Министерство образования и науки Российской Федерации

Санкт-Петербургский государственный электротехнический университет

«ЛЭТИ»

Факультет компьютерных технологий и информатики

Кафедра автоматики и процессов управления

отчет

**по лабораторной работе №6**

**по дисциплине «ПСРВ»**

Тема: Отладка программ в интегрированной системе Momentix

|  |  |  |
| --- | --- | --- |
| Студент гр. 3391 |  | Малина А. |
| Преподаватель |  | Дорогов А.Ю. |

Санкт-Петербург

## 2018

## Цель работы: Освоить технологию отладки программ средствами интегрированной системы Momentix. Изучить механизм синхронизации Mutex и передачи сигналов и сообщений.

**Ход работы:**

**Упражнение 1.**

Скриншот конфигурации отладчика приведен на Рис. 1

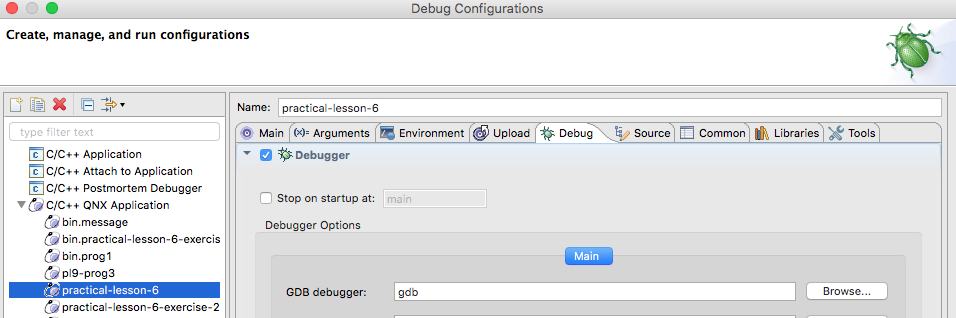


Рис. 1 Скриншот конфигурации отладчика

После запуска отладчика, открывается Debug-перспектива, скриншот представлен на Рис. 2

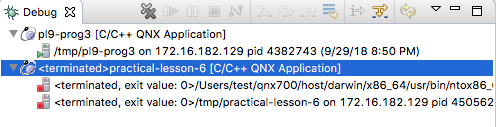
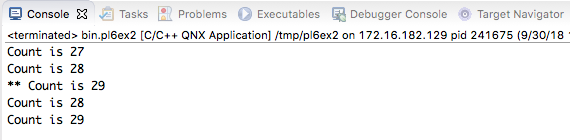


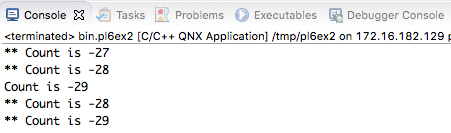
Рис. 2 Скриншот Debug-перспективы

**Упражнение 2. Механизм синхронизации Mutex**

Последние 4 строки в окне Console



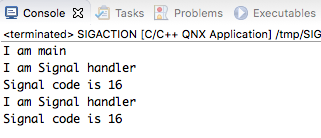
После смены значений в операторах sleep(), в консоли получены следующие последние 4 строки



Объяснение результата: в первом случае function1() выполняется в 2 раза чаще функции function2(), т.е. успевает еще раз увеличить счетчик count, перед тем как function2() его уменьшит на 1. Во втором случае function2() выполняется в 2 раза чаще function1(), поэтому значение счетчика в итоге меняет свой знак.

**Упражнение 3. Передача и прием сигналов**

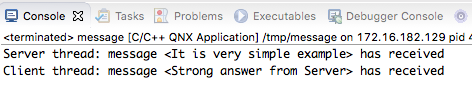
Содержимое Console после запуска программы



В функции main() выполняется оператор printf("I am Signal handler\n"), затем посылается сигнал SIGUSR1 kill( getpid(), SIGUSR1 ), его обработчик – функция handler(), в которой печатается текст printf("I am Signal handler\n"); Т.к. это ее первый запуск, то в ней посылается сигнал SIGUSR1 kill( getpid(), SIGUSR1 ) и SIGUSR2 kill( getpid(), SIGUSR2 ), но для сигнала SIGUSR2 не было назначено действие sigaction(), только для SIGUSR1, поэтому функция handler() для SIGUSR2 не вызывается.

**Упражнение 4. Передача сообщений**

Содержимое Console после запуска программы



Описание работы программы:

1) В функции main() создаются два потока - клиент и сервер, main() блокируется в ожидании завершения потока клиента.

2) Сервер создает канал, по которому будут общаться потоки клиента и сервера, засыпает на 1 секунду, таким образом дает время на инициализацию потока клиента.

3) Сервер ждет ответа от клиента, затем выводит его в консоль.

4) Сервер формирует ответ клиенту и посылает его.

5) Поток клиента при старте устанавливает соединение с сервером через оператор ConnectAttach().

6) Клинет формирует сообщение и отправляет его серверу, при этом указывается переменная, в которую будет помещен ответ от сервера.

7) Клиент выводит в консоль ответ от сервера, разрывает связь с каналом, завершает свою работу.

8) Функция main() получает управление и завершает свою работу.

**Приложение**

**Листинг программы mutex.c (Упражнение 2**)

#include <stdio.h>

#include <pthread.h>

#include <unistd.h>

pthread\_mutex\_t mutex = PTHREAD\_MUTEX\_INITIALIZER;

int count = 0;

void\* function1( void\* arg )

{

int tmp = 0;

// пока истина, выполнять цикл

while(1)

{

// захватить мьютекс

pthread\_mutex\_lock( &mutex );

// увеличить счетчик на 1

tmp = count++;

// освободить мьютекс

pthread\_mutex\_unlock( &mutex );

// вывод значения счетчика в консоль

printf( "Count is %d\n", tmp );

/\* заснуть на 1 секунду \*/

sleep(2);

}

return 0;

}

void\* function2( void\* arg )

{

int tmp = 0;

while(1) {

// захватить мьютекс

pthread\_mutex\_lock( &mutex );

// уменьшить счетчик на 1

tmp = count--;

// освободить мьютекс

pthread\_mutex\_unlock( &mutex );

// вывод значения счетчика в консоль

printf( "\*\* Count is %d\n", tmp );

/\* заснуть на 2 секунды \*/

sleep(1);

}

return 0;

}

int main()

{

// создать поток, который начнет свою работу с вызова function1(), аргумент функции - пустой

pthread\_create( NULL, NULL, &function1, NULL );

// создать поток, который начнет свою работу с вызова function1(), аргумент функции - пустой

pthread\_create( NULL, NULL, &function2, NULL );

/\* Let the threads run for 60 seconds. \*/

sleep(60);

return 0;

}

**Листинг программы sigaction.c (Упражнение 3)**

#include <stdio.h>

#include <stdlib.h>

#include <signal.h>

#include <unistd.h>

int main()

{

extern void handler();

struct sigaction act;

sigset\_t set;

sigemptyset( &set );

sigaddset( &set, SIGUSR1 );

sigaddset( &set, SIGUSR2 );

/\*

\* Define a handler for SIGUSR1 such that when

\* entered both SIGUSR1 and SIGUSR2 are masked.

\*/

act.sa\_flags = 0;

/\*

В sa\_mask задаётся маска сигналов, к

оторые должны блокироваться (т.е. добавляется к маске сигналов нити,

в которой вызывается обработчик сигнала)

при выполнении обработчика сигнала.

\*/

act.sa\_mask = set;

// sa\_handler указывает действие, которое должно быть связано с signum;

act.sa\_handler = &handler;

/\*

Системный вызов sigaction() используется для

изменения выполняемого процессом действия

при получении определённого сигнала

\*/

sigaction( SIGUSR1, &act, NULL );

printf("I am main\n");

// Посылка сигнала SIGUSR1 от одного процесса другому

kill( getpid(), SIGUSR1 );

/\* Program will terminate with a SIGUSR2 \*/

return EXIT\_SUCCESS;

}

void handler(int signo)

{

static int first = 1;

printf("I am Signal handler\n");

printf("Signal code is %i\n", signo);

if(first)

{

first = 0;

// Посылка сигнала SIGUSR1 от одного процесса другому

kill( getpid(), SIGUSR1 ); /\* Prove signal masked \*/

// Посылка сигнала SIGUSR2 от одного процесса другому

kill( getpid(), SIGUSR2 ); /\* Prove signal masked \*/

}

}

**Листинг программы massage.c (Упражнение 4)**

**#include** <pthread.h>

**#include** <stdlib.h>

**#include** <string.h>

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

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

**#include** <unistd.h>

**#include** <stdio.h>

**#define** BUFFER\_LENGTH 25

**void**\* **server**()

{

**int** chid;

**int** rcvd;

**char** receive\_buf[BUFFER\_LENGTH];

**char** reply\_buf[BUFFER\_LENGTH];

// Создать канал для обмена сообщениями

chid = **ChannelCreate**(0);

// Заснуьб на 1 секунду для того, чтобы клиент успел инициализироваться

**sleep**(1);

// Ожидать сообщение в канале chid, поместить его в строку receive\_buf

rcvd = **MsgReceive**(chid, &receive\_buf, **sizeof**(receive\_buf), NULL);

// Вывод сообщения в консоль

**printf**("Server thread: message <%s> has received \n", &receive\_buf);

// Создать ответ для клиента

**strcpy**(reply\_buf, "Strong answer from Server");

// Ответить клиенту с идентификатором rcvd, ответ - строка reply\_buf

**MsgReply**(rcvd, 1500052, &reply\_buf, **sizeof**(reply\_buf));

// Закрыть канал

**ChannelDestroy**(chid);

// Завершение работы потока

**pthread\_exit**(NULL);

}

**void**\* **client**()

{

**int** coid;

pid\_t PID;

**char** send\_buf[BUFFER\_LENGTH];

**char** reply\_buf[BUFFER\_LENGTH];

// Получить идентификатор текущего процесса.

PID = **getpid**();

// Установить соединение между процессом c идентификатором PID и каналом с идентификатором 1

coid = **ConnectAttach**(0, PID, 1, 0, 0);

// Поместить текст для отправки на сервер в строку send\_buf

**strcpy**(send\_buf, "It is very simple example");

// Отправка сообщения send\_buf по соединению с идетификатором coid, ответ хранить в reply\_buf

**MsgSend**(coid, &send\_buf, **sizeof**(send\_buf), &reply\_buf, **sizeof**(reply\_buf));

// Вывод ответа в консоль

**printf**("Client thread: message <%s> has received \n", &reply\_buf);

// Разорвать соединение

**ConnectDetach**(coid);

// Завершение работы потока

**pthread\_exit**(NULL);

}

**int** **main**()

{

pthread\_t server\_tid, client\_tid;

**pthread\_create**(&server\_tid, NULL, &server, NULL);

**pthread\_create**(&client\_tid, NULL, &client, NULL);

// Функция pthread\_join() блокирует вызывающий поток, пока указанный поток client\_tid не завершится.

**pthread\_join**(client\_tid,NULL);

**return** EXIT\_SUCCESS;

}