Министерство науки и высшего образования

Российской Федерации

Федеральное государственное бюджетное образовательное   
учреждение высшего образования

«Новосибирский государственный технический университет»



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.

1. **Задание на лабораторную**
2. Скомпилировать и выполнить примеры программ. Проанализировать результаты выполнения.
3. Получить значение системного тика в OC QNX.
4. 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. |

1. **Ход работы.**
   1. Анализ работы программы-примера из пункта 2.2. работа с таймером

Результаты выполнения кода-примера:

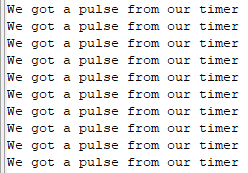


Рисунок 1. Результат выполнения программы с применением таймера в работе

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.

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

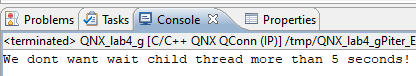


Рисунок 2. Результат выполнения программы с использованием тайм-аута ядра

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.

* 1. Получение значения системного тика в ОС QNX.

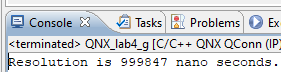


Рисунок 3. Значение системного тика

* 1. Программа успешно реализована

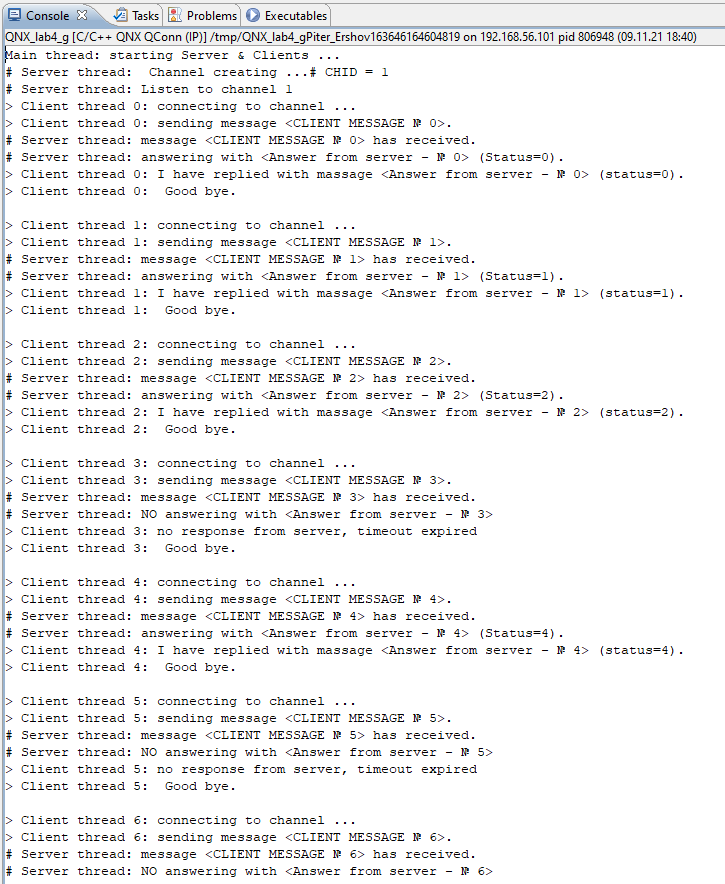


Рисунок 4. Выполнения разработанной программы

1. **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 соединияемся

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;

}