Лабораторная работа #3. Разработка многопоточного администратора ресурсов QNX Neutrino

# 1. Задача

Разработать многопоточную версию администратора ресурсов ГПСЧ BBS (лабораторная работа #2) *cryptobbs v2.0*, поддерживающую корректную параллельную работу с несколькими клиентами.

Требования:

* **в качестве инструмента организации многопоточной схемы должен использоваться** [**пул потоков QNX Neutrino**](http://www.qnx.com/developers/docs/6.5.0/index.jsp?topic=%2Fcom.qnx.doc.neutrino_resmgr%2Fmultithread.html)**; +**
* протокол взаимодействия “cryptobbs-client” v 2.0 должен поддерживать работу с сессиями, каждый клиент должен иметь возможности генерации “своей” ПСП с индивидуальными параметрами;
* в качестве идентификатора клиента используется идентификатор отправителя входящего сообщения;
* **предусмотреть хранение внутри АР контекстов взаимодействия с каждым клиентом, выделение такого контекста (io\_open) и уничтожение (io\_close). \***

# 2. Пример скелета многопоточного АР

Листинг 1

#include <errno.h>

#include <stdio.h>

#include <stddef.h>

#include <stdlib.h>

#include <unistd.h>

/\*

\* определяем константу THREAD\_POOL\_PARAM\_T чтобы отключить предупреждения

\* компилятора при использовании функций семейства dispatch\_\*()

\*/

#define THREAD\_POOL\_PARAM\_T dispatch\_context\_t

#include <sys/iofunc.h>

#include <sys/dispatch.h>

static resmgr\_connect\_funcs\_t connect\_funcs;

static resmgr\_io\_funcs\_t io\_funcs;

static iofunc\_attr\_t attr;

main(int argc, char \*\*argv)

{

thread\_pool\_attr\_t pool\_attr;

resmgr\_attr\_t resmgr\_attr;

dispatch\_t \*dpp;

thread\_pool\_t \*tpp;

dispatch\_context\_t \*ctp;

int id;

/\* инициализация интерфейса диспетчеризации \*/

if((dpp = dispatch\_create()) == NULL) {

fprintf(stderr,

"%s: Unable to allocate dispatch handle.\n",

argv[0]);

return EXIT\_FAILURE;

}

/\* инициализация атрибутов АР - параметры IOV \*/

memset(&resmgr\_attr, 0, sizeof resmgr\_attr);

resmgr\_attr.nparts\_max = 1;

resmgr\_attr.msg\_max\_size = 2048;

/\* инициализация структуры функций-обработчиков сообщений \*/

iofunc\_func\_init(\_RESMGR\_CONNECT\_NFUNCS, &connect\_funcs,

\_RESMGR\_IO\_NFUNCS, &io\_funcs);

/\* инициализация атрибутов устройства\*/

iofunc\_attr\_init(&attr, S\_IFNAM | 0666, 0, 0);

/\* прикрепление к точке монтирования в пространстве имён путей \*/

id = resmgr\_attach(

dpp, /\* хэндл интерфейса диспетчеризации \*/

&resmgr\_attr, /\* атрибуты АР \*/

"/dev/sample", /\* точка монтирования \*/

\_FTYPE\_ANY, /\* open type \*/

0, /\* флаги \*/

&connect\_funcs, /\* функции установления соединения \*/

&io\_funcs, /\* функции ввода-вывода \*/

&attr); /\* хэндл атрибутов устройства \*/

if(id == -1) {

fprintf(stderr, "%s: Unable to attach name.\n", argv[0]);

return EXIT\_FAILURE;

}

/\* инициализация атрибутов пула потоков \*/

memset(&pool\_attr, 0, sizeof pool\_attr);

pool\_attr.handle = dpp;

pool\_attr.context\_alloc = dispatch\_context\_alloc;

pool\_attr.block\_func = dispatch\_block;

pool\_attr.unblock\_func = dispatch\_unblock;

pool\_attr.handler\_func = dispatch\_handler;

pool\_attr.context\_free = dispatch\_context\_free;

pool\_attr.lo\_water = 2;

pool\_attr.hi\_water = 4;

pool\_attr.increment = 1;

pool\_attr.maximum = 50;

/\* инициализация пула потоков \*/

if((tpp = thread\_pool\_create(&pool\_attr,

POOL\_FLAG\_EXIT\_SELF)) == NULL) {

fprintf(stderr, "%s: Unable to initialize thread pool.\n",

argv[0]);

return EXIT\_FAILURE;

}

/\* запустить потоки, блокирующая функция \*/

thread\_pool\_start(tpp);

/\* здесь вы не окажетесь, грустно \*/

}