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 № 2 по дисциплине «Проектирование Систем Реального Времени»

### Механизмы сообщений

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 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 механизма сообщений QNX.

1. **Задание на лабораторную**
2. Скомпилировать и запустить программу из раздела 2.6, проанализировать результаты.
3. Написать и отладить программу, реализующую задание в соответствии с таблицей 1:

|  |  |
| --- | --- |
| № варианта | Задание |
| 7 | Клиент пересылает серверу имя директории. Сервер возвращает список файлов и поддиректорий данной директории (рекурсивно). |

1. **Ход работы.**
   1. Анализ работы программы-примера

Результаты выполнения кода-примера:

Main thread: starting Server & Clients ...

# Server thread: Channel creating ...# CHID = 1

# Server thread: Listen to channel 1

> Client thread 3: connecting to channel ... COID = 3

> Client thread 20: sending pulse - code=12345, value=0.

# Server thread: received pulse - code=20, value=12345.

> Client thread 3: Good bye.

Main thread: waiting for child threads exiting ...

> Client thread 4: connecting to channel ... COID = 3

> Client thread 4: sending message <It's very simple example>.

# Server thread: message <It's very simple example> has received.

# Server thread: answering with <Answer from server> (Status=0).

> Client thread 4: I have replied with massage <Answer from server> (status=0).

> Client thread 4: Good bye.

Main thread: the end.

После запуска программы в основном потоке (Main функция) запускаются три потока:

1. Поток с сервером
2. Два потока клиентов.

В потоке-сервера создаётся канал с номером 1 для передачи данных.

Первый клиент с номером потока 3 устанавливает соединение с каналом (номер соединение coid=3), затем посылает по каналу импульс (потоки с нечетным номером посылают импульсы). Сервер подтверждает получение импульса и 3 поток завершает работу.

Поток с номером 4 выполняет те же действия, что и поток с номером 3, при этом получая тот же номер coid=3, поскольку это соединение уже было освобождено потоком с номером 3. Затем 4 поток посылает сообщение серверу, сервер подтверждает получение и отправляет ответ клиенту. Клиент получает ответ и завершает работу. Основной поток, дождавшись завершения работы клиентов, завершает работу программы.

* 1. Программа успешно разработана.

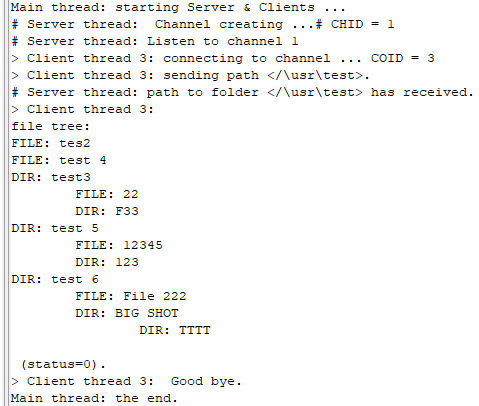
****

Рисунок 1. Результаты поиска для пути “/\\usr\\test”

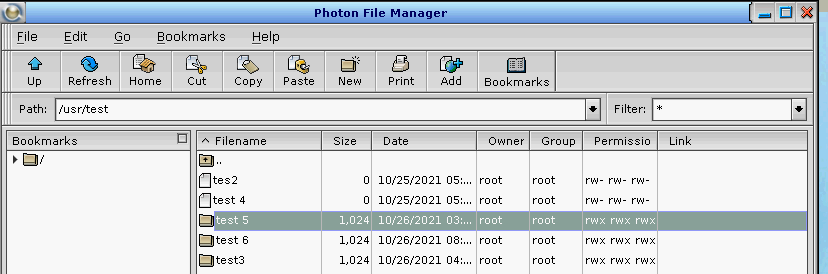


Рисунок 2. Скриншот файловой системы QNX, показывающий целевую директорию

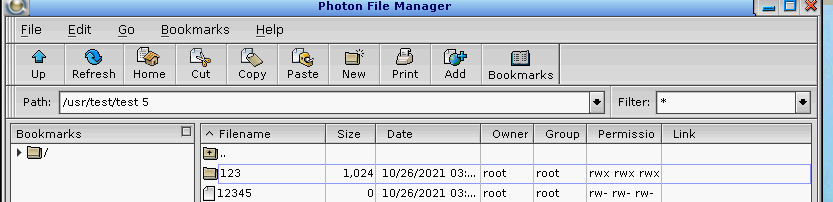


Рисунок 3. Скриншот файловой системы QNX, показывающий одну из директорий, лежащую в целевой директории

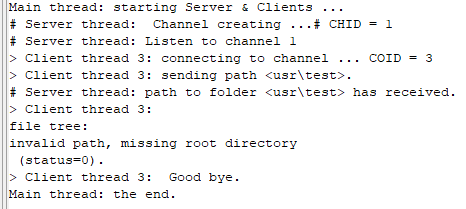


Рисунок 4. Результаты поиска для некорректного пути “usr\\test”

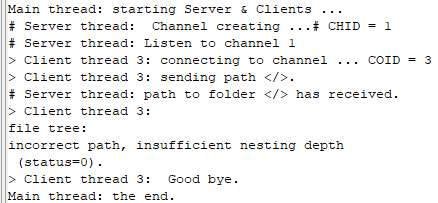


Рисунок 5. Результаты поиска для корневого каталога “/”

1. **Код программы**

Код программы-примера

**#include** <pthread.h>

**#include** <stdlib.h>

**#include** <stdio.h>

**#include** <string.h>

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

**#define** BUFFERSIZE 50

**int** chid; // идентификатор канала

**void** \***server**() // поток-сервер

{

**int** rcvid;

**int** i=0;

\_int8 code;

**int** value;

**char** receive\_buf[BUFFERSIZE], reply\_buf[BUFFERSIZE];

**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);

**if** (rcvid>0) // получили обычное сообщение

{

**printf**("# Server thread: message <%s> has received.\n", receive\_buf);

**strcpy**(reply\_buf, "Answer from server");

**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));

i++;

}

**if** (rcvid==0) // получили импульс

{

code=receive\_buf[4]; // 4-ый байт - это код импульса (по структуре \_pulse)

// байты с 8 по 11 - это данные, соберем их в переменную value

value=receive\_buf[11];

value<<=8; value+=receive\_buf[10];

value<<=8; value+=receive\_buf[9];

value<<=8; value+=receive\_buf[8];

**printf**("# Server thread: received pulse - code=%d, value=%d.\n", code, value);

}

}

}

**void** \***client**(**void** \*parametr) // поток-клиент

{

**int** coid, status;

\_int8 code;

**int** value;

pid\_t PID;

pthread\_t client;

**char** send\_buf[BUFFERSIZE], reply\_buf[BUFFERSIZE];

PID=getpid();

client=**pthread\_self**(); // получаем идентификатор потока-клиента

**printf**("> Client thread %d: connecting to channel ... ", client);

// 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);

}

**printf**("COID = %d\n", coid);

**if** (client%2==0) // четные потоки будут отправлять сообщения

{

**strcpy**(send\_buf, "It's very simple example");

**printf**("> Client thread %d: sending message <%s>.\n", client, send\_buf);

// отправляем сообщение из буфера send\_buf в соединение coid

// ответ принимаем в буфер reply\_buf и статус записывается в переменную status

status=**MsgSend**(coid, &send\_buf, **sizeof**(send\_buf), &reply\_buf, **sizeof**(reply\_buf));

**printf**("> Client thread %d: I have replied with massage <%s> (status=%d).\n", client, re-ply\_buf, status);

}

**else**

{ // нечетные потоки будут отправлять импульсы

code=20;

value=12345;

**printf**("> Client thread %d: sending pulse - code=%d, value=%d.\n", code, value);

// посылаем импульс в соединение coid

// приоритет импульса 20

// код - code, данные - value

**MsgSendPulse**(coid, 20, code, value);

}

// разрываем соединение coid

**ConnectDetach**(coid);

**printf**("> Client thread %d: Good bye.\n", client);

**pthread\_exit**(NULL);

}

**int** **main**()

{

pthread\_t client\_tid1, client\_tid2;

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

// создаем потоки сервера и двух клиентов

**pthread\_create**(NULL, NULL, server, NULL);

sleep(1);

**pthread\_create**(&client\_tid1, NULL, client, NULL);

**pthread\_create**(&client\_tid2, NULL, client, NULL);

**printf**("Main thread: waiting for child threads exiting ...\n");

// ждем их завершения

**pthread\_join**(client\_tid1, NULL); **pthread\_join**(client\_tid2, NULL);

**printf**("Main thread: the end.\n");

**return** EXIT\_SUCCESS;

}

Код разработанной программы

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

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

**#include** <stdlib.h>

**#include** <dirent.h>

**#include** <stdio.h>

**#include** <unistd.h>

**#include** <errno.h>

**#include** <string.h>

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

**#define** FILENAME\_MAX 255

**#define** MAXLENHGT\_STR 1000

**int** chid; // идентификатор канала

**int** **GetDeep**(**char** path[]) // получить глубину вложенности

{

**char** sep[2]="\\";

**char** buff[FILENAME\_MAX] = "";

**char** \*istr;

**int** deep = 0;

**strcpy**(buff, path);

istr = **strtok** (buff, sep);

**while**(istr != NULL)

{

istr = **strtok** (NULL, sep);

deep++;

}

**return** deep;

}

**int** **GetDirNum**() // получить количество папок в текущей папке

{

DIR \*dip;

**struct** dirent \*dit;

**struct** stat statbuf;

**int** dirNum = 0;

**if**((dip = **opendir**(".")) == NULL)

**return** errno;

**while**((dit = **readdir**(dip)) != NULL)

{

**if**(**strcmp**(dit->d\_name, ".") !=0 && **strcmp**(dit->d\_name, "..") != 0)

{

**if**(**stat**(dit->d\_name, &statbuf) == -1)

{

**perror**("stat");

**return** errno;

}

**if**(S\_ISDIR(statbuf.st\_mode))

dirNum++;

}

}

**return** dirNum;

}

**int** **CheckRoot**(**char** \*\*path)

{

**if**(**strcmp**(path[0], "/") != 0)

**return** 0;

**else**

**return** 1;

}

**int** **CheckDeep**(**int** deep) // проверить глубину вложенности

{

**if**(deep < 2)

**return** 0;

**else**

**return** 1;

}

**char** \*\* **GetPath**(**int** deep, **char** inputpath[]) // получить путь до целевой папки

{

**char** sep[2]="\\";

**char** buff[FILENAME\_MAX] = "";

**char** \*istr;

**char** \*\* path = (**char**\*\*)**malloc**(deep \* **sizeof**(**char**\*));

**int** i = 0;

**strcpy**(buff, inputpath);

istr = **strtok** (buff, sep);

path[i] = (**char**\*)**malloc**(100 \* **sizeof**(**char**));

**strcpy**(path[i], istr);

**while**(istr != NULL && i < deep)

{

istr = **strtok** (NULL, sep);

i++;

**if**(i < deep)

{

path[i] = (**char**\*)**malloc**(100 \* **sizeof**(**char**));

**strcpy**(path[i], istr);

}

}

**return** path;

}

**char** \*\***GetDirs**() // получить массив папок

{

DIR \*dip;

**struct** dirent \*dit;

**struct** stat statbuf;

**int** i = 0;

**int** dirsNum = GetDirNum();

**char** \*\*dirs = (**char**\*\*)**malloc**(dirsNum \* **sizeof**(**char**\*));

**if**((dip = **opendir**(".")) == NULL)

**return** errno;

**while**((dit = **readdir**(dip)) != NULL && i < dirsNum)

{

**if**(**strcmp**(dit->d\_name, ".") !=0 && **strcmp**(dit->d\_name, "..") != 0)

{

**if**(**stat**(dit->d\_name, &statbuf) == -1)

{

**perror**("stat");

**return** errno;

}

**if**(S\_ISDIR(statbuf.st\_mode))

{

dirs[i] = (**char**\*)**malloc**(FILENAME\_MAX \* **sizeof**(**char**));

**strcpy**(dirs[i], dit->d\_name);

i++;

}

}

}

**return** dirs;

}

**void** **GoToPath**(**int** deep, **char** \*\*path) // перейти по пути

{

**int** i = 0;

**while**(i < deep)

{

**chdir**(path[i]);

i++;

}

}

**void** **ReturnRoot**() // перейти в корневой каталог

{

**char** currentPath[FILENAME\_MAX];

**if**((**getcwd**(currentPath, FILENAME\_MAX)) == NULL)

**return** errno;

**while**(**strcmp**(currentPath, "/") != 0)

{

**chdir**("..");

**getcwd**(currentPath, FILENAME\_MAX);

}

}

**int** **GetIndex**(**int** dirnum, **char** \*\*dirs, **char** checkedDirs[100][FILENAME\_MAX]) // получить индекс папки, которую еще не проверяли

{

**int** i = 0;

**int** flag = 0;

**while**(i < dirnum)

{

**int** j = 0;

**while**(j < 100)

{

**if**(**strcmp**(dirs[i], checkedDirs[j]) == 0)

{

flag = 1;

**break**;

}

flag = 0;

j++;

}

**if**(flag == 0)

**return** i;

i++;

}

**return** -1;

}

**char** \***GetCurrentDir**(**char** \*currentPath) // получить имя текущего каталога

{

**char** \*out = (**char**\*)**malloc**(FILENAME\_MAX \* **sizeof**(**char**));

**char** sep[2]="/";

**char** buff[FILENAME\_MAX] = "";

**char** \*istr;

**strcpy**(buff, currentPath);

istr = **strtok** (buff, sep);

**while**(istr != NULL)

{

**strcpy**(out, istr);

istr = **strtok** (NULL, sep);

}

**return** out;

}

**int** **CheckDir**(**char** \*dir, **char** checkedData[100][FILENAME\_MAX]) // проверить уникальность папки

{

**int** i = 0;

**while**(i < 100)

{

**if**(**strcmp**(dir, checkedData[i]) == 0)

{

**return** 1;

}

i++;

}

**return** 0;

}

**char** \* **GetTab**(**int** deep, **char** \*tree) // получить отступ, в зависимости от глубины вложенности файла

{

**char** tab[10] = " ";

**int** i = 0;

**while**(i < deep)

{

**strcat**(tree, tab);

i++;

}

**return** tree;

}

**char** \***GetFiles**(**char** \* tree, **int** deep) // получить файлы, не являющиется папками

{

DIR \*dip;

**struct** dirent \*dit;

**struct** stat statbuf;

**int** i = 0;

**int** dirsNum = GetDirNum();

//char \*dirs = (char\*)malloc(FILENAME\_MAX \* sizeof(char));

**if**((dip = **opendir**(".")) == NULL)

**return** errno;

**while**((dit = **readdir**(dip)) != NULL && i < dirsNum)

{

**if**(**strcmp**(dit->d\_name, ".") !=0 && **strcmp**(dit->d\_name, "..") != 0)

{

**if**(**stat**(dit->d\_name, &statbuf) == -1)

{

**perror**("stat");

**return** errno;

}

**if**(S\_ISDIR(statbuf.st\_mode) == 0)

{

//printf("FILE: %s", dit->d\_name);

tree = GetTab(deep, tree);

**strcat**(tree, "FILE: ");

**strcat**(tree, dit->d\_name);

**strcat**(tree, "\n");

i++;

}

}

}

**return** tree;

}

**void** **ShowAll**() // вывести всё содержимое папки

{

DIR \*dip;

**struct** dirent \*dit;

**if**((dip = **opendir**(".")) == NULL)

**return** errno;

**while**((dit = **readdir**(dip)) != NULL)

{

**if**(**strcmp**(dit->d\_name, ".") !=0 && **strcmp**(dit->d\_name, "..") != 0)

{

**printf**("%s\n", dit->d\_name);

}

}

}

**char** \***GetTree**(**char** \*inputPath) // получить дерево файлов

{

**char** checkedDirs[100][FILENAME\_MAX];

**int** deep = GetDeep(inputPath);

**int** dirsNum = 0;

**int** checkedIndex = 0;

**char** \*\*path;

**int** globalDeep = 0;

path = GetPath(deep, inputPath);

**char** \*thee = (**char**\*)**malloc**(MAXLENHGT\_STR \* **sizeof**(**char**));

**if**(CheckDeep(deep))

{

**if**(CheckRoot(path))

{

ReturnRoot();

GoToPath(deep, path);

**int** index = 0;

**int** flag = 0;

**while**(1)

{

**char** currentPath[FILENAME\_MAX];

**getcwd**(currentPath, FILENAME\_MAX);

dirsNum = GetDirNum();

**char** \*\*dirs = GetDirs();

**if**(dirsNum != 0)

index = GetIndex(dirsNum, dirs, checkedDirs);

**if**(index == -1 && **strcmp**(GetCurrentDir(currentPath), path[deep - 1]) != 1)

{

**break**;

}

**else**

{

**if**(index != -1)

{

**if**(dirsNum != 0)

{

**if**(CheckDir(dirs[index], checkedDirs) == 0)

{

**if**(**strcmp**(GetCurrentDir(currentPath), path[deep - 1]) == 0 && flag != 1)

{

flag = 1;

thee = GetFiles(thee, globalDeep);

}

**if**(**strcmp**(GetCurrentDir(currentPath), path[deep - 1]) != 0)

{

thee = GetFiles(thee, globalDeep);

}

thee = GetTab(globalDeep, thee);

**strcat**(thee, "DIR: ");

**strcat**(thee, dirs[index]);

**strcat**(thee, "\n");

}

**strcpy**(checkedDirs[checkedIndex], dirs[index]);

**chdir**(dirs[index]);

globalDeep++;

checkedIndex++;

}

**else**

{

globalDeep--;

**chdir**("..");

}

}

**else**

{

globalDeep--;

**chdir**("..");

}

}

}

}

**else**

**strcat**(thee, "invalid path, missing root directory");

}

**else**

**strcat**(thee, "incorrect path, insufficient nesting depth");

**return** thee;

}

**void** \***server**() // поток-сервер

{

**int** rcvid;

**int** i=0;

\_int8 code;

**int** value;

**char** receive\_buf[FILENAME\_MAX];

**char** \*reply\_buf;

**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);

**if** (rcvid > 0) // получили обычное сообщение

{

**printf**("# Server thread: path to folder <%s> has received.\n", receive\_buf);

//strcpy(reply\_buf, "Answer from server");

reply\_buf = GetTree(receive\_buf);

//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));

i++;

}

}

}

**void** \***client**(**void** \*parametr) // поток-клиент

{

**int** coid, status;

\_int8 code;

**int** value;

pid\_t PID;

pthread\_t client;

**char** send\_buf[FILENAME\_MAX];

**char** \*reply\_buf;

PID=**getpid**();

client=**pthread\_self**(); // получаем идентификатор потока-клиента

**printf**("> Client thread %d: connecting to channel ... ", client);

// 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);

}

**printf**("COID = %d\n", coid);

**strcpy**(send\_buf, parametr);

**printf**("> Client thread %d: sending message <%s>.\n", client, send\_buf);

// отправляем сообщение из буфера send\_buf в соединение coid

// ответ принимаем в буфер reply\_buf и статус записывается в переменную status

status=**MsgSend**(coid, &send\_buf, **sizeof**(send\_buf), &reply\_buf, **sizeof**(reply\_buf));

**printf**("> Client thread %d: \nfile tree: \n%s \n (status=%d).\n", client, reply\_buf, status);

**ConnectDetach**(coid);

**printf**("> Client thread %d: Good bye.\n", client);

**pthread\_exit**(NULL);

}

**int** **main**()

{

**char** \*a = "/\\usr\\test";

pthread\_t client\_tid1, client\_tid2;

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

// создаем потоки сервера и двух клиентов

**pthread\_create**(NULL, NULL, server, NULL);

**sleep**(1);

**pthread\_create**(&client\_tid1, NULL, client, (**void**\*)a); // передаёт путь до целевой папки, получаем рекурсивно всё её содержимое

// ждем их завершения

**pthread\_join**(client\_tid1, NULL);

**printf**("Main thread: the end.\n");

**return** EXIT\_SUCCESS;

}