МИНОБРНАУКИ РОССИИ

Государственное образовательное учреждение высшего профессионального Образования

“Санкт-Петербургский государственный электротехнический университет “ЛЭТИ” им. В.И.Ульянова (Ленина)” (СПбГЭТУ)

Кафедра «Автоматики и процессов управления»

Курсовая работа

Программа управления роботом-манипулятором

Выполнили:
студенты гр. 8391
ФКТИ
Быстрова Н.А.
Спиридонов Р.Е.
Спиридонова Д.В.
Хусаинова Э.В.

Проверил:
Дорогов А.Ю.

Санкт-Петербург
2013 г.

Оглавление

1. Задание на разработку программы

ВАРИАНТ 1

задания на разработку программы управления роботом

1. Способ передачи события, связанного с изменением состояния импульсного датчика по координатам X,Y,Z - импульсное сообщение.

2. Способ передачи события, связанного с изменением состояния датчиков конечных положений по координатам W,F - импульсное сообщение.

3. Источник событий для счетчика шагов по F,W – виртуальный таймер генерирующий импульсные сообщения.

4. Способ передачи параметров в порожденный поток - через глобальные переменные.

5. Способ хранения текущего значения регистров робота - локально в одном из потоков.

Программа управления роботом-манипулятором Пульт-1

Программа должна обеспечивать следующие функции:

1. Прием команд оператора вводимых с клавиатуры. Их исполнение.

2. Управление роботом манипулятором в соответствии с командами оператора.

3. Непрерывное отображение состояния робота на экране монитора.

Команды оператора

Для ввода команд оператора используются функциональные клавиши Esc,Enter,F1,F2,...,F12.

К клавиши F1,F2,...,F12 используются для ввода команд управления роботом на основе следующего протокола:

- первое нажатие клавиши задает соответствующую команду,

- повторное нажатие клавиши прекращает действие команды.

Команды управления роботом

Клавиша	Описание команды	
	Первое нажатие	Повторное нажатие
F1	Двигаться по X вперед	Остановить движение по X вперед
F2	Двигаться по X назад	Остановить движение по X назад
F3	Двигаться по Y вперед	Остановить движение по Y вперед
F4	Двигаться по Y назад	Остановить движение по Y назад
F5	Двигаться по Z вперед	Остановить движение по Z вперед

F6	Двигаться по Z назад	Остановить движение по Z назад
F7	Двигаться по F вперед	Остановить движение по F вперед
F8	Двигаться по F назад	Остановить движение по F назад
F9	Двигаться по W вперед	Остановить движение по W вперед
F10	Двигаться по W назад	Остановить движение по W назад
F11	Включить схват S	Выключить схват S
F12	Включить дрель D	Выключить дрель D

Enter Двигаться в начальное положение по всем координатам и сбросить значения датчиков положений.

Команды программы

Клавиша	Описание команды
Esc	Завершение программы

Управление роботом манипулятором

Управление роботом выполняется в соответствии с программным интерфейсом, приведенным в руководстве к практическому занятию 9, и требованиями конкретного варианта задания.

Отображение информации о текущем состоянии робота

Отображаемые данные о состоянии робота:

1. Положение по координате X
2. Положение по координате Y
3. Положение по координате Z
4. Положение по координате F
5. Положение по координате W
6. Состояние датчика начального положения по X (включен/выключен)
7. Состояние датчика начального положения по Y (включен/выключен)
8. Состояние датчика начального положения по Z (включен/выключен)
9. Состояние датчика начального положения по F (включен/выключен)
10. Состояние датчика конечного положения по F (включен/выключен)
11. Состояние датчика начального положения по W (включен/выключен)
12. Состояние датчика конечного положения по W (включен/выключен)
13. Состояние схвата S (включен/выключен)

14.Состояние дрели D (включена/выключена)

Отображение координат положения - непрерывное при движении.

Отображение состояния датчиков - по запросу (клавише «+»).

Требования к реализации программы

1. Программа должна обеспечивать минимизацию ошибки определения текущего положения робота по всем координатам.

2. Программа должна отображать в краткой форме назначение функциональных клавиш, используемых для ввода команд.

3. При запуске программа должна выполнить:

- определить и отобразить состояние робота,

- если робот находится не в начальном положении - автоматически выполнить команду Enter.

4. При достижении конечных положений робота программа должна обеспечивать автоматическое прекращение действия команд, инициировавших движение. Сброс датчиков не выполнять.

2. Описание эмулятора робота

Задание курсовой работы связано с разработкой прикладной программы для управления программной моделью технологического робота (далее эмулятор робота). Реальным прототипом эмулятора является учебный робототехнический комплекс (УРТК), который в течение нескольких лет использовался для выполнения лабораторных работ по курсу Системное программное обеспечение. Робототехнический комплекс состоит из манипулятора, блока управления, устройства ввода-вывода информации и управляющей ЭВМ (Рис.1). Эмулятор робота в режиме реального времени моделирует работу манипулятора робота, блока внешнего управления и интерфейсного контроллера.

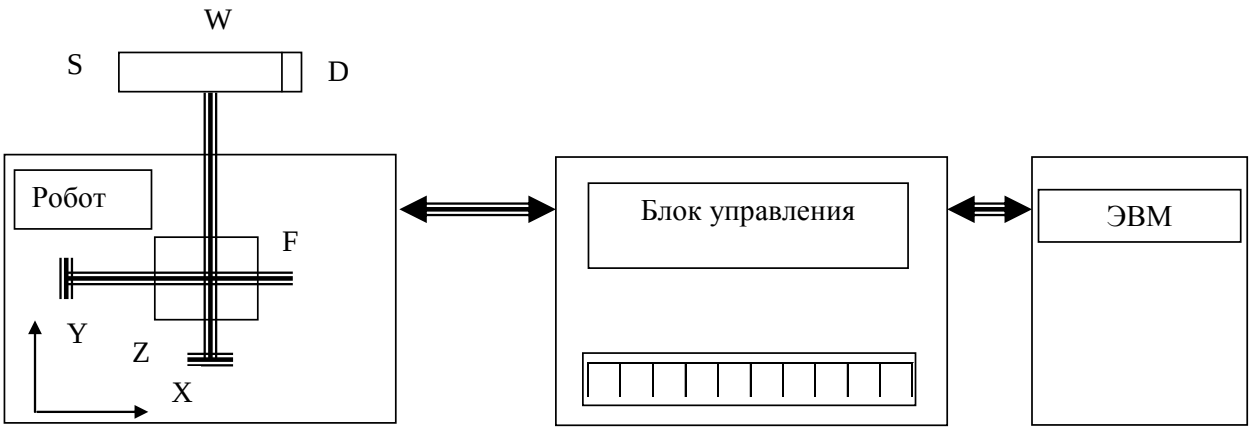


Рисунок 1. Структурная схема УРТК. X,Y,Z - координаты линейного перемещения, F - поворот основания, W - поворот головки, S - управление схватом, D - управление двигателем.

Манипулятор. Манипулятор представляет собой устройство из трех взаимно-перпендикулярных ходовых винтов, установленных на подвижном основании и

поворотной головки. Привод на ходовые винты и поворотную головку осуществляется от электродвигателей постоянного тока со встроенным редуктором. Поворотная головка оборудована схватом и двигателем, имитирующим привод сверлильного станка. УРТК позволяет имитировать работу обрабатывающих и транспортно-складских устройств.

Блок управления. Блок управления и устройство ввода-вывода информации представляет собой электронное устройство, которое позволяет осуществлять работу манипулятора в режиме ручного и автоматического управления. Управление УРТК в режиме ручного управления осуществляется с клавиатуры блока управления, а в режиме автоматического от ЭВМ. В режиме автоматического управления для определения текущего положения каретки манипулятора используются фотодатчики импульсного типа, установленные по координатам **X,Y,Z**. Импульсы датчика порождаются вращением 6-ти лепестковой крыльчатки, расположенной на валу винтовой пары. ЭВМ осуществляет подсчет импульсов поступивших от датчика с момента начала движения, что позволяет с высокой точностью определить текущее положение каретки манипулятора.

Для координат **X,Y,Z** существуют герконовые датчики начального положения. Для координат **W,F** - импульсные датчики перемещения отсутствуют, есть только герконовые датчики начального и конечного положения.

Контроллер робота-манипулятора. Контроллер робота построен на основе программируемой микросхемы 580ВВ55 (рис.2). Микросхема предназначена для организации обмена 8-ми битовыми данными и содержит три независимых регистра.

Рисунок 2. Функциональная схема контроллера робота.

Регистры контроллера настроены на выполнение следующих операций:

- регистр A[0...7] - на вывод данных;
- регистр C[0...3] -на ввод данных;
- регистр C[4...5] -на вывод данных;
- регистр B[0...7] -на ввод данных.

3. Структурная схема программы на уровне программных потоков

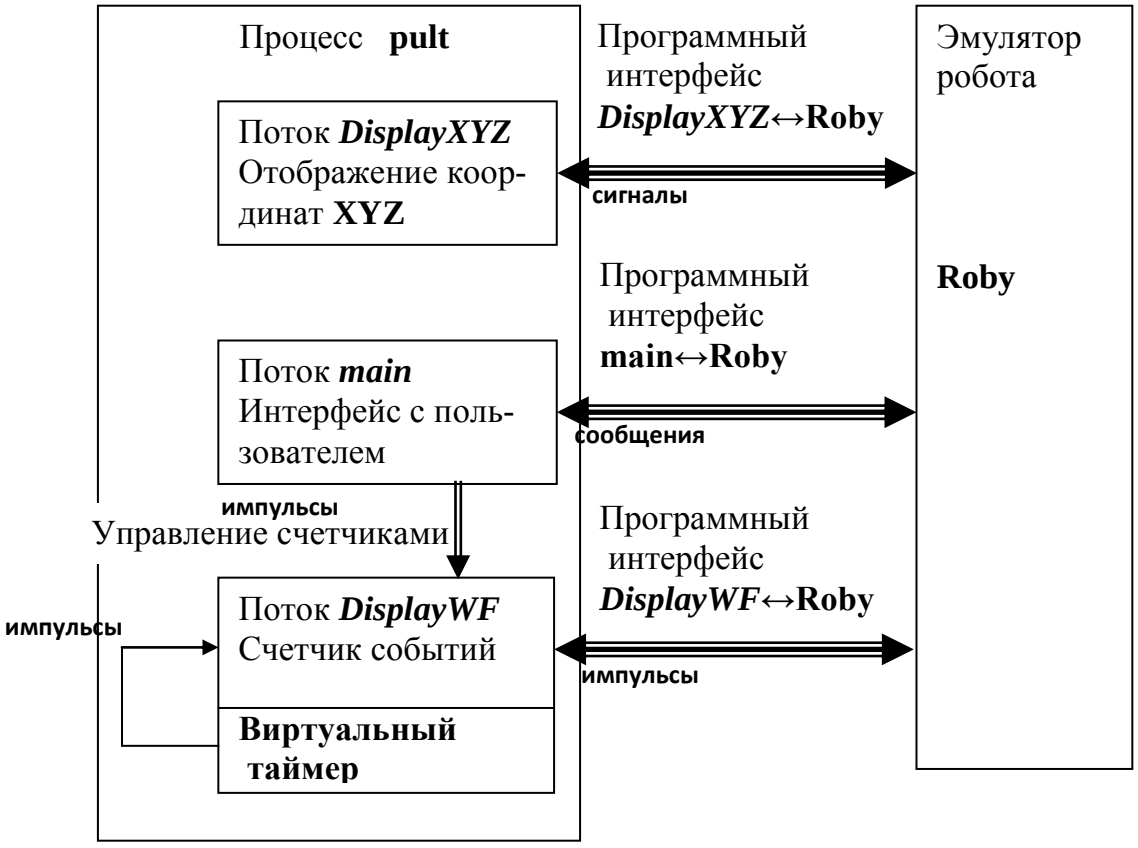


Рисунок 3. Программный интерфейс трехпоточного приложения для управления роботом по координатам X, Y, Z, W, F.

4. Описание алгоритма функционирования программных потоков с указанием их характеристик

4.1. Приоритеты потоков

Ни один поток при разработке программы не был выделен по приоритету. Каждый поток имеет одинаковый приоритет больше нуля, наследуемый от потока Main.

При просмотре характеристик потоков в системе QNX через представление QNX System Information в Momentics IDE видим, что все три потока программы CourseWork работают с приоритетом 10r, где r показывает дисциплину диспетчеризации потоков «циклическая». При данной дисциплине поток исполняется в течение некоторого кванта времени и передает управление следующеу потоку с таким же приоритетом.

При разработке был расчет на то, что потоки сами уступают остальным потокам право выполнения, что обеспечивается функциями sleep(…) во время инициализации датчиков и установления каналов связи. Диспетчеризация FIFO в данном случае не подходит, так как поток main принимает все управление с клавиатуры, ожидая команды от пользователя, что привело бы к недееоспособности других потоков во время этого ожидания. Дисциплина round или other могут решить эту проблему и возможность непрерывного управления роботом равномерно распределится между выполнением потоков программы.

4.2. Возможные состояния и причины их изменения в процессе сеанса работы программы

В процессе сеанса работы потоки могут находиться в одном из состояний:

Ready – готов к выполнению;

Running – исполняется;

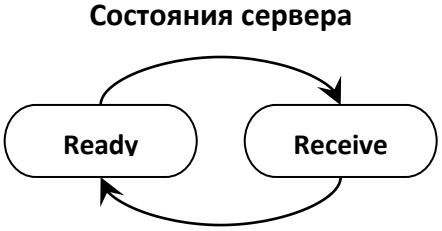
Blocked – заблокирован.

Блокировка потока может происходить в случае самостоятельной передачи управления (простаивание sleep()) другим потокам, либо в случае срабатывания дисциплины диспетчеризации, либо при пересылке сообщений. Пересылка импульсов или сигналов – это виды асинхронного уведомления потоков, не вызывающие их блокировки.

Рассмотрим состояния, в которых потоки находятся при пересылке сообщений.

Существует поток-клиент, посылающий сообщение (запрос) потоку-серверу. При этом поток-сервер должен вернуть ответ.

Первоначально сервер ждёт от кого-нибудь сообщение и находится в так называемом состоянии блокировки по приёму (Receive-blocked).



При получении сообщения сервер переходит в состояние готовности (Ready) и становится способен выполнять работу. Если сервер имеет наивысший приоритет среди готовых к исполнению (Ready) в данный момент потоков, то он получает процессор и обрабатывает поступившее сообщение, после чего сервер сможет дать некоторый ответ клиенту.

Клиент работает самостоятельно (Running), пока не посылает сообщение. Тогда клиент переключится в состояние блокировки в зависимости от состояния сервера: Send-blocked – блокировка по передаче или Receive-blocked – блокировка по приёму.

Состояние Send-blocked возникнет у клиента, если:

клиент ждёт, пока сервер занимается чем-то другим (обработкой другого сообщения и т. д.), но не приёмом сообщения от данного клиента.

Клиент переходит в состояние Reply-blocked, когда:

сервер вернётся к приёму сообщения от данного клиента, тогда ожидающий ответа клиент переходит в состояние ожидания ответа (Reply-blocked).

Клиент может сравнительно долго находиться в одном из блокированных состояний, но может быстро получить ответ от свободного сервера и вернуться в состояние готовности (Ready). Смена состояний клиента при передаче сообщений определяется сменой состояний сервера.

5. Текст программы с комментариями

Файл declarations.h

```
#ifndef DECLARATIONS_H_
#define DECLARATIONS_H_

#include <cstdlib>
#include <iostream>
#include <pthread.h>
#include <process.h>
#include <unistd.h>
#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 <sys/siginfo.h>
#include <termios.h>

#define NSEC 5000000
//#define INT_NSEC 165000000
//#define INT_NSECF 165000000
#define INT_NSEC 5036898//интервал для таймера W – для 2000 единиц по шкале
#define INT_NSECF 7517070//интервал для таймера F – для 1000 единиц по шкале

#endif /* DECLARATIONS_H_ */
```

Файл roby.h

```
#ifndef ROBY_H_
#define ROBY_H_

//Биты регистра A
#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

//Биты регистра B
#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

//Биты регистра C
#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

//Биты сообщения концевиков
#define TIMER_W_SET           1
#define     TIMER_F_SET       2

#define W_END       0x08
#define W_BEGIN     0x10
#define F_END       0x20
#define F_BEGIN     0x40
//Коды импульсов таймера
#define     TIMER_W_COUNT 33 // Таймер координаты W
#define     TIMER_F_COUNT 65 // Таймер координаты F

#endif /* ROBY_H_ */
```

Файл CourseWork.cc

```
#include "declarations.h" //Объявления библиотек
#include "roby.h"//Макросы для робота
int coid_roby;//connection id
int chidWF;//channel id WF
int stateW;//Ин- Де- кремент для W
int stateF;//Ин- Де- кремент для F
pid_t pid;//Process ID
int displayWF;//идентификатор канала связи с потоком displayWF
int x,y,z;//Счетчики для координат X,Y,Z
//Структура для пересылаемого сообщения
struct MESSAGE {
	unsigned char type;
	unsigned int buf;
};

struct MESSAGE Amsg; // Декларация структуры для записи в регистр A
struct MESSAGE Cmsg; // Декларация структуры для записи в регистр C
int f_cnt; //Счетчик координаты f
int w_cnt; //Счетчик координаты w

//Перевод терминала в режим нередактируемого ввода
int raw(int fd) {
	struct termios termios_p;
	if (tcgetattr(fd, &termios_p))
		return (-1);
	termios_p.c_cc[VMIN] = 1;
	termios_p.c_cc[VTIME] = 0;
	termios_p.c_lflag &= ~(ECHO | ICANON | ISIG | ECHOE | ECHOK | ECHONL);
	termios_p.c_oflag &= ~(OPOST);
	return (tcsetattr(fd, TCSADRAIN, &termios_p));
}

//Перевод терминала в режим редактируемого ввода
int unraw(int fd) {
	struct termios termios_p;
	if (tcgetattr(fd, &termios_p))
		return (-1);
	termios_p.c_lflag |= (ECHO | ICANON | ISIG | ECHOE | ECHOK | ECHONL);
	termios_p.c_oflag |= (OPOST);
	return (tcsetattr(fd, TCSADRAIN, &termios_p));
}

//Поток отображения координат XYZ
void* DisplayXYZ(void* arg) {
	int chid;//идентификатор канала связи Roby->DisplayXYZ
	x = -1, y = -1, z = -1;//Переменны для отображения координат
	struct MESSAGE msg;//Сообщение для инициализации датчиков
	struct _pulse pulse;//Структура для приема импульсов от Roby
	chid = ChannelCreate(NULL);//Создание коммуникационного канала
	msg.buf = chid;//Сохранение id канала в сообщение инициализации
	int st = 0;//Статус отправки сообщения
	for (int j = 4; j < 7; ++j) {//Цикл инициализации датчиков XYZ
		msg.type = j;//Тип сообщения для соответствующего датчика
		printf("\nXYZ - before %d", j);
		st = MsgSend(coid_roby, &msg, sizeof(msg), 0, 0);//отправка
		printf("\nXYZ - after %d, st = %d", j, st);
	}
	printf("\nXYZ - initialized");
	while (1) {//Основной цикл приема сообщений
		MsgReceivePulse(chid, &pulse, sizeof(pulse), NULL);//Прием
импульса от Roby
		switch (pulse.code) {//Разбор импульса
```

```
                case B_X:
                        x = pulse.value.sival_int;
                        break;
                case -2://Для движения по Y вперед
                case B_Y:
                        y = pulse.value.sival_int;
                        break;
                case B_Z:
                        z = pulse.value.sival_int;
                        break;
                }
                printf("\n\rx = %d y = %d z = %d ", x, y, z);//Вывод текущего
положения
                //Обнуление битов управления для остановки движения в конечных
положениях
                if (x == 0)
                        Amsg.buf &= ~A_X_BACK;
                else if (x == 1024)
                        Amsg.buf &= ~A_X_FORWARD;
                else if (y == 0)
                        Amsg.buf &= ~A_Y_BACK;
                else if (y == 1024)
                        Amsg.buf &= ~A_X_FORWARD;
                else if (z == 0)
                        Amsg.buf &= ~A_Z_BACK;
                else if (z == 1024)
                        Amsg.buf &= ~A_Z_FORWARD;
        }//end while
        pthread_exit(NULL);
}

//Поток отображения координат WF
void* DisplayWF(void *arg) {
        struct MESSAGE msg;//Структура для инициализации
        struct _pulse pulse;//Структура для импульсных сообщений
        struct _itimer it, old_it;//Структуры для параметров таймера W
        struct _itimer itF, old_itF;//Структуры для параметров таймера F
        //Инициализация параметров таймеров
        it.nsec = NSEC;//время "заводки" таймера
        it.interval_nsec = INT_NSEC;//интервал повторного запуска таймера
        itF.nsec = NSEC;
        itF.interval_nsec = INT_NSECF;
        chidWF = ChannelCreate(0);//Создание канала displayWF-Roby-Main
        printf("channel has chid = %d \n", chidWF);
        int coidW = ConnectAttach(0, pid, chidWF, 0, 0);//соединения для событий
таймеров
        int coidF = ConnectAttach(0, pid, chidWF, 0, 0);
        struct sigevent eventW, eventF;//структуры событий
        //Макросы для привязки отправки импульсных сообщений по событиям
        SIGEV_PULSE_INIT(&eventW,coidW,SIGEV_PULSE_PRIO_INHERIT,TIMER_W_COUNT,NU
LL);
        SIGEV_PULSE_INIT(&eventF,coidF,SIGEV_PULSE_PRIO_INHERIT,TIMER_F_COUNT,NU
LL);
        //Создание таймеров-"будильников" со "звонками"-событиями
        int timer_W = TimerCreate(CLOCK_REALTIME, &eventW);
        int timer_F = TimerCreate(CLOCK_REALTIME, &eventF);
        sleep(3);
        for (int i = 7; i < 9; i++) {//Цикл инициализации датчиков WF
                msg.type = i;
                msg.buf = chidWF;
                printf("Init W sensor \n");
                MsgSend(coid_roby, &msg, sizeof(msg), 0, 0);
                printf("Init F sensor \n");
```

```
	}
	f_cnt = 500;//Условные начальные положения
	w_cnt = 1000;//в серединах шкалы
	while (1) {
		MsgReceivePulse(chidWF, &pulse, sizeof(pulse), NULL);
		switch (pulse.code) {//Разбор полученного импульса
		case TIMER_W_SET:
			//Если есть движение по координате - запуск таймера (при
NSEC) , иначе - останов (при 0)
			it.nsec = Cmsg.buf & (C_W_BACK | C_W_FORWARD) ? NSEC : 0;
			TimerSettime(timer_W, NULL, &it, &old_it);//Запись в таймер
новых параметров, запуск при nsec != 0
			break;
		case W_END:
			if (w_cnt != 2000) {//В случае необходимости
				w_cnt = 2000;//Корректировка положения на конечное
				Cmsg.buf &= ~C_W_FORWARD;//обнуление бита движения в
направлении за диапазон
			}
			stateW = 0;
			it.nsec = 0;
			TimerSettime(timer_W, NULL, &it, &old_it);//Запись в таймер
новых параметров
			break;
		case W_BEGIN:
			if (w_cnt) {//В случае необходимости
				w_cnt = 0;//Корректировка положения на начальное
				Cmsg.buf &= ~C_W_BACK;
			}
			stateW = 0;
			it.nsec = 0;//Для остановки таймера
			TimerSettime(timer_W, NULL, &it, &old_it);
			break;
		case TIMER_W_COUNT:
			w_cnt += w_cnt == 2001 || w_cnt == -1 ? 0 : stateW;
			break;
		case F_END:
			if (f_cnt != 1000) {
				f_cnt = 1000;
			}
			stateF = 0;
			itF.nsec = 0;
			TimerSettime(timer_F, NULL, &itF, &old_itF);
			break;
		case F_BEGIN:
			if (f_cnt != 0) {
				f_cnt = 0;
				Cmsg.buf &= ~C_F_BACK;
			}
			stateF = 0;
			itF.nsec = 0;
			TimerSettime(timer_F, NULL, &itF, &old_itF);
			break;
		case TIMER_F_SET:
			itF.nsec = Cmsg.buf & (C_F_BACK | C_F_FORWARD) ? NSEC : 0;
			TimerSettime(timer_F, NULL, &itF, &old_itF);
			break;
		case TIMER_F_COUNT:
			f_cnt += f_cnt == 1001 || f_cnt == -1 ? 0 : stateF;
			break;//Для отображения некорректной работы шкалы оставлены
значения,
		};//выходящие за диапазон шкалы, чтобы показать, что мы еще
движемся в этом направлении
```

```
            printf("w = %d, f = %d\n\r", w_cnt, f_cnt);
      };
      pthread_exit(NULL);
}
//Функция разбора нажатой клавиши и управления
void control(int c3) {
      switch (c3) {//Разбор 3 числа кода F1-F12
      case 80://F1 X->
            Amsg.buf &= ~A_X_BACK;//Обнуляем бит в противоположном направлении
            Amsg.buf = Amsg.buf ^ A_X_FORWARD;//Меняем состояние бита
соответств. нажатой клавише
            break;
      case 81://F2 X<-
            Amsg.buf &= ~A_X_FORWARD;//Обнуляем бит в противоположном
направлении
            Amsg.buf = Amsg.buf ^ A_X_BACK;//Меняем состояние нажатого
            break;
      case 82://F3 Y->
            Amsg.buf &= ~A_Y_BACK;
            Amsg.buf = Amsg.buf ^ A_Y_FORWARD;
            break;
      case 83://F4 Y<-
            Amsg.buf &= ~A_Y_FORWARD;
            Amsg.buf = Amsg.buf ^ A_Y_BACK;
            break;
      case 84://F5 Z->
            Amsg.buf &= ~A_Z_BACK;
            Amsg.buf = Amsg.buf ^ A_Z_FORWARD;
            break;
      case 85://F6 Z<-
            Amsg.buf &= ~A_Z_FORWARD;
            Amsg.buf = Amsg.buf ^ A_Z_BACK;
            break;
      case 86://F7 F->
            Cmsg.buf &= ~C_F_BACK;
            Cmsg.buf = Cmsg.buf ^ C_F_FORWARD;
            stateF = Cmsg.buf & C_F_FORWARD ? 1 : 0;//Устанавливаем/Снимаем
инкремент по F
            MsgSendPulse(displayWF, 10, TIMER_F_SET, 0);//Отправляем импульс
потоку displayWF
            break;
      case 87://F8 F<-
            Cmsg.buf &= ~C_F_FORWARD;
            Cmsg.buf = Cmsg.buf ^ C_F_BACK;
            stateF = Cmsg.buf & C_F_BACK ? -1 : 0;//Устанавливаем/Снимаем
декремент по F
            MsgSendPulse(displayWF, 10, TIMER_F_SET, 0);//Отправляем импульс
потоку displayWF
            break;
      case 88://F9 W->
            Cmsg.buf &= ~C_W_BACK;
            Cmsg.buf = Cmsg.buf ^ C_W_FORWARD;
            stateW = Cmsg.buf & C_W_FORWARD ? 1 : 0;
            MsgSendPulse(displayWF, 10, TIMER_W_SET, 0);
            break;
      case 89://F10 W<-
            Cmsg.buf &= ~C_W_FORWARD;
            Cmsg.buf = Cmsg.buf ^ C_W_BACK;
            stateW = Cmsg.buf & C_W_BACK ? -1 : 0;
            MsgSendPulse(displayWF, 10, TIMER_W_SET, 0);
            break;
      case 90://F11 S On/Off
            Amsg.buf ^= A_S;//Меняем состояние хвата
```

```
            break;
      case 65://F12 D On/Off
            Amsg.buf ^= A_D;//Меняем состояние дрели
            break;
      };
      if (c3 >= 86 && c3 <= 89)//Если это в регистр C
            MsgSend(coid_roby, &Cmsg, sizeof(Cmsg), NULL, NULL);
      else
            //Если это в регистр A
            MsgSend(coid_roby, &Amsg, sizeof(Amsg), NULL, NULL);
}
//Перевод робота в начальное положение
void goStart() {
      Amsg.buf = 0;//Обнуляем буферы сообщений
      Cmsg.buf = 0;//для прекращения других движений
      Amsg.buf = A_X_BACK | A_Z_BACK | A_Y_BACK;//88 -Формируем слово для
управления
      MsgSend(coid_roby, &Amsg, sizeof(Amsg), NULL, NULL);//Посылаем команду
      control(87);//F<- двигаться в начало по F
      control(89);//W<- двигаться в начало по W
      Amsg.buf = 0;//Обнуляем буферы сообщений
      Cmsg.buf = 0;//так как в начальном состоянии робот стоит на месте
}

//Вывод данных от концевиков. Аргументы - для локального хранения регистров
void showSensors(unsigned char* _regC, unsigned char* _regB) {
      struct MESSAGE msg;//Структура для сообщения
      msg.type = 2;//для считывания порта C
      MsgSend(coid_roby, &msg, sizeof(msg), _regC, sizeof(unsigned char));
      msg.type = 3;//для считывания порта B
      MsgSend(coid_roby, &msg, sizeof(msg), _regB, sizeof(unsigned char));
      printf("\n\r Sensors condition:");//Вывод значений датчиков-концевиков
      printf("\n\rXb: %c,", (*_regB) & B_X_BEGIN ? '1' : '0');
      printf(" Yb: %c,", (*_regB) & B_Y_BEGIN ? '1' : '0');
      printf(" Zb: %c,", (*_regB) & B_Z_BEGIN ? '1' : '0');
      printf(" Fb: %c,", (*_regC) & C_F_BEGIN ? '1' : '0');
      printf(" Fe: %c,", (*_regC) & C_F_END ? '1' : '0');
      printf(" Wb: %c,", (*_regB) & B_W_BEGIN ? '1' : '0');
      printf(" We: %c,", (*_regB) & B_W_END ? '1' : '0');
      printf(" S: %c,", Amsg.buf & A_S ? '1' : '0');//не считываем, так как
      printf(" D: %c", Amsg.buf & A_D ? '1' : '0');//нет ОС по ним
}

//Вывод в консоль меню
void showMenu() {
      printf("\nControl Roby \n");
      printf("\n    1st press    2nd press");
      printf("\nF1      X ->         X STOP ->");
      printf("\nF2      X <-         X STOP <-");
      printf("\nF3      Y ->         Y STOP ->");
      printf("\nF4      Y <-         Y STOP <-");
      printf("\nF5      Z ->         Z STOP ->");
      printf("\nF6      Z <-         Z STOP <-");
      printf("\nF7      F ->         F STOP ->");
      printf("\nF8      F <-         F STOP <-");
      printf("\nF9      W ->         W STOP ->");
      printf("\nF10     W <-         W STOP <-");
      printf("\nF11     S ON         S OFF    ");
      printf("\nF12     D ON         D OFF    ");
      printf("\nPress '+' for showing sensors data\n");
      printf("\nPress 'Enter' to go to start position\n\r");
}
```

```
int main() {
	unsigned char regC, regB;//Переменные для хранения регистров C и B
	//Создание соединения с Roby
	const char coname[9] = "apu/roby";
	coid_roby = name_open(coname, 0);
	if (coid_roby == -1) {
		printf("\n Name not opened");
		return -1;
	}
	printf("\napu/roby has coid=%d", coid_roby);
	pid = getpid();//Получение Process Id
	//Создание дочерних потоков
	pthread_t displayXYZ_tid;
	pthread_t displayWF_tid;
	pthread_create(&displayXYZ_tid, NULL, &DisplayXYZ, NULL);
	sleep(2);
	pthread_create(&displayWF_tid, NULL, &DisplayWF, NULL);
	sleep(2);
	//Соединение с потоком displayWF
	displayWF = ConnectAttach(0, pid, chidWF, 0, 0);
	sleep(2);
	showMenu();//Вывод меню
	Amsg.buf = 0;
	Amsg.type = 2;//Чтение регистра C
	MsgSend(coid_roby, &Amsg, sizeof(Amsg), &regC, sizeof(unsigned char));
	Amsg.type = 3;//Чтение регистра B
	MsgSend(coid_roby, &Amsg, sizeof(Amsg), &regB, sizeof(unsigned char));
	Amsg.type = 0;//Значение для записи регистра A
	unsigned char startB = 0, startC = 0;
	startB = B_W_BEGIN | B_X_BEGIN | B_Y_BEGIN | B_Z_BEGIN;//0xF0
	startC = C_F_BEGIN;//0x08
	Cmsg.buf = 0;
	Cmsg.type = 1;
	x = regB & B_X_BEGIN ? 0 : x;//Если мы в начальном или конечном
положениях,
	y = regB & B_Y_BEGIN ? 0 : y;// то выставляем это положение
	z = regB & B_Z_BEGIN ? 0 : z;// для соответствующих координат
	f_cnt = regC & C_F_BEGIN ? 0 : regC & C_F_END ? 1000 : f_cnt;
	w_cnt = regB & B_W_BEGIN ? 0 : regB & B_W_END ? 2000 : w_cnt;
	 if (regB != startB || regC != startC)//Если мы не в начале
		goStart();//То уводим робота в начало
	int c1, c2, c3;
	raw(0);//Перевод терминала в режим нередактируемого ввода
	do {
		c1 = getchar();
		//printf("\r\nc1 = %d", c1);
		if (c1 == 27) {//ESC
			c2 = getchar();
			//printf("\r\nc2 = %d", c2);
			if (c2 == 79) {//F1-F12
				c3 = getchar();
				//printf("\r\nc3 = %d", c3);
				control(c3);
			} else if (c2 == 27)//ESC
				break;
		} else if (c1 == 43) { //'+'
			showSensors(&regC, &regB);
		} else if (c1 == 10) { //Enter
			goStart();
		}
	} while (1);
	unraw(0);
	printf("\nExit\n");}
```