# Введение

## Цель работы:

1. Освоение низкоуровневого интерфейса с роботом манипулятором.
2. Изучение системных функций QNX идентификации процесса по имени.
3. Изучение функции посылки сообщения

## Общие сведения

Учебный робототехнический комплекс состоит из манипулятора, блока управления, устройства ввода-вывода информации и ПЭВМ (Рис.1)

 Рис. 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 ПЭВМ. В режиме автоматического управления для определения текущего положения каретки манипулятора используются датчики 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 правильной работы системы в автоматическом режиме, перед началом работы манипулятор должен быть установлен в исходное состояние, которое затем принимается за нулевое. Для координат X,Y,Z существуют герконовые датчики начального положения. Для координат W,F - импульсные датчики перемещения отсутствуют, есть только герконовые датчики конечных положений.

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.



Рис.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.

Общие сведения об эмуляторе робота

Имя программы эмулятора - roby.

Эмулятор при инициализации регистрирует свое имя - "apu/roby".

Программный интерфейс с эмулятором обеспечивается посредством передачи сообщений. Каждое сообщение состоит из:

- фиксированной части - код команды длиной в 1 байт:

- переменной части - данные команды – слова (переменная типа unsigned int):

Эмулятор roby возвращает ответ: на команды чтения - ответ содержит прочитанные данные. На команды записи и инициализации ответа не предусмотрено (т.е. формируется ответ нулевой длины),

Формат управляющих команд и сообщений программного интерфейса:

1. Запись в регистр А код – 0, данные - записываемое слово;
2. Запись в регистр С код – 1, данные - записываемое слово;
3. Чтение регистра С код – 2, ответ - прочитанный байт;
4. Чтение регистра В код – 3, ответ - прочитанный байт;
5. Инициализация датчика координаты X код – 4, данные - идентификатор канала, который используется для реализации счетчика шагов по Х или 0 длиной в 1 слово;
6. Инициализация датчика координаты Y код – 5, данные - идентификатор канала, который используется для реализации счетчика шагов по Х или 0 длиной в 1 слово;
7. Инициализация датчика координаты Z код – 6, данные - идентификатор канала, который используется для реализации счетчика шагов по Х или 0 длиной в 1 слово.

# Создание интерфейсной программы для низкоуровневого управления эмулятором робота

## Интерфейсная программа управления эмулятором робота

**#include** <stdlib.h>

**#include** <stdio.h>

**struct** MESSAGE // пересылаемое сообщение send message structure

{

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

**unsigned** **int** buf; // send data

};

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

**int** status;

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

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

**char** command[10]; // Буфер команд оператора

**int** coid;

coid=name\_open("apu/roby",0);

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

**do** {

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

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

**switch** (command[0]) {

**case** 'A': //Write port A

msg.type = 0;

**sscanf**(command + 1, "%X", &msg.buf);

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

**printf**("Write port A: message <%s>\n", &rmsg);

**break**;

**case** 'C': //Write port C

msg.type = 1;

**sscanf**(command + 1, "%X", &msg.buf);

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

**printf**("Write port C: message <%d>\n", msg.buf);

**break**;

**case** 'c': //Read port C

msg.type = 2;

**sscanf**(command + 1, "%X", &msg.buf);

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

**printf**("Read port c: message <%s>\n", &rmsg);

**break**;

**case** 'b': //Read port B

msg.type = 3;

**sscanf**(command + 1, "%X", &msg.buf);

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

**printf**("Read port b: message <%s>\n", &rmsg);

**break**;

**case** 'E':

**return** 0; //Return from main

**default**:

**printf**("Unknown command\n");

**break**;

}

} **while** (1);

**return** EXIT\_SUCCESS;

}

## Файл макроопределений 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

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

## Системные вызовы и их описание

### int name\_open( const char \* name, int flags );

Функция name\_open () открывает имя для соединения с сервером. Результат не гарантируется при доступе к ресурсам на других узлах.

Аргументы:

* name: имя сервера с которым должно быть открыто соединение
* flags: флаги, которые влияют на поведение функции

Возвращает неотрицательное целое число, представляющее идентификатор канала соединения или -1, если произошла ошибка.

### int MsgSend( int coid, const void\* smsg, int sbytes, void\* rmsg, int rbytes );

Функции ядра, вызов отправляет сообщение в канал процесса по coid

Аргументы:

* coid: ID канала по которому должно быть передано сообщение, может быть получен при вызовах ConnectAttach() или name\_open().
* smsg: Указатель к буферу, содержащему сообщение к отправке.
* sbytes: Количество байт для отправки.
* rmsg: Указатель к буферу в котором должен быть сохранен ответ.
* rbytes: Размер буфера ответа в байтах.

Возвращает значение статуса из MsgReply\*() или -1, если ошибка.

## Запуск программы эмулятора робота

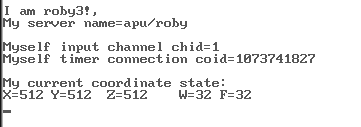


Рис. 3 Скриншот вывода эмулятора робота при запуске

## Проверка действия команды A

Проверка действия команды А, производится запуском робота на движение по координате X в обратном направлении. Поведение робота контролируем по значениям координаты в терминальном окне.



Рис.4 Скриншот переданной команды

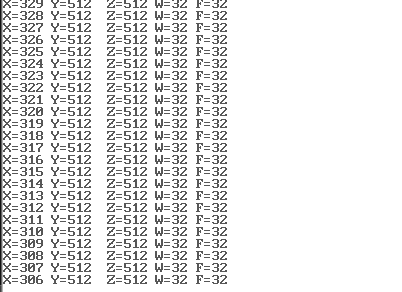


Рис. 5 Значение координаты в терминальном окне эмулятора

## Исследование низкоуровневого интерфейса и определение временных характеристик