파이프 통신(Named FIFO)을 이용하는 버전

Makefile

CC = gcc

CFLAGS = -W -Wall

TARGET = paint

OBJECTS = main.o event.o drawing.o ipc.o typo.o

all : $(TARGET)

$(TARGET) : $(OBJECTS)

$(CC) $(CFLAGS) -o $@ $^ -lX11 -lpthread

rm \*.o Drawing Answer

clean :

rm \*.o $(TARGET)

header.h

#include <stdio.h>

#include <pthread.h>

#include <X11/Xlib.h>

#include <X11/Xutil.h>

#include <X11/keysymdef.h>

//

#include <termios.h>

#include <stdlib.h>

#include <string.h>

#include <fcntl.h>

#include <unistd.h>

#include <errno.h>

#include <time.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <sys/ipc.h>

#include <sys/msg.h>

typedef enum bool{ false, true } bool;

#define MARGIN 50

#define CANVAS\_X MARGIN

#define CANVAS\_Y MARGIN

#define CANVAS\_WIDTH 600

#define CANVAS\_HEIGHT 300

#define PLAYER\_X MARGIN + CANVAS\_WIDTH + MARGIN

#define PLAYER\_Y MARGIN

#define PLAYER\_WIDTH 300

#define PLAYER\_HEIGHT CANVAS\_HEIGHT

#define PALET\_AREA\_X MARGIN

#define PALET\_AREA\_Y 370

#define PALET\_ITEM\_WIDTH 30

#define PALET\_ITEM\_HEIGHT 30

#define BRUSH\_ITEM\_HEIGHT 30

#define BRUSH\_ITEM\_WIDTH 30

#define BRUSH\_AREA\_X MARGIN

#define BRUSH\_AREA\_Y MARGIN + CANVAS\_HEIGHT + BRUSH\_ITEM\_HEIGHT

// main window

extern Display \*dpy;

extern Window w;

extern XEvent xe;

// other ui stuff (context)

extern GC gc;

extern int lineWidth;

extern int color;

// ipc

extern char strAnswerCorrect[100];

extern bool isDrawer;

extern bool bGameOver;

// drawn line

#define MAX\_INDEX\_PATH 100000

extern int indexPath;

extern long pathColor[MAX\_INDEX\_PATH];

extern int pathWidth[MAX\_INDEX\_PATH];

extern XPoint path[MAX\_INDEX\_PATH];

// drawing

extern void DrawPallete();

extern void RepaintPath();

extern void Clear();

extern bool EventCursorIsWithinCanvas(XEvent);

extern bool EventCursorIsWithinColorPick(XEvent);

extern bool EventCursorIsWithinWidthPick(XEvent);

extern bool EventCursorIsBeingClicked(XEvent);

extern bool GetClearPick(XEvent);

extern int GetColorPick(XEvent);

extern int GetWidthPick(XEvent);

extern void SetForegroundToColorIndex(int i);

extern void SetForegroundToColor(unsigned long);

extern void SetLineWidth(int width);

//typo

extern void TypoSetInputAnswer();

extern void TypeInputLoopWriter();

extern void TypeInputLoopReader();

//ipc

extern void SndPath(int, int, int, int, int);

extern void SndAnswerCorrect(char \*strAnswer);

extern void SndGameOver();

//init

extern void InitDisplay();

extern void IpcInit();

extern void IpcInitClear();

// user\_input // ipc

extern void \*Thread1();

extern void \*Thread2Writer();

extern void \*Thread2Reader();

main.c

#include "header.h"

#define N\_THREADS 2

static pthread\_t thid[N\_THREADS];

//externs//

char strAnswerCorrect[100] = "nn45nn3c8n";

bool bGameOver = false;

bool isDrawer = false;

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

int i;

if (argc == 1){

printf(

"not enough argument. \n"

"usage: \n"

"[%s -w] for writer \n"

"[%s -r] for reader \n",

argv[0], argv[0]

);

return 1;

}

if (strcmp(argv[1], "-w") == 0) {

isDrawer = true;

printf("starting as a writer. \n");

}

else if (strcmp(argv[1], "-r") == 0) {

isDrawer = false;

printf("starting as a reader. \n");

}

else {

printf("invalid argument \n");

return 1;

}

printf("createing threads.. \n");

if (isDrawer){

InitDisplay();

TypoSetInputAnswer();

pthread\_create(&thid[0], NULL, Thread1, NULL); // user\_input

pthread\_create(&thid[1], NULL, Thread2Writer, NULL); // ipc

printf("hello there \n");

TypoInputLoopWriter();

}

else{

InitDisplay();

pthread\_create(&thid[0], NULL, Thread1, NULL); // user\_input

pthread\_create(&thid[1], NULL, Thread2Reader, NULL); // ipc

printf("hello there \n");

TypoInputLoopReader();

}

/\*

for (i = 0; i < N\_THREADS; ++i) {

if (pthread\_join(thid[i], NULL) != 0) {

printf("end of process (-1) \n");

return -1;

}

}

\*/

printf("end of process (0) \n");

return 0;

}

event.c

#include "header.h"

// main window

Display \*dpy;

Window w;

XEvent xe;

// other ui stuff (context)

GC gc;

int lineWidth;

int color;

// drawn line

int indexPath;

XPoint path[MAX\_INDEX\_PATH];

long pathColor[MAX\_INDEX\_PATH];

int pathWidth[MAX\_INDEX\_PATH];

extern void InitDisplay() {

dpy = XOpenDisplay(NULL);

w = XCreateSimpleWindow(dpy, RootWindow(dpy, 0), 50, 50, 1050, 500,

5, BlackPixel(dpy, 0), WhitePixel(dpy, 0));

XSelectInput(dpy, w, ExposureMask |

ButtonPressMask | ButtonReleaseMask |

// EnterWindowMask | LeaveWindowMask |

PointerMotionMask

);

XMapWindow(dpy, w);

XFlush(dpy);

gc = XCreateGC(dpy, w, 0L, (XGCValues \*)NULL);

color = BlackPixel(dpy, 0);

lineWidth = 5;

XSetFont(dpy, gc, XLoadFont(dpy, "fixed"));

XSetLineAttributes(dpy, gc, lineWidth, LineSolid, CapRound, JoinRound);

XFlush(dpy);

} // func

// (XEvent xe)

extern void onDraw() {

int i;

long preColor;

int preWidth;

preColor =pathColor[0];

preWidth = pathWidth[0];

SetForegroundToColor(preColor);

SetLineWidth(preWidth);

DrawPallete();

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

if ( path[i-1].x != -1 && path[i-1]. y != -1 &&

path[i].x != -1 && path [i].y != -1){

XDrawLine(dpy, w, gc,

path[i-1].x, path[i-1].y,

path[i].x, path[i].y);

}

else{

if(path[i-1].x == -1 && path[i].x != -1){

preColor = pathColor[i];

preWidth = pathWidth[i];

SetForegroundToColor(preColor);

SetLineWidth(preWidth);

}

}

}

// repaint(xe);

} // func

static void onMouseMoved(XEvent xe) {

if (EventCursorIsWithinCanvas(xe) && EventCursorIsBeingClicked(xe) && indexPath < MAX\_INDEX\_PATH) {

path[indexPath].x = xe.xmotion.x;

path[indexPath].y = xe.xmotion.y;

if (indexPath == 0) { return; }

XDrawLine(dpy, w, gc,

path[indexPath - 1].x, path[indexPath - 1].y,

path[indexPath].x, path[indexPath].y);

++indexPath;

} // if

} // func

static void onButtonPress(XEvent xe) {

if (EventCursorIsWithinCanvas(xe) && indexPath < MAX\_INDEX\_PATH) {

path[indexPath].x = xe.xmotion.x;

path[indexPath].y = xe.xmotion.y;

pathColor[indexPath] = color;

pathWidth[indexPath] = lineWidth;

++indexPath;

}

} // func

static void onButtonRelease(XEvent xe) {

int result;

XEvent local\_xe = xe;

if (EventCursorIsWithinColorPick(local\_xe)) {

// printf("EventCursorIsWithinColorPick\n");

if ((result = GetColorPick(local\_xe)) != -1) {

SetForegroundToColorIndex(result);

} else if (GetClearPick(local\_xe)) {

Clear();

}

} // if

else if (EventCursorIsWithinWidthPick(local\_xe)) {

// printf("EventCursorIsWithinWidthPick\n");

if ((result = GetWidthPick(local\_xe)) != -1) {

SetLineWidth(result \* 5 + 5);

}

}else if (EventCursorIsWithinCanvas( local\_xe)){

path[indexPath].x = -1;

path[indexPath].y = -1;

pathColor[indexPath] = color;

pathWidth[indexPath] = lineWidth;

++indexPath;

} // elif

} // func

// input processing routine for main thread

static void UserInputMessageLoop() {

for (;;) {

XNextEvent(dpy, &xe);

if(isDrawer){

switch (xe.type) {

case Expose:

onDraw(xe);

break;

case MotionNotify:

onMouseMoved(xe);

break;

case ButtonPress:

onButtonPress(xe);

break;

case ButtonRelease:

onButtonRelease(xe);

break;

case EnterNotify:

break;

case LeaveNotify:

break;

default: // Neither of above

printf("xe.type unknown: check either XSelectInput \n");

break;

}

}else{

switch (xe.type) {

case Expose:

onDraw(xe);

break;

case MotionNotify:

break;

case ButtonPress:

break;

case ButtonRelease:

break;

case EnterNotify:

break;

case LeaveNotify:

break;

default: // Neither of above

printf("xe.type unknown: check either XSelectInput() \n");

break;

}

}

} // for(;;)

} // func

extern void \*Thread1() {

UserInputMessageLoop();

pthread\_exit(NULL);

} // func

ipc.c

#include "header.h"

#include <fcntl.h>

#include <errno.h>

#define YO = 100

extern int fdDrawing;

extern int fdAnswer;

extern int nwrite;

static char \*strDrawing = "Drawing";

static char \*strAnswer = "Answer";

static int wroteNumber;

static int fdDrawing;

static int fdAnswer;

char answerStr[100];

void fatal(char \*err){

perror(err);

exit(0);

}

static void InitFifoFdDrawing() { // for reader

// ... .. .. ..

wroteNumber = 0;

if (isDrawer){

if ((fdDrawing = open(strDrawing, O\_WRONLY | O\_NONBLOCK)) <0) {

fatal("fifo open failed");

}

}

else{

if (mkfifo(strDrawing, 0666) == -1){

if (errno != EEXIST) {

fatal("fifo exist");

}

}

if ((fdDrawing = open(strDrawing, O\_RDWR))< 0) {

fatal("fifo open failed");

}

}

}

static void InitFifoFdAnswer(){

if (!isDrawer){

if ((fdAnswer = open(strDrawing, O\_WRONLY | O\_NONBLOCK)) <0) {

fatal("fifo open failed");

}

}

else{

if (mkfifo(strAnswer, 0666) == -1){

if (errno != EEXIST) {

fatal("fifo exist");

}

}

if ((fdAnswer = open(strAnswer, O\_RDWR))< 0) {

fatal("fifo poen failed");

}

}

}

extern void SndGameOver() {

int nwrite;

int yo = 100;

char yooo[4];

sprintf(yooo, "%d", yo);

if ((nwrite = write(fdAnswer, yooo, sizeof(yooo))) == -1)

fatal("failed to send a signal");

//read fifo gameover fifo

}

extern void SndAnswerCorrect(char \*strAnswer) {

char bufAnswer[100];

int nwrite;

sprintf(bufAnswer, "%s", strAnswer);

printf("%s \n", bufAnswer);

if ((nwrite = write(fdAnswer, bufAnswer, sizeof(bufAnswer))) == -1)

fatal("answer write failed");

//SendFifo

}

static void IpcLoopWriter() {

char yooo[4];

char buf[10];

int nwrite;

int i;

int yo;

//printf("defug: ans: %s", strAnswerCorrect)ead failed");

for (;;) {

for (i = wroteNumber; i<indexPath; i++){

sprintf(buf, "%d", path[i].x);

if ((nwrite = write(fdDrawing, buf, 10)) == -1){

fatal("message write failed");

}

sprintf(buf, "%d", path[i].y);

if ((nwrite = write(fdDrawing, buf, 10)) == -1){

fatal("message write failed");

}

sprintf(buf, "%ld", pathColor[i]);

if ((nwrite = write(fdDrawing, buf, 10)) == -1){

fatal("message write failed");

}

sprintf(buf, "%d", pathWidth[i]);

if ((nwrite = write(fdDrawing, buf, 10)) == -1){

fatal("message write failed");

}

//read gameover signal

if (read(fdAnswer, yooo, sizeof(yooo))>0){

/\* yo = atoi(yooo);

if(yo==YO){

bGameOver = true;

}\*/

}

wroteNumber = i;

}

}

}

static void IpcLoopReader() {

char answerBuf[100];

char buf[10];

if (read(fdAnswer, answerBuf, sizeof(answerBuf))<0)

strcpy(strAnswerCorrect, answerBuf);

printf("defug: ans: %s", strAnswerCorrect);

for (;;){

if (read(fdDrawing, buf, 10)<0) fatal("message read failed");

path[indexPath].x = atoi(buf);

// printf("x : %d",path[indexPath].x);

if (read(fdDrawing, buf, 10) <= 0) fatal("message read failed");

path[indexPath].y = atoi(buf);

// printf("y : %d",path[indexPath].y);

if (read(fdDrawing, buf, 10) <= 0) fatal("message read failed");

pathColor[indexPath] = atoi(buf);

// printf("color:%d",pathColor[indexPath]);

if (read(fdDrawing, buf, 10) <= 0) fatal("message read failed");

pathWidth[indexPath] = atoi(buf);

// printf("width:%d\n",pathWidth[indexPath]);

indexPath++;

RepaintPath();

}

}

extern void \*Thread2Writer() {

InitFifoFdDrawing();

InitFifoFdAnswer();

IpcLoopWriter();

pthread\_exit(NULL);

}

extern void \*Thread2Reader() {

InitFifoFdDrawing();

InitFifoFdAnswer();

IpcLoopReader();

pthread\_exit(NULL);

}

drawing.c

#include "header.h"

//////////////////// color ////////////////////

// color related

#define MAX\_COLOR 18 // number of basic colors available to pick

#define WHITE 17

static char \*color\_name[MAX\_COLOR] = {

"Black", "Red", "Green", "Blue", "Navy",

"Orange", "Yellow", "Pink", "Magenta", "Cyan",

"Brown", "Grey", "LimeGreen", "Turquoise", "Violet",

"Wheat", "Purple", "White"

};

// "Black", "Red", "Green", "Blue", "Navy",

// "Orange", "Yellow", "Pink", "Magenta", "Cyan",

// "Brown", "Grey", "LimeGreen", "Turquoise", "Violet",

// "Wheat", "Purple", "White"

// ex) XSetForeground(dpy, gc, UsrColorPixel("blue"));

static unsigned long UsrColorPixel(char \*name) { // get color

XColor c0, c1;

Colormap cmap;

cmap = DefaultColormap(dpy, 0);

XAllocNamedColor(dpy, cmap, name, &c0, &c1);

return (c0.pixel);

}

extern void SetForegroundToColorIndex(int i) {

color = UsrColorPixel(color\_name[i]);

XSetForeground(dpy, gc, color);

}

/\*

static void SetForegroundToColorName(char\* name) {

color = UsrColorPixel(name);

XSetForeground(dpy, gc, color);

}

\*/

extern void SetForegroundToColor(unsigned long \_color) {

color = \_color;

XSetForeground(dpy, gc, color);

}

//////////////////// ////////////////////

extern void SetLineWidth(int width) {

XGCValues gv;

lineWidth = width;

gv.line\_width = width;

XChangeGC(dpy, gc, GCLineWidth, &gv);

}

#define MAX\_NPICK\_WIDTH 6

extern void DrawPallete() {

int i;

XGCValues gv;

//canvas

XSetForeground(dpy, gc, BlackPixel(dpy, 0));

gv.line\_width = 4;

XChangeGC(dpy, gc, GCLineWidth, &gv);

XDrawRectangle(dpy, w, gc, CANVAS\_X, CANVAS\_Y, CANVAS\_WIDTH, CANVAS\_HEIGHT);

//answer

XSetForeground(dpy, gc, UsrColorPixel("blue"));

gv.line\_width = 4;

XChangeGC(dpy, gc, GCLineWidth, &gv);

XDrawRectangle(dpy, w, gc, PLAYER\_X, PLAYER\_Y, PLAYER\_WIDTH, PLAYER\_HEIGHT);

XSetForeground(dpy, gc, BlackPixel(dpy, 0));

if (isDrawer) {

XDrawString(dpy, w, gc,

PLAYER\_X + (PLAYER\_WIDTH / 5),

PLAYER\_Y + (PLAYER\_HEIGHT / 2),

strAnswerCorrect, strlen(strAnswerCorrect));

}

else {

XDrawString(dpy, w, gc,

PLAYER\_X + (PLAYER\_WIDTH / 5),

PLAYER\_Y + (PLAYER\_HEIGHT / 2),

"?????????? : guess what!", strlen("?????????? : guess what!"));

}

// palets draw

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

XSetForeground(dpy, gc, UsrColorPixel(color\_name[i]));

XFillRectangle(dpy, w, gc,

MARGIN + (PALET\_AREA\_X \* i), MARGIN + PALET\_AREA\_Y,

PALET\_ITEM\_WIDTH, PALET\_ITEM\_HEIGHT);

}

// draw white palet

i = WHITE;

XSetForeground(dpy, gc, BlackPixel(dpy, 0));

gv.line\_width = 1;

XChangeGC(dpy, gc, GCLineWidth, &gv);

XDrawRectangle(dpy, w, gc,

MARGIN + (PALET\_AREA\_X \* i), MARGIN + PALET\_AREA\_Y,

PALET\_ITEM\_WIDTH, PALET\_ITEM\_HEIGHT);

//clear palet

XSetFont(dpy, gc, XLoadFont(dpy, "fixed"));

XDrawString(dpy, w, gc,

MARGIN + (PALET\_AREA\_X \* (i + 1)), MARGIN + PALET\_AREA\_Y + 15,

"CLEAR", 5);

///////// brush width draw////////////

for (i = 0; i< MAX\_NPICK\_WIDTH; i++){

XSetForeground(dpy, gc, BlackPixel(dpy, 0));

gv.line\_width = i \* 5 + 5;

XChangeGC(dpy, gc, GCLineWidth, &gv);

XDrawLine(dpy, w, gc,

MARGIN + (BRUSH\_AREA\_X \* i) + (BRUSH\_ITEM\_WIDTH / 2) - 1,

BRUSH\_AREA\_Y + (BRUSH\_ITEM\_HEIGHT / 2) - 1,

MARGIN + (BRUSH\_AREA\_X \* i) + (BRUSH\_ITEM\_WIDTH / 2) + 1,

BRUSH\_AREA\_Y + (BRUSH\_ITEM\_HEIGHT / 2) + 1);

}

SetForegroundToColor(color);

gv.line\_width = lineWidth;

XChangeGC(dpy, gc, GCLineWidth, &gv);

}

extern void Clear() {

XClearArea(dpy, w, CANVAS\_X, CANVAS\_Y, CANVAS\_WIDTH, CANVAS\_HEIGHT, 0);

indexPath = 0;

}

extern bool EventCursorIsWithinCanvas(XEvent xe) {

return (xe.xmotion.x > CANVAS\_X && xe.xmotion.x < CANVAS\_WIDTH + CANVAS\_X)

&& (xe.xmotion.y > CANVAS\_Y && xe.xmotion.y < CANVAS\_HEIGHT + CANVAS\_Y);

}

extern bool EventCursorIsWithinColorPick(XEvent xe) {

return xe.xmotion.y > MARGIN + PALET\_AREA\_Y && xe.xmotion.y < MARGIN + PALET\_AREA\_Y + PALET\_ITEM\_HEIGHT;

}

extern bool EventCursorIsWithinWidthPick(XEvent xe) {

return xe.xmotion.y > BRUSH\_AREA\_Y && xe.xmotion.y < BRUSH\_AREA\_Y + BRUSH\_ITEM\_HEIGHT;

}

extern bool EventCursorIsBeingClicked(XEvent xe) {

return xe.xmotion.state & Button1Mask;

}

extern bool GetClearPick(XEvent xe) {

int i = MAX\_COLOR;

return xe.xmotion.x > MARGIN + (BRUSH\_AREA\_X \* i) && xe.xmotion.x < MARGIN + (BRUSH\_AREA\_X \* i) +

BRUSH\_ITEM\_WIDTH;

}

extern int GetColorPick(XEvent xe) {

int i;

for (i = 0; i < MAX\_COLOR + 1; ++i) {

if (xe.xmotion.x > MARGIN + (BRUSH\_AREA\_X \* i)

&& xe.xmotion.x < MARGIN + (BRUSH\_AREA\_X \* i) + BRUSH\_ITEM\_WIDTH) {

break;

} // if

} // for

if (i >= MAX\_COLOR) { return -1; }

// printf("color %d \n", i);

return i;

}

extern int GetWidthPick(XEvent xe) {

int i;

for (i = 0; i < MAX\_COLOR + 1; ++i) {

if (xe.xmotion.x > MARGIN + (PALET\_AREA\_X \* i)

&& xe.xmotion.x < MARGIN + (PALET\_AREA\_X \* i) + PALET\_ITEM\_WIDTH) {

break;

} // if

} // for

if (i >= MAX\_NPICK\_WIDTH) { return -1; }

// printf("width arr n %d \n", i);

return i;

}

extern void RepaintPath(){

int i; // for variable

int contextColor = color; // ex color

int contextWidth = lineWidth; // ex width

SetForegroundToColor(pathColor[0]);

SetLineWidth(pathWidth[0]);

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

if (path[i].x != -1) { //

if (path[i - 1].x != -1) { //

XDrawLine(dpy, w, gc, path[i - 1].x, path[i - 1].y, path[i].x, path[i].y); //

}

else if (path[i - 1].x == -1) { //

SetForegroundToColor(pathColor[i]); //

SetLineWidth(pathWidth[i]);

}

}

else {}

}

SetForegroundToColor(contextColor);

SetLineWidth(contextWidth);

} // func

typo.c

#include "header.h"

extern void TypoSetInputAnswer() {

printf("tell me what your answer is: ");

tcflush(0, TCIFLUSH);

scanf("%s", strAnswerCorrect);

getchar();

printf("the answer is %s\n", strAnswerCorrect);

tcflush(0, TCIFLUSH);

SndAnswerCorrect(strAnswerCorrect);

}

extern void TypoInputLoopWriter() {

int i;

for (; !bGameOver; sleep(3));

for (i = 0; i < 5; ++i) {

printf("%s, your mate got that one right\n", strAnswerCorrect);

}

}

extern void TypoInputLoopReader() {

int i;

char tmp\_strAnswer[100];

for (;; tcflush(0, TCIFLUSH)) {

printf("\ntell me what you think the answer is: \n");

tcflush(0, TCIFLUSH);

scanf("%s", tmp\_strAnswer); // .. ...

if (strcmp(strAnswerCorrect, tmp\_strAnswer) == 0) { // ....

for (i = 0; i < 5; ++i) {

printf("%s, you got that right\n", tmp\_strAnswer);

}

SndGameOver();

return;

}

else { // ....

printf("your answer is %s, which is incorrect. lol\n", tmp\_strAnswer);

}

}

}