МИНОБРНАУКИ РОССИИ

Федеральное государственное автономное образовательное

учреждение высшего образования

«ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

**Инженерно-технологическая академия**

**Институт компьютерных технологий и информационной безопасности**

**Кафедра Математического обеспечения и применения ЭВМ**

**ОТЧЁТ**

по лабораторной работе

по курсу «ОСРВ»

на тему «Разработка клиент-серверного приложения для ОСРВ QNX»

Выполнили:

ст. группы КТмо1-3

Калиниченко Г.А.

Цветков С.В.

Проверил:

преподаватель МОП ЭВМ

Пирская Л. В.

Оценка

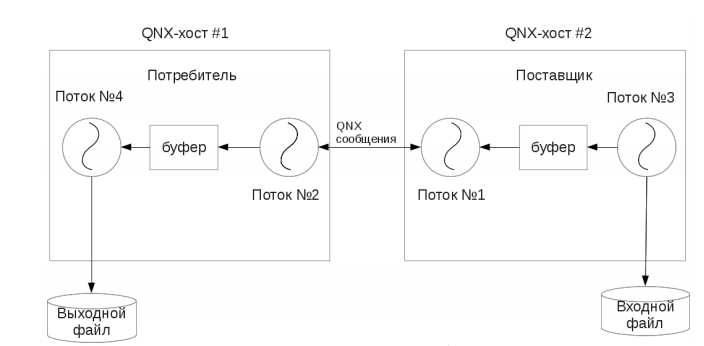
«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2017 г.

Таганрог 2017

**Цель работы.** Необходимо разработать сетевое многопоточное приложение, реализующее классическую задачу из области синхронизации процессов «поставщики-потребители» в распределенной среде QNX.

**Задание**

Необходимо разработать сетевое многопоточное приложение, реализующее классическую задачу из области синхронизации процессов «поставщики-потребители» (http://cs.mipt.ru/docs/courses/osstud/06/ch6.htm) в распределенной среде QNX. Поставщик должен прочитать файл с любым изображением в формате BMP или TGA, повернуть его на заранее заданный угол и отправить его потребителю. При этом поставщик должен отправлять файл по частям через равные промежутки времени (100 мс). Размер части в байтах должен задаваться в параметрах приложения при запуске. Части файла должны отправляться не последовательно, а в случайном порядке (перемешиваться), но так, чтобы их последовательность можно было восстановить на принимающей стороне (можно, например, присвоить целочисленные идентификаторы). Потребитель, получив все части, должен масштабировать изображение (увеличить или уменьшить, фактор задается в параметрах приложения при запуске). Приложение должно быть организовано следующим образом:



* поток №1 — периодически выбирает из буфера часть и отсылает по сети потоку №2;
* поток №3 — считывает входной файл, поворачивает изображение и разбивает его на части, далее помещая их в буфер;
* поток №2 — принимает части изображения по сети и кладет их в буфер;
* поток №4 — берет из буфера все части, составляет изображение, масштабирует его и сохраняет в файл;
* примечание: а) размер буферов задается константой, б) в случае переполнения буфера поток №3 блокируется, возобновляя работу при наличии места в буфере; в) алгоритм(алгоритмы) обработки изображений может быть любым (например, афинные преобразования);
* т. к. буфер потребителя также конечного размера, то поток №4 должен успевать обрабатывать части из буфера так, чтобы буфер не переполнялся, иначе необходимо генерировать исключительную ситуацию, с ее последующей обработкой как на стороне Потребителя, так и на стороне Поставщика;
* в случае успешной работы приложения, выходной файл должен содержать первоначальное изображение повернутое на заданный угол и корректно масштабированное (увеличенное или уменьшенное)

**Описание используемых при разработке средств для создания, взаимодействия и синхронизации сетевых процессов и потоков**

Для взаимодействия между сетевыми потоками используются стандартные QNX-сообщения MsgSend, MsgReceive,MsgReply. Данные сообщения предназначены для осуществления коммуникациями между двумя клиентами. Так MsgSend тип сообщения который отвечает за отправку пакета сообщения после посылки клиент будет ожидать от второго клиента по средствам сообщения MsgReply. Сообщение MsgReceive необходимо для получения сообщения.

Схема работы сообщения MsgReceive заключается в ожидании сообщение отправленного клиентом. Если сообщение не было получено клиентом или сервером, сторона, инициировавшее сообщение MsgReceive будет находится в состоянии ожидания пока кто-то не инициализирует сообщение MsgSend для отправки пакета данных.

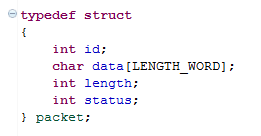
MsgReply предназначен для отправки ответа клиенту. При вызове функции MsgReply() блокировка не происходит, т.е. сервер будет продолжать работать дальше. Это сделано потому, что клиент уже находится в заблокированном состоянии (REPLY) и не требуется какая-либо дополнительная синхронизация

Сообщение MsgSend может принимать 2 состояния send и reply. Состояние send означает, что данные были отправлены серверу но подтверждение что сервер получил эти данные нет. Когда сообщение будет доставлено серверу клиент переходит в состояние reply. В данном состоянии клиент ожидает получения данных с сервера при помощи сообщение MsgReply в котором храниться ответное сообщение.

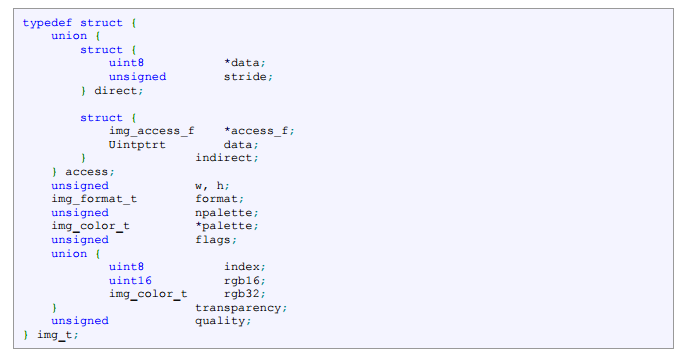
Для работы с потоками используется стандартная структура pthread\_t. Для запуска потоков используется pthread\_create которая принимает id потока и функцию обработчик потоков.

**Описание структуры и алгоритмов разработанной программы**

Для пересылки данных от сервера к клиенту используются пакеты из которых состаит исходное ихображение



Для обработки изображений использовалась стандартная библиотека qnx img.h. Были разработаны функции для поворота изображения и изменения масштаба. Изображение хранится в структуре img\_t



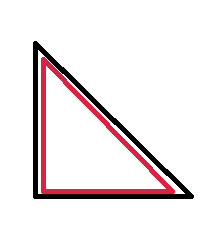
Для поворота изображения, была разработана функция char\* GetImageAndRotate(char\* fileName, int degree), которая принимает на вход имя файла с изображением и угол поворота, выходные данные – это имя файла, сохраненного на диск, повернутого на определенный угол.

Для изменения масштаба изображения, была разработана функция int GetImageAndScale(char\* file\_name, int value), которая принимает на вход имя файла с изображением и значением с коэффициентом масштаба, выходные данные – это имя файла, сохраненного на диск, измененного масштаба. В данной функции происходит масштабирование изображения и подается на вход стандартной функции qnx img img\_resize\_fs().

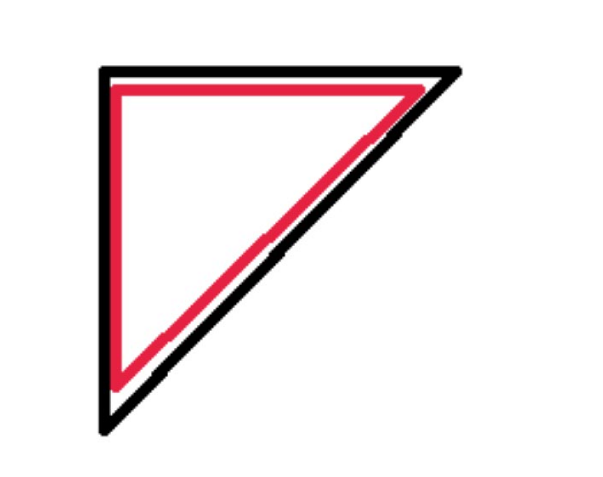
Алгоритм синхронизации пакетов заключается в следующем. Вначале происходит поиск пакета с минимальным значением id. Затем проходя по буферу мы получаем id на 1 больше предыдущего и записываем в файл прямого доступа. Тем самым пакеты соединяется в 1 файл.

**Результаты работы**

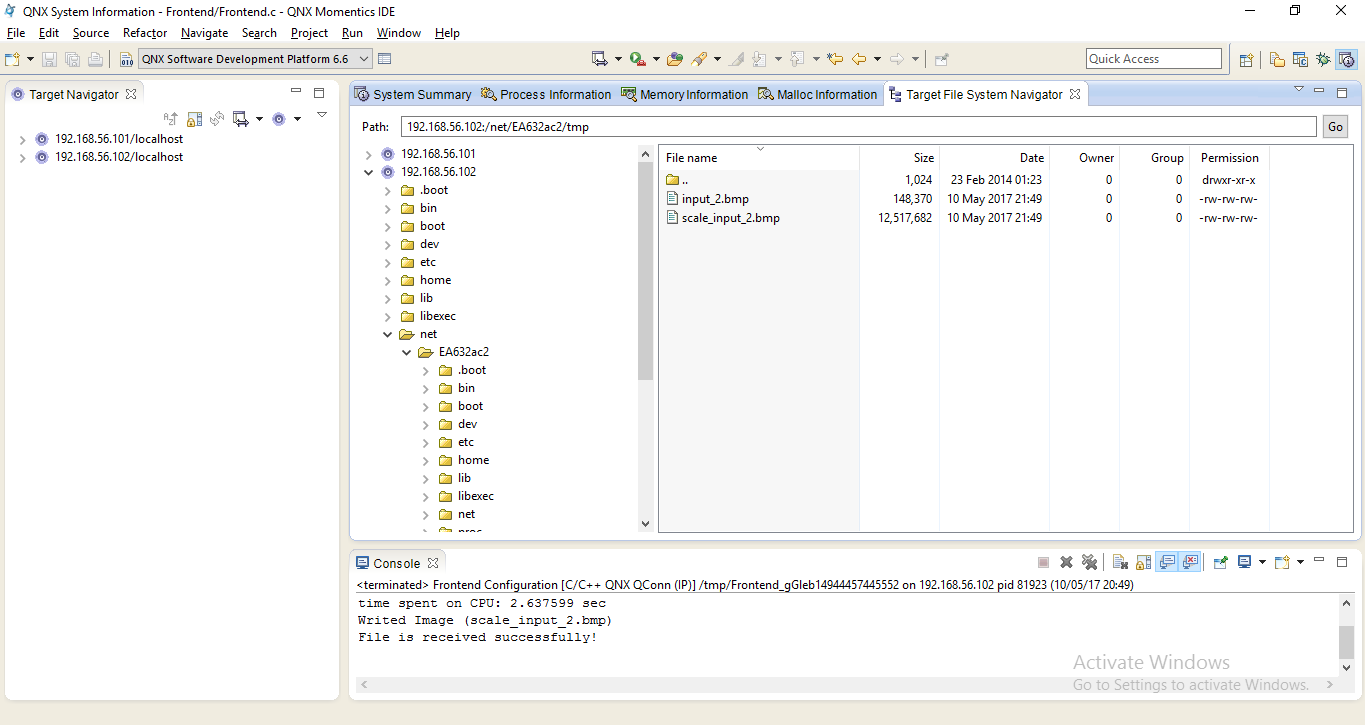
Исходное изображение (261x284, формат bmp). Угол поворота 90 градусов.

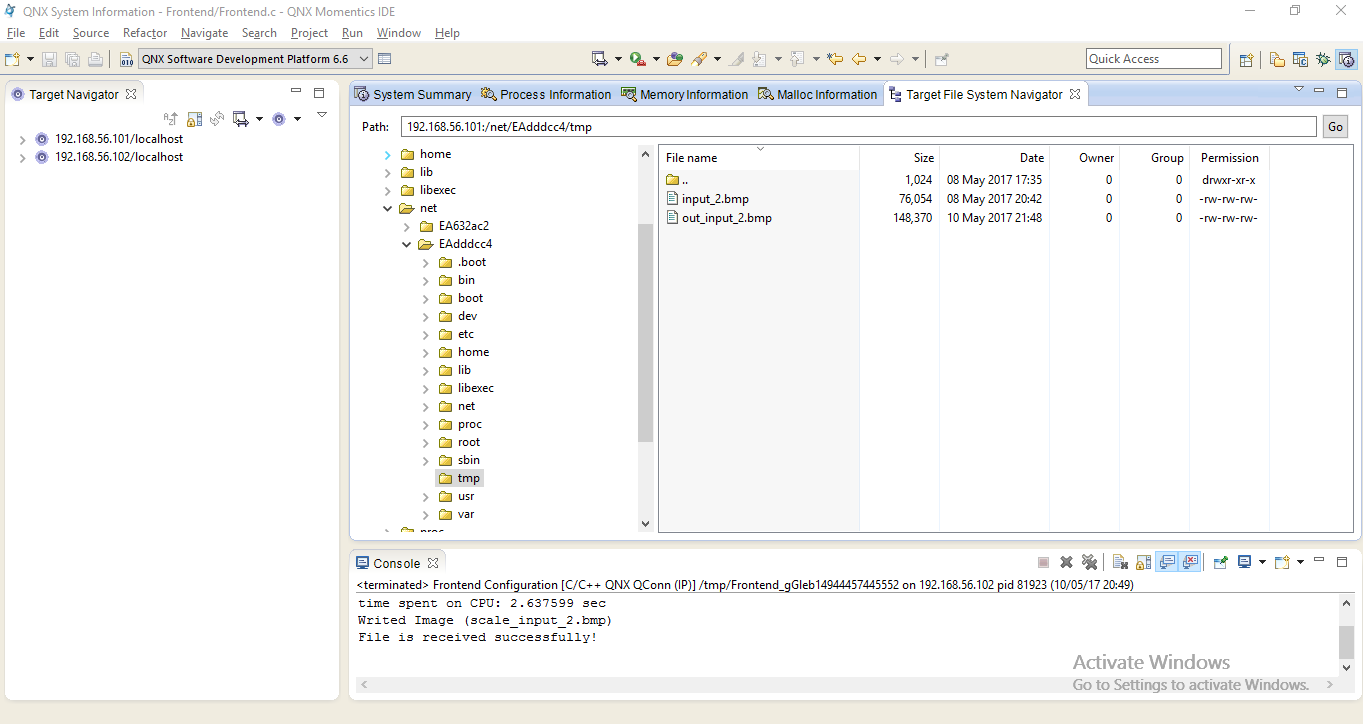


Изображение на выходе (2610х2844, формат bmp). Масштаб х10.



Файловая системы виртуальных машин





**Выводы**

Благодаря данной лабораторной работе, мы познакомились с операционной системой реального времени QNX, средой разработки программного обеспечения QNX Momentics IDE и средствами передачи данных по сети.

**Листинг**

Backend.c

**#include** <stdlib.h>

**#include** <stdio.h>

**#include** <pthread.h>

**#include** <time.h>

**#include** <sys/iofunc.h>

**#include** <sys/dispatch.h>

**#include** <string.h>

**#include** <errno.h>

**#include** <time.h>

**#include** <img/img.h>

**#define** LENGTH\_BUFFER 32

**#define** LENGTH\_WORD 2048

**#define** NAME\_IN\_FILE "input\_2.bmp"

**#define** DEGREE 90

**typedef** **struct**

{

**int** id;

**char** data[LENGTH\_WORD];

**int** length;

**int** status;//free or busy

} packet;

**typedef** **struct**

{

**char** file\_name[30];

**long** **int** size;

} info\_file;

info\_file info;

packet buffer[2][LENGTH\_BUFFER];

**int** buffer\_full[2] = {0 , 0};

**int** select\_buf = 1;

**int** exit\_code = 0;

**int** end\_read\_file = 0;

**char** file\_to\_send[30];

// Ïåðåîäè÷åñêè (100 ìñ) âûáèðàåò èç áóôåðà ÷àñòü è îòïðàâëÿåò å¸ ïàêåòàìè (îäèí ïàêåò =

// îäíî ñîîáùåíèå) ïî ñåòè ïîòîêó ¹2.

**void**\* **handleThreadOne**( **void**\* arg ){

**int** server\_coid = -1;

**while**(server\_coid == -1){

server\_coid = name\_open("One", NAME\_FLAG\_ATTACH\_GLOBAL);

}

// Áóôåð ñ êàêèì ðàáîòàåì.

**int** num\_buf = 1;

// Óêàçûâàåò íà êîíåö ïåðåäà÷è.

**int** end\_transmitted = 0;

// Ìàññèâ èíäåêñîâ íåîáõîäèì äëÿ ñìåøèâàíèÿ ïàêåòîâ.

**int** mas\_index[LENGTH\_BUFFER];

**int** i;

**for** (i=0; i < LENGTH\_BUFFER; i++)

mas\_index[i] = i;

SendInformationMsgToClient(server\_coid);

**while**(end\_transmitted != 1)

{

**switch**(num\_buf)

{

**case** 1:

**if**(buffer\_full[0] == 1)

{

RandomArrayPacket(mas\_index); // Ïåðåìåøèâàåì ïàêåòû.

SendPacketToClient(mas\_index, server\_coid, buffer[0]);

buffer\_full[0] = 0; num\_buf = 2;

}

**if**(end\_read\_file == 1 && buffer\_full[0] == 0 && buffer\_full[1] == 0)

{

end\_transmitted = SendLastPacketToClient(server\_coid);

}

**break**;

**case** 2:

**if**(buffer\_full[1] == 1)

{

RandomArrayPacket(mas\_index);

SendPacketToClient(mas\_index, server\_coid, buffer[1]);

buffer\_full[1] = 0; num\_buf = 1;

}

**if**(end\_read\_file == 1 && buffer\_full[1] == 0 && buffer\_full[0] == 0)

{

end\_transmitted = SendLastPacketToClient(server\_coid);

}

**break**;

} // EndSwitch.

} // EndWhile.

name\_close(server\_coid);

exit\_code = -1;

printf("End send file, thread 1 - CLOSE\n");

**return**( 0 );

}

//get input file and drop on pieces and save in buffer

**void**\* **handleThreadThree**( **void**\* arg )

{

FILE\* file = fopen( file\_to\_send, "rb" );

**if**( file != NULL )

{

**int** i, stop = 0, id = 0;

**char**\* byte[1];

**int** result\_read = 1;

**while**(result\_read == 1 && exit\_code == EOK)

{

**switch**(select\_buf)

{

**case** 1:

**if**(buffer\_full[0] == 0)

{

**int** free\_idbuf = FindFreePlaceInBuffer(buffer[0]);

**if**(free\_idbuf != -1)

{

result\_read = SetDataToBuffer(file, free\_idbuf, buffer[0], &id, byte, 0);

}

**else** buffer\_full[0] = 1;

}

**break**;

**case** 2:

**if**(buffer\_full[1] == 0)

{

**int** free\_idbuf = FindFreePlaceInBuffer(buffer[1]);

**if**(free\_idbuf != -1)

{

result\_read = SetDataToBuffer(file, free\_idbuf, buffer[1], &id, byte, 1);

}

**else** buffer\_full[1] = 1;

}

**break**;

}

}

fclose(file); end\_read\_file = 1;

}

printf("End read file, thread 3 - CLOSE\n");

**return**( 0 );

}

**char**\* **GetImageAndRotate**(**char**\* fileName, **int** degree);

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

strncat(file\_to\_send, GetImageAndRotate(NAME\_IN\_FILE,DEGREE), 30);

SetInfoFile(file\_to\_send);

pthread\_t thread1\_tid, thread3\_tid;

pthread\_create(&thread3\_tid, NULL, &handleThreadThree, NULL );

pthread\_create(&thread1\_tid, NULL, &handleThreadOne, NULL );

**while**(exit\_code == EOK)

{

**switch**(select\_buf)

{

**case** 1:

select\_buf = (buffer\_full[0] == 1 && buffer\_full[1] == 0) ? 2 : 1;

**break**;

**case** 2:

select\_buf = (buffer\_full[1] == 1 && buffer\_full[0] == 0) ? 1 : 2;

**break**;

}

}

sleep(1);

printf("File has been sent!");

sleep(1);

**return** EXIT\_SUCCESS;

}

**char**\* **GetImageAndRotate**(**char**\* fileName, **int** degree)

{

img\_lib\_t ilib = NULL;

img\_lib\_attach(&ilib);

**int** rc;

img\_t img, imgOut;

/\* initialize an img\_t by setting its flags to 0 \*/

img.flags = 0;

img.format = IMG\_FMT\_PKLE\_ARGB1555;

img.flags |= IMG\_FORMAT;

imgOut.flags = 0;

imgOut.format = IMG\_FMT\_PKLE\_ARGB1555;

imgOut.flags |= IMG\_FORMAT;

/\* char\* type;

switch(fileType){

case 1:

type = ".bmp";

break;

case 2:

type = ".tga";

break;

default:

return fileName;

break;

}

\*/

**char** filename[30];

strcpy(filename, fileName);

**if** ((rc = img\_load\_file(ilib, filename, NULL, &img)) != IMG\_ERR\_OK) {

fprintf(stderr, "img\_load\_file(%s) failed: %d\n", filename, rc);

**return** fileName;

}

fprintf(stdout, "Uploaded Image is %dx%dx%d (%s)\n", img.w, img.h, IMG\_FMT\_BPP(img.format), filename);

img\_fixed\_t angel = (**int**)((degree \* 3.141592 / 180) \* 65536.0);

**if**((rc = img\_rotate\_ortho(&img, &imgOut, angel))!= IMG\_ERR\_OK){

fprintf(stderr, "img\_rotate\_ortho failed: %d\n", rc);

**return** fileName;

}

printf("Rotated Image\n");

**char** outfile[60];

strcpy(outfile, "out\_");

strncat(outfile,filename,30);

**if** ((rc = img\_write\_file(ilib,outfile,NULL,&imgOut)) != IMG\_ERR\_OK) {

fprintf(stderr, "img\_write\_file(%s) failed: %d\n", outfile, rc);

**return** fileName;

}

printf("Writed Image (%s)\n", outfile);

/\* for our purposes we''re done with the img lib \*/

img\_lib\_detach(ilib);

**return** outfile;

}

/\*

\* Function for read byte and save in buffer

\* @free\_idbuf position in buffer

\* @byte container for reading

\* \*/

**int** **SetDataToBuffer**(FILE\* file, **int** free\_idbuf, packet\* buffer, **int**\* id\_packet, **char**\* byte, **int** num\_buf){

**int** i, rc = 1, stop = 0;

**for**(i=0; i<LENGTH\_WORD; i++)

{

rc = fread(byte, **sizeof**(**char**), 1, file);

**if**(rc == 1)

{

buffer[free\_idbuf].length = i+1;

buffer[free\_idbuf].data[i] = byte[0];

}

**else**{

buffer[free\_idbuf].status = 1;

buffer[free\_idbuf].id = \*id\_packet;

buffer\_full[num\_buf] = 1;

stop = 1;

**break**;

}

}

**if**(stop != 1)

{

buffer[free\_idbuf].status = 1;

buffer[free\_idbuf].id = \*id\_packet;

\*id\_packet = \*id\_packet + 1;

}

**return** rc;

}

**int** **SendInformationMsgToClient**(**int** server\_coid)

{

**int** reply;

// Îòïðàâëÿåì ïàêåò.

MsgSend(server\_coid, &info, **sizeof**(info), &reply, **sizeof**(**int**));

**while**(reply != EOK)

{

MsgSend(server\_coid, &info, **sizeof**(info), &reply, **sizeof**(**int**));

}

**return** reply;

}

**void** **SendPacketToClient**(**int**\* mas\_index, **int** server\_coid, packet\* buffer ){

**int** i, reply\_client;

**for**(i=0; i<LENGTH\_BUFFER;i++)

{

**if**(buffer[mas\_index[i]].status != 0){

// Îòïðàâëÿåì ïàêåò.

MsgSend(server\_coid, &buffer[mas\_index[i]], **sizeof**(buffer[mas\_index[i]]), &reply\_client, **sizeof**(**int**));

**while**(reply\_client != EOK)

{

MsgSend(server\_coid, &buffer[mas\_index[i]], **sizeof**(buffer[mas\_index[i]]), &reply\_client, **sizeof**(**int**));

}

buffer[mas\_index[i]].status = 0;

}

}

}

**int** **SendLastPacketToClient**(**int** server\_coid){

**int** reply\_client;

packet end\_packet;

end\_packet.status = 230894;

MsgSend(server\_coid, &end\_packet, **sizeof**(end\_packet), &reply\_client, **sizeof**(**int**));

/\* while(reply\_client != EOK)

{

MsgSend(server\_coid, &end\_packet, sizeof(end\_packet), &reply\_client, sizeof(int));

}

\*/

**return** 1;

}

**void** **SetInfoFile**(**char**\* filename){

FILE\* file;

file = fopen(filename,"rb");

fseek(file,0,SEEK\_END);

info.size = ftell(file);

strcpy(info.file\_name, NAME\_IN\_FILE);

fclose(file);

}

**int** **FindFreePlaceInBuffer**(packet\* buffer)

{

**int** i;

**for**(i = 0; i<LENGTH\_BUFFER; i++)

{

//free position 0 free 1 busy

**if**(buffer[i].status==0)

{

**return** i;

}

}

**return** -1;

}

**void** **RandomArrayPacket**(**int**\* arrayIndex)

{

**int** i; **int** j, temp;

**for**(i = 0; i < LENGTH\_BUFFER; i++)

{

j = LENGTH\_BUFFER - 1;

temp = arrayIndex[i];

arrayIndex[i] = arrayIndex[j];

arrayIndex[j] = temp;

}

}

Frontend.c

**#include** <stdlib.h>

**#include** <stdio.h>

**#include** <sys/iofunc.h>

**#include** <sys/dispatch.h>

**#include** <errno.h>

**#include** <time.h>

**#include** <img/img.h>

**#define** LENGTH\_BUFFER 32

**#define** LENGTH\_WORD 2048

**#define** SCALE 10

**#define** NAME\_OUT\_FILE "data.dat"

/\* We specify the header as being at least a pulse \*/

**typedef** **struct** \_pulse msg\_header\_t;

**typedef** **struct**

{

**int** id;

**char** data[LENGTH\_WORD];

**int** length;

**int** status;

} packet;

**typedef** **struct**

{

**char** file\_name[30];

**long** **int** size;

} info\_file;

info\_file info;

**int** threadBuf = 1;

**int** buffer1Full = 0;

**int** buffer2Full = 0;

**int** EXITCODE = 0;

packet buffer1[LENGTH\_BUFFER];

packet buffer2[LENGTH\_BUFFER];

**void**\* **ThreadTwo**( **void**\* arg ){

msg\_header\_t header;

name\_attach\_t\* attach;

**int** rcvid, ENDPACKET = 0;

**if** (!( attach = name\_attach(NULL,"One",NAME\_FLAG\_ATTACH\_GLOBAL) ) )

{

EXITCODE = ECONNREFUSED;

**return** EXIT\_FAILURE;

}

rcvid = MsgReceive(attach->chid, &header, **sizeof**(header), NULL);

**if** (rcvid == -1)

{

ConnectDetach(header.scoid);

EXITCODE = ECONNREFUSED;

**return** EXIT\_FAILURE;

}

// name\_open() sends a connect message, must EOK this.

**if** (header.type == \_IO\_CONNECT ) {

MsgReply( rcvid, EOK, NULL, 0 );

}

GetInformationMsgFromServer(&rcvid, attach->chid);

**while**(ENDPACKET == 0)

{

**switch**(threadBuf)

{

**case** 1:

**if**(buffer1Full == 0)

{

GetDataFromServer(attach->chid, &ENDPACKET, buffer1 );

buffer1Full = 1;

}

**break**;

**case** 2:

**if**(buffer2Full == 0)

{

GetDataFromServer(attach->chid, &ENDPACKET, buffer2 );

buffer2Full = 1;

}

**break**;

}

}

/\* Remove the name from the space \*/

name\_detach(attach, 0);

**return**(0);

}

**void**\* **ThreadFour**( **void**\* arg ){

FILE \*file;

file = fopen(NAME\_OUT\_FILE,"wb");

fclose(file);

clock\_t ca, ce;

**int** numBuf = 1;

ca = clock();

**while**(EXITCODE == 0)

{

**switch**(numBuf)

{

**case** 1:

**if**(buffer1Full == 1)

{

**int** i;

file = fopen(NAME\_OUT\_FILE,"ab");

**int** x = ReturnIndexMinIdInBuffer(buffer1);

**if**(x == -2){

EXITCODE = 1;

fclose(file);

buffer1Full = 0;

numBuf = 2;

**break**;

}

fwrite(buffer1[x].data, **sizeof**(**char**),buffer1[x].length, file );

buffer1[x].status = 0;

**for**(i=0; i<LENGTH\_BUFFER - 1;i++)

{

x = FindNextIndexId(buffer1[x].id,buffer1);

**switch**(x)

{

**case** -1:

**break**;

**case** -2:

EXITCODE = 1;

**break**;

**default**:

fwrite(buffer1[x].data, **sizeof**(**char**),buffer1[x].length, file );

buffer1[x].status = 0;

**break**;

}

}

fseek(file,0,SEEK\_END);

printf("%ld / %ld byte \n", ftell(file), info.size);

fclose(file);

buffer1Full = 0;

numBuf = 2;

}

**break**;

**case** 2:

**if**(buffer2Full == 1)

{

**int** i;

file = fopen(NAME\_OUT\_FILE,"ab");

**int** x = ReturnIndexMinIdInBuffer(buffer2);

**if**(x==-2){

EXITCODE = 1;

fclose(file);

buffer2Full = 0;

numBuf = 1;

**break**;

}

fwrite(buffer2[x].data, **sizeof**(**char**),buffer2[x].length, file );

buffer2[x].status = 0;

**for**(i=0; i<LENGTH\_BUFFER - 1;i++)

{

x = FindNextIndexId(buffer2[x].id,buffer2);

**switch**(x)

{

**case** -1:

**break**;

**case** -2:

EXITCODE = 1;

**break**;

**default**:

fwrite(buffer2[x].data, **sizeof**(**char**),buffer2[x].length, file );

buffer2[x].status = 0;

**break**;

}

}

fseek(file,0,SEEK\_END);

printf("%ld / %ld byte \n", ftell(file), info.size);

fclose(file);

buffer2Full = 0;

numBuf = 1;

}

**break**;

}

}

ce = clock();

printf("time spent on CPU: %lf sec\n", (ce - ca)/(**double**)CLOCKS\_PER\_SEC);

GetImageAndScale(info.file\_name, SCALE);

EXITCODE = -1;

**return**(0);

}

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

pthread\_t thread2\_tid, thread4\_tid;

pthread\_create(&thread2\_tid, NULL, &ThreadTwo, NULL );

pthread\_create(&thread4\_tid, NULL, &ThreadFour, NULL );

**while**(EXITCODE != -1)

{

**switch**(threadBuf)

{

**case** 1:

threadBuf = (buffer1Full == 1 && buffer2Full == 0) ? 2 : 1;

**break**;

**case** 2:

threadBuf = (buffer2Full == 1 && buffer1Full == 0) ? 1 : 2;

**break**;

}

}

RenameInputFile(info.file\_name);

sleep(1);

printf("File is received successfully!\n");

**return** EXIT\_SUCCESS;

}

**int** **GetImageAndScale**(**char**\* file\_name, **int** value){

img\_lib\_t ilib = NULL;

img\_lib\_attach(&ilib);

**int** rc;

img\_t img, imgOut;

/\* initialize an img\_t by setting its flags to 0 \*/

img.flags = 0;

img.format = IMG\_FMT\_PKLE\_ARGB1555;

img.flags |= IMG\_FORMAT;

imgOut.flags = IMG\_W;

imgOut.format = IMG\_FMT\_PKLE\_ARGB1555;

**char** filename[30];

strcpy(filename, NAME\_OUT\_FILE);

**if** ((rc = img\_load\_file(ilib, filename, NULL, &img)) != IMG\_ERR\_OK) {

fprintf(stderr, "img\_load\_file(%s) failed: %d\n", filename, rc);

**return** -2;

}

**if**(value != 0){

**if**(value < 0){

imgOut.h = img.h / (value \* -1);

imgOut.w = img.w / (value \* -1);

}

**else**{

imgOut.h = img.h \* value;

imgOut.w = img.h \* value;

}

} **else** {

imgOut.h = img.h;

imgOut.w = img.w;

}

**if** ((rc = img\_resize\_fs(&img, &imgOut)) != IMG\_ERR\_OK) {

fprintf(stderr, "img\_resize\_fs failed: %d\n", rc);

**return** -3;

}

**char** outfile[60];

strcpy(outfile, "scale\_");

strncat(outfile,file\_name,30);

**if** ((rc = img\_write\_file(ilib,outfile,NULL,&imgOut)) != IMG\_ERR\_OK) {

fprintf(stderr, "img\_write\_file(%s) failed: %d\n", outfile, rc);

**return** -4;

}

printf("Writed Image (%s)\n", outfile);

/\* for our purposes we''re done with the img lib \*/

img\_lib\_detach(ilib);

**return** 0;

}

**void** **GetDataFromServer**(**int** chid, **int**\* end\_packet, packet\* buffer\_r ){

**int** i, rcvid;

packet buffer;

**for**(i=0; i<LENGTH\_BUFFER; i++)

{

rcvid = MsgReceive(chid, &buffer, **sizeof**(buffer), NULL);

**if**(buffer.status == 230894)

{

\*end\_packet = 1;

buffer\_r[i].status = buffer.status;

MsgReply( rcvid, EOK, NULL, 0 );

**break**;

}

**if**(buffer.status == 1)

{

buffer\_r[i].id = buffer.id;

buffer\_r[i].length = buffer.length;

buffer\_r[i].status = buffer.status;

**int** j;

**for**(j=0;j<LENGTH\_WORD; j++){

buffer\_r[i].data[j] = buffer.data[j];

}

}

MsgReply( rcvid, EOK, NULL, 0 );

}

}

**void** **GetInformationMsgFromServer**(**int**\* rcvid, **int** chid){

\*rcvid = MsgReceive(chid, &info, **sizeof**(info), NULL);

MsgReply( \*rcvid, EOK, NULL, 0 );

}

**void** **RenameInputFile**(**char**\* file\_name)

{

rename(NAME\_OUT\_FILE, file\_name);

}

**int** **ReturnIndexMinIdInBuffer**(packet\* src)

{

**int** i, j = -1;

packet tmp;

**for**(i=0; i<LENGTH\_BUFFER; i++){

**if**(src[i].status == 1){

tmp = src[i];

j = i;

**break**;

}

}

**if**(j == -1){

**for**(i=0; i<LENGTH\_BUFFER; i++){

**if**(src[i].status == 230894) **return** -2;

}

**return** j;

}

**for** (i = 0; i < LENGTH\_BUFFER; i++)

{

**if** (src[i].status == 1 && tmp.id > src[i].id ){

tmp=src[i];

j=i;

}

}

**return** j;

}

**int** **FindNextIndexId**(**int** id, packet\* src)

{

**int** i = -1;

**for**(i=0; i<LENGTH\_BUFFER; i++){

**if**(src[i].status == 1 && src[i].id == id + 1) **return** i;

}

**for**(i=0; i<LENGTH\_BUFFER; i++){

**if**(src[i].status == 230894) **return** -2;

}

**return** i;

}