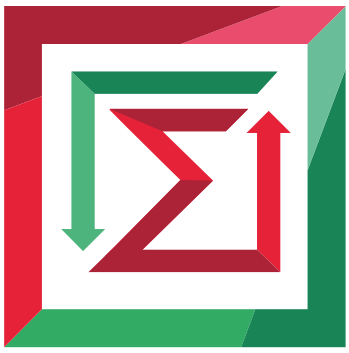
Министерство науки и высшего образования
Российской Федерации

Федеральное государственное бюджетное образовательное
учреждение высшего образования
«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

НГТУ
НЭТИ

Кафедра теоретической и прикладной информатики

Лабораторная работа № 4
по дисциплине «Проектирование Систем Реального Времени»
Синхронизация потоков



ФАКУЛЬТЕТ:	ПМИ
ГРУППА:	ПМИМ-01
СТУДЕНТЫ:	Ершов П. К. Грициенко И. Г.
БРИГАДА:	7
ПРЕПОДАВАТЕЛЬ:	Кобылянский В. Г.

Новосибирск
2021

1. Цель работы

Целью работы является изучение механизмов синхронизации.

2. Задание на лабораторную

1. Скомпилировать и выполнить примеры программ. Проанализировать результаты выполнения.
2. Получить значение системного тика в ОС QNX.
3. Написать программу, реализующую задание в соответствии с вариантом:

№ варианта	Задание
7	Клиент отсылает сообщение серверу. Сервер отвечает некоторым клиентам (функция MsgReply()), а некоторым нет – сделать это с помощью функции rand(). Предусмотреть в клиенте тайм-аут на блокировку. Сделать программу на основе 2-ой лабораторной работы.

3. Ход работы.

3.1. Анализ работы программы-примера из пункта 2.2. работа с таймером

Результаты выполнения кода-примера:

```
We got a pulse from our timer
We got a pulse from our timer
We got a pulse from our timer
We got a pulse from our timer
We got a pulse from our timer
We got a pulse from our timer
We got a pulse from our timer
We got a pulse from our timer
We got a pulse from our timer
We got a pulse from our timer
```

Рисунок 1. Результат выполнения программы с применением таймера в работе

Основной поток в начале создаёт канал, с которым сам же и соединяется. После этого создаётся два таймера: основной, срабатывающий через 1,8 секунды (0,8 секунды задаётся через значение структуры itime.it_value.tv_nsec в наносекундах) и вторичный, срабатывающий через 2,5 секунды. После этого запускается цикл на 10 повторов, в котором основной поток сам себе принимает импульсы: первичный импульс через 1,8 секунды и повторные ещё через 2,5 секунды.

3.2. Анализ работы программы-примера из пункта 2.3. работа с тайм-аутами ядра

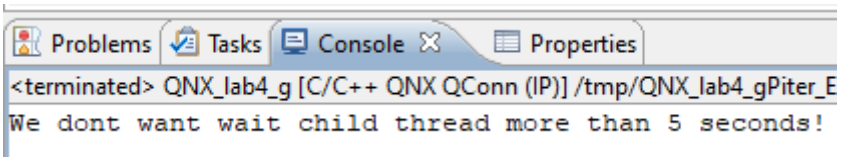


Рисунок 2. Результат выполнения программы с использованием тайм-аута ядра

Основной поток устанавливает время тайм-аута ядра на 5 секунд ровно. Создаётся вторичный поток, в котором установлена задержка на 100 секунд. В основном потоке создаётся тайм-аут, после чего поток блокируется до тех пор, пока вторичный поток не завершит исполнение. Так как время работы вторичного потока превышает время тайм-аута, блокировка снимается спустя 5 секунд и основной поток завершается.

3.3. Получение значения системного тика в ОС QNX.

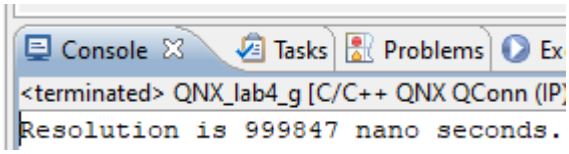


Рисунок 3. Значение системного тика

3.4. Программа успешно реализована

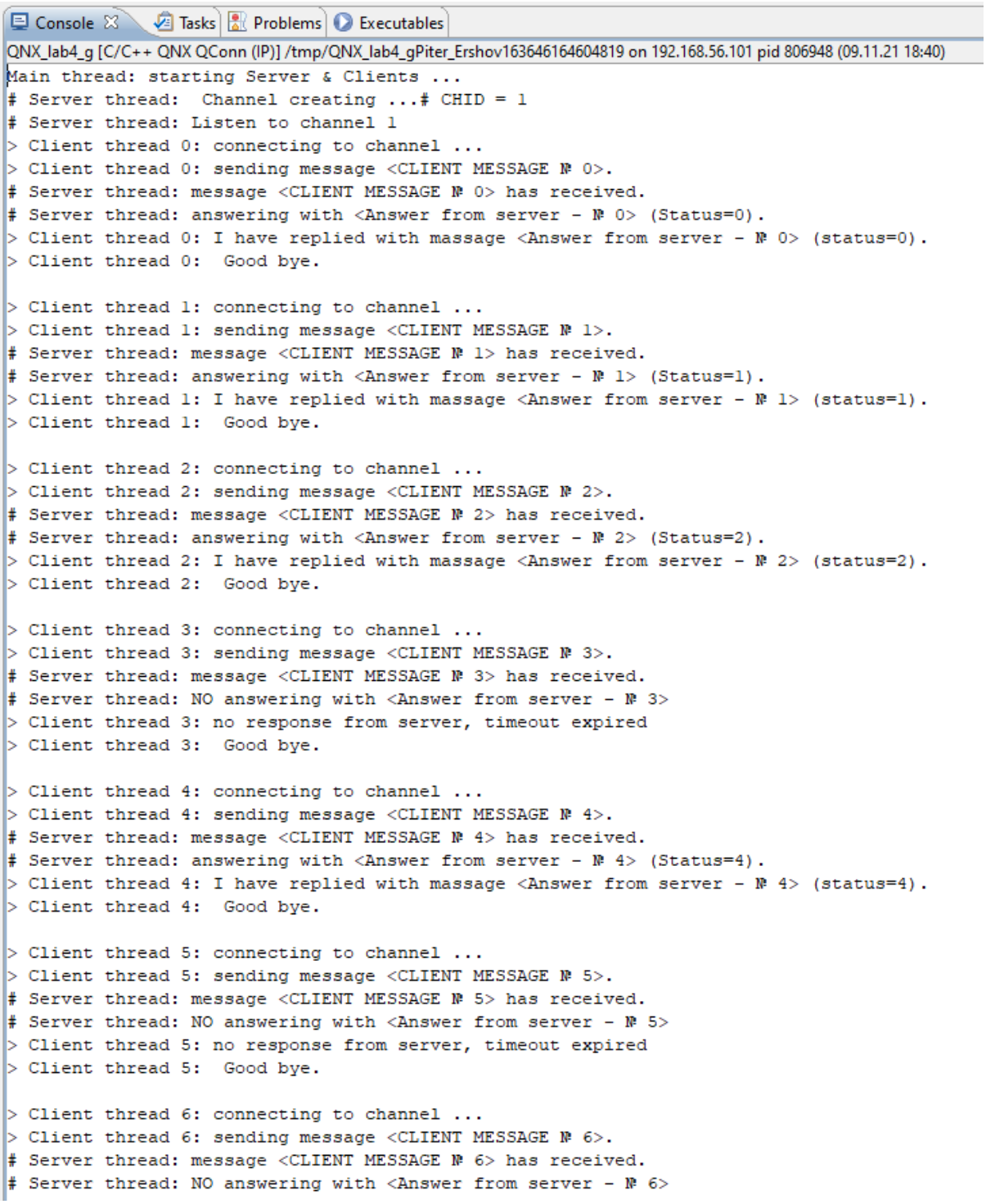


Рисунок 4. Выполнения разработанной программы

4. Код программы

Код программы-примера использования таймера

```
#include <stdio.h>
#include <time.h>
#include <sys/netmgr.h>
#include <sys/neutrino.h>

// Задаем код импульса
// он должен быть между _PULSE_CODE_MINAVAIL и _PULSE_CODE_MAXAVAIL
#define MY_PULSE_CODE _PULSE_CODE_MINAVAIL

int main()
{
        int i;
        struct sigevent event;
        struct itimerspec itime;
        timer_t timer_id;
        int chid, coid;
        struct _pulse impulse;

        // создаем канал
        chid=ChannelCreate(0);
        // и сами к нему соединяемся
        coid=ConnectAttach(ND_LOCAL_NODE, 0, chid, 0, 0);

        // устанавливаем структуру event на уведомление
        //   импульсом с кодом MY_PULSE_CODE в канал coid.
        // импульс передается с приоритетом текущего потока,
        // который получен с помощью функции getprio(0).
        // последний аргумент равен нулю - данные импульса
        SIGEV_PULSE_INIT(&event, coid, getprio(0), MY_PULSE_CODE, 0);
        // создаем таймер
        timer_create(CLOCK_REALTIME, &event, &timer_id);

// задаем время срабатывания таймера
        // таймер сработает через 1.8 секунды
        // (1 сек + 800000000 наносек = 1.8 секунды)
        itime.it_value.tv_sec=1;
        itime.it_value.tv_nsec=800000000;

        // таймер повторно сработает через 2.5 секунды
        // (2 сек + 500000000 наносек = 2.5 секунды)
        itime.it_interval.tv_sec=2;
        itime.it_interval.tv_nsec=500000000;

        // создаем относительный таймер (второй параметр равен нулю)
        timer_settime(timer_id, 0, &itime, NULL);

        // теперь мы получим импульс через 1.8 секунды и будем
        // получать повторные через 2.5 секунды

        for(i=10; i>0; i--)
        {
                MsgReceivePulse(chid, &impulse, sizeof(impulse), NULL);
                printf("We got a pulse from our timer\n");
        }
        // получили 10 импульсов и выходим
        return 0;
}
```

Код программы-примера использования тайм-аутов ядра

```
#include <pthread.h>
#include <errno.h>
#include <time.h>
#include <sys/neutrino.h>

void *the_thread(void *notused)
{       // дочерний поток
        // поток может выполнять работу,
```

```
        //   результат которой, если она будет
        //   выполнена не в срок, будет не нужен
        sleep(100); //дочерний поток простаивает 100 секунд
        return NULL;
}

int main(void)
{
        int return_val;
        struct sigevent event;
        struct timespec time;
        pthread_t thread_id;

        // устанавливаем структуру event на разблокирование
        // при срабатывании тайм-аута
        SIGEV_UNBLOCK_INIT(&event);

// устанавливаем время тайм-аута на 5 секунд
        time.tv_sec=5;
        time.tv_nsec=0;

        // создаем поток, который будет выполнять функцию the_thread()
        pthread_create(&thread_id, NULL, the_thread, NULL);

        // устанавливаем тайм-аут на блокировку типа STATE_JOIN
        timer_timeout(CLOCK_REALTIME, _NTO_TIMEOUT_JOIN, &event, &time, NULL);

        // вызываем функцию pthread_join() из-за которой
        //   главный поток блокируется до тех пор, пока не завершится поток thread_id
        return_val=pthread_join(thread_id, NULL);
        if(return_val==ETIMEDOUT)
        {
                // сработал тайм-аут - мы не дождались завершения дочернего потока
                puts("We dont want wait child thread more than 5 seconds!");
        }
        if(return_val==EOK)
        {
                puts("Child thread success terminated.");
        }
        return 0;
}
```

Код разработанной программы

```
#include <pthread.h>
#include <errno.h>
#include <time.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <sys/neutrino.h>
#include <time.h>

#define BUFFERSIZE 50

int chid;           // идентификатор канала

void *server()    // поток-сервер
{
        int rcvid;
        int i=0, stime;
        long ltime;
         int8 code;
        int value;
        char receive_buf[BUFFERSIZE], reply_buf[BUFFERSIZE];
        char * buff[BUFFERSIZE];

        stime = (unsigned int) ltime/2;
        srand(stime);

        printf("# Server thread:  Channel creating ...");
        // создание канала с опциями по умолчанию и запись в chid номера канала
        chid=ChannelCreate(0);
```

```
        if (chid<0)
        {
           perror("Server error");
           exit(EXIT_FAILURE);
        }
        printf("# CHID = %d\n", chid);
        printf("# Server thread: Listen to channel %d\n", chid);
        while (1)       // сервер работает в цикле
        {
          // принимаем сообщение из канала с номером chid в буфер receive_buf
          // в rcvid записывается идентификатор полученного сообщения
                rcvid=MsgReceive(chid, &receive_buf, sizeof(receive_buf), NULL);
                printf("# Server thread: message <%s> has received.\n", receive_buf);
                int flag = (rand() % 2 + 0);

                strcpy(reply_buf, "Answer from server - № ");
                sprintf(buff, "%d", i);
                strcat(reply_buf, buff);
                if(flag == 0)
                {
                        printf("# Server thread: answering with <%s> (Status=%d).\n", reply_buf, i);
                        // отправляем ответ (буфер reply_buf) по номеру полученного сообщения (rcvid)
                        // второй параметр (в данном случае переменная i)
                        // статус ответа, обрабатывается клиентом.
                        MsgReply(rcvid, i, &reply_buf, sizeof(reply_buf));
                }
                else
                        printf("# Server thread: NO answering with <%s> \n", reply_buf);

                i++;
        }
}


void *client(void *parametr)    // поток-клиент
{
        int coid, status;
        _int8 code;
        int value;
        struct sigevent event;
        struct timespec time;
        pid_t PID;
        pthread_t client;
        char send_buf[BUFFERSIZE], reply_buf[BUFFERSIZE] = "1";
        PID=getpid();

        SIGEV_UNBLOCK_INIT(&event);

        time.tv_sec=4;
        time.tv_nsec=0;

        client=pthread_self(); // получаем идентификатор потока-клиента
        printf("> Client thread %s: connecting to channel ... \n", parametr);
        // создаем соединение с каналом на текущем узле (0)
        // канал принадлежит процессу с идентификатором PID
        // номер канала - chid
        // наименьшее значение для COID - 0
        // флаги соединения не заданы - 0
        coid=ConnectAttach(0, PID, chid, 0, 0);
        // в coid записан идентификатор соединения или ошибочное значение меньше нуля
        if (coid<0)
        {
           perror("Client error");
           exit(EXIT_FAILURE);
        }

        strcpy(send_buf, "CLIENT MESSAGE № ");
        strcat(send_buf, parametr);
        printf("> Client thread %s: sending message <%s>.\n", parametr, send_buf);
        // отправляем сообщение из буфера send_buf в соединение coid
        // ответ принимаем в буфер reply_buf и статус записывается в переменную status

        timer_timeout(CLOCK_REALTIME, _NTO_TIMEOUT_REPLY, &event, &time, NULL);

        status = MsgSend(coid, &send_buf, sizeof(send_buf), &reply_buf, sizeof(reply_buf));
```

```
        int flag = strcmp(reply_buf, "1");
        if(flag != 1)
                printf("> Client thread %s: no response from server, timeout expired\n", parametr);
        else
                printf("> Client thread %s: I have replied with massage <%s> (status=%d).\n", parametr,
reply_buf, status);

        // разрываем соединение coid
        ConnectDetach(coid);
        printf("> Client thread %s:  Good bye.\n", parametr);
        pthread_exit(NULL);
}


int main(void)
{

        printf("Main thread: starting Server & Clients ...\n");

        // создаем потоки сервера и двух клиентов
        pthread_create(NULL, NULL, server, NULL);
        sleep(1);

        int i=0;
        while(i < 10)
        {
                pthread_t client_tid1;
                char * buff[BUFFERSIZE];
                sprintf(buff, "%d", i);
                pthread_create(&client_tid1, NULL, client, (void*)buff);
                pthread_join(client_tid1, NULL);
                printf("\n");
                sleep(6);
                i++;
        }
        printf("Main thread: the end.\n");
        return 0;
}
```