Указания по созданию многопоточного сервера на RPC с взаимоисключением на pthread\_mutex, моделирующего алгоритм Лампорта «Булочная»

1. Установить rpc и rpcgen по файлу с соответствующими указаниями.
2. Создать файл bakery.x:

/\* Запрос к серверу \*/

struct bakery\_request {

/\* Номер клиента \*/

int client;

/\* Номер в очереди \*/

int number;

/\* Место для помещения ответа \*/

char response;

};

program bakery\_program

{

version bakery\_version

{

/\* Взятие номера \*/

struct bakery\_request bakery\_take\_number(struct bakery\_request) = 1;

/\* Обработка \*/

struct bakery\_request bakery\_process(struct bakery\_request) = 2;

} = 1;

} = 0x20000001;

1. Запустить rpcgen:

rpcgen -aC bakery.x

1. Создать два файла: bakery\_worker.h и bakery\_worker.c, в которых реализовать структуры и функции для рабочих потоков:

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <memory.h>

#include "bakery.h"

/\* Количество рабочих потоков \*/

#define WORKERS 5

/\* Структура, которая передается потоку \*/

struct bakery\_worker\_request

{

/\* Запрос от кода клиента \*/

struct bakery\_request client\_request;

/\* Запрос от кода rpc \*/

struct svc\_req svc\_request;

};

/\* Массив запросов к потокам \*/

struct bakery\_worker\_request request[WORKERS];

/\* Массив рабочих потоков \*/

pthread\_t worker[WORKERS];

/\* Массив мьютексов рабочих потоков. Мьютекс заблокирован, если у потока нет запроса или он обрабатывает запрос \*/

pthread\_mutex\_t worker\_mutex[WORKERS];

/\* Мьютекс, обеспечивающий взаимное исключение при доступе к переменным состояния ниже \*/

pthread\_mutex\_t data\_mutex;

/\* Какой номер запроса будет обслужен следующим? \*/

int next\_to\_serve = 1;

/\* Ответ на следующий запрос \*/

char response = 'a';

/\* Функция рабочего потока, принимает номер от 0 до WORKERS - 1 \*/

static void bakery\_worker\_run(void \*param);

/\* Инициализация рабочих потоков и связанных структур \*/

void bakery\_worker\_create();

/\* Остановка потоков и уничтожение связанных структур \*/

void bakery\_worker\_destroy();

/\* Передача запроса потоку \*/

int bakery\_worker\_add\_request(struct bakery\_request \*req, struct svc\_req \*rqstp);

1. В файл bakery.h добавить строки:

**#include <unistd.h>**

#include <rpc/rpc.h>

struct bakery\_request {

int client;

int number;

char response;

};

typedef struct bakery\_request bakery\_request;

**#include "bakery\_worker.h"**

1. Вызвать bakery\_worker\_create() в функции main() файла bakery\_svc.c:

int

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

{

SVCXPRT \*transp = NULL;

int sock;

int proto = 0;

struct sockaddr\_in saddr;

int asize = sizeof (saddr);

**bakery\_worker\_create();**

**…**

1. Вызвать bakery\_worker\_destroy() в функции closedown() файла bakery\_svc.c:

static void

closedown()

{

**bakery\_worker\_destroy();**

…

}

1. В функции bakery\_program\_1 файла bakery\_svc.c добавить флаг немедленного ответа на запрос:

**int reply\_now = 1;**

switch (rqstp->rq\_proc)

{

…

case bakery\_process:

xdr\_argument = (xdrproc\_t) xdr\_bakery\_request;

xdr\_result = (xdrproc\_t) xdr\_bakery\_request;

local = (char \*(\*)(char \*, struct svc\_req \*)) bakery\_process\_1\_svc;

**reply\_now = 0;**

break;

…

}

…

**if(reply\_now)**

**{**

if (result != NULL && !svc\_sendreply(transp, (xdrproc\_t) xdr\_result, result))

{

svcerr\_systemerr(transp);

}

**}**

1. В функцию bakery\_take\_number\_1\_svc файла bakery\_server.c добавить простановку номера запроса в поле number структуры bakery\_request вместо комментария

/\*

\* insert server code here

\*/

1. В функции bakery\_process\_1\_svc вместо комментария выше добавить строку

bakery\_worker\_add\_request(argp, rqstp);

1. В файле bakery\_client.c обеспечить отправку номера клиента в поле client в каждом запросе, а также отправку номера в очереди в поле number при запросе обработки. Установить случайную задержку между получением номера и отправкой запроса на обслуживание.
2. Собрать и запустить приложение.

Для взаимоисключения используются мьютексы для потоков и мьютекс для данных. Мьютекс потока в начале работы заблокирован. Как только потоку приходит запрос, мьютекс потока разблокируется основным потоком сервера, после этого поток блокирует мьютекс и начинает обработку, после обработки вновь ждет разблокировки мьютекса. Перед входом в критическую секцию (изменение буквы и номера очереди) поток блокирует мьютекс данных после выхода – разблокирует его. Перед блокировкой своего мьютекса поток разблокирует мьютексы других потоков, у которых есть запрос, во избежание тупика.

В функции bakery\_worker\_create() необходимо:

* создать потоки, передать в каждый номер потока (= номер клиента)
* инициализировать мьютексы для потоков и мьютекс для данных.

В функции bakery\_worker\_destroy() необходимо:

* остановить потоки;
* уничтожить мьютексы потоков и мьютекс для данных.

В функции bakery\_worker\_add\_request() необходимо:

* записать в структуру request[i] пришедший запрос;
* разблокировать соотвествующий мьютекс;

В функции bakery\_worker\_run() необходимо:

* принять параметр – id потока = id клиента;
* в бесконечном цикле обрабатывать запросы клиента.

Две последние функции приводятся ниже.

int bakery\_worker\_add\_request(struct bakery\_request \*req, struct svc\_req \*rqstp)

{

if (req == NULL || rqstp == NULL)

{

return 1;

}

if (req->client >= 0 && req->client < WORKERS)

{

memmove(&request[req->client].client\_request, req, sizeof(struct bakery\_request));

memmove(&request[req->client].svc\_request, rqstp, sizeof(struct svc\_req));

pthread\_mutex\_unlock(worker\_mutex + req->client);

return 0;

}

return 1;

}

static void bakery\_worker\_run(void \*param)

{

short int id = \*(short int \*) param;

printf("worker %hd is running\n", id);

while (1)

{

pthread\_mutex\_lock(worker\_mutex + id);

struct bakery\_request client\_request = request[id].client\_request;

struct svc\_req svc\_request = request[id].svc\_request;

if (next\_to\_serve == client\_request.number)

{

pthread\_mutex\_lock(&data\_mutex);

static struct bakery\_request result;

result.client = client\_request.client;

result.number = client\_request.number;

result.response = response++;

next\_to\_serve++;

if (!svc\_sendreply(svc\_request.rq\_xprt, (xdrproc\_t) xdr\_bakery\_request, (char \*) &result))

{

perror("svc\_sendreply()");

svcerr\_systemerr(svc\_request.rq\_xprt);

}

pthread\_mutex\_unlock(&data\_mutex);

printf("sent response %c to client %d number %d\n", result.response, result.client, result.number);

for (int i = 0; i < WORKERS; ++i)

{

if (i != id && request[i].client\_request.number > 0)

{

pthread\_mutex\_unlock(worker\_mutex + i);

}

}

}

pthread\_mutex\_lock(worker\_mutex + id);

}

}

Ниже приводится функция клиента:

void

bakery\_program\_1(char \*host, int client)

{

srand(getpid());

CLIENT \*clnt;

struct bakery\_request \*result\_1;

struct bakery\_request bakery\_take\_number\_1\_arg;

struct bakery\_request \*result\_2;

struct bakery\_request bakery\_process\_1\_arg;

clnt = clnt\_create(host, bakery\_program, bakery\_version, "udp");

if (clnt == NULL)

{

clnt\_pcreateerror(host);

exit(1);

}

bakery\_take\_number\_1\_arg.client = client;

result\_1 = bakery\_take\_number\_1(&bakery\_take\_number\_1\_arg, clnt);

if (result\_1 == NULL)

{

clnt\_perror(clnt, "call failed:");

exit(1);

}

printf("client %d got number %d\n", client, result\_1->number);

sleep(rand() % (client+1) + 1);

bakery\_process\_1\_arg.client = client;

bakery\_process\_1\_arg.number = result\_1->number;

result\_2 = bakery\_process\_1(&bakery\_process\_1\_arg, clnt);

if (result\_2 == NULL)

{

clnt\_perror(clnt, "call failed:");

exit(1);

}

printf("client %d number %d got response %c\n", client, result\_2->number, result\_2->response);

clnt\_destroy(clnt);

printf("client %d finished\n", client);

}

Ниже представлены скриншоты работы сервера и клиентов:



