Министерство образования Республики Беларусь

Учреждение образования

Белорусский государственный университет

информатики и радиоэлектроники

Факультет инженерно-экономический

Кафедра экономической информатики

Лабораторная работа №4

Создание клиент-серверных приложений с установлением логического соединения (TCP/IP), используя алгоритм параллельной обработки запросов (отдельный процесс для обработки каждого запроса)

Вариант 12

Выполнил: Томильчик В. С.,

группа 074004

Проверил: Федосенко В. А.

Минск, 2022

**Задание (Вариант 12)**

Осуществить взаимодействие клиента и сервера на основе протокола TCP/IP. Реализовать параллельное соединение с использованием отдельного процесса для обработки каждого запроса. На сервере хранится список товаров, имеющихся на складе. Каждая запись списка содержит следующую информацию о товарах

- страна-изготовитель;

- фирма-изготовитель;

- наименование товара;

- количество единиц товара.

Таких записей должно быть не менее 5.

Клиент посылает на сервер наименование товара. Назад он получает количество единиц этого товара.

**Серверная часть:**

#include <sys/types.h>

#include <sys/socket.h>

#include <sys/signal.h>

#include <sys/wait.h>

#include <sys/resource.h>

#include <netinet/in.h>

#include <string.h>

#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

struct Book {

char country[10];

char firm[100];

char name[30];

char expirationdate[10];

char count[20];

char cost[10];

char info[256];

} product[5];

int Func(int newS)

{

long int num;

char info[256];

int i, m;

char\* name;

char mas[100];

char p, p1;

char buf[256], buffer[256];

while (true)

{

recv(newS, buf, sizeof(buf), 0);

p = buf[0];

printf("Your choise %c\n", p);

switch (p)

{

case '1':

for (int i = 1; i <= 5; i++) {

strcpy(buf, product[i].country);

send(newS, buf, sizeof(buf), 0);

printf("Sended message: %s\n", buf);

}

recv(newS, buf, sizeof(buf), 0);

printf("Received message: %s\n", buf);

if (strcmp(buf, "1") == 0 || strcmp(buf, "2") == 0 || strcmp(buf, "3") == 0 || strcmp(buf, "4") == 0 || strcmp(buf, "5") == 0)

{

send(newS, buf, sizeof(buf), 0);

num = atoi(buf);

strcpy(info, product[num].country);

for (int i = 1; i <= 5; i++) {

if (strcmp(product[i].country, info) == 0) {

strcpy(buf, product[i].firm); send(newS, buf, sizeof(buf), 0);

strcpy(buf, product[i].name); send(newS, buf, sizeof(buf), 0);

strcpy(buf, product[i].count);send(newS, buf, sizeof(buf), 0);

strcpy(buf, "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n"); send(newS, buf, sizeof(buf), 0);

}

if (i == 5) {

strcpy(buf, "\*break");

send(newS, buf, sizeof(buf), 0);

}

}

break;

}

else {

strcpy(buf, "Invalid input! Try again!\n");

send(newS, buf, sizeof(buf), 0);

printf("Sended message: %s\n", buf);

break;

}

case '2':

recv(newS, buf, sizeof(buf), 0);

num = atoi(buf);

printf("Your value: %d\n", num);

recv(newS, buf, sizeof(buf), 0);

p1 = buf[0];

printf("Your choise %c\n", p1);

switch (p1)

{

case '1':

recv(newS, buf, sizeof(buf), 0);

strcpy(product[num].country, buf);

break;

case '2':recv(newS, buf, sizeof(buf), 0);

strcpy(product[num].firm, buf);

break;

case '3':recv(newS, buf, sizeof(buf), 0);

strcpy(product[num].name, buf);

break;

case '4':recv(newS, buf, sizeof(buf), 0);

strcpy(product[num].expirationdate, buf);

break;

case '5':recv(newS, buf, sizeof(buf), 0);

strcpy(product[num].count, buf);

break;

case '6':recv(newS, buf, sizeof(buf), 0);

strcpy(product[num].cost, buf);

}

printf("The recieved string: %s\n", buf);

printf("Editing is allowed!\n");

break;

case '3':

for (i = 1; i <= 5; i++) {

buf[0] = '\0';

strcat(buf, product[i].country); strcat(buf, " ");

strcat(buf, product[i].firm); strcat(buf, " ");

strcat(buf, product[i].name); strcat(buf, " ");

strcat(buf, product[i].expirationdate); strcat(buf, " ");

strcat(buf, product[i].count); strcat(buf, " ");

strcat(buf, product[i].cost); strcat(buf, "\n");

send(newS, buf, sizeof(buf), 0);

}

break;

case '4':

exit(0);

}

}

}

void print(int a)

{

if (a) printf("%d client connected\n", a);

else printf("No clients connected\n");

}

void reaper(int sig)

{

int status;

while (wait3(&status, WNOHANG, (struct rusage\*)0) >= 0);

}

int main()

{

int numcl = 0;

strcpy(product[1].country, "USA");

strcpy(product[1].firm, "Coca-cola");

strcpy(product[1].name, "Sprite");

strcpy(product[1].expirationdate, "29.05.2023");

strcpy(product[1].count, "1000");

strcpy(product[1].cost, "3");

strcpy(product[2].country, "UA");

strcpy(product[2].firm, "Roshen");

strcpy(product[2].name, "Sweets");

strcpy(product[2].expirationdate, "29.05.2023");

strcpy(product[2].count, "1050");

strcpy(product[2].cost, "5");

strcpy(product[3].country, "Germany");

strcpy(product[3].firm, "Paulo");

strcpy(product[3].name, "Chocolate");

strcpy(product[3].expirationdate, "29.05.2023");

strcpy(product[3].count, "1000");

strcpy(product[3].cost, "10");

strcpy(product[4].country, "Itali");

strcpy(product[4].firm, "Barilla");

strcpy(product[4].name, "Pasta");

strcpy(product[4].expirationdate, "29.05.2023");

strcpy(product[4].count, "900");

strcpy(product[4].cost, "7");

strcpy(product[5].country, "Poland");

strcpy(product[5].firm, "Tea");

strcpy(product[5].name, "Curtis");

strcpy(product[5].expirationdate, "29.05.2023");

strcpy(product[5].count, "750");

strcpy(product[5].cost, "14");

struct sockaddr\_in local;

int s, newS, rc;

local.sin\_family = AF\_INET;

local.sin\_port = htons(7500);

local.sin\_addr.s\_addr = htonl(INADDR\_ANY);

puts("Server is online!");

s = socket(AF\_INET, SOCK\_STREAM, 0);

rc = bind(s, (struct sockaddr\*)&local, sizeof(local));

rc = listen(s, 7);

(void)signal(SIGCHLD, reaper);

while (true) {

newS = accept(s, NULL, NULL);

numcl++;

print(numcl);

switch (fork()) {

case 0:

(void)close(s);

exit(Func(newS));

default:

(void)close(newS);

}

}

return 0;

}

**Клиентская часть:**

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

int main() {

struct sockaddr\_in peer;

int s, t, t1;

int rc;

char buf[256], p, p1, b[256];

peer.sin\_family = AF\_INET;

peer.sin\_port = htons(7500);

peer.sin\_addr.s\_addr = inet\_addr("127.0.0.1");

s = socket(AF\_INET, SOCK\_STREAM, 0);

rc = connect(s, (struct sockaddr\*)&peer, sizeof(peer));

while (true) {

puts("Choose:");

puts("\t1 - Select record");

puts("\t2 - Edit record");

puts("\t3 - View records");

puts("\t4 - Exit");

scanf("%s", buf);

buf[1] = '\0';

send(s, buf, sizeof(buf), 0);

p = buf[0];

switch (p)

{

case '1':

int i;

printf("\nChoose a country (1-5) from the list:\n");

for (i = 1; i <= 5; i++) {

recv(s, buf, sizeof(buf), 0);

printf("\t%d) %s\n", i, buf);

}

printf("Your choice: ");

scanf("%s", buf);

send(s, buf, sizeof(buf), 0);

recv(s, buf, sizeof(buf), 0);

if (strcmp(buf, "Invalid input! Try again!\n") == 0) {

printf("Invalid input! Try again!\n");

break;

}

else {

i = 1;

printf("\nThe list of products from this country:\n\t\_\_\n\n");

while (1) {

recv(s, buf, sizeof(buf), 0);

if (strcmp(buf, "\*break") == 0)

break;

printf("\tBRAND: %s\n", buf);

recv(s, buf, sizeof(buf), 0);

printf("\tNAME OF PRODUCT: %s\n", buf);

recv(s, buf, sizeof(buf), 0);

printf("\tTOTAL NUMBER: %s\n\n", buf);

recv(s, buf, sizeof(buf), 0);

printf("\t%s\n", buf);

}

break;

}

case '2':

puts("Which number do you want to edit(1-5)?");

scanf("%s", buf);

send(s, buf, sizeof(buf), 0);

puts("What field (1-6) to edit");

puts("\t1 - Country");

puts("\t2 - Firm");

puts("\t3 - Name");

puts("\t4 - expirationdate");

puts("\t5 - Count");

puts("\t6 - Cost");

scanf("%s", buf);

send(s, buf, sizeof(buf), 0);

p1 = buf[0];

buf[0] = '\0';

switch (p1) {

case '1':

int i;

printf("\nChoose a country (1-5) from the list:\n");

for (i = 1; i <= 5; i++) {

recv(s, buf, sizeof(buf), 0);

printf("\t%d) %s\n", i, buf);

}

printf("Your choice: ");

scanf("%s", buf);

send(s, buf, sizeof(buf), 0);

recv(s, buf, sizeof(buf), 0);

if (strcmp(buf, "Invalid input! Try again!\n") == 0) {

printf("Invalid input! Try again!\n");

break;

}

else {

i = 1;

printf("\nThe list of product from this country:\n\t\_\_\n\n");

while (1) {

recv(s, buf, sizeof(buf), 0);

if (strcmp(buf, "\*break") == 0)

break;

printf("\tFirm: %s\n", buf);

recv(s, buf, sizeof(buf), 0);

printf("\tName: %s\n", buf);

recv(s, buf, sizeof(buf), 0);

printf("\tNumber: %s\n\n", buf);

recv(s, buf, sizeof(buf), 0);

printf("\t%s\n", buf);

}

break;

}

case '2':printf("Firm: ");

fflush(stdin); fflush(stdout);

scanf("%s", buf);

send(s, buf, sizeof(buf), 0);

break;

case '3':

fflush(stdin); fflush(stdout);

printf("Name: "); scanf("%s", buf);

send(s, buf, sizeof(buf), 0);

break;

case '4':

fflush(stdin); fflush(stdout);

printf("expirationdate: "); scanf("%s", buf);

send(s, buf, sizeof(buf), 0);

break;

case '5':

fflush(stdin); fflush(stdout);

printf("Count: "); scanf("%s", buf);

send(s, buf, sizeof(buf), 0);

break;

case '6':

fflush(stdin); fflush(stdout);

printf("Cost: "); scanf("%s", buf);

send(s, buf, sizeof(buf), 0);

default:

{

printf("Invalid value: ");

}

}

break;

case '3':

recv(s, buf, sizeof(buf), 0); printf("%s", buf);

recv(s, buf, sizeof(buf), 0); printf("%s", buf);

recv(s, buf, sizeof(buf), 0); printf("%s", buf);

recv(s, buf, sizeof(buf), 0); printf("%s", buf);

recv(s, buf, sizeof(buf), 0); printf("%s", buf);

break;

case '4':

exit(0);

default:

{printf("Invalid value: ");}

}

}

}

**Контрольные вопросы:**

1 По какой причине последовательная реализация сервера, может оказаться неудовлетворительной?

Последовательная реализация сервера может оказаться неудовлетворительной, поскольку клиенты будут вынуждены ждать завершения обработки всех предыдущих запросов на установление соединения. Если клиент решит передать большие объемы данных (например, несколько мегабайт), последовательный сервер отложит обслуживание всех других клиентов до тех пор, пока не выполнит этот запрос.

2 Какие преимущества предоставляет параллельная реализация сервера?

Параллельная реализация сервера дает возможность обойтись без продолжительных задержек, так как не позволяет одному клиенту захватить все ресурсы. Вместо этого параллельный сервер поддерживает обмен данными сразу с несколькими клиентами для того, чтобы их запросы выполнялись одновременно.

1. Какие основные задачи выполняет ведущий процесс?

Ведущий процесс создает ведомый процесс для обслуживания соединения и предоставляет возможность ведомому процессу работать параллельно с ним. В любое время в сервере функционирует один ведущий процесс, а число ведомых процессов может составлять от нуля и более. Ведущий процесс сервера принимает каждый входящий запрос на установление соединения и создает ведомый процесс для его обслуживания.

1. Для чего используется блокирующий вызов функции *accept()*?

Для получения очередного запроса на установление соединения из общепринятого порта используется блокирующий вызов функции accept, что позволяет предотвратить бесполезное расходование ресурсов процессора во время ожидания очередных запросов на установление соединения в ведущем процессе сервера.

1. Когда и для чего вызывается функция fork()?

После возврата управления из функции accept() ведущий процесс сервера создает ведомый процесс для обслуживания соединения. Для этого ведущий процесс вызывает функцию fork(), чтобы разделиться на два процесса.

1. Каким образом решается проблема не полностью завершившихся процессов (процессов, информация о которых остается в системных таблицах)?

В системе Linux эта проблема решается путем передачи специального сигнала родительскому процессу после завершения работы каждого дочернего процесса. Завершившийся процесс остается в виде так называемого процесса-зомби до тех пор, пока родительским процессом не будет выполнен системный вызов wait3.

1. Какой код используется в процессе для обозначения нормального завершения?

В соответствии с общепринятым соглашением, в процессе для обозначения нормального завершения используется код завершения нуль.