Министерство образования и науки Российской Федерации

Санкт-Петербургский государственный электротехнический университет

«ЛЭТИ»

Факультет компьютерных технологий и информатики

Кафедра автоматики и процессов управления

## Курсовая работа

## Программа управления роботом-манипулятором

Выполнил: Малина А.

Группа: 3391

Преподаватель: Дорогов А.Ю.

Санкт-Петербург

2018

**СОДЕРЖАНИЕ**

1. Задание на разработку программы……………………………………………………...3

2. Описание эмулятора робота……………………………………………………………...4

3. Программный интерфейс эмулятора робота…………………………………………..6

4. Структурная схема программы на уровне программных потоков………………….8

5. Описание алгоритма функционирования программных потоков с указанием

их характеристик……………………………………………………………………………….9

5.1. Приоритеты потоков……………………………………………………………...9

5.2. Дисциплины диспетчеризации………………………………………………..10

5.3. Возможные состояния и причины их изменения в процессе сеанса работы программы…………………………………………………………………...10

6. Описание интерфейса между потоками……………………………………………….11

7. Текст программы с комментариями……………………………………………………12

1. Задание на разработку программы.

**ПРОГРАММА УПРАВЛЕНИЯ РОБОТОМ-МАНИПУЛЯТОРОМ ПУЛЬТ-1n**

**Задание на разработку**

Программа должна обеспечивать следующие функции:

1. Прием команд оператора вводимых с клавиатуры и 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.

**ВАРИАНТ 2**

**задания на разработку программы управления роботом**

1. Способ передачи события, связанного с изменением состояния

импульсного датчика по координатам X,Y,Z - сигнал.

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 хранения текущего значения регистров робота - ло-

кально в одном из потоков.

5. Способ передачи параметров в порожденный поток - через

глобальные переменные.

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.



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 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.



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.

3. Программный интерфейс эмулятора робота.

Эмулятор (программа **roby)** представляет собой процесс, реализующий одновременно функции сервера и клиента. Для управляющих команд эмулятор выступает как неполный сервер, регистрирующий для межпроцессного взаимодействия имя **"apu/roby".** Локализация сервера выполняется командой name\_open(), возвращающей управляющему процессу идентификатор соединения. Программный интерфейс с серверной компонентой обеспечивается посредством передачи QNX-сообщений. Каждое сообщение состоит из фиксированной части - кода команды длиной в 1 байт и значения команды длиной в 1 байт. На команды чтения эмулятор возвращает ответ длиной в один байт, который содержит прочитанные данные. На команды записи и инициализации ответа не предусмотрено (т.е. формируется ответ нулевой длины). Формат управляющих команд и сообщений программного интерфейса приведен в таблице 1.

Таблица 1

|  |  |  |  |
| --- | --- | --- | --- |
| Команда | Код сообщения | Данные | Ответ |
| Запись в регистр **А** | 0 | Значение регистра | - |
| Запись в регистр **С** | 1 | Значение регистра | - |
| Чтение регистра **С** | 2 | - | Значение регистра |
| Чтение регистра **В** | 3 | - | Значение регистра |
| Инициализация импульсного датчика **X** | 5 | идентификатор канала или 0 | - |
| Инициализация импульсного датчика **Y** | 6 | идентификатор канала или 0 | - |
| Инициализация датчиков конечных положений координаты **W** | 7 | идентификатор канала или 0 | - |
| Инициализация датчиков конечных положений координаты **F** | 8 | идентификатор канала или 0 | - |

Эмулятор содержит внутренний таймер и счетчики по каждой координате. Счетчики выполняют подсчет импульсов таймера, эмулируя пошаговые перемещения каретки робота. При движении робота значения внутренних счетчиков по всем координатам непрерывно выводятся в терминальном окне. Темп работы счетчиков зависит от координаты, а возможные состояния ограничены диапазоном: 0-1024 для координат **X,Y,Z** и 0-64 для координат **W,F**. Счетчики запускаются, когда получена команда на перемещение и прекращают счет при достижении границ допустимого диапазона. Состояние импульсных датчиков координат **X,Y,Z** изменяется на каждом шаге перемещения, и отображается битовыми переменными регистра **B** контроллера. При достижении крайних положений изменяют свое состояние датчики конечных положений, их значение отображается битовыми переменными регистров **B** и **C**.

Управление реальным роботом (УРТК) осуществлялось за счет непрерывного циклического опроса регистров контроллера **B** и **C.**  Счетчики перемещений в этом случае поддерживались в управляющей программе. Такой вариант управления возможен и для эмулятора, но для учебных целей функции эмулятора расширены возможностью передачи численных значений внутренних счетчиков положений с помощью сигналов и импульсных сообщений. Для перехода в режим расширенного использования датчики эмулятора должны быть инициализированы номером канала или нулевым значением. Инициализация датчиков номером канала активизирует эмулятор робота для работы в режиме клиента – источника сообщений. Сервером в этом случае выступает управляющая программа, которая открывает канал (при помощи вызова ChannelCreate()) и передает эмулятору номер этого канала (в общем случае на каждый датчик может быть использован свой собственный канал). Эмулятор использует номер канала для установления соединений, через которые посылаются импульсные сообщения (pulse) при каждом изменении значений импульсных фотодатчиков.

Для импульсных фотодатчиков каждое значение импульсного сообщения несет информацию о значении координаты **X, Y** или **Z,** а код определяет вид координаты.Эти сообщения передаются только, когда робот движется, т.е. когда предварительно была подана команда на движение робота вперед или назад по какой либо координате. Темп генерации импульсных сообщений определяется атрибутами встроенного таймера эмулятора.

Если при инициализации вместо номера канала, передано нулевое значение, то информация о состоянии координат будет передаваться сигналами (**SIGUSR1**). Код сигнала несет информацию о координате, а значение - о состоянии соответствующего счетчика.

Датчики начальных и конечных положений инициализируются также как и импульсные фотодатчики, и предают информацию с помощью сигналов (**SIGUSR2**) и импульсных сообщений, когда состояние датчиков изменяется. В таблице 2 приведены значения использованных кодов импульсов и сигналов. Все датчики инициализируются независимо, поэтому часть из них может быть активизирована на передачу импульсных сообщений, а другая часть – на передачу сигналов.

Таблица 2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Координата робота | Код импульса | Значение импульса | Код сигнала | Значение сигнала |
| X-импульсный | 0x01 | Счетчик X | 0x01 | Счетчик X |
| Y-импульсный | 0x02 | Счетчик Y | 0x02 | Счетчик Y |
| Z-импульсный | 0x04 | Счетчик Z | 0x04 | Счетчик Z |
| W-начало | 0x10 | Счетчик W | 0x10 | Счетчик W |
| W-конец | 0x08 | Счетчик W | 0x08 | Счетчик W |
| F-начало | 0x40 | Счетчик F | 0x40 | Счетчик F |
| F-конец | 0x20 | Счетчик F | 0x20 | Счетчик F |

Передача информации о состоянии датчиков начальных положений по координатам **X,Y,Z** не предусмотрена, их состояние может контролироваться только в режиме опроса регистра **B** контроллера.

4. Структурная схема программы на уровне программных потоков.

На следующем рис. 3 показан программный интерфейс трехпоточного приложения для управления роботом по координатам **X,Y,Z** и **W,F**.



**сигналы**

**сообщения**

**импульсы**

**импульсы**

**импульсы**

*Рис. 3.*

5. Описание алгоритма функционирования программных потоков

с указанием их характеристик.

*5.1. Приоритеты потоков.*

При разработке программы ни один поток особо не был выделен. Все потоки имеют равный приоритет, но больше, чем 0, чтобы у них была возможность выполняться, не конкурируя с потоком Idle (системный поток с приоритетом 0, который готов к выполнению).

Чтобы конкретно определить назначенные в работающей программе приоритет для каждого потока нужно выполнить следующее.

**Исследование конфигурацию каналов и соединений с помощью интегрированной системы Momentics IDE.**

1. Запустите в терминальном окне программу **qconn** – целевой агент сбора информации.
2. Создайте новое окно в интегрированной системе.
3. Выберите в созданном окне перспективу QNX System Information ( нажмите последовательно **Window>Open Perspective>QNX System Information**).
4. Создайте новый проект типа QNX Target System Project (**File>New>Project>QNX (в левом окне)> QNX Target System Project (в правом окне)>Next>)**.
5. В предложенной форме укажите имя проекта (любое) и IP-адрес целевой системы. IP адрес можно взять из меню консоли (**Configuration/Network**). Нажмите **Finish**. Интегрированная система создаст исследовательский проект с указанным именем в поле **Target Navigator**.
6. Нажмите на «**+**» возле имени проекта, чтобы развернуть список процессов. Разыщите в списке процессы **roby** и **pult**. В скобках рядом с именем процессов указаны их идентификаторы **pid**, запишите значения идентификаторов **roby** и **pult** в отчет.
7. Выделите процесс **pult** и выберите ярлычок **Thread Information**. В отчет запишите приоритеты потоков и дисциплину диспетчеризации (символ **r** – циклическая, **f** – FIFO, **o**-спорадическая).
8. Выделите процесс **roby** и выберите ярлычок **Connection Information**. Запишите в отчет идентификаторы соединений и расшифруйте их назначение. Если соединение устанавливается с помощью функций open() для файловых операций, то идентификатором соединения является файловый дескриптор (который обозначается цифрами 0,1,2,3,..). Когда же соединение устанавливается функцией name\_open() для обмена сообщениями, то идентификаторы соединений будут обозначаться 0s,1s,2s,3s,..

*5.2. Дисциплины диспетчеризации.*

Рассмотрим 3 возможных дисциплины диспетчеризации (Scheduling Policy), используемых в ОС QNX.

* FIFO (дисциплина “первый вошёл – первый вышел”): если потоку назначена диспетчеризация FIFO, то другой поток с равным приоритетом получает управление в случае блокировки первого потока (или если первый поток самостоятельно уступает управление, например, заснув sleep()).
* Циклическая (round) диспетчеризация: поток исполняется в течение некоторого кванта времени и передаёт управление следующему потоку с таким же приоритетом.
* Спорадическая диспетчеризация: устанавливается лимит (бюджет) использования потоком процессора в течение определённого периода времени, после исчерпания которого приоритет потока снижается и он уступает право управления другим потокам (с более нормальным приоритетом) до тех пор, пока приоритет первого потока не восстановится до нормального значения.

При разработке данной многопоточной программы предполагалось, что потоки стараются самостоятельно уступать остальным потокам право выполнения (с помощью функции sleep() вызывалась процедура передиспетчеризации потоков для инициализации датчиков и т. д.). Но диспетчеризация FIFO не подходит в данном случае, так как поток main забирает всё управление, ожидая команды от пользователя (нажатие клавиши).

Дисциплина диспетчеризации round (циклическая) или other (спорадическая) могут решить эту проблему, и возможность управления сможет равномернее распределиться между потоками данной программы.

*5.3. 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).

**Ready**

**Receive**

**Состояния сервера**

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.

**Ready**

**Send**

**Состояния клиента**

**Reply**

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.

6. Описание интерфейса между потоками.

*Интерфейс* main↔Roby

Поток main определяет идентификатор соединения coid\_roby, используемого для связи с процессом Roby, с помощью функции name\_open(), задав имя “apu/roby”, под которым процесс Roby зарегистрировался в системе. Поток main, зная coid\_roby, передаёт его значение остальным потокам в процессе pult, так как coid\_roby является глобальной переменной.

Поток main обрабатывает команды пользователя (нажатие клавиш) и посылает в качестве клиента соответствующие сообщения (записать в порт A, C, прочитать порт B) с помощью вызова MsgSend() к процессу-серверу Roby.

На время обработки сервером Roby посланного сообщения поток main блокируется до момента получения ответа от Roby.

*Интерфейс* DisplayXYZ↔Roby

Поток DisplayXYZ (дисплей датчиков линейного перемещения по X, Y и Z координатам) создаётся потоком main, выполняет инициализацию датчиков X, Y и Z с помощью инициализирующих сообщений MsgSend(), не указывая канал chid (тогда chid==0). Процесс Roby, приняв эти сообщения, инициализирует импульсные датчики движения (Xp, Yp и Zp) так, что они начинают генерировать сигналы.

С этого момента процесс Roby перестаёт быть сервером для потока DisplayXYZ и сам периодически пересылает сигналы от датчиков перемещения по X, Y и Z, которые принимаются с помощью функции sigwaitinfo(). Для этой цели применяются сигналы SIGUSR1.

*Интерфейс* DisplayWF↔main

Поток main создаёт поток DisplayWF (дисплей датчиков перемещения головки по W и основания по F), а затем в том случае, если начинается движение по W или F (вперёд или назад), main посылает импульсы к потоку DisplayWF, которые не блокируют ни принимающий поток DisplayWF ни отправителя.

Получив данные импульсы и проанализировав их код, поток DisplayWF запускает работу таймеров для отсчёта времени перемещения по W и F. Если же робот двигается по

координатам W и F на текущий момент времени, то импульсы от потока main не посылаются.

*Интерфейс* DisplayWF↔Roby

Поток DisplayWF после инициализации датчиков крайних положений по W и F может принять от датчиков эмулятора Roby импульсы о достижении того или иного крайнего положения (W\_BEGIN, W\_END, F\_BEGIN или F\_END). В таком случае движение по соответствующей координате будет остановлено, а также будет остановлен соответствующий ей виртуальный таймер.

*Интерфейс* DisplayWF↔виртуальные таймеры по W и F

Поток DisplayWF заполняет структуру sigevent для каждого виртуального таймера (структуры pulse\_event\_W и pulse\_event\_F для W и F соответственно), указывая chid=chidWF для созданного канала, открытые соединения coidW и coidF по этому каналу, предназначенные для пересылки импульсов от виртуальных таймеров.

Указав тип уведомления struct sigevent и создав виртуальные таймеры (относительные интервальные таймеры реального времени типа CLOCK\_REALTIME), при движении робота по координате W или F, поток DisplayWF будет принимать импульсы от этих таймеров, соответствующие некоторому перемещению по координате W или F.

*Интерфейс потоков программы*

Интерфейс потоков программы обеспечивается с помощью глобальных переменных, содержащих значения координат X, Y, Z, W и F (см. счётчики w\_cnt и f\_cnt).

Среди глобальных переменных особо следует выделить следующие:

● coid\_roby – идентификатор для соединения с Roby: определяется по имени “apu/roby” в потоке main и распространяется через эту глобальную переменную остальным потокам программы для посылки сообщений к эмулятору Roby (например, для инициализации датчиков);

● chidWF – идентификатор канала, созданного потоком DisplayWF, к которому с помощью образованных соединений можно посылать импульсы: от потока main по соединению coid\_DisplayWF, от таймеров по W и по F, от датчиков крайних положений по W и F робота.

7. Текст программы с комментариями.

**Файл “makefile”**

#This is "makefile".

OBJ=pult.o

all: $(OBJ)

qcc -o -g pult $(OBJ)

$(OBJ): pult.c roby.h

qcc -c -g pult.c

clean:

$(RM) $(OBJ)

**Файл “roby.h”**

#ifndef ROBY\_H\_

#define ROBY\_H\_

// Биты регистра A:

#define A\_D 0x01 // D: управление дрелью;

#define A\_S 0x02 // S: управление схватом;

#define A\_X\_FORWARD 0x04 // +X: движение по X вперёд;

#define A\_X\_BACK 0x08 // -X: движение по X назад;

#define A\_Z\_BACK 0x10 // -Z: движение по Z назад;

#define A\_Z\_FORWARD 0x20 // +Z: движение по Z вперёд;

#define A\_Y\_BACK 0x40 // -Y: движение по Y назад;

#define A\_Y\_FORWARD 0x80 // +Y: движение по Y вперёд.

// Биты регистра B:

#define B\_X 0x01 // Xp (Xpulse): импульсный датчик движения по X;

#define B\_Y 0x02 // Yp (Ypulse): импульсный датчик движения по Y;

#define B\_Z 0x04 // Zp (Zpulse): импульсный датчик движения по Z;

#define B\_W\_END 0x08 // WE (WEnd): датчик конечного положения по W (для головки);

#define B\_W\_BEGIN 0x10 // WB (WBegin): датчик начального положения по W (для головки);

#define B\_Z\_BEGIN 0x20 // ZN: датчик начального положения по Z;

#define B\_Y\_BEGIN 0x40 // YN: датчик начального положения по Y;

#define B\_X\_BEGIN 0x80 // XN: датчик начального положения по X.

// Биты регистра C: (при этом биты PC0 и PC1 не используются)

#define C\_F\_END 0x04 // FE (FEnd): датчик конечного положения по F (для основания);

#define C\_F\_BEGIN 0x08 // FB (FBegin): датчик начального положения по F (для основания);

#define C\_W\_FORWARD 0x10 // +W: движение по W вперёд (для головки);

#define C\_W\_BACK 0x20 // -W: движение по W назад (для головки);

#define C\_F\_FORWARD 0x40 // +F: движение по F вперёд (для основания);

#define C\_F\_BACK 0x80 // -F: движение по F назад (для основания).

// Для потока DisplayXYZ

//Коды сигналов, передающих значения координат X, Y и Z от робота Roby:

#define X\_VALUE -1 //Для кодов сигналов используются отрицательные значения,

#define Y\_VALUE -2 //так как положительные значения кодов сигналов

#define Z\_VALUE -4 //использует система.

// Для потока DisplayWF

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

#define W\_END 0x08

#define W\_BEGIN 0x10

#define F\_END 0x20

#define F\_BEGIN 0x40

// Коды импульсных сообщений для запуска таймеров в потоке DisplayWF:

#define TIMER\_W\_SET 1

#define TIMER\_F\_SET 2

// Коды импульсов таймера для заполнения структуры (уведомления) sigevent в потоке DisplayWF:

#define TIMER\_W\_COUNT 33 // таймер координаты W;

#define TIMER\_F\_COUNT 65 // таймер координаты F.

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

**Файл “pult.с”**

#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 <pthread.h>

#include <signal.h>

#include <time.h>

#include <sys/siginfo.h>

#include "roby.h"

struct MESSAGE //Пересылаемое сообщение send message structure

{

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

unsigned int buf;// send data

};

void\* DisplayXYZ(); // Прототип функции DisplayXYZ().

void handler(int); // Прототип функции handler().

void\* DisplayWF(); // Прототип функции DisplayWF().

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.

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

unsigned int X=0, Y=0, Z=0; //Эти переменные содержат значения координат X, Y и Z

//для робота Roby.

int main\_pulse\_priority=10; //Приоритет импульсов, поступающих от потока main

//к потоку DisplayWF.

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

unsigned char DisplayState=0; //Режим отображения состояния датчиков:

//0 - выключен; 1 - включен.

//-----Определения функций-----------------------------------------------------------

int main()

{

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

//робота Roby.

unsigned char ch; //Символ, введённый пользователем с клавиатуры.

char Help[]="Unknown command. Please, use next keys: <F1>-<F12>, <Enter>, <+/=> or <Esc> for exit."; //Справка.

char command; //Тип команды.

int status;

struct MESSAGE msg; //Посылаемое сообщение.

unsigned char rmsg; //Буфер ответного сообщения.

//char command[10]; //Буфер для команд оператора.

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

pthread\_t DisplayXYZ\_tid; //TID (идентификатор потока) для потока DisplayXYZ.

pthread\_t DisplayWF\_tid; //TID для потока DisplayWF.

pthread\_create(&DisplayXYZ\_tid, NULL, &DisplayXYZ, NULL); //Создать поток-сын DisplayXYZ

pthread\_create(&DisplayWF\_tid, NULL, &DisplayWF, NULL); //и поток-сын DisplayWF.

//pthread\_create(&tid, &attr, &Display, NULL);

//....Вставлены операторы.

coid\_roby=coid=name\_open("apu/roby", NULL); //Определить coid (идентификатор соединения)

//по имени "apu/roby".

//int name\_open( const char \* name,

// int flags );

//...........

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

sleep(5); //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).

pid\_t PID=getpid(); int coid\_DisplayWF=ConnectAttach(0, PID, chidWF, 0, 0); //Создать

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

/\*\*\*\*\*Syntax for ConnectAttach()\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* int ConnectAttach(int nd, //Node Descriptor - дескриптор узла в сети (nd=0, если

\* //локальное соединение);

\* pid\_t pid, //PID (Process ID);

\* int chid, //CHID (Channel ID);

\* unsigned index,

\* int flags); //Флаги.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

printf("\nPult1\_var2: main thread: please, press any key.");

do { //Бесконечный цикл.

sleep(1); //Передать управление потоку DisplayXYZ или

//потоку DisplayWF (на некоторое время).

//.....Интерфейс с пользователем.

printf("\n>"); //Приглашение выбрать команду с помощью клавиши (для пользователя).

//scanf("%s",command); //Считать команду оператора с клавиатуры.

ch=getch(); //Считать символ, введённый с клавиатуры.

switch(ch){ //Анализ ASCII-кода нажатой клавиши.

case 27: //Нажата клавиша <Esc>.

printf("\n<Esc> - exit..."); //Завершение программы.

command='E'; //Тип команды.

return 0;

case 13: //Нажата клавиша <Enter>.

printf("\n<Enter> - move to start position on all coordinates"

"\nand clear position values.");

//Двигаться в начальное положение по всем координатам

//и сбросить значения датчиков положений.

//command=...; //Тип команды.

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 '=': //Нажата клавиша <+/=>.

//Переключить режим отображения состояния датчиков.

DisplayState^=0x01; //Поразрядное исключающее ИЛИ (XOR)

//для переменной DisplayState сбрасывает её в 0

//либо устанавливает в 1.

printf("\n<+/=> - display state = %d.", DisplayState);

//command=...; //Тип команды.

break;

case 0: //Нажата управляющая или функциональная клавиша.

switch(ch=getch()){ //Это требует дополнительного анализа.

case 59: //Нажата клавиша <F1>.

//Начать/остановить движение по X вперёд.

printf("\n<F1> - toggle moving on X forward.");

PA^=A\_X\_FORWARD; //Переключить бит A\_X\_FORWARD в регистре PA.

command='A'; //Тип команды.

break;

case 60: //Нажата клавиша <F2>.

//Начать/остановить движение по X назад.

printf("\n<F2> - toggle moving on X backward.");

PA^=A\_X\_BACK; //Переключить бит A\_X\_BACK в регистре PA.

command='A'; //Тип команды.

break;

case 61: //Нажата клавиша <F3>.

//Начать/остановить движение по Y вперёд.

printf("\n<F3> - toggle moving on Y forward.");

PA^=A\_Y\_FORWARD; //Переключить бит A\_Y\_FORWARD в регистре PA.

command='A'; //Тип команды.

break;

case 62: //Нажата клавиша <F4>.

//Начать/остановить движение по Y назад.

printf("\n<F4> - toggle moving on Y backward.");

PA^=A\_Y\_BACK; //Переключить бит A\_Y\_BACK в регистре PA.

command='A'; //Тип команды.

break;

case 63: //Нажата клавиша <F5>.

//Начать/остановить движение по Z вперёд.

printf("\n<F5> - toggle moving on Z forward.");

PA^=A\_Z\_FORWARD; //Переключить бит A\_Z\_FORWARD в регистре PA.

command='A'; //Тип команды.

break;

case 64: //Нажата клавиша <F6>.

//Начать/остановить движение по Z назад.

printf("\n<F6> - toggle moving on Z backward.");

PA^=A\_Z\_BACK; //Переключить бит A\_Z\_BACK в регистре PA.

command='A'; //Тип команды.

break;

case 65: //Нажата клавиша <F7>.

//Начать/остановить движение по F вперёд.

printf("\n<F7> - toggle rotating on F forward.");

PC^=C\_F\_FORWARD; //Переключить бит C\_F\_FORWARD в регистре PC.

command='C'; //Тип команды.

ActiveF=1; //F - активная координата.

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

stateF=0; //то стоп по F.

else //Иначе

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

break;

case 66: //Нажата клавиша <F8>.

//Начать/остановить движение по F назад.

printf("\n<F8> - toggle rotating on F backward.");

PC^=C\_F\_BACK; //Переключить бит C\_F\_BACK в регистре PC.

command='C'; //Тип команды.

ActiveF=1; //F - активная координата.

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

stateF=0; //то стоп по F.

else //Иначе

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

break;

case 67: //Нажата клавиша <F9>.

//Начать/остановить движение по W вперёд.

printf("\n<F9> - toggle rotating head on W forward.");

PC^=C\_W\_FORWARD; //Переключить бит C\_W\_FORWARD в регистре PC.

command='C'; //Тип команды.

ActiveW=1; //W - активная координата.

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

stateW=0; //то остановить его по W.

else //Иначе

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

break;

case 68: //Нажата клавиша <F10>.

//Начать/остановить движение по W назад.

printf("\n<F10> - toggle rotating head on W backward.");

PC^=C\_W\_BACK; //Переключить бит C\_W\_BACK в регистре PC.

command='C'; //Тип команды.

ActiveW=1; //W - активная координата.

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

stateF=0; //то остановить его по W.

else //Иначе

stateF=-1; //запустить движение назад по W.

break;

case 133: //Нажата клавиша <F11>.

//Включить/выключить схват S.

printf("\n<F11> - toggle S.");

PA^=A\_S; //Переключить бит A\_S в регистре PA.

command='A'; //Тип команды.

break;

case 134: //Нажата клавиша <F12>.

//Включить/выключить дрель D.

printf("\n<F12> - toggle drill work.");

PA^=A\_D; //Переключить бит A\_D в регистре PA.

command='A'; //Тип команды.

break;

default: //Нажата другая (управляющая или функциональная) клавиша.

printf("\n%s", Help);

command='U'; //Неизвестный тип команды.

} //End of switch().

break;

default: //Нажата другая клавиша.

printf("\n%s", Help); //puts(Help);

command='U'; //Неизвестный тип команды.

} //End of switch().

if(DisplayState==1){ //Если включен режим отображения состояния датчиков, то

//command='b'; //Команда чтения из порта PB роботу Roby.

command='c'; //Команда чтения из порта PC роботу Roby.

} //End of if().

//...............................

//.....Анализ типа команды command.

switch(command) { //Анализировать тип команды, заданный символом command.

//switch(command[0]) { //Проверить символ, с которого начинается команда.

case 'A': //Write port A.

msg.type=0; //Тип сообщения.

msg.buf=PA; //Записать в буфер сообщения msg.buf значение переменной PA.

//sscanf(command+1,"%X", &msg.buf); //Преобразовать строку (command+1)

//из 16-ичного вида и сохранить результат в msg.buf.

//.....Вставлены операторы.

printf("Write port A.");

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

//int MsgSend( int coid, //Послать сообщение, где coid - идентификатор соединения;

// const void\* smsg, //smsg - указатель на посылаемое сообщение;

// int sbytes, //sbytes - размер (в байтах) посылаемого сообщения;

// void\* rmsg, //rmsg - указатель на буфер принимаемого ответа от roby;

// int rbytes ); //rbytes - размер (в байтах) буфера для принимаемого ответа.

break;

case 'C': //Write port C.

msg.type=1; //Тип сообщения.

//.....Вставлены операторы.

msg.buf=PC; //Записать в буфер сообщения msg.buf значение переменной PC.

//sscanf(command+1,"%X", &msg.buf); //Считать 16-ичную строку из (command+1)

//в буфер msg.buf.

printf("Write port C.");

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

MsgSendPulse(coid\_DisplayWF, main\_pulse\_priority,

TIMER\_W\_SET, NULL); //запустить таймер по W.

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

MsgSendPulse(coid\_DisplayWF, main\_pulse\_priority,

TIMER\_F\_SET, NULL); //запустить таймер по F.

//.......................................

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

//сообщение msg к roby.

break;

case 'c': //Read port C.

msg.type=2; //Тип сообщения.

//......Вставлены операторы.

//Буфер сообщения msg.buf при чтении порта C заполнять не требуется.

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

printf("Read port C.");

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

//сообщение msg к roby и принять ответ rmsg от roby.

printf("Port C: 0x%X", rmsg); //Вывод на экран (в поток stdout) ответ rmsg

//в 16-ичном виде.

break;

case 'b': //Read port B.

msg.type=3; //Тип сообщения.

//........Вставлены операторы.

//Буфер сообщения msg.buf при чтении порта B заполнять не требуется.

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

printf("Read port B.");

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

//сообщение msg к roby и принять ответ rmsg от roby.

printf("Port B: 0x%X", rmsg); //Вывод на экран (в поток stdout) ответ rmsg

//в 16-ичном виде.

break;

case 'E': //Выход из программы.

return 0; //Return from main().

default: //Другой символ.

printf("Unknown command\n");

//break;

}

} while(1);

}

void\* DisplayXYZ()

{

struct sigaction act;

sigset\_t set; //Набор сигналов.

siginfo\_t info; //Информация о сигнале.

int changed = 0;

int coid = coid\_roby;

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

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

struct MESSAGE msg;

sigemptyset(&set); // Очистить набор сигналов set.

sigaddset(&set, SIGUSR1); // Добавить в набор set сигнал SIGUSR1.

act.sa\_mask = set; // Маска сигналов.

act.sa\_handler = &handler; // Указать обработчик сигналов.

/\*

Если в sa\_flags указан SA\_SIGINFO,

то sa\_sigaction (вместо sa\_handler) задаёт функцию обработки сигнала signum.

В первом аргументе функция принимает номер сигнала

\*/

act.sa\_flags = SA\_SIGINFO;

/\* Установить связь между получением

данным процессом сигнала SIGUSR1 и действием act,

которое предусматривает вызов обработчика сигналов handler()

\*/

sigaction(SIGUSR1, &act, NULL);

// Инициализация датчиков перемещения по X, Y и Z робота roby

msg.buf = 0;

msg.type = 4;

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

msg.type = 5;

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

msg.type = 6;

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

while(1)

{

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

// Приём СИГНАЛА SIGUSR1 от roby

// Ожидать СИГНАЛ из набора set.

sigwaitinfo(&set, &info);

// Анализ сигнала по коду

switch(info.si\_code)

{

case -B\_X:

x\_cnt = info.si\_value.sival\_int;

changed = 1;

break;

case -B\_Y:

y\_cnt = info.si\_value.sival\_int;

changed = 1;

break;

case -B\_Z:

z\_cnt = info.si\_value.sival\_int;

changed = 1;

break;

default:

changed = 0;

printf("\nDisplay thread: unknown signal code: %i", info.si\_code);

}

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);

}

}

// Завершить работу потока Display

pthread\_exit(NULL);

}

void handler(int signo)

{

// Обработчик сигналов SIGUSR1.

}

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.

int coid = coid\_roby;

// 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);

}