Advanced IPC 기법(message passing)을 이용한 버전

header.h

// header.h : 유일한 헤더

#include <stdio.h>

#include <pthread.h>

#include <X11/Xlib.h>

#include <X11/Xutil.h>

#include <X11/keysymdef.h>

//

#include <termios.h>

#include <stdio.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;

// 창에 요소가 배치될 때 창의 왼쪽이나 위쪽 여백

#define MARGIN 50

// 캔버스 영역: 그림 그리는 곳

#define CANVAS\_X MARGIN

#define CANVAS\_Y MARGIN

#define CANVAS\_WIDTH 600

#define CANVAS\_HEIGHT 300

// 플레이어 영역: 오른쪽에 그려지는 사각형 두 개

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

#define PLAYER\_Y MARGIN

#define PLAYER\_WIDTH 300

#define PLAYER\_HEIGHT CANVAS\_HEIGHT

// 물감 (색상 선택) 영역

#define PALET\_AREA\_X MARGIN

#define PALET\_AREA\_Y 370

#define PALET\_ITEM\_WIDTH 30

#define PALET\_ITEM\_HEIGHT 30

// 붓의 굵기 선택 영역

#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 XPoint path[MAX\_INDEX\_PATH];

extern int pathColor[MAX\_INDEX\_PATH];

extern int pathWidth[MAX\_INDEX\_PATH];

// drawing

extern void ContinuePath(int x, int y);

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 SetLineWidth(int width);

// typo

extern void TypoSetInputAnswer();

extern void TypoInputLoopWriter();

extern void TypoInputLoopReader();

// ipc

extern void SndPath(int \_index, int \_x, int \_y, int \_color, int \_width);

extern void SndAnswerCorrect(char \*strAnswer);

extern void SndGameOver();

// init

extern void InitDisplay();

extern void IpcInit();

extern void IpcInitClear();

// threads

extern void \*Thread1();

extern void \*Thread2Writer();

extern void \*Thread2Reader();

main.c

// main.c : 주함수와 스레드 관리

#include "header.h"

#define N\_THREADS 2

static pthread\_t thid[N\_THREADS];

// externs

char strAnswerCorrect[100] = "nn45nn3c8n";

bool isDrawer = false;

bool bGameOver = false;

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

// 옵션 지정하지 않았을 때 에러 출력하고 종료

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;

}

// writer와 reader 인자 구분

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;

}

// 본편 실행

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

if (isDrawer) {

IpcInitClear(); // ipc 초기화

InitDisplay(); // x window 초기화

TypoSetInputAnswer(); // 사용자에게 제시어(맞혀야 할 단어) 입력 받음

pthread\_create(&thid[0], NULL, Thread1, NULL); // 스레드0: x window 입력 받음

pthread\_create(&thid[1], NULL, Thread2Writer, NULL); // 스레드1: ipc 메시지 들음

printf("hello there \n");

TypoInputLoopWriter();

} else /\* if (!isDrawer) \*/ {

IpcInit();

InitDisplay();

pthread\_create(&thid[0], NULL, Thread1, NULL); // x window

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;

}

}

\*/

// 종료

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

return 0;

}

event.c

// event.c : 창의 초기화와 이벤트 처리 거시적 관점

#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];

int pathColor[MAX\_INDEX\_PATH];

int pathWidth[MAX\_INDEX\_PATH];

extern void InitDisplay() {

// 1. 창 띄우고 이벤트(입력) 연결

dpy = XOpenDisplay(NULL);

w = XCreateSimpleWindow(dpy, RootWindow(dpy, 0), 50, 50, 1050, 500, // Display, Window, x, y, width, height

5, BlackPixel(dpy, 0), WhitePixel(dpy, 0)); // border width, border color, bg color

XSelectInput(dpy, w, ExposureMask |

ButtonPressMask | ButtonReleaseMask | // 마우스 버튼 눌림, 뗌

// EnterWindowMask | LeaveWindowMask | // 창에 들어옴, 나감

PointerMotionMask // 커서 이동 (MotionNotify 발생)

);

XMapWindow(dpy, w);

XFlush(dpy);

// 2. gc, font, line, width, color 설정

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)

static void onExpose() {

DrawPallete();

RepaintPath();

} // func

static void onMouseMoved(XEvent xe) {

if (!isDrawer) { return; } // 이 프로세스가 그림 그리는 측인 경우에만 그린다

// 캔버스 안에서 커서가 눌린 채 움직였고 그림 데이터 저장공간이 여유 있다면

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

// 현재위치 저장

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

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

// 직전위치부터 현재위치까지 잇는 선을 그린다 (드로우)

if (indexPath == 0) { return; }

XDrawLine(dpy, w, gc,

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

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

// 다음으로 (현재위치는 직전위치가 된다)

++indexPath;

// 기본 UI 그려주기

// DrawPallete();

// 통신 추가

if (isDrawer) {

SndPath(indexPath - 1, xe.xmotion.x, xe.xmotion.y, color, lineWidth);

}

} // if

} // func

static void onButtonPress(XEvent xe) {

if (!isDrawer) { return; } // 이 프로세스가 그림 그리는 측인 경우에만 그린다

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

ContinuePath(xe.xmotion.x, xe.xmotion.y);

}

} // func

static void onButtonRelease(XEvent xe) {

int result;

if (!isDrawer) { return; } // 이 프로세스가 그림 그리는 측인 경우에만 그린다

XEvent local\_xe = xe;

if (EventCursorIsWithinColorPick(local\_xe)) { // 색상 선택

// printf("EventCursorIsWithinColorPick\n");

if ((result = GetColorPick(local\_xe)) != -1) { // 몇 번째 요소인지 확인하고

SetForegroundToColorIndex(result); // 그 요소의 색상을 적용한다

} else if (GetClearPick(local\_xe)) { // "CLEAR" 선택시

Clear();

}

} else if (EventCursorIsWithinWidthPick(local\_xe)) { // 붓 굵기 선택

// printf("EventCursorIsWithinWidthPick\n");

if ((result = GetWidthPick(local\_xe)) != -1) { // 몇 번째 요소인지 확인하고

SetLineWidth(result \* 5 + 5); // 그 요소의 굵기를 적용한다

}

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

ContinuePath(-1, -1);

} // elif

} // func

// input processing routine for main thread

static void UserInputMessageLoop() {

for (;;) {

XNextEvent(dpy, &xe);

switch (xe.type) {

case Expose: // 창 크기 변경 (resize)

onExpose(xe);

break;

case MotionNotify: // 커서 이동

onMouseMoved(xe);

break;

case ButtonPress: // 마우스 클릭: 누름

onButtonPress(xe);

break;

case ButtonRelease: // 마우스 클릭: 뗌

onButtonRelease(xe);

break;

case EnterNotify: // 커서가 창 안으로 들어옴

break;

case LeaveNotify: // 커서가 창 밖으로 나감

break;

default: // Neither of above

printf("xe.type unknown: check either XSelectInput() or switch case (warning) \n");

break;

}

} // for(;;)

} // func

extern void \*Thread1() {

UserInputMessageLoop();

pthread\_exit(NULL);

} // func

ipc.c

// ipc1 : 그리는 입장

// ipc.c : 프로세스간 통신

#include "header.h"

#define KEY (key\_t)600201

#define PERMISSION IPC\_CREAT | 0660

#define MSGTYPE\_PATH 1

#define MSGTYPE\_ANSWER 2

#define MSGTYPE\_GAMEOVER 3

static int qid;

// 주고 받을 데이터: 그린 점

struct path {

long data\_type;

int index;

int x;

int y;

int color;

int width;

};

// 주고 받을 데이터: 정답

struct answer {

long data\_type;

int n;

char str[100];

};

#define YO 100 // 관련: struct gameover

// reader가 정답을 맞히면 게임이 끝났음을 알리는 메시지

struct gameover {

long data\_type;

int yo;

};

// 오류

static void fatal(char \*err) {

perror(err);

exit(0);

}

// 큐를 비운다

static void ClearQueue() {

struct path buf1;

struct answer buf2;

struct gameover buf3;

for (;msgrcv(qid, &buf1, sizeof(struct path) - sizeof(long), MSGTYPE\_PATH, IPC\_NOWAIT) == 0;);

for (;msgrcv(qid, &buf2, sizeof(struct answer) - sizeof(long), MSGTYPE\_ANSWER, IPC\_NOWAIT) == 0;);

for (;msgrcv(qid, &buf3, sizeof(struct gameover) - sizeof(long), MSGTYPE\_GAMEOVER, IPC\_NOWAIT) == 0;);

}

// includes msgctl() msgget()

extern void IpcInit() {

msgctl(qid, IPC\_RMID, NULL);

if (-1 == (qid = msgget(KEY, PERMISSION))) {

fatal("failed to init queue");

}

}

// includes msgctl() msgget() // ClearQueue()

extern void IpcInitClear() {

msgctl(qid, IPC\_RMID, NULL);

if (-1 == (qid = msgget(KEY, PERMISSION))) {

fatal("failed to init queue");

}

ClearQueue();

}

// 게임 끝내기

extern void SndGameOver() {

struct gameover buf;

buf.data\_type = MSGTYPE\_GAMEOVER;

buf.yo = YO;

if (-1 == msgsnd(qid, &buf, sizeof(struct gameover) - sizeof(long), 0)) {

fatal("SndGameOver msgsnd() failed");

}

}

// 정답 설정하기

extern void SndAnswerCorrect(char \*strAnswer) {

struct answer buf;

//

buf.data\_type = MSGTYPE\_ANSWER;

sprintf(buf.str, "%s", strAnswer); // strcpy(buf.str, strAnswer);

printf("%s \n", buf.str);

if (-1 == msgsnd(qid, &buf, sizeof(struct answer) - sizeof(long), 0)) {

fatal("SndAnswerCorrect msgsnd() failed");

}

}

// 그린 지점 한 단위를 메시지 전송

extern void SndPath(int \_index, int \_x, int \_y, int \_color, int \_width) {

struct path buf;

// copy path

buf.data\_type = MSGTYPE\_PATH;

buf.index = \_index;

buf.x = \_x;

buf.y = \_y;

buf.color = \_color;

buf.width = \_width;

if (-1 == msgsnd(qid, &buf, sizeof(struct path) - sizeof(long), 0)) {

fatal("SndPath msgsnd() failed");

}

}

// 받은 그린 데이터(구조체 메시지; 점 하나)를 풀어서 저장

static void RcvPath(struct path rcvpath) {

if (indexPath >= MAX\_INDEX\_PATH) { return; } // 여분의 공간이 없으면 추가할 수 없음

// 현재위치 저장

path[rcvpath.index].x = rcvpath.x;

path[rcvpath.index].y = rcvpath.y;

// 현재위치와 함께 다른 (추가) 정보 저장

pathColor[rcvpath.index] = rcvpath.color;

pathWidth[rcvpath.index] = rcvpath.width;

// CLEAR 입력 받은 경우

if (rcvpath.index == -1) {

Clear(); // 지운다

indexPath = 0;

}

// 다음으로

if (indexPath <= rcvpath.index) {

indexPath = rcvpath.index + 1;

}

}

// 문제를 내고 그리는 측(writer)이 메시지를 듣게 함

// ("Loop"은 쓰기(메시지의 발생)가 아닌 듣게(읽게) 하기 위한 것... 타이머 이벤트가 아닌 이상)

// (메시지의 전송은 메시지가 발생한 이벤트에서 수행하는 것이 옳다)

static void IpcLoopWriter() {

struct gameover buf;

// gameover 메시지를 체크하다가 받는 순간 게임 끝낸다

for (;;) {

if (-1 == msgrcv(qid, &buf, sizeof(struct path) - sizeof(long), MSGTYPE\_GAMEOVER, 0)) {

fatal("failed to msgrcv()");

}

if (buf.yo == YO) {

bGameOver = true;

}

}

} // func

// 정답을 맞히는 측(reader)이 메시지를 듣게 함

// ("Loop"은 쓰기(메시지의 발생)가 아닌 듣게(읽게) 하기 위한 것... 타이머 이벤트가 아닌 이상)

// (메시지의 전송은 메시지가 발생한 이벤트에서 수행하는 것이 옳다)

static void IpcLoopReader() {

struct path buf;

// 먼저 제시어를 읽어들인다

struct answer buf\_answer;

if (-1 == msgrcv(qid, &buf\_answer, sizeof(buf\_answer) - sizeof(long), MSGTYPE\_ANSWER, 0)) {

fatal("failed to msgrcv()");

}

strcpy(strAnswerCorrect, buf\_answer.str);

printf("\n your mate came up with a word. guess what! \n\n");

// printf("debug: ans: %s \n", strAnswerCorrect);

// 그린 데이터를 읽는다

for (;;) {

if (-1 == msgrcv(qid, &buf, sizeof(struct path) - sizeof(long), MSGTYPE\_PATH, 0)) {

fatal("failed to msgrcv()");

}

RcvPath(buf);

RepaintPath();

}

} // func

// 이 루틴을 writer(그리는 측)만이 호출한다

extern void \*Thread2Writer() {

IpcLoopWriter();

pthread\_exit(NULL);

}

// 이 루틴을 reader(읽는 측)만이 호출한다

extern void \*Thread2Reader() {

IpcLoopReader();

pthread\_exit(NULL);

}

// 프로그램은 writer(그리는 측)를 먼저 실행한 뒤 reader(읽는 측)를 실행해야 한다

drawing.c

// drawing.c 모든 자잘한 (미시적인) 그리는 처리와 그에 연관된 것

#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);

}

// GC의 선의 색상을 지정

extern void SetForegroundToColorIndex(int i) {

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

XSetForeground(dpy, gc, color);

}

/\*

// GC의 선의 색상을 지정

static void SetForegroundToColorName(char\* name) {

color = UsrColorPixel(name);

XSetForeground(dpy, gc, color);

}

\*/

// GC의 선의 색상을 지정

static void SetForegroundToColor(unsigned long \_color) {

color = \_color;

XSetForeground(dpy, gc, color);

}

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

// GC의 선의 굵기를 지정

extern void SetLineWidth(int width) {

XGCValues gv;

lineWidth = width;

gv.line\_width = width;

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

}

// path에 점 하나를 추가함

extern void ContinuePath(int x, int y) {

if (indexPath >= MAX\_INDEX\_PATH) { return; } // 여분의 공간이 없으면 추가할 수 없음

// 현재위치 저장

path[indexPath].x = x;

path[indexPath].y = y;

// 현재위치와 함께 다른 (추가) 정보 저장

pathColor[indexPath] = color;

pathWidth[indexPath] = lineWidth;

// 다음으로 (현재위치는 직전위치가 된다)

++indexPath;

// 통신 추가

if (isDrawer) {

SndPath(indexPath - 1, x, y, color, lineWidth);

}

}

#define MAX\_NPICK\_WIDTH 6

// 창에 기본 UI 그리는 함수

extern void DrawPallete() {

int i;

XGCValues gv;

// 캔버스 영역 드로우

XSetForeground(dpy, gc, BlackPixel(dpy, 0)); // 임시로 그릴 거니까 static void SetForegroundToColor() 사용하지 않음

gv.line\_width = 4;

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

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

// 정답 영역 드로우

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);

// 정답 영역 문자

XSetForeground(dpy, gc, BlackPixel(dpy, 0)); // 임시로 그릴 거니까 static void SetForegroundToColor() 사용하지 않음

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!"));

}

// 물감 영역 드로우: 흰색 차례 직전까지 색깔 순서대로

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);

}

// 물감 영역 드로우: 흰색

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);

// 물감 영역 드로우: CLEAR

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

XDrawString(dpy, w, gc,

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

"CLEAR", 5);

// 붓의 굵기 드로우

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);

}

// 이전 콘텍스트로 되돌아간다

SetForegroundToColor(color);

gv.line\_width = lineWidth;

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

}

// 다시 따라 그린다

extern void RepaintPath() {

int i; // 반복문 제어 변수

int contextColor = color; // 뒷정리 준비: 이 함수가 끝나면 돌아갈 곳 1 (스택 프레임처럼 시작할 때 한다)

int contextWidth = lineWidth; // 뒷정리 준비: 이 함수가 끝나면 돌아갈 곳 2

SetForegroundToColor(pathColor[0]);

SetLineWidth(pathWidth[0]);

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

if (path[i].x != -1) { // 지금 지점이 끊어진 점(-1)이 아닌 유효한 점이니까, 그리기를 시작하는 중이던지 그리는 한 중간이던지 둘 중 하나다.

if (path[i - 1].x != -1) { // 이전 지점도 지금 지점도 끊어진 점(-1)이 아니니까, 붓칠을 하는 중간이다.

XDrawLine(dpy, w, gc, path[i - 1].x, path[i - 1].y, path[i].x, path[i].y); // 두 점을 잇는 선을 따라 붓칠한다.

} else if (path[i - 1].x == -1) { // 이전 지점은 끊어졌고 지금 지점은 있으니까, 두점을 따라 잇는 선을 그리기 시작하는 때다.

SetForegroundToColor(pathColor[i]); // 붓을 든다.

SetLineWidth(pathWidth[i]);

}

} else { /\* 지금 지점이 끊어진 점(-1)이 아닌 유효한 점이니까, 그리기를 중단한 상태다. 붓을 내린다. \*/ }

}

SetForegroundToColor(contextColor); // 뒷정리

SetLineWidth(contextWidth);

} // func

// 캔버스에 그린 것을 지운다

extern void Clear() {

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

indexPath = 0;

// 통신 추가

if (isDrawer) {

SndPath(-1, -1, -1, color, lineWidth);

}

}

// 판별

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;

}

//

버튼이 눌렸는지 여부 반환

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;

}

// 몇 번째 버튼이 눌렸는지 반환

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;

}

// 몇 번째 버튼이 눌렸는지 반환

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;

}

typo.c

// 키보드 입력 받기 (본 프로그램은 이것을 메인 스레드에서 실행한다)

#include "header.h"

// 그리는(writer) 측에게 제시어 입력 받음

// 주의: ipc 초기화된 이후에 호출되어야 한다

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);

}

// 그리는(writer) 측의 경우 정답을 맞힐 때까지 실행

// 주의: ipc 초기화된 이후에 호출되어야 한다

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);

}

}

// 맞히는(reader) 측에게 제시어 입력 받음

// 주의: ipc 초기화된 이후에 호출되어야 한다

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); // 입력 받아서

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

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

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

}

SndGameOver();

return;

} else { // 틀렸다면

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

}

}

}