МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

**Федеральное государственное автономное образовательное учреждение высшего образования «Самарский национальный исследовательский университет   
имени академика С.П. Королева»**

**(Самарский университет)**

ИНСТИТУТ ИНФОРМАТИКИ И КИБЕРНЕТИКИ

Кафедра программных систем

Дисциплина

Технологии промышленного программирования

**ОТЧЁТ**

по лабораторной работе

**Запуск и организация взаимодействия**

**параллельных процессов**

Студент: В.А. Артамонов

Группа: 6231-020402D

Преподаватель: Баландин А.В.

Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_

Самара  
2022

Задание

ВАРИАНТ №7

Разработать приложение, состоящее из пяти взаимодействующих процессов. Требуется написать три программных модуля – М1, М2, М3. На базе модуля М1 из shell запускается стартовый процесс Р1(М1).

Процесс Р1 создает свой канал и, используя функцию семейства spawn\*(), запускает процессы Р2(М2) и Р3(М2), передавая им в качестве аргумента chid своего канала, затем переходит в состояние приема сообщений по своему каналу.

Процесс Р2 создает свой канал и, используя функцию семейства spawn\*(), запускает процесс Р4(М3), передавая в качестве аргумента chid своего канала, затем переходит в состояние приема сообщений по созданному каналу.

Процесс Р3 создает свой канал и, используя функцию семейства spawn\*(), запускает процесс Р5(М3), передавая в качестве аргумента chid своего канала, затем переходит в состояние приема сообщений по созданному каналу.

Процесс Р?(М3) устанавливает соединение c каналом родительского процесса Р?(М2) и посылает запрос на получение pid процесса Р1 и chid его канала, затем, после получения ответа (pid и chid), устанавливает соединение с каналом процесса Р1 и посылает ему сообщение "Р? loaded". После получения ответа выводит на экран "Р? ОК" и терминируется.

Процесс Р?(М2) после ответа на запрос процесса Р?(М3) выводит на терминал "Р? ОК" и терминируется.

Процесс Р1, получив сообщение от процесса Р1 или Р5, выводит его на экран и посылает ответ. После взаимодействия с Р1 и Р5 процесс Р1 выводит на экран "Р1 ОК" и терминируется.

Код программы

Процесс P1

**#include** <cstdlib>

**#include** <iostream>

**#include** <stdlib.h>

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

**#include** <unistd.h>

**#include** <process.h>

**#include** <string.h>

/\* Процесс M1 создаёт свой канал и, используя функцию семейства spawn\*(),

\* запускает процессы М2 и М3,

\* передавая им в качестве аргумента chid своего канала,

\* затем переходит в состояние приёма сообщений по своему каналу. \*/

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

**const** **char** \*MODULE\_2\_PATH = "/home/Test1.2";

/\* Создание канала.

\* flags - набор флагов свойств канала,

\* 0 – устанавливает свойства по умолчанию \*/

**int** chid = **ChannelCreate**(0);

std::cout << "P1" << " Запущен с своим чидом = " << chid << std::endl;

// Преобразование числа в Си-строку

**char** buffer[20];

**const** **char** \*strChid = **itoa**(chid, buffer, 10);

/\* Вызов дочерних процессов.

\* Функции семейства spawn\*() способны

\* создавать новый (дочерний) процесс со своим pid,

\* параллельно выполняющийся вместе с родительским процессом.

\* P\_NOWAIT – родительский процесс не блокируется,

\* выполняется параллельно с дочерним процессом

\* и должен ожидать завершения дочернего процесса.

\* path – пусть к .exe

\*

\* В случае ошибки возвращается -1 (устанавливается errno). \*/

**int** p2pid = **spawnl**(P\_NOWAIT, MODULE\_2\_PATH, MODULE\_2\_PATH, strChid, "2",

NULL);

**if** (p2pid < 0)

std::cout << "P2 ошибка при создании процесса" << std::endl;

**int** p3pid = **spawnl**(P\_NOWAIT, MODULE\_2\_PATH, MODULE\_2\_PATH, strChid, "3",

NULL);

**if** (p3pid < 0)

std::cout << "P3 ошибка при создании процесса" << std::endl;

/\* Приём сообщения.

\* Процесс-сервер должен принять сообщение и послать ответ клиенту.

\*

\* Если сообщение уже было в канале, когда нить сервера вызывает MsgReceive(),

\* то оно немедленно копируется ядром в адресное пространство сервера.

\* Если сообщения в канале нет, то принимающая сообщение нить сервера

\* переходит в RECEIVE-блокированное состояние,

\* ожидая пока сообщение от клиента не поступит в канал.

\* При получении сообщения нить переходит в

\* состояние готовности к выполнению (READY).

\*

\* Получив сообщение от клиента сервер должен послать ему ответное сообщение.

\* При этом нить должна находится в REPLY-блокированном состоянии.

\* Ответ может послать любая нить сервера.

\* Важно только, чтобы на каждое

\* принятое сервером сообщение следовал бы ответ и только один.

\* Выполнение функции MsgSend(), вызванной нитью клиента,

\* которой соответствует rcvid, завершается разблокированием нити

\* и возвратом функцией MsgSend() значения статуса, заданного сервером

\* в аргументе status при выполнении функции MsgReply().

\*

\* Функция MsgReply() не блокирует нить сервера,

\* передача ответа выполняется немедленно. \*/

**int** rcvid = -1; // ссылка на нить клиента

**char** msg[512]; // буфер приема сообщения

\_msg\_info info; // информация о сообщении

**int** msgCounter = 0;

**while** (msgCounter < 2) { // получить два сообщения

rcvid = **MsgReceive**(chid, msg, **sizeof**(msg), &info); // получить сообщение

**if** (rcvid == -1)

std::cout << "P1 не получил сообщение" << std::endl;

**else**

std::cout << msg << std::endl;

**MsgReply**(rcvid, 0, msg, **sizeof**(msg)); // посылка ответа клиенту

msgCounter++;

}

std::cout << "P1 OK" << std::endl;

**return** EXIT\_SUCCESS;

}Процесс P2

**#include** <cstdlib>

**#include** <iostream>

**#include** <stdlib.h>

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

**#include** <unistd.h>

**#include** <process.h>

**#include** <string.h>

**#include** <string>

/\* Процесс M2 создаёт свой канал и, используя функцию семейства spawn\*(),

\* запускает процесс М3, передавая в качестве аргумента chid своего канала,

\* затем переходит в состояние приёма сообщений по созданному каналу.

\*

\* Процесс Р?(М2) после ответа на запрос процесса Р?(М3)

\* выводит на терминал "Р? ОК" и терминируется. \*/

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

**const** **char**\* MODULE\_3\_PATH = "/home/Test1.3";

std::string pnum = argv[2];

std::string pname = "P" + pnum;

**int** grandparentChid = atoi(argv[1]); // id канала родительского процесса

**int** chid = **ChannelCreate**(0); // создание канала

std::cout << pname << " Запущен с своим чидом = " << chid << std::endl;

// Преобразование числа в Си-строку

**char** buffer[20];

**const** **char**\* strChid = **itoa**(chid, buffer, 10);

// Вызов дочернего процесса

**int** childPid = **spawnl**(P\_NOWAIT, MODULE\_3\_PATH, MODULE\_3\_PATH, strChid,

argv[2], NULL);

**if** (childPid < 0)

std::cout << pname << "1 ошибка при создании процесса" << std::endl;

// Приём сообщения

**int** rcvid = -1; // ссылка на нить клиента

**char** msg[512]; // буфер приёма сообщения

\_msg\_info info; // информация о сообщении

**int** replyMsg[2]; // буфер ответа

replyMsg[0] = grandparentChid; // chid P1

replyMsg[1] = **getppid**(); // pid P1

**while** (rcvid == -1) { // получить одно сообщение

rcvid = **MsgReceive**(chid, msg, **sizeof**(msg), &info); // получить сообщение

**if** (rcvid == -1)

std::cout << pname << " не получил сообщение" << std::endl;

**else**

std::cout << msg << std::endl;

**MsgReply**(rcvid, 0, replyMsg, **sizeof**(replyMsg)); // посылка ответа клиенту

}

std::cout << pname << " OK" << std::endl;

**return** EXIT\_SUCCESS;

}

Процесс P3

**#include** <cstdlib>

**#include** <iostream>

**#include** <stdlib.h>

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

**#include** <unistd.h>

**#include** <process.h>

**#include** <string.h>

**#include** <string>

/\* Процесс М3 устанавливает соединение c каналом родительского процесса М2

\* и посылает запрос на получение pid процесса Р1 и chid его канала,

\* затем, после получения ответа (pid и chid),

\* устанавливает соединение с каналом процесса Р1

\* и посылает ему сообщение "Р? loaded".

\* После получения ответа выводит на экран "Р? ОК" и терминируется. \*/

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

std::string pnum = argv[2];

std::string pname = "P" + pnum + "1";

std::cout << pname << " запущен" << std::endl;

**int** parentChid = atoi(argv[1]); // id канала родительского процесса

/\* Передача сообщения родителю.

\* connectionid — id канала для отправки сообщения.

\* Установление соединения с каналом: \*/

**int** connectionId = **ConnectAttach**(0, **getppid**(), parentChid,

\_NTO\_SIDE\_CHANNEL, 0);

**if** (connectionId == -1) {

std::cout << pname << " ошибка соединения " << std::endl;

exit(EXIT\_FAILURE);

}

// Запрос на получение pid процесса Р1 и chid его канала

**int** replyMsg1[2]; // буфер ответа

**const** **char**\* msg = (pname + " отправка запроса").c\_str(); // сообщение процессу серверу

**if** (MsgSend(connectionId, msg, strlen(msg) + 1, // послать сообщение P2

replyMsg1, **sizeof**(replyMsg1)) == -1) {

std::cout << pname << " ошибка получения сообщения" << std::endl;

exit(EXIT\_FAILURE);

}

**if** (replyMsg1 != NULL) {

std::cout << pname << " сервер ответил P1 с chid = "

<< replyMsg1[0] << " и P1 pid = " << replyMsg1[1]

<< std::endl;

}

// Установление соединения с каналом P1

connectionId = **ConnectAttach**(0, replyMsg1[1], replyMsg1[0],

\_NTO\_SIDE\_CHANNEL, 0);

**if** (connectionId == -1) {

std::cout << pname << " ошибка соединения" << std::endl;

exit(EXIT\_FAILURE);

}

// Посылка сообщения P1

**char** replyMsg2[512];//буфер ответа

std::string temp = pname + " загружен";

msg = temp.c\_str();

**if** (MsgSend(connectionId, msg, strlen(msg) + 1, // послать сообщение P1

replyMsg2, **sizeof**(replyMsg2)) == -1) {

std::cout << pname << " ошибка получения сообщения" << std::endl;

exit(EXIT\_FAILURE);

}

**if** (replyMsg2 != NULL)

std::cout << pname << " OK " << std::endl;

**return** EXIT\_SUCCESS;

}