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 №9**

**по дисциплине «ПСРВ»**

Тема: Разработка многопоточной программы управления роботом по координатам угловых перемещений.

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

Санкт-Петербург

## 2018

## Цель работы:

Изучить и освоить системные средства создания виртуальных таймеров и способов обработки генерируемых ими событий.

Разработать программу управления роботом по координатам **W** и **F**. Робот манипулятор не имеет датчиков перемещения по этим координатам. Текущее положение робота по координате **W** и **F** можно измерить количеством элементарных временных интервалов, которое затрачивается на перемещение робота из начального положения в заданное текущее. В этом случае шаг о перемещения равен длительности элементарного временного интервала.

Для задания временного интервала и формирования события по его истечению можно использовать виртуальный таймер. Передача события от интервального периодического таймера в программный процесс выполняется с помощью сигнала или импульсного сообщения.

На следующем рисунке показан программный интерфейс трехпоточного приложения для управления роботом по координатам **X,Y,Z** и **W,F**.



Приложение рекомендуется построить на основе программы **PROG2** (см. Практическое занятие 8), дополнив ее дополнительным потоком, содержащим таймеры и счетчики событий.

По координатам **W** и **F** робот оснащен датчиками конечных положений. При достижений крайних положений (начального и конечного) робот может формировать сигнал **SIGUSR2** или импульсное сообщение. Предварительно датчики конечных положений должны быть инициализированы. Формат инициализирующих команд для передачи импульсов:

struct MESSAGE // пересылаемое сообщение send message structure

{

unsigned char type; // 0,1,2,3,4,5,6

unsigned int buf;// send data

};

* Инициализация датчика координаты **W**  type = 7, buf = номер канала,
* Инициализация датчика координаты **F** type = 8, buf = номер канала,

Если значение buf=0 , то датчики инициализируются для передачи сигналов SIGUSR2.

//Коды **(поле code)** импульсных сообщений от датчиков конечных положений по координатам **W и F**

#define W\_END 0x08

#define W\_BEGIN 0x10

#define F\_END 0x20

#define F\_BEGIN 0x40

Поле значений (поле **value**) в импульсном сообщении равно нулю, если робот находится в начальном положении по данной координате и больше нуля – если в конечном. При использовании сигналов их поля значений такие же, как и для импульсных сообщений, а поля кодов всегда отрицательны, но численно совпадают с кодами импульсов.

## Задание 3. Таймеры с импульсным выходом, датчики с импульсным выходом

В этом задании дополнительный поток ***displayWF*** должен обрабатывать импульсные сообщения от собственных таймеров, и от датчиков конечных положений эмулятора **roby**.

**Ход работы:**

1) В среде Momentix создан проект PROG3

2) В потоке main добавлен вызов phread\_create() для создания потока displayWF и displayXYZ

**pthread\_create**(NULL, NULL, &DisplayXYZ, NULL);

**pthread\_create**(NULL, NULL, &DisplayWF, NULL);

3) В потоке displayWF создан канал, используя вызов ChannelCreate() и канальные соединения для двух таймеров используя вызовы ConnectAttach(). Идентификатор канала описан глобальной переменной chidWF, так чтобы он доступен для всех потоков процесса

// Создать канал для передачи импульсов

chidWF = **ChannelCreate**(0);

**printf**("\nDisplayWF thread: created channel has chidWF = %d\n", chidWF);

4) В потоке main создано соединение с каналом потока displayWF, используя вызов ConnectAttach(). Это соединение необходимо для запуска таймеров. Соединение с каналом создано после запуска потока displayWF и создания в нем канала, поэтому после создания потока displayWF выполнена задержка исполнения потока main на одну секунду, используя системный вызов sleep().

/\*

Приостановить поток main, чтобы позволить потокам DisplayXYZ и DisplayWF

подготовиться к работе (например, инициализировать датчики движения по X, Y и Z

и датчики крайних положений по W и F эмулятора робота roby).

\*/

**sleep**(5);

pid\_t PID = **getpid**();

// Создать соединение с каналом chidWF (для запуска таймеров).

**int** coid\_DisplayWF = **ConnectAttach**(0, PID, chidWF, 0, 0);

5) В потоке main в фрагмент кода, соответствующей записи данных в порт C , добавлен анализ входной команды. Цель анализа: определить координату движения (W или/и F), установить значения глобальных переменных stateW и stateF, указывающих направление движения, и послать импульсные сообщения для запуска таймеров потоку displayWF, используя системные вызовы MsgSendPulse().

case 'C':

msg.type = 1;

msg.buf = PC;

/\*

Анализ входной комманды оператора:

определить координату движения (W или F); (stateW или stateF уже установлены)

- послать импульсные сообщения для запуска таймеров

потоку DisplayWF, используя MsgSendPulse()

\*/

if(ActiveW) // Если стала активной координата W, то

MsgSendPulse(coid\_DisplayWF, main\_pulse\_priority, TIMER\_W\_SET, 0); // запустить таймер по W

if(ActiveF) // Если стала активной координата F, то

MsgSendPulse(coid\_DisplayWF, main\_pulse\_priority, TIMER\_F\_SET, 0); // запустить таймер по F

MsgSend(coid, &msg, sizeof(msg), NULL, 0);

break;

6) В потоке displayWF заполннена структура sigevent для таймеров, используя макровызовы SIGEV\_PULSE\_INIT()

SIGEV\_PULSE\_INIT(&pulse\_event\_W, coidW, SIGEV\_PULSE\_PRIO\_INHERIT, 0, TIMER\_W\_COUNT);

SIGEV\_PULSE\_INIT(&pulse\_event\_F, coidF, SIGEV\_PULSE\_PRIO\_INHERIT, 0, TIMER\_F\_COUNT);

7) В потоке displayWF созданы виртуальные таймеры для каждой из координат, используя системные вызовы TimerCreate().

**timer\_create**(CLOCK\_REALTIME, &pulse\_event\_W, &timerW);

**timer\_create**(CLOCK\_REALTIME, &pulse\_event\_F, &timerF);

8) В потоке displayWF посылаются инициализирующие команды для конечных датчиков эмулятора roby.

// Инициализация датчиков крайних положений по координатам W и F

//Буфер сообщения msg.buf для команд инициализации датчиков содержит номер канала chidWF.

msg.buf = chidWF;

// Инициализация датчика координаты W

msg.type = 7;

**MsgSend**(coid, &msg, **sizeof**(msg), NULL, 0);

// Инициализация датчика координаты F

msg.type = 8;

**MsgSend**(coid, &msg, **sizeof**(msg), NULL, 0);

9) В потоке displayWF с помощью системного вызова MsgReceivePulse(), организован в бесконечном цикле прием импульсных сообщений по открытому каналу

**while**(1)

{

//sleep(1); //Вернуть управление потоку main (на некоторое время).

// Приём импульсных сообщений по открытому каналу chidWF

**MsgReceivePulse**(chidWF, &pulse, **sizeof**(pulse), NULL);

10) В потоке displayWF в теле бесконечного цикла используя конструкцию switch/case выполняется анализ кода импульса, управление счетчиками w\_cnt и f\_cnt, а также обрабатка импульсов от датчиков конечного положения и от потока main. Инкрементация или декрементация счетчиков положения определяется по значениям глобальных переменных stateW и stateF

// Анализ кода импульса

switch(pulse.code)

{

// Импульс от потока main: запустить таймер по W

case TIMER\_W\_SET:

ActiveW = 0;

/\*

Таймер по W будет запущен, а команду на его запуск

(для будущих импульсов) сделать неактивной,

т. е., когда работает запущенный таймер,

не нужно его снова инициализировать.

\*/

printf("\nDisplayWF thread: pulse code = %d from main thread: start timer W.", pulse.code);

/\*

В потоке DisplayWF запускать и останавливать таймеры нужно

с помощью системных вызовов TimerSettime()

\*/

/\*

timer\_settime(timer\_t timerid,

int flags,

struct itimerspec \*value,

struct itimerspec \*oldvalue), где

timerid - идентификатор таймера;

flags - флаги;

\*/

itspec.it\_value.tv\_nsec = 5000000; // 5 миллисекунд

// При flags == NULL задан относительный таймер по W

timer\_settime(timerW, 0, &itspec, &old\_itspec);

break;

// Импульс от потока main: запустить таймер по F

case TIMER\_F\_SET:

ActiveF = 0;

printf("\nDisplayWF thread: pulse code = %d from main thread: start timer F\n", pulse.code);

itspec.it\_value.tv\_nsec = 5000000; // 5 миллисекунд

timer\_settime(timerF, 0, &itspec, &old\_itspec);

break;

// Импульс от таймера по W: изменить значение счётчика w\_cnt

case TIMER\_W\_COUNT:

/\*

Инкрементация или декрементация счётчиков положения по W и F определяется

значениями глобальных переменных stateW и stateF,

т. е. направлениями движения по W и F (w\_cnt+=stateW; f\_cnt+=stateF;)

\*/

printf("\nDisplayWF thread: pulse code = %d from timer W: change w\_cnt\n", pulse.code);

w\_cnt += stateW;

break;

// Импульс от таймера по F: изменить значение счётчика f\_cnt

case TIMER\_F\_COUNT:

printf("\nDisplayWF thread: pulse code = %d from timer F: change f\_cnt\n", pulse.code);

f\_cnt += stateF;

break;

// Импульс от датчика крайнего положения по W

case W\_BEGIN: case W\_END:

printf("\nDisplayWF thread: pulse code = %d from end position on W: stop timer W\n", pulse.code);

itspec.it\_value.tv\_nsec = 0; // Для остановки таймера по W

timer\_settime(timerW, 0, &itspec, &old\_itspec); // Остановить таймер по W

stateW = 0; // Нет движения по W

break;

// Импульс от датчика крайнего положения по F

case F\_BEGIN: case F\_END:

printf("\nDisplayWF thread: pulse code = %d from end position on F: stop timer F\n", pulse.code);

itspec.it\_value.tv\_nsec = 0; // Для остановки таймера по F

timer\_settime(timerF, 0, &itspec, &old\_itspec); // Остановить таймер по F

stateF = 0; // Нет движения по F

break;

default:

// Если код импульса отрицателен, то принят системный импульс.

if(pulse.code < 0)

{

printf("\nDisplayWF thread: pulse code = %d from system.", pulse.code);

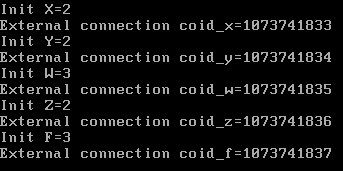
}

}

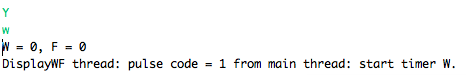
11) В потоке displayWF добавлен в тело бесконечного цикла оператор printf() для вывода значений координат W и F.

printf("\nW = %d, F = %d", w\_cnt, f\_cnt);

После запуска программы эмулятор выводит информацию о соединениях на экран



Скриншот работы программы для команды «движение по W назад» представлен ниже



Скриншот работы программы для команды «движение по W назад» после приема сигнала об остановке движения по W.



**Приложение**

**Листинг программы prog3.c**

**#include** <stdio.h>

**#include** <string.h>

**#include** <stdlib.h>

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

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

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

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

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

**#include** <fcntl.h>

**#include** "roby.h"

**#define** TIMER\_W\_SET 1

**#define** TIMER\_W\_UNSET 4

**#define** TIMER\_F\_SET 2

**#define** TIMER\_F\_UNSET 8

**#define** TIMER\_W\_COUNT 33

**#define** TIMER\_F\_COUNT 65

**#define** ATTACH\_POINT "apu/roby"

// пересылаемое сообщение

**struct** MESSAGE

{

// message type

**unsigned** **char** type;

// send data

**unsigned** **int** buf;

};

**int** coid; // Идентификатор соединения, который использует эмулятор робота Roby.

**int** chidWF; // Идентификатор канала, которого должен создать поток DisplayWF.

**int** stateW = 0; // Направления движения по координатам W и

**int** stateF = 0; // F соответственно: -1 - назад; 0 - нет движения; +1 - вперёд.

**int** ActiveW = 0, ActiveF = 0; // Активные координаты W и F: переменная ActiveW или ActiveF

//устанавливается в 1, когда нужно заставить робота Roby начать движение

//по координате W или F соответственно.

**int** main\_pulse\_priority = 10; //Приоритет импульсов, поступающих от потока main к потоку DisplayWF.

**void**\* **DisplayXYZ**()

{

**struct** MESSAGE msg;

**struct** \_pulse pulse;

**int** chid, x\_cnt, y\_cnt, z\_cnt;

x\_cnt = y\_cnt = z\_cnt = 0;

**int** changed = 0;

chid = **ChannelCreate**(0);

**printf**("Создан канал %d\n ", chid);

// Инициализация датчиков координат X,Y,Z

**for**(**int** i = 4; i < 7; i++)

{

msg.type = i;

msg.buf = chid;

**MsgSend**(coid, &msg, **sizeof**(msg), NULL, 0);

}

**for**(;;)

{

**MsgReceivePulse**(chid, &pulse, **sizeof**(pulse), NULL);

**switch**(pulse.code)

{

**case** B\_X: // Если изменилась координата X.

x\_cnt = pulse.value.sival\_int;

changed = 1;

**break**;

**case** B\_Y: // Если изменилась координата Y.

y\_cnt = pulse.value.sival\_int;

changed = 1;

**break**;

**case** B\_Z: // Если изменилась координата Z.

z\_cnt = pulse.value.sival\_int;

changed = 1;

**break**;

**default**: // Если изменились другие координаты.

changed = 0;

**break**;

}

**if**(changed)

{

// Вывод текущих координат на экран.

**printf**("Roby's coordinates:\r\n\tX=%d\r\n\tY=%d\r\n\tZ=%d\r\n",

x\_cnt,

y\_cnt,

z\_cnt);

}

}

}

**void**\* **DisplayWF**()

{

**struct** MESSAGE msg; // Посылаемое сообщение.

//unsigned char rmsg; // Буфер ответного сообщения.

**struct** \_pulse pulse;

**unsigned** **int** w\_cnt = 0, f\_cnt = 0; // Счётчики положения по W и F соответственно.

// Создать канал для передачи импульсов

chidWF = **ChannelCreate**(0);

**printf**("\nDisplayWF thread: created channel has chidWF = %d\n", chidWF);

// Создать канальное соединение для двух таймеров по каналу chidWF.

pid\_t PID = **getpid**();

**int** coidW = **ConnectAttach**(0, PID, chidWF, 0, 0);

**int** coidF = **ConnectAttach**(0, PID, chidWF, 0, 0);

/\*

Заполнить структуру sigevent для таймеров,

используя макровызовы SIGEV\_PULSE\_INIT()

\*/

**struct** sigevent pulse\_event\_W, pulse\_event\_F; //Структуры уведомлений для таймеров по W и по F.

/\*

Синтаксис для макроса SIGEV\_PULSE\_INIT()

SIGEV\_PULSE\_INIT(eventp, coid, priority, code, value), где

eventp - указатель на заполняемую структуру sigevent,

при этом eventp соответствует полю sigev\_notify в структуре struct sigevent

и задаёт способ уведомления (в данном случае SIGEV\_PULSE - импульсы);

coid - идентификатор соединения,

priority - приоритет, например, значение priority = SIGEV\_PULSE\_PRIO\_INHERIT

предотвращает изменение приоритета принимающего потока;

code - код импульса,

value - значение импульса.

\*/

SIGEV\_PULSE\_INIT(&pulse\_event\_W, coidW, SIGEV\_PULSE\_PRIO\_INHERIT, 0, TIMER\_W\_COUNT);

SIGEV\_PULSE\_INIT(&pulse\_event\_F, coidF, SIGEV\_PULSE\_PRIO\_INHERIT, 0, TIMER\_F\_COUNT);

/\*

Создать виртуальные таймеры для координат W и F,

используя системные вызовы TimerCreate()

\*/

/\*

Синтаксис для timer\_create()

timer\_create(clockid\_t clock\_id, struct sigevent \*event, timer\_t \*timer\_id), где

clock\_id - тип таймера: CLOCK\_REALTIME, CLOCK\_SOFTTIME, CLOCK\_MONOTONIC;

\*event - указатель на структуру sigevent;

\*timer\_id - указатель на идентификатор таймера.

\*/

timer\_t timerW, timerF; // Идентификаторы таймеров по W и F.

**timer\_create**(CLOCK\_REALTIME, &pulse\_event\_W, &timerW);

**timer\_create**(CLOCK\_REALTIME, &pulse\_event\_F, &timerF);

/\*

Послать инициализирующие команды для датчиков крайних положений

по W и F к Roby, используя глобальную переменную coid

\*/

// Инициализация датчиков крайних положений по координатам W и F

//Буфер сообщения msg.buf для команд инициализации датчиков содержит номер канала chidWF.

msg.buf = chidWF;

// Инициализация датчика координаты W

msg.type = 7;

**MsgSend**(coid, &msg, **sizeof**(msg), NULL, 0);

// Инициализация датчика координаты F

msg.type = 8;

**MsgSend**(coid, &msg, **sizeof**(msg), NULL, 0);

// Установка значений временных интервалов

/\*

Новая и старая спецификации интервальных

таймеров соответственно.

\*/

**struct** itimerspec itspec, old\_itspec;

/\*

Задержка перед однократной (первой) генерацией

события - это время (для относительного таймера - относительно

текущего момента времени), после истечения которого таймер

запустится на генерирование событий.

\*/

itspec.it\_value.tv\_sec = 0;

/\*

Задаётся интервал для каждой циклической перезагрузки таймера.

Данные значения для структуры itspec соответствуют таймеру с интервалом

генерации событий, равным 5 миллисекундам.

\*/

itspec.it\_interval.tv\_sec = 0;

itspec.it\_interval.tv\_nsec = 5000000;

**while**(1)

{

//sleep(1); //Вернуть управление потоку main (на некоторое время).

// Приём импульсных сообщений по открытому каналу chidWF

**MsgReceivePulse**(chidWF, &pulse, **sizeof**(pulse), NULL);

// Анализ кода импульса

**switch**(pulse.code)

{

// Импульс от потока main: запустить таймер по W

**case** TIMER\_W\_SET:

ActiveW = 0;

/\*

Таймер по W будет запущен, а команду на его запуск

(для будущих импульсов) сделать неактивной,

т. е., когда работает запущенный таймер,

не нужно его снова инициализировать.

\*/

**printf**("\nDisplayWF thread: pulse code = %d from main thread: start timer W.", pulse.code);

/\*

В потоке DisplayWF запускать и останавливать таймеры нужно

с помощью системных вызовов TimerSettime()

\*/

/\*

timer\_settime(timer\_t timerid,

int flags,

struct itimerspec \*value,

struct itimerspec \*oldvalue), где

timerid - идентификатор таймера;

flags - флаги;

\*/

itspec.it\_value.tv\_nsec = 5000000; // 5 миллисекунд

// При flags == NULL задан относительный таймер по W

**timer\_settime**(timerW, 0, &itspec, &old\_itspec);

**break**;

// Импульс от потока main: запустить таймер по F

**case** TIMER\_F\_SET:

ActiveF = 0;

**printf**("\nDisplayWF thread: pulse code = %d from main thread: start timer F\n", pulse.code);

itspec.it\_value.tv\_nsec = 5000000; // 5 миллисекунд

**timer\_settime**(timerF, 0, &itspec, &old\_itspec);

**break**;

// Импульс от таймера по W: изменить значение счётчика w\_cnt

**case** TIMER\_W\_COUNT:

/\*

Инкрементация или декрементация счётчиков положения по W и F определяется

значениями глобальных переменных stateW и stateF,

т. е. направлениями движения по W и F (w\_cnt+=stateW; f\_cnt+=stateF;)

\*/

**printf**("\nDisplayWF thread: pulse code = %d from timer W: change w\_cnt\n", pulse.code);

w\_cnt += stateW;

**break**;

// Импульс от таймера по F: изменить значение счётчика f\_cnt

**case** TIMER\_F\_COUNT:

**printf**("\nDisplayWF thread: pulse code = %d from timer F: change f\_cnt\n", pulse.code);

f\_cnt += stateF;

**break**;

// Импульс от датчика крайнего положения по W

**case** W\_BEGIN: **case** W\_END:

**printf**("\nDisplayWF thread: pulse code = %d from end position on W: stop timer W\n", pulse.code);

itspec.it\_value.tv\_nsec = 0; // Для остановки таймера по W

**timer\_settime**(timerW, 0, &itspec, &old\_itspec); // Остановить таймер по W

stateW = 0; // Нет движения по W

**break**;

// Импульс от датчика крайнего положения по F

**case** F\_BEGIN: **case** F\_END:

**printf**("\nDisplayWF thread: pulse code = %d from end position on F: stop timer F\n", pulse.code);

itspec.it\_value.tv\_nsec = 0; // Для остановки таймера по F

**timer\_settime**(timerF, 0, &itspec, &old\_itspec); // Остановить таймер по F

stateF = 0; // Нет движения по F

**break**;

**default**:

// Если код импульса отрицателен, то принят системный импульс.

**if**(pulse.code < 0)

{

**printf**("\nDisplayWF thread: pulse code = %d from system.", pulse.code);

}

}

/\*

Вывод значений координат W и F, а также кодов

и значений принимаемых импульсов

\*/

**printf**("\nW = %d, F = %d", w\_cnt, f\_cnt);

}

**ChannelDestroy**(chidWF);

**pthread\_exit**(NULL);

}

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

{

**unsigned** **char** PA=0, PC=0; //Эти переменные хранят значения регистров PA и PC

**struct** MESSAGE msg; // буфер посылаемого сообщения

**unsigned** **char** rmsg; // Буфер ответного сообщения

**char** command, ch[3];

coid = **name\_open**(ATTACH\_POINT, 0);

**if**(coid == -1)

{

**return** EXIT\_FAILURE;

}

**printf**("apu/roby has coid = %d\n", coid);

**pthread\_create**(NULL, NULL, &DisplayXYZ, NULL);

**pthread\_create**(NULL, NULL, &DisplayWF, NULL);

/\*

Приостановить поток main, чтобы позволить потокам DisplayXYZ и DisplayWF

подготовиться к работе (например, инициализировать датчики движения по X, Y и Z

и датчики крайних положений по W и F эмулятора робота roby).

\*/

**sleep**(5);

pid\_t PID = **getpid**();

// Создать соединение с каналом chidWF (для запуска таймеров).

**int** coid\_DisplayWF = **ConnectAttach**(0, PID, chidWF, 0, 0);

**do**

{

**sleep**(1); //Передать управление потоку DisplayXYZ или потоку DisplayWF (на некоторое время)

**printf**("Enter command>\n");

**scanf**("%s", ch);

**switch**(ch[0])

{

**case** 'I':

// Двигаться в начальное положение по всем координатам

// и сбросить значения датчиков положений.

PA = 0; PC = 0; //Для этого сбросить в 0 переменные PA и PC.

command ='C'; //Подать команду записи в регистр PC роботу Roby.

**break**;

**case** '+': **case** '=':

command = 'c';

**break**;

**case** '!':

command = 'B';

**break**;

**case** 'Y':

**scanf**("%s", ch);

**switch**(ch[0])

{

**case** 'X':

PA ^= A\_X\_FORWARD;

command ='A';

**break**;

**case** 'x':

PA ^= A\_X\_BACK;

command ='A';

**break**;

**case** 'Y':

PA ^= A\_Y\_FORWARD;

command ='A';

**break**;

**case** 'y':

PA ^= A\_Y\_BACK;

command ='A';

**break**;

**case** 'Z':

PA ^= A\_Z\_FORWARD;

command ='A';

**break**;

**case** 'z':

PA ^= A\_Z\_BACK;

command ='A';

**break**;

**case** 'F':

PC ^= C\_F\_FORWARD;

command ='C';

ActiveF = 1; // F - активная координата.

**if**(stateF == 1) //Если Roby движется вперёд по F,

stateF = 0; //то стоп по F.

**else**

stateF = 1; // Иначе запустить движение вперёд по F.

**break**;

**case** 'f':

PC ^= C\_F\_BACK;

command ='C';

ActiveF = 1; // F - активная координата.

**if**(stateF == -1) // Если Roby движется назад по F,

stateF = 0; //то стоп по F.

**else**

stateF = -1; // Иначе запустить движение назад по F.

**break**;

**case** 'W':

PC ^= C\_W\_FORWARD;

command ='C';

ActiveW = 1; // W - активная координата.

**if**(stateW == 1 ) // Если Roby двигался вперёд по W,

stateW = 0; // то остановить его по W.

**else**

stateW = 1; // Иначе запустить движение вперёд по W.

**break**;

**case** 'w':

PC ^= C\_W\_BACK;

command ='C';

ActiveW = 1; //W - активная координата.

**if**(stateW == -1) // Если Roby двигался назад по W,

stateW = 0; // то остановить его по W.

**else**

stateW = -1; // Иначе запустить движение назад по W.

**break**;

**case** 'S':

PA ^= A\_S;

command ='A';

**break**;

**case** 'D':

PA ^= A\_D;

command ='A';

**break**;

}

}

**switch**(command)

{

**case** 'A':

msg.buf = PA;

msg.type = 0;

**MsgSend**(coid, &msg, **sizeof**(msg), NULL, 0);

**break**;

**case** 'B':

msg.type = 3;

**MsgSend**(coid, &msg, **sizeof**(msg), &rmsg, **sizeof**(rmsg));

**printf**("Port B: 0x%X\n", rmsg);

**break**;

**case** 'c':

msg.type = 2;

**MsgSend**(coid, &msg, **sizeof**(msg), &rmsg, **sizeof**(rmsg));

**printf**("Port C: 0x%X\n", rmsg);

**break**;

**case** 'C':

msg.type = 1;

msg.buf = PC;

/\*

Анализ входной комманды оператора:

определить координату движения (W или F); (stateW или stateF уже установлены)

- послать импульсные сообщения для запуска таймеров

потоку DisplayWF, используя MsgSendPulse()

\*/

**if**(ActiveW) // Если стала активной координата W, то

**MsgSendPulse**(coid\_DisplayWF, main\_pulse\_priority, TIMER\_W\_SET, 0); // запустить таймер по W

**if**(ActiveF) // Если стала активной координата F, то

**MsgSendPulse**(coid\_DisplayWF, main\_pulse\_priority, TIMER\_F\_SET, 0); // запустить таймер по F

**MsgSend**(coid, &msg, **sizeof**(msg), NULL, 0);

**break**;

**default**:

**printf**("default last\n");

**break**;

}

} **while**(1);

}

**Листинг заголовочного файла roby.h**

**#ifndef** ROBY\_H\_

**#define** ROBY\_H\_

**#define** A\_D 0x01

**#define** A\_S 0x02

**#define** A\_X\_FORWARD 0x04

**#define** A\_X\_BACK 0x08

**#define** A\_Z\_BACK 0x10

**#define** A\_Z\_FORWARD 0x20

**#define** A\_Y\_BACK 0x40

**#define** A\_Y\_FORWARD 0x80

**#define** B\_X 0x01

**#define** B\_Y 0x02

**#define** B\_Z 0x04

**#define** B\_W\_END 0x08

**#define** B\_W\_BEGIN 0x10

**#define** B\_Z\_BEGIN 0x20

**#define** B\_Y\_BEGIN 0x40

**#define** B\_X\_BEGIN 0x80

**#define** C\_F\_END 0x04

**#define** C\_F\_BEGIN 0x08

**#define** C\_W\_FORWARD 0x10

**#define** C\_W\_BACK 0x20

**#define** C\_F\_FORWARD 0x40

**#define** C\_F\_BACK 0x80

**#define** X\_MIN 0

**#define** X\_MAX 1024

**#define** Y\_MIN 0

**#define** Y\_MAX 1024

**#define** Z\_MIN 0

**#define** Z\_MAX 1024

**#define** W\_MIN 0

**#define** W\_MAX 100

**#define** F\_MIN 0

**#define** F\_MAX 50

**#define** S\_MIN 0

**#define** S\_MAX 1

**#define** D\_MIN 0

**#define** D\_MAX 1

**#endif** /\* ROBY\_H\_ \*/