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**

**ОТЧЕТ**

по лабораторной работе

по курсу «ОСРВ»

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

Выполнили:

ст. группы КТмо1-3

Шепель И. О.

Куприянова А. А.

Проверил:

преподаватель МОП ЭВМ

Пирская Л. В.

Оценка

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

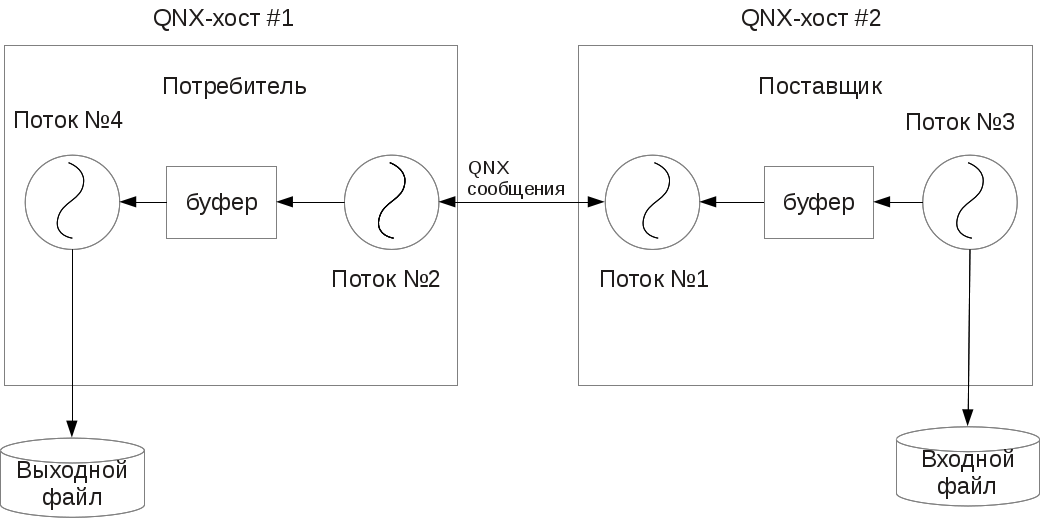
Таганрог 2017

**Цель работы.**

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

**Задание.**

Поставщик (далее – сервер) должен прочитать файл с любым изображением в формате BMP или TGA, повернуть его на заранее заданный угол и отправить его потребителю (далее – клиент). При этом сервер должен отправлять файл по частям через равные промежутки времени (100 мс). Размер части в байтах задается в параметрах приложения при запуске. Части файла отправляются не последовательно, а перемешанными, но так, чтобы их последовательность можно было восстановить на принимающей стороне. Клиент, 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 (100 мс) 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;

Размер буферов задается константой. В случае переполнения буфера поток №3 блокируется, возобновляя работу при наличии места в буфере. Алгоритм обработки изображений может быть любым.

Т.к. буфер клиента 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 клиента, так и на стороне сервера.

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 синхронизации сетевых процессов и потоков использовались следующие средства:

* Qnet — собственный протокол QNX сетевой подсистемы io-pkt. Протокол Qnet распространяет механизм обмена сообщениями на сеть микроядер QNX Neutrino. Необходимый модуль подключается следующей командой:

mount -Tio-pkt lsm-qnet.so.

* SRR-механизм обмена сообщениями в QNX. Назван так по первым буквам трех основных функций, применяемых при обмене сообщениями1. MsgSend() служит для отправки сообщения, MsgReceive() — для приема сообщения, и MsgReply() — для передачи ответа вызывающей стороне.
* GNS – менеджер глобальных служб имен. GNS является автономным менеджером ресурсов. С помощью утилиты gns приложение может выполнять объявление, поиск и использование (подключение) служб в сети при отсутствии информации о местоположении службы и ее поставщике.

GNS работает в двух различных режимах: сервер и клиент. Менеджер в режиме сервера представляет собой централизованную базу данных, в которой хранятся объявленные службы, а также обрабатываются запросы на поиск и подключение. Менеджер в режиме клиента выполняет передачу запросов на объявление, поиск и подключение между локальным приложением и сервером gns.

Запуск GNS в роли сервера выполняется следующей командой:

gns -s

Запуск GNS в роли клиента:

gns -с

Для вывода расширенной информации используются команды gns –sv и gns -сv соответственно.

**Описание структуры и алгоритмов разработанной программы.**

В целях оптимизации кода подключаемые библиотеки, объявление структур и глобальных переменных было вынесено в отдельный файл ptoperties.h, подключаемый в коде сервера и клиента.

Для работы с изображениями использовалась библиотека img.h.

Функции поворота (char\* Rotate(char\* inputFileName, int angle)) и масштабирования (void Scale()) изображения основаны на функциях img\_rotate\_ortho и img\_resize\_fs соответственно. Параметры трансформации (угол поворота и параметр масштабирования) задаются через консоль.

Функция поворота изображения Rotate принимает на вход имя входного файла (inputFileName) и угол поворота изображения (angle). Для работы с входным и выходным изображениями была использована структура img\_t, выбран формат \_FMT\_PKLE\_ARGB1555, выделяющий 16 бит на хранение одного пикселя.

Изображение считывается при помощи функции img\_load\_file. Далее угол приводится к формату img\_fixed\_t, который принимает на вход функция поворота изображения img\_rotate\_ortho, для корректной работы которой угол поворота angle должен быть кратным 90 градусам. Функция img\_write\_file записывает результат в файл rotated.bmp. В случае успешного выполнения функция Rotate возвращает имя выходного файла. В случае ошибок в выполнении функций библиотеки img будет выведено соответствующее сообщение в консоль. В этом случае Rotate вернет имя исходного файла.

Функция масштабирования Scale ничего не принимает и ничего не возвращает. Параметр масштабирования хранится в глобальной переменной scale.

В самом начале выполнения функция проверяет значение параметра scale. Если этот параметр задан меньше или равным нулю, он переназначается на 1. При значении параметра масштабирования больше 1 изображение увеличивается, при меньше 1 – уменьшается. Изменение размеров изображения выполняет функция img\_resize\_fs. Перед ее вызовом задаются размеры выходного изображения: они вычисляются путем умножения размеров входного изображения на параметр scale, после чего полученные значения приводятся к типу unsigned, в результате чего происходит округление значений в меньшую сторону. Таким образом возможно задавать дробные значения параметра scale. Выходное изображение записывается в файл result.bmp, которое затем можно получить на виртуальной машине клиента.

Описываемые в схеме проекта буферы реализованы в виде двойной буферизации. Таким образом, буфер клиента разбит на два: пока один принимает данные, второй записывает на диск, аналогично с буфером сервера.

Пакеты перемешиваются отдельной функцией MixPackets, которая получает на вход массив индексов и возвращает перемешанный массив. В данной работе массив инвертируется (первый элемент меняется с последним, второй – с предпоследним и так далее), хотя способ перемешивания может быть любым. Для восстановления порядка пакетов используется целочисленный идентификатор в структуре пакета. Также эта структура содержит поле status, содержащее информацию о пакете: 0 – свободен, 1 – занят, 2 – последний.

Для отслеживания прогресса выполнения передачи сервер передает клиенту сообщение, содержащее информацию о размере передаваемого сообщения. Информация о прогрессе выводится в консоль.

Для отслеживания ошибок выполнения библиотечных функций значения, выдаваемые ими, заносятся в переменную status, после чего функция strerror(status) преобразует код ошибки в словесное описание, которое выводится в консоль. Некоторые ошибки присваивают системной переменной errno ненулевое значение, в этом случае переменная errno также несет информацию об ошибке.

В случае отсутствия ошибок программа по завершении выводит в консоль сообщение Image is scaled successfully. После этого преобразованный файл можно получить из папки tmp виртуальной машины клиента.

**Результаты работы программы.**

Входное изображение.



Изображение, повернутое на 270 градусов и увеличенное в 2 раза.



**Выводы.**

В ходе выполнения лабораторной работы было разработано сетевое многопоточное приложение, реализующее задачу «поставщики-потребители» в распределенной среде QNX.

**Исходный код разработанного приложения.**

Сервер.

#include "D:\QNX2\properties.h"

char\* Rotate(char\* inputFileName, int angle) {

char\* outputFileName = "rotated.bmp";

img\_lib\_t ilib = NULL;

img\_lib\_attach(&ilib);

int status;

img\_t inputImage, outputImage;

inputImage.flags = 0;

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

inputImage.flags |= IMG\_FORMAT;

outputImage.flags = 0;

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

outputImage.flags |= IMG\_FORMAT;

if ((status = img\_load\_file(ilib, inputFileName, NULL, &inputImage)) != IMG\_ERR\_OK) {

printf("img\_load\_file(%s) has been failed: %s\n", inputFileName, strerror(status));

if (errno) {

printf("Error: %s\n", strerror( errno ));

}

return inputFileName;

}

img\_fixed\_t ang = (int)((angle \* 3.141592 / 180) \* 65536.0);

if((status = img\_rotate\_ortho(&inputImage, &outputImage, ang))!= IMG\_ERR\_OK){

printf("img\_rotate\_ortho has been failed: %s\n", strerror(status));

if (errno) {

printf("Error: %s\n", strerror( errno ));

}

return inputFileName;

}

if ((status = img\_write\_file(ilib,outputFileName,NULL,&outputImage)) != IMG\_ERR\_OK) {

printf("img\_write\_file(%s) has been failed: %s\n", outputFileName, strerror(status));

if (errno) {

printf("Error: %s\n", strerror( errno ));

}

return inputFileName;

}

img\_lib\_detach(ilib);

return outputFileName;

}

int SendSizeInfo(int serverID) {

int reply;

MsgSend(serverID, &size, sizeof(size), &reply, sizeof(int));

while(reply != EOK) MsgSend(serverID, &size, sizeof(size), &reply, sizeof(int));

return reply;

}

void MixPackets(int\* index)

{

int temp;

int i;

for(i = 0; i < BUFFER\_SIZE; i++) {

temp = index[i];

index[i] = index[BUFFER\_SIZE - 1 - i];

index[BUFFER\_SIZE - 1 - i] = temp;

}

}

void SendPacket(int\* index, int serverID, dataPacket\* buffer ){

int reply;

int i;

for(i=0; i<BUFFER\_SIZE;i++) {

if(buffer[index[i]].status == 0) continue;

MsgSend(serverID, &buffer[index[i]], sizeof(buffer[index[i]]), &reply, sizeof(int));

while(reply != EOK) MsgSend(serverID, &buffer[index[i]], sizeof(buffer[index[i]]), &reply, sizeof(int));

buffer[index[i]].status = 0;

}

}

void SendLastPacket(int serverID){

int reply;

dataPacket lastPacket;

lastPacket.status = 2;

MsgSend(serverID, &lastPacket, sizeof(lastPacket), &reply, sizeof(int));

}

int FindFreePlace(dataPacket\* buffer) {

int i;

for(i = 0; i<BUFFER\_SIZE; i++) {

if(buffer[i].status!=0) continue;

return i;

}

return -1;

}

int SetToBuffer(FILE\* file, int freePlace, dataPacket\* buffer, int\* id) {

int status = 1,

stop = 0;

int i;

for(i=0; i<PACKET\_SIZE; i++) {

status = fread(byte, sizeof(char), 1, file);

if(status != 1) {

buffer[freePlace].status = 1;

buffer[freePlace].id = \*id;

bufferIsFull[selectedBuffer] = true;

stop = 1;

break;

}

buffer[freePlace].size = i+1;

buffer[freePlace].data[i] = byte[0];

}

if(stop == 1) return status;

buffer[freePlace].status = 1;

buffer[freePlace].id = \*id;

\*id = \*id + 1;

return status;

}

void\* FirstThread( void\* arg ) {

int serverID = -1;

while(serverID == -1) serverID = name\_open("One", NAME\_FLAG\_ATTACH\_GLOBAL);

int bufferNumber = 0, index[BUFFER\_SIZE];

bool transferIsFinished = false;

int i;

for (i = 0; i < BUFFER\_SIZE; i++) index[i] = i;

SendSizeInfo(serverID);

while(!transferIsFinished) {

if(bufferIsFull[bufferNumber]) {

MixPackets(index);

SendPacket(index, serverID, buffers[bufferNumber]);

bufferIsFull[bufferNumber] = false;

bufferNumber = 1 - bufferNumber;

}

if(!fileIsRead || bufferIsFull[bufferNumber] || bufferIsFull[1-bufferNumber]) continue;

SendLastPacket(serverID);

transferIsFinished = true;

}

name\_close(serverID);

exitLabel = -1;

}

void\* ThirdTread( void\* arg ) {

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

if (file == NULL) {

printf("No data");

return 0;

}

int stop = 0, id = 0;

bool result = true;

while(result && (exitLabel == EOK)) {

if(bufferIsFull[selectedBuffer]) continue;

int freePlace = FindFreePlace(buffers[selectedBuffer]);

if(freePlace == -1) {

bufferIsFull[selectedBuffer] = true;

} else result = (bool)SetToBuffer(file, freePlace, buffers[selectedBuffer], &id);

}

fclose(file);

fileIsRead = true;

}

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

int angle;

printf("enter angle\n");

scanf("%d", &angle);

strncat(sentFileName, Rotate(INPUT,angle), 20);

FILE\* tmp;

tmp = fopen(sentFileName,"rb");

fseek(tmp,0,SEEK\_END);

size = ftell(tmp);

fclose(tmp);

pthread\_t firstThread, thirdTread;

pthread\_create(&thirdTread, NULL, &ThirdTread, NULL );

pthread\_create(&firstThread, NULL, &FirstThread, NULL );

while(exitLabel == EOK) selectedBuffer = (bufferIsFull[selectedBuffer] && !bufferIsFull[1-selectedBuffer]) ? 1-selectedBuffer : selectedBuffer;

sleep(1);

return EXIT\_SUCCESS;

}

Клиент.

#include "D:\QNX2\properties.h"

typedef struct \_pulse msg\_header\_t;

void Scale() {

if (scale<=0) scale=1;

img\_lib\_t ilib = NULL;

img\_lib\_attach(&ilib);

int status;

img\_t inputImage, outputImage;

inputImage.flags = 0;

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

inputImage.flags |= IMG\_FORMAT;

outputImage.flags = 0;

outputImage.flags = IMG\_W;

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

if ((status = img\_load\_file(ilib, GOTDATA, NULL, &inputImage)) != IMG\_ERR\_OK) {

printf("img\_load\_file(%s) failed: %s\n", GOTDATA, strerror(status));

if (errno) {

printf("Error: %s\n", strerror( errno ));

}

return;

}

outputImage.h = (unsigned)(inputImage.h \* scale);

outputImage.w = (unsigned)(inputImage.h \* scale);

if ((status = img\_resize\_fs(&inputImage, &outputImage)) != IMG\_ERR\_OK) {

printf("img\_resize\_fs failed: %s\n", strerror(status));

if (errno) {

printf("Error: %s\n", strerror( errno ));

}

return;

}

char\* outputFileName="result.bmp";

if ((status = img\_write\_file(ilib,outputFileName,NULL,&outputImage)) != IMG\_ERR\_OK) {

printf("img\_write\_file(%s) failed: %s\n", outputFileName, strerror(status));

if (errno) {

printf("Error: %s\n", strerror( errno ));

}

return;

}

img\_lib\_detach(ilib);

}

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

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

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

}

void GetPacket(int chid, int\* lastPacket, dataPacket\* buffer ){

int i, rcvid;

dataPacket packet;

for(i=0; i<BUFFER\_SIZE; i++) {

rcvid = MsgReceive(chid, &packet, sizeof(packet), NULL);

if(packet.status == 2) {

\*lastPacket = 1;

buffer[i].status = packet.status;

MsgReply( rcvid, EOK, NULL, 0 );

break;

}

if(packet.status == 1) {

buffer[i].id = packet.id;

buffer[i].size = packet.size;

buffer[i].status = packet.status;

int j;

for(j=0;j<PACKET\_SIZE; j++) buffer[i].data[j] = packet.data[j];

}

MsgReply( rcvid, EOK, NULL, 0 );

}

}

int FindIndex(dataPacket\* buffer) {

int i, j = -1;

dataPacket tmp;

for(i=0; i<BUFFER\_SIZE; i++) {

if (buffer[i].status != 1) continue;

tmp = buffer[i];

j = i;

break;

}

if(j == -1) {

for(i=0; i<BUFFER\_SIZE; i++)

if(buffer[i].status == 2) return -2;

return j;

}

for (i = 0; i < BUFFER\_SIZE; i++)

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

tmp=buffer[i];

j=i;

}

return j;

}

int FindNextIndex(int id, dataPacket\* buffer) {

int i = -1;

for(i=0; i<BUFFER\_SIZE; i++)

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

for(i=0; i<BUFFER\_SIZE; i++)

if(buffer[i].status == 2) return -2;

return i;

}

void\* SecondTread(void\* arg){

msg\_header\_t header;

name\_attach\_t\* attach;

int rcvid, lastPacket = 0;

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

clientExitLabel = ECONNREFUSED;

return (void\*)EXIT\_FAILURE;

}

if ((rcvid = MsgReceive(attach->chid, &header, sizeof(header), NULL)) == -1) {

ConnectDetach(header.scoid);

clientExitLabel = ECONNREFUSED;

return (void\*)EXIT\_FAILURE;

}

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

MsgReply( rcvid, EOK, NULL, 0 );

}

GetSizeInfo(&rcvid, attach->chid);

while(lastPacket == 0) {

if (bufferIsFull[clientSelectedBuffer+2]) continue;

GetPacket(attach->chid, &lastPacket, buffers[clientSelectedBuffer+2] );

bufferIsFull[clientSelectedBuffer+2] = true;

}

name\_detach(attach, 0);

return(0);

}

void\* FourthTread(void\* arg) {

FILE \*file;

int choisedBuffer = 0;

while(clientExitLabel == 0) {

if(!bufferIsFull[choisedBuffer+2]) continue;

int i;

file = fopen(GOTDATA,"ab");

int index = FindIndex(buffers[choisedBuffer+2]);

if(index == -2){

clientExitLabel = 1;

fclose(file);

bufferIsFull[choisedBuffer+2] = false;

choisedBuffer = 1-choisedBuffer;

break;

}

fwrite(buffers[choisedBuffer+2][index].data, sizeof(char),buffers[choisedBuffer+2][index].size, file );

buffers[choisedBuffer+2][index].status = 0;

for(i=0; i<BUFFER\_SIZE - 1;i++) {

index = FindNextIndex(buffers[choisedBuffer+2][index].id, buffers[choisedBuffer+2]);

if (index==-1) continue;

if (index==-2) {

clientExitLabel = 1;

continue;

}

fwrite(buffers[choisedBuffer+2][index].data, sizeof(char),buffers[choisedBuffer+2][index].size, file );

buffers[choisedBuffer+2][index].status = 0;

}

fseek(file, 0, SEEK\_END);

done = ftell(file)\*1.0;

progress = done/size\*100;

printf("progress: %.2f% \n", progress);

fclose(file);

bufferIsFull[choisedBuffer+2] = false;

choisedBuffer = 1-choisedBuffer;

}

Scale();

clientExitLabel = -1;

return(0);

}

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

printf("enter scale\n");

scanf("%f", &scale);

pthread\_t secondTread, fourthTread;

pthread\_create(&secondTread, NULL, &SecondTread, NULL );

pthread\_create(&fourthTread, NULL, &FourthTread, NULL );

while(clientExitLabel != -1) clientSelectedBuffer = (bufferIsFull[clientSelectedBuffer+2] && !bufferIsFull[3-clientSelectedBuffer]) ? 1-clientSelectedBuffer : clientSelectedBuffer;

remove(GOTDATA);

sleep(1);

printf("Image is scaled successfully\n");

return EXIT\_SUCCESS;

}

Отдельный файл properties.h:

#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>

#include <stdbool.h>

#define BUFFER\_SIZE 16

#define PACKET\_SIZE 2048

#define INPUT "pic1.bmp"

#define GOTDATA "tmp.dat"

typedef struct {

int id;

char data[PACKET\_SIZE];

int size;

int status;

} dataPacket;

dataPacket buffers[4][BUFFER\_SIZE];

bool bufferIsFull[4] = {false, false, false , false};

int selectedBuffer = 0;

int clientSelectedBuffer = 0;

int exitLabel = 0;

int clientExitLabel = 0;

bool fileIsRead = false;

char sentFileName[20];

float scale;

float progress;

float done;

long int size;

char\* byte[1];