Федеральное агентство по образованию

Государственное образовательное учреждение

высшего профессионального образования

**САМАРСКИЙ ГОСУДАРСТВЕННЫЙ АЭРОКОСМИЧЕСКИЙ**

**УНИВЕРСИТЕТ им. академика С.П. КОРОЛЕВА**

**(национальный исследовательский университет)**

**Факультет информатики**

Отчёт к лабораторной работе

по дисциплине «Параллельные вычислительные системы»

Тема:

**«Многопроцессорные приложения»**

**Вариант №9**

Выполнил Проценко В. И.

Группа М618

Преподаватель Баландин А.В.

Самара 2012

Оглавление

[Цель и постановка задания 3](#_Toc336119693)

[Приложения 7](#_Toc336119694)

[Исходный код 7](#_Toc336119695)

# Цель и постановка задания

**Тема лабораторной работы**: проектирование и разработка многопроцессных приложений, организация взаимодействия параллельных процессов.

**Цель задания**: изучить и освоить функции операционной системы QNX для запуска параллельных процессов и организации межпроцессного взаимодействия с помощью механизма обмена сообщениями.

**Задание**:

1. Перед выполнением полученного задания необходимо внимательно изучить следующие стандартные функции операционной системы QNX для запуска и организации взаимодействия параллельных процессов:

* семейство функций spawn();
* флаги запуска P\_WAIT, P\_NOWAIT, P\_NOWAITO, P\_OVERLAY;
* функции управления каналом ChannelCreate(), ChannelDestroy(), ConnectAttach(),ConnectDetach();
* функции передачи сообщений MsgSend(), MsgReceive(), MsgReply(), MsgRead(),MsgWrite().

1. Написать программу, реализующую указанный вариант задания, максимально используя при программировании приведенные выше функции. Программа должна выводить на экран подробные комментарии к выполняемым действиям.

**Форма отчетности**: для отчета по лабораторной работе требуется представить следующее:

1. результат выполнения программы на компьютере в среде операционной системы QNX;
2. письменный отчет по результатам выполнения лабораторной работы, содержащий схему взаимодействия параллельных процессов в виде UML-диаграммы последовательностей, описание механизма взаимодействия процессов, программный код.

**ВАРИАНТ №9**

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

Процесс Р1 создает канал и, используя функцию семейства spawn\*(), запускает процесс Р2(М2), передавая ему в качестве параметра chid созданного канала, затем переходит в состояние приема сообщений по созданному каналу.

Процесс Р2 создает свой канал, и, используя функцию семейства spawn\*(), запускает процесс Р3(М3), передавая ему в качестве параметра chid созданного канала, затем устанавливает соединение с каналом процесса Р1, передает ему chid своего канала, затем переходит в состояние приема сообщений по своему каналу.

Процесс Р3 устанавливает соединение с каналом процесса Р2 и посылает ему запрос на получение pid процесса Р1 и chid его канала. Получив от Р2 ответ (pid и chid), устанавливает соединение с каналом процесса Р1 и посылает ему свой pid и chid своего канала, после чего переходит в состояние приема сообщений по своему каналу.

Процесс Р1 устанавливает соединение с каналом процесса Р2 и передает ему сообщение "Р1 send message to Р2", получив ответ, выводит его на терминал, затем устанавливает соединение с каналом процесса Р3 и передает ему сообщение "Р1 send message to Р3", получает ответ, выводит его на экран и терминируется.

Процесс Р2, получив сообщение от процесса Р1, выводит его на терминал, посылает ответ "Р2 ОК" и терминируется.

Процесс Р3, получив сообщение от процесса Р1, выводит его на экран, посылает ответ "Р3 ОК" и терминируется.

# D:\Temp\Коммуникация процессов.pngДиаграмма последовательностей

Рис . Коммуникация процессов

# Результат работы программы

# C:\QNX_Shared\Lab1Snapshot.png

Рис. Результат работы программы

# Приложения

### Исходный код

### M1.c

#include <stdlib.h>

#include <stdio.h>

#include <stddef.h>

#include <process.h>

#include <errno.h>

#include <sys/neutrino.h>

#include <sys/netmgr.h>

const int BUFFER\_LENGTH = 30;

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

pid\_t pid\_1 = getpid();

char \*msg = (char\*)malloc(BUFFER\_LENGTH\*sizeof(char));

char \*rmsg = (char\*)malloc(BUFFER\_LENGTH\*sizeof(char));

// Step 1 Creating channel - chidP1.

int chidP1 = ChannelCreate(0);

printf("P1>\n");

printf("P1> pid: %i, chid: %i\n", pid\_1, chidP1);

// Step 2 Spawning P2 process - pid\_2.

char argv0[12];

sprintf(argv0, "%i", chidP1);

int pid\_2 = spawnl(P\_NOWAITO, "/tmp/M2", argv0, NULL); // Spawned P2

printf("P1> Spawned new process. res: %i\n", pid\_2);

// Step 6 Receiving channel id of P2 process - chidP2.

int rcvid = MsgReceive(chidP1, msg, sizeof(int), NULL);

printf("P1> Received, rcvid: %i\n", rcvid);

printf("P1> Message: %s\n", msg);

int chidP2 = \*(int\*)((int)msg);

// Step 7 Reply to P2 process.

strcpy(msg, "P1 received chidP2");

MsgReply(rcvid, EOK, msg, sizeof(msg));

printf("P1> Message: %s Size of message: %i\n", msg, sizeof(msg));

printf("P1> chidP2: %i\n", chidP2);

// Step 10 Receiving process and channel id's of P3 - pid\_3, chidP3.

rcvid = MsgReceive(chidP1, msg, 20, NULL);

int pid\_3 = \*(int\*)((int)msg);

int chidP3 = \*(int\*)((int)msg + 4);

// Step 11 Reply to P3 process.

strcpy(rmsg, "P1 received pid\_3, chidP3");

MsgReply(rcvid, EOK, rmsg, sizeof(rmsg));

printf("P1> pid\_3: %i chidP3: %i\n", pid\_3, chidP3);

// Step 12 Sending to P2 message "P1 send message to P2"

int coid\_2 = ConnectAttach(ND\_LOCAL\_NODE, pid\_2, chidP2, \_NTO\_SIDE\_CHANNEL, 0);

if (coid\_2==-1) {

fprintf(stderr, "Connection error.\n");

exit(EXIT\_FAILURE);

}

sprintf(msg, "P1 send message to P2");

int msg\_res = MsgSend(coid\_2, msg, strlen(msg) + 1, rmsg, BUFFER\_LENGTH\*sizeof(char)); //sizeof(bytes)

if (msg\_res < 0) {

msg\_res = errno;

fprintf(stderr, "P1> Error MsgSend\n");

fprintf(stderr, "P1> Error code:%i\n", msg\_res);

exit(EXIT\_FAILURE);

}

printf("P1> Server replied: %s\n", rmsg);

//17 step Sending to P3 message "P1 send message to P3"

int coid\_3 = ConnectAttach(ND\_LOCAL\_NODE, pid\_3, chidP3, \_NTO\_SIDE\_CHANNEL, 0);

if (coid\_3==-1) {

fprintf(stderr, "Connection error.\n");

exit(EXIT\_FAILURE);

}

sprintf(msg, "P1 send message to P3");

msg\_res = MsgSend(coid\_3, msg, strlen(msg) + 1, rmsg, BUFFER\_LENGTH\*sizeof(char)); //sizeof(bytes)

if (msg\_res < 0) {

msg\_res = errno;

fprintf(stderr, "P1> Error MsgSend\n");

fprintf(stderr, "P1> Error code:%i\n", msg\_res);

exit(EXIT\_FAILURE);

}

printf("P1> Server replied: %s\n", rmsg);

printf("P1> Good by!\n");

return EXIT\_SUCCESS;

}

### M2.c

#include <stdlib.h>

#include <stdio.h>

#include <stddef.h>

#include <process.h>

#include <sys/neutrino.h>

#include <sys/netmgr.h>

#include <errno.h>

const int BUFFER\_LENGTH = 30;

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

pid\_t pid\_2 = getpid();

pid\_t pid\_1 = getppid();

char \*msg = (char\*)malloc(BUFFER\_LENGTH\*sizeof(char));

char \*rmsg = (char\*)malloc(BUFFER\_LENGTH\*sizeof(char));

printf(" P2>\n");

printf(" P2> Parent pid: %i\n", pid\_1);

int chidP1 = atoi(argv[0]);

// Step 3 Creating channel - chidP2.

int chidP2 = ChannelCreate(0);

printf(" P2> pid: %i, P1 chid: %i, P2 chid: %i\n", pid\_2, chidP1, chidP2);

// Step 4 Spawning P3 process - pid\_3.

char argv0[12];

sprintf(argv0, "%i", chidP2);

int pid\_3 = spawnl(P\_NOWAITO, "/tmp/M3", argv0, NULL);

// Step 6 Sending chidP2 to P1.

int coid\_1 = ConnectAttach(ND\_LOCAL\_NODE, pid\_1, chidP1, \_NTO\_SIDE\_CHANNEL, 0);

if (coid\_1==-1) {

fprintf(stderr, "Connection error.\n");

exit(EXIT\_FAILURE);

}

printf(" P2> P2 coid: %i\n", coid\_1);

int msg\_res = MsgSend(coid\_1, &chidP2, sizeof(int), rmsg, BUFFER\_LENGTH\*sizeof(char)); //sizeof(bytes)

if (msg\_res < 0) {

msg\_res = errno;

fprintf(stderr, " P2> Error MsgSend\n");

fprintf(stderr, " P2> Error code:%i\n", msg\_res);

exit(EXIT\_FAILURE);

}

printf(" P2> Server replied \n%s\n", rmsg);

// Step 8 Receiving query from P3.

int rcvid = MsgReceive(chidP2, msg, sizeof(msg), NULL);

printf(" P2> Received, rcvid: %i\n", rcvid);

printf(" P2> Message: %s\n", msg);

// Step 9 Replying to P3. Sending pid\_1 and chidP1.

\*(int\*)((int)msg) = pid\_1;

\*(int\*)((int)msg + 4) = chidP1;

MsgReply(rcvid, EOK, msg, 8);

// Step 12 Receiving message from P1. Reply.

rcvid = MsgReceive(chidP2, msg, sizeof(msg), NULL);

printf(" P2> Received, rcvid: %i\n", rcvid);

printf(" P2> Message: %s\n", msg);

strcpy(msg, "P2 OK");

MsgReply(rcvid, EOK, msg, strlen(msg) + 1);

printf(" P2> Good luck!\n");

return EXIT\_SUCCESS;

}

### M3.c

#include <stdlib.h>

#include <stdio.h>

#include <stddef.h>

#include <process.h>

#include <sys/neutrino.h>

#include <sys/netmgr.h>

#include <errno.h>

const int BUFFER\_LENGTH = 30;

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

pid\_t pid\_3 = getpid();

pid\_t pid\_2 = getppid();

char \*msg = (char\*)malloc(BUFFER\_LENGTH\*sizeof(char));

char \*rmsg = (char\*)malloc(BUFFER\_LENGTH\*sizeof(char));

printf(" P3> P3\n");

printf(" P3> Parent pid: %i\n", pid\_2);

int chidP2 = atoi(argv[0]);

// Step 5 Creating channel - chidP3.

int chidP3 = ChannelCreate(0);

printf(" P3> pid: %i, P2 chid: %i, P3 chid: %i\n", pid\_3, chidP2, chidP3);

// Step 8 Sending "query" to P2.

int coid\_2 = ConnectAttach(ND\_LOCAL\_NODE, pid\_2, chidP2, \_NTO\_SIDE\_CHANNEL, 0);

if (coid\_2==-1) {

fprintf(stderr, "Connection error.\n");

exit(EXIT\_FAILURE);

}

printf(" P3> P3 coid: %i\n", coid\_2);

sprintf(msg, "pid\_1, chidP1?");

int msg\_res = MsgSend(coid\_2, msg, strlen(msg) + 1, rmsg, BUFFER\_LENGTH\*sizeof(char));

if (msg\_res < 0) {

msg\_res = errno;

fprintf(stderr, " P3> Error MsgSend\n");

fprintf(stderr, " P3> Error code:%i\n", msg\_res);

exit(EXIT\_FAILURE);

}

printf(" P3> Server replied \n");

// Step 9 Receiving from P2 pid\_1 and chidP1.

int pid\_1 = \*(int\*)((int)rmsg);

int chidP1 = \*(int\*)((int)rmsg + 4);

printf(" P3> pid: %i chidP1: %i\n", pid\_1, chidP1);

// Step 10 Sending pid\_3 and chidP3 to P1. Packing int 8 byte buffer. |0\_\_pid\_3\_\_|4\_\_chidP3\_\_|8

int coid\_1 = ConnectAttach(ND\_LOCAL\_NODE, pid\_1, chidP1, \_NTO\_SIDE\_CHANNEL, 0);

if (coid\_1==-1) {

fprintf(stderr, "Connection error.\n");

exit(EXIT\_FAILURE);

}

printf(" P3> P3 coid: %i\n", coid\_1);

\*(int\*)((int)msg) = pid\_3;

\*(int\*)((int)msg + 4) = chidP3;

msg\_res = MsgSend(coid\_1, msg, 20, rmsg, BUFFER\_LENGTH\*sizeof(char));

if (msg\_res < 0) {

msg\_res = errno;

fprintf(stderr, " P3> Error MsgSend\n");

fprintf(stderr, " P3> Error code:%i\n", msg\_res);

exit(EXIT\_FAILURE);

}

printf("P3> Server replied \n");

// Step 17 Receiving message from P1. Reply.

int rcvid = MsgReceive(chidP3, msg, BUFFER\_LENGTH, NULL);

printf(" P3> Received, rcvid: %i\n", rcvid);

printf(" P3> Message: %s\n", msg);

strcpy(msg, "P3 OK");

MsgReply(rcvid, EOK, msg, strlen(msg)+1);

printf(" P3> Have a nice day!\n");

return EXIT\_SUCCESS;

}