\_dev);

return 1;

}

if (ioctl(exp\_dev,LCD\_DATA\_WRITE, b[q]) < 0 )

{

printf("Error in writing the LCD \r\n");

close(exp\_dev);

return 1;

}

sleep(3);

getchar();

if (ioctl(exp\_dev,LCD\_OFF, NULL) < 0 )

{

printf("Error in Switching OFF the LCD \r\n");

close(exp\_dev);

return 1;

}

//

/\* open as blocking mode \*/

exp\_dev1 = open(NODE\_EXP1, O\_RDWR);

if (exp\_dev1 < 0){

fprintf(stderr, "Open error: %s\n", NODE\_EXP1);

return 1;

}

if (res = ioctl(exp\_dev1,SEG\_7\_LED\_ON, NULL) < 0 ){

printf("%d---> Error in switching OFF the LED \r\n",res);

close(exp\_dev1);

return 1;

}

d = h%10;

sd = (h/10)\*16;

dd = d+sd;

sum = sum++;

if (res=ioctl(exp\_dev1,SEG\_7\_LED\_WRITE, sum) < 0 ){

printf("%d---> Error in writing to the 7-Seg LED \r\n",res);

close(exp\_dev1);

return 1;

}

sleep(1);

if(h==0){

if (res=ioctl(exp\_dev1,SEG\_7\_LED\_OFF, NULL) < 0){

printf("%d---> Error in switching off the LED \r\n",res);

close(exp\_dev1);

return 1;

}

}

printf("\n Total count :\n");

printf("\nDMK = %d ",dmk);

printf("\nADMK = %d ",admk);

printf("\nBJP = %d ",bjp);

printf("\nNOTA = %d ",nota);

printf("\n\n");

close(exp\_dev1);

close(exp\_dev);

close(fbfd);

}

return 0;

}