**Отчет о лабораторной работе 3 по курсу «Разработка ПО ОС UNIX”**

**Тема: «Удаленный вызов процедур»**

Выполнил:

Студент группы Б13-001

Спирихин М.А.

1. **Задание**

Разработать программу для Solaris. Программа состоит из серверной и клиентской частей. Серверная часть делит полином на полином, клиентская часть вводит исходные данные и выводит результат. Программа должна собираться из нескольких файлов с использованием make.

1. **Тип данных и файл rpcgen**

Для представления многочлена вводится тип Polynom. Тип содержит степень и массив коэффициентов. Используется массив постоянной длины, подходящий для полинома макс. степени.

Для передачи от клиента к серверу вводится тип DivPolRequest. Тип содержит исходный многочлен и многочлен делитель.

Для передачи от сервера к клиенту вводится тип DivPolResponse. Тип содержит многочлен результат и многочлен остаток от деления.

Ниже представлено описание типов и процедуры в файле DivPol.x.

**const** POLYNOM\_MAX\_POWER=10;

**struct** Polynom {

**int** power;

**int** coefs[POLYNOM\_MAX\_POWER];

};

**struct** DivPolRequest {

Polynom source;

Polynom divider;

};

**struct** DivPolResponse {

Polynom result;

Polynom remainder;

};

**program** DIVPOL\_PROG {

version DIVPOL\_VERS {

DivPolResponse DIVPOL(DivPolRequest) = **1**;

} = **1**;

} = **0x23451111**;

1. **Клиентская часть**

Клиентская часть представляет собой утилиту для командной строки, которая позволяет через меню ввести исходные многочлены, затем она обращается к серверу и выводит результат.

В качестве аргумента передается адрес сервера, производится подключение к нему, в случае неудачи, программа останавливается. Далее через меню пользователь вводит степени и коэффициенты исходных многочленов с проверкой на корректность данных. Функция PolToString переводит многочлен в готовый для отображения вид.

Код содержится в файле DivPol\_client.c.

/\*

\* RPC-client

\* Program do division of two polynoms

\*/

#include "DivPol.h"

//turn Polynom into string

**const** **char**\* **PolToString**(Polynom pol)

{

**char**\* result = (**char** \*) malloc(pol.power\***15**);

**int** i;

**for** (i=**0**;i<=pol.power;i++)

{

**int** p = pol.power - i;

**double** coef = pol.coefs[p];

**if** (coef==**0** && p!=**0**)

**continue**;

**char**\* s = (**char** \*) malloc(**15**);

**if** (p>**0**)

**if** (coef==**1**)

sprintf(s,"x^%d + ",p);

**else**

sprintf(s,"%.2f\*x^%d + ",coef,p);

**else**

sprintf(s,"%.2f",coef);

**if** (i==**0**)

strcpy(result,s);

**else**

strcat(result,s);

}

**return** result;

}

//enter polynom throw console

**void** **PolFromConsole**(Polynom\* pol, **const** **char**\* name)

{

printf("Enter the power of %s:**\n**",name);

**if** (scanf("%d",&(pol->power)) != **1**)

{

printf("Not a number. Program has stopped.**\n**");

exit(**1**);

}

**while** (pol->power < **0** || pol->power >=POLYNOM\_MAX\_POWER)

{

printf("The power of polynom must be from 0 to %d. Enter again:**\n**",POLYNOM\_MAX\_POWER-**1**);

**if** (scanf("%d",&(pol->power)) != **1**)

{

printf("Not a number. Program has stopped.**\n**");

exit(**1**);

}

}

printf("Enter the coefs of %s starting with c0:**\n**",name);

**for** (**int** i = **0**;i<=pol->power;i++)

**if** (scanf("%lf",&(pol->coefs[i])) != **1**)

{

printf("Not a number. Program has stopped.**\n**");

exit(**1**);

}

}

//CheckIfPolynom Is incorrect (coef on max power cant be zero)

**int** **PolIsIncorrect**(Polynom\* pol)

{

**if** (pol->coefs[pol->power]==**0**)

**return** **1**;

**return** **0**;

}

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

{

// arguments check

**if** (argc < **2**) {

printf ("usage: %s server\_host [-l use literal]**\n**", argv[**0**]);

exit (**1**);

}

// try handle to server

**char** \*host;

host = argv[**1**];

CLIENT \*clnt = clnt\_create (host, DIVPOL\_PROG, DIVPOL\_VERS, "udp");

**if** (clnt == NULL) {

clnt\_pcreateerror(host);

exit(**1**);

}

// data entrance

Polynom source, divider;

**if** (argc == **3** && strcmp(argv[**2**], "-l") == **0**)

{

// Data by Literal

source.power = **3**;

source.coefs[**0**]=**2**;

source.coefs[**1**]=**0**;

source.coefs[**2**]=**7**;

source.coefs[**3**]=**1**;

divider.power=**1**;

divider.coefs[**0**]=-**1**;

divider.coefs[**1**]=**9**;

}

**else**

{

// Menu for data entrance

PolFromConsole(&(source),"source polynom");

PolFromConsole(&(divider),"divider");

**if** (PolIsIncorrect(&(source)) == **1** || PolIsIncorrect(&(divider)) == **1**)

{

printf("Incorrect polynom**\n**");

exit(**1**);

}

}

printf("Source = %s.**\n**",PolToString(source));

printf("Divider = %s.**\n**",PolToString(divider));

// prepare data for server and do request

DivPolResponse \*response;

DivPolRequest request;

request.source = source;

request.divider = divider;

response = divpol\_1(&request, clnt);

**if** (response == (DivPolResponse \*) NULL) {

clnt\_perror (clnt, "call failed");//request error

} **else** {

//request complete. Show result

printf("Result = %s.**\n**",PolToString(response->result));

printf("Remainder = %s.**\n**",PolToString(response->remainder));

printf("%s = (%s)\*(%s) + %s.**\n**",PolToString(source),PolToString(divider),

PolToString(response->result),PolToString(response->remainder));

}

//delete clnt and exit

clnt\_destroy (clnt);

exit (**0**);

}

1. **Серверная часть и алгоритм деления**

В серверной части находится функция DivPolResponse DIVPOL(DivPolRequest), которая запрашивается с сервера. Функция принимает DivPolRequest, запускает внутреннюю функцию Divide, которая осуществляет деление многочленов и возвращает результат.

Функция Divide осуществляет деление многочлена source на многочлен divider, записывает результат в result, остаток в reminder. Используется стандартный алгоритм деления многочленов столбиком. После цикла с помощью функции NormalizePower производится пересчет степени результирующих многочленов, чтобы не было нуля в макс. степени.

Код содержится в файле DivPol\_server.c.

/\*

\* RPC-server

\* Program do division of two polynoms

\*/

#include "DivPol.h"

//turn Polynom into string (Like in client)

**const** **char**\* **PolToString**(Polynom pol)

{

**…**

}

//Get Real Power of polynom

**int** **NormalizePower**(Polynom pol)

{

**int** result = **0**;

**for** (**int** i=pol.power;i>=**0**;i--)

{

**if** (pol.coefs[i] > **0.000001**)

{

**return** i;

}

}

**return** **0**;

}

//polynoms division

**void** **Divide**(Polynom source, Polynom divider, Polynom\* result, Polynom\* remainder)

{

remainder->power = source.power;

**for**(**int** i=**0**;i<=remainder->power;i++)

remainder->coefs[i] = source.coefs[i];

result->power = remainder->power - divider.power;

**for** (**int** i = **0**; i <= result->power; i++)

{

**double** coeff = remainder->coefs[remainder->power - i] / divider.coefs[divider.power];

result->coefs[result->power - i] = coeff;

**for** (**int** j = **0**; j <= divider.power; j++)

{

remainder->coefs[remainder->power - i - j] -= coeff \* divider.coefs[divider.power - j];

}

}

remainder->power = NormalizePower(\*remainder);

result->power = NormalizePower(\*result);

}

//main on server

DivPolResponse\* **divpol\_1\_svc**(DivPolRequest \*request, **struct** svc\_req \*rqstp)

{

**static** DivPolResponse response;

printf("Source = %s.**\n**",PolToString(request->source));

printf("Divider = %s.**\n**",PolToString(request->divider));

Divide(request->source,request->divider, &(response.result),&(response.remainder));

printf("Result = %s.**\n**",PolToString(response.result));

printf("Remainder = %s.**\n**",PolToString(response.remainder));

**return** &response;

}

1. **MakeFile**

Используются 4 исходных файла

1. Файл для rpcgen – DivPol.x
2. Серверное приложение – DivPol\_server.c
3. Клиентское приложение – DivPol\_client.c
4. Файл для программы make – MakeFile.DivPol

Сборка осуществляется командой

make –f MakeFile.DivPol

Очистку всех файлов кроме исходных можно произвести

make –f MakeFile.DivPol clean

Код файла MakeFile.DivPol.

# Parameters

CLIENT = DivPol\_client

SERVER = DivPol\_server

SOURCES\_CLNT.c =

SOURCES\_CLNT.h =

SOURCES\_SVC.c =

SOURCES\_SVC.h =

SOURCES.x = DivPol.x

TARGETS\_SVC.c = DivPol\_svc.c DivPol\_server.c DivPol\_xdr.c

TARGETS\_CLNT.c = DivPol\_clnt.c DivPol\_client.c DivPol\_xdr.c

TARGETS = DivPol.h DivPol\_xdr.c DivPol\_clnt.c DivPol\_svc.c DivPol\_client.c DivPol\_server.c

TARGETS\_CLEAN = DivPol.h DivPol\_xdr.c DivPol\_clnt.c DivPol\_svc.c

OBJECTS\_CLNT = **$(**SOURCES\_CLNT.c:%.c=%.o**)** **$(**TARGETS\_CLNT.c:%.c=%.o**)**

OBJECTS\_SVC = **$(**SOURCES\_SVC.c:%.c=%.o**)** **$(**TARGETS\_SVC.c:%.c=%.o**)**

# Compiler flags

CC = gcc

CFLAGS += -g -DRPC\_SVC\_FG

LDLIBS += -lnsl

RPCGENFLAGS = -C

# Targets

all : **$(**CLIENT**)** **$(**SERVER**)**

**$(**TARGETS**)** : **$(**SOURCES.x**)**

rpcgen **$(**RPCGENFLAGS**)** **$(**SOURCES.x**)**

**$(**OBJECTS\_CLNT**)** : **$(**SOURCES\_CLNT.c**)** **$(**SOURCES\_CLNT.h**)** **$(**TARGETS\_CLNT.c**)**

**$(**OBJECTS\_SVC**)** : **$(**SOURCES\_SVC.c**)** **$(**SOURCES\_SVC.h**)** **$(**TARGETS\_SVC.c**)**

**$(**CLIENT**)** : **$(**OBJECTS\_CLNT**)**

**$(**LINK.c**)** -o **$(**CLIENT**)** **$(**OBJECTS\_CLNT**)** **$(**LDLIBS**)**

**$(**SERVER**)** : **$(**OBJECTS\_SVC**)**

**$(**LINK.c**)** -o **$(**SERVER**)** **$(**OBJECTS\_SVC**)** **$(**LDLIBS**)**

clean:

**$(**RM**)** core **$(**TARGETS\_CLEAN**)** **$(**OBJECTS\_CLNT**)** **$(**OBJECTS\_SVC**)** **$(**CLIENT**)** **$(**SERVER**)**