Ташкентский университет информационных технологий

Ферганский филиал

Кафедра «Информационно - образовательных технологий»

Методические пособия для самостоятельное изучению и проведению практических занятий по предмету

«СИСТЕМЫ РЕАЛЬНОГО ВРЕМЕНИ»

Направления образование: « 5330500 - Компьютерний инжиниринг»

Фергана 2017г.

Данная методические пособия предназначены для студентов по направлению подготовки: «Компьютерний инжиниринг». В нем приведены краткие теоритические материалы по определенным тематикам предмета, по которому выполняется практические занятии. Последовательности тематики соответствуют программу дисциплины «системы реального времени» утвержденной мииистерством ВиССО РУз от 12. 04. 2012г.

Методические пособия подготовлена на основе материала ОРЕНБУРГСКИЙ ГОСУДАРСТВЕННЫЙ АГРАРНЫЙ УНИВЕРСИТЕТА опубликованный в сети интернета.

Пособия также полезно для преподавателей ВУЗов, которые занимаются разработкой программные продукт.

Составитель: С.М. Абдурахмонов

Рецензент: И.У. Билолов

«УТВЕРЖДЕНО»
На заседание научного совета
филиала
Протокол № 2
«____»_____201__г.
Заместитель директор по УНИР
_____ Ф.Ю. Полвонов

«РЕКОМЕНДОВАНО»
Начальник учебно-методического
отдела
_____ М.Андапулатов

Заседание кафедры «Информационно-образовательных технологий»
протокол №____, «_____»_____ 201___ года
заведующей кафедрой _____ Абдурахмонов С.

СОДЕРЖАНИЕ

СОДЕРЖАНИЕ

1.2.2. Краткое содержание вопросов:

Требование 1. Операционная система должна быть мультипрограммной и мультизадачной (многопоточной — multi-threaded), а также активно использовать прерывания для диспетчеризации. Максимальное время выполнения того или иного действия в ОСРВ должно быть известно заранее и соответствовать требованиям приложения. Операционная система так же должна быть многопоточной на принципе абсолютного приоритета (прерываемой). То есть планировщик должен иметь возможность прервать любой поток выполнения и предоставить ресурс тому потоку, которому он более необходим. Операционная система (и аппаратура) должна также обеспечивать прерывания на уровне обработки прерываний.

Приоритеты задач.

Требование 2. В системе реального времени должны существовать гарантии того, что событие с высоким приоритетом будет обработано перед событием более низкого приоритета. ОСРВ должна обладать развитой системой приоритетов. Во-первых, это требуется потому, что система сама может рассматриваться как набор приложений, подразделяющихся на потоки, и несколько высоких уровней приоритетов должны быть выделены системным процессам и потокам. Во-вторых, в сложных приложениях необходимо все потоки реального времени помещать на разные приоритетные уровни, а потоки не реального времени помещать на один уровень (ниже, чем любые потоки реального времени). При этом потоки не реального времени можно обрабатывать в режиме циклического планирования при котором каждому процессу предоставляется квант времени процессора, а когда квант заканчивается, контекст процесса сохраняется, и он ставится в конец очереди. Во многих ОСРВ для планирования задач на одном уровне используется режим циклического планирования. Наследование приоритетов

Требование 3. Должна существовать система наследования приоритетов. Комбинация приоритетов потоков и разделение ресурсов между ними приводит к проблеме инверсии приоритетов. Это можно проиллюстрировать на примере, где есть как минимум три потока. Когда поток низшего приоритета захватил ресурс, разделяемый с потоком высшего приоритета, и начал выполняться поток среднего приоритета, выполнение потока высшего приоритета будет приостановлено, пока не освободится ресурс и не отработает поток среднего приоритета. В этой ситуации время, необходимое для завершения потока высшего приоритета, зависит от нижних уровней приоритетов — это и есть инверсия приоритетов. В такой ситуации трудно выдержать ограничение на время исполнения.Чтобы устранить такие инверсии, ОСРВ должна допускать наследование приоритета, то есть повышение уровня приоритета потока до уровня потока, который его вызывает.Наследование означает, что блокирующий ресурс поток наследует приоритет потока, который он блокирует, но это справедливо только в случае, когда блокируемый поток имеет более высокий приоритет.

Синхронизация процессов и задач

Требование 4. Операционная система должна обеспечивать мощные, надежные и удобные механизмы синхронизации задач. Так как задачи разделяют данные ресурсы) и должны сообщаться друг с другом, представляется логичным существование механизмов блокирования и коммуникации. То есть необходимы

механизмы, гарантированно предоставляющие возможность оперативно обменяться сообщениями и синхросигналами между параллельно выполняющимися задачами и процессами. Эти системные механизмы должны быть всегда доступны процессам, требующим реального времени. Следовательно, системные ресурсы для их функционирования должны быть распределены заранее. Пренебрежение вопросами синхронизации процессов, выполняющихся в режиме мультипрограммирования, может привести к их неправильной работе или даже к краху системы. Предсказуемость.

Требование 5. Поведение операционной системы должно быть известно и достаточноточно прогнозируемо. Время выполнения системных вызовов и временные характеристики поведения системы в различных обстоятельствах должны быть известны разработчику. Поэтому создатель ОСРВ должен приводить следующие характеристики:

• задержку прерывания, то есть время от момента прерывания до момента запуска задачи: она должна быть предсказуема и согласована с требованиями приложения;

1.3. : Тема « Аппаратурная среда систем реального времени.»

1.3.1. План:

1) Виды вычислительных установок систем реального времени.

2) Требования к вычислительным установкам систем реального времени.

3) Особенности аппаратного обеспечения систем реального времени.

1.3.2. Краткое содержание вопросов:

Систему реального времени можно разделить как бы на три слоя:

1. Ядро - содержит только строгий минимум, необходимый для работы системы: управление задачами, их синхронизация и взаимодействие, управление памятью и устройствами ввода/вывода; размер ядра очень ограничен: часто несколько килобайт.

2. Система управления - содержит ядро и ряд дополнительных сервисов, расширяющих его возможности: расширенное управление памятью, вводом/выводом, задачами, файлами и т.д., обеспечивает также взаимодействие системы и управляющего/управляемого оборудования.

3. Система реального времени - содержит систему управления и набор утилит: средства разработки (компиляторы, отладчики и т.д.), средства визуализации (взаимодействия человека и операционной системы). Вычислительные установки, на которых применяются СРВ, можно условно разделить на три группы.

1. «Обычные» компьютеры. По логическому устройству совпадают с настольными системами. Аппаратное устройство несколько отличается. Для обеспечения минимального времени простоя в случае технической неполадки процессор, память и т.д. размещены на съемной плате, вставляемой в специальный разъем так называемой «пассивной» основной платы. В другие разъемы этой платы вставляются платы периферийных контроллеров и другое оборудование. Сам компьютер помещается в специальный корпус, обеспечивающий защиту от пыли и механических повреждений. В качестве мониторов часто используются жидкокристаллические

дисплеи, иногда с сен-сочувствительным покрытием. По экономическим причинам среди процессоров этих компьютеров доминирует семейство Intel 80x86.

Подобные вычислительные системы обычно не используются для непосредственного управления промышленным оборудованием. Они, в основном, служат как терминалы для взаимодействия с промышленными компьютерами и встроенными контроллерами, для визуализации состояния оборудования и технологического процесса. На таких компьютерах в качестве операционных систем часто используются «обычные» операционные системы с дополнительными программными комплексами, адаптирующими их к требованиям «реального времени».

2. Промышленные компьютеры. Состоят из одной платы, на которой размещены:

- процессор, контроллер памяти, память 4-х видов:

- ПЗУ, постоянное запоминающее устройство (ROM, read-only memory), где обычно размещена сама операционная система реального времени, типичная емкость – 0,5-1 Мб;

- ОЗУ, оперативное запоминающее устройство (RAM, random access memory), куда загружается код и данные ОСРВ; обычно организована на базе динамической памяти (dynamic RAM, DRAM); типичная емкость – 16-128 Мб;

- статическое ОЗУ (static RAM, SRAM) (то же, что и ОЗУ, но питается от имеющейся на плате батарейки), где размещаются критически важные данные, которые не должны пропадать при выключении питания; типичная емкость - 2Мб; типичное время сохранения данных - 5 лет;

- флеш-память (flash RAM) (электрически программируемое ПЗУ), которая играет роль диска для ОСРВ; типичная емкость - 4Мб.

Контроллеры периферийных устройств: SCSI (Small Computer System Interface), Ethernet,

COM портов, параллельного порта, несколько программируемых таймеров. На плате находится также контроллер и разъем шины, через которую компьютер управляет внешними устройствами. В качестве шины в подавляющем большинстве случаев используется шина VME, которую в последнее время стала теснить шина Compact PCI.

Несмотря на наличие контроллера SCSI, обычно ОСРВ работает без дисковых

накопителей, поскольку они не удовлетворяют предъявляемым к системам реального времени требованиям по надежности, устойчивости к вибрации, габаритам и времени готовности после включения питания.

Плата помещается в специальный корпус (крейт), в котором разведены разъемы шины и установлен блок питания. Корпус обеспечивает надлежащий температурный режим, защиту от пыли и механических повреждений. В этот же корпус вставляются платы аналого-цифровых и/или цифро-аналоговых преобразователей (АЦП и/или ЦАП), через которые осуществляется ввод/вывод управляющей информации, платы управления электромоторами. В тот же корпус могут вставляться другие такие же (или иные) промышленные компьютеры, образуя многопроцессорную систему

1.4. Тема: « Понятие операционных систем реального времени»

1.4.1. План:

1) Характеристики компьютерных операционных систем.

2) Операционные системы общего назначения.

1.4.2. Краткое содержание вопросов:

Одно из коренных внешних отличий систем реального времени от систем общего назначения - четкое разграничение систем разработки и систем исполнения. Система исполнения операционных системах реального времени - набор инструментов (ядро, драйверы, исполняемые модули), обеспечивающих функционирование приложения реального времени. Большинство современных ведущих операционных систем реального времени поддерживают целый спектр аппаратных архитектур, на которых работают системы исполнения (Intel, Motorola, RISC,MIPS, PowerPC, и другие). Это объясняется тем, что набор аппаратных средств является частью комплекса реального времени и аппаратура должн а быть также адекватна решаемой задаче. Поэтому ведущие операционные системы реального времени перекрывают целый ряд наиболее популярных архитектур, чтобы удовлетворить самым разным требованиям по части аппаратуры. Систему исполнения операционных системах реального времени и компьютер, на котором она исполняется называют "целевой" (target) системой.

Система разработки – это набор средств, обеспечивающих создание и отладку приложения реального времени.

Системы разработки работают, как правило, в популярных и распространенных ОС, таких, как UNIX. Кроме того, многие операционные системы реального времени имеют и так называемые резидентные средства разработки, исполняющиеся в среде самой операционной системы реального времени - особенно это относится к операционным системам реального времени класса "ядра".

Функционально средства разработки операционных систем реального времени

отличаются от привычных систем разработки, таких, например, как Developers Studio, TaskBuilder, так как часто они содержат средства удаленной отладки, средства профилирования (измерение времен выполнения отдельных участков кода), средства эмуляции целевого процессора, специальные средства отладки взаимодействующих задач, а иногда и средства моделирования. Рассмотрим основные параметры операционных системах реального времени.

Время реакции системы. Практически все производители систем реального времени приводят такой параметр, как время реакции системы на прерывание (interrupt latency). Если главным для системы реального времени является ее способность вовремя отреагировать на внешние события, то такой параметр, как время реакции системы является ключевым.

В настоящее время нет общепринятых методологий измерения этого параметра, поэтому он является полем битвы маркетинговых служб производителей систем реального времени. Но уже появился проект сравнения операционных систем реального времени, который включает в себя, в том числе, и разработку методологии тестирования. Рассмотрим времена, которые необходимо знать для того, чтобы предсказать время реакции системы. События, происходящие на объекте, регистрируются

датчиками, данные с датчиков передаются в модули ввода-вывода (интерфейсы) системы. Модули ввода-вывода, получив информацию от датчиков и преобразовав ее, генерируют запрос на прерывание в управляющем компьютере, подавая ему тем самым сигнал о том, что на объекте произошло событие. Получив сигнал от модуля ввода-вывода, система должна запустить программу обработки этого события. Интервал времени - от события на объекте и до выполнения первой инструкции в программе обработки этого события и является временем реакции системы на события. Проектируя систему реального времени, разработчики должны уметь вычислять этот интервал и знать из чего он складывается.

Время выполнения цепочки действий - от события на объекте до генерации прерывания -не зависит от операционных систем реального времени и целиком определяется аппаратурой, а интервал времени - от возникновения запроса на прерывание и до выполнения первой инструкции обработчика определяется целиком свойствами операционной системы и архитектурой компьютера. Причем это время нужно уметь оценивать в худшей для системы ситуации, то есть в предположении, что процессор загружен, что в это время могут происходить другие прерывания, что система может выполнять какие-то действия, блокирующие прерывания. Основанием для оценки времен реакции системы могут служить результаты тестирования с подробным описанием архитектуры целевой системы, в которой проводились измерения с точным указанием, какие промежутки времени измерялись. Некоторые производители операционных систем реального времени результаты такого тестирования предоставляют. Время переключения контекста.

В операционные системы реального времени заложен параллелизм, возможность одновременной обработки нескольких событий. Поэтому все операционные системы реального времени являются многозадачными (многопроцессными, многонитиевыми). Чтобы уметь оценивать накладные расходы системы при обработке параллельных событий, необходимо знать время, которое система затрачивает на передачу управления от процесса к процессу (от задачи к задаче, от нити к нити), то есть время переключения контекста.

1.5. Тема: « Обязательные требования к ОСРВ»

1.5.1.План:

1) Требования многозадачности.

2) Требования к работе с памятью.

1.5.2. Краткое содержание вопросов:

Требование 1. ОСРВ должна быть многонитиевой или многозадачной и поддерживать диспетчеризацию с вытеснением. Поведение ОСРВ должно быть предсказуемым. Это не означает, что ОСРВ должна быть быстрой, но означает, что максимальный промежуток времени для выполнения любой операции должен быть известен заранее и должен быть согласован с требованиями приложения. Например, Windows 3.11 - даже на процессоре Pentium Pro с тактовой частотой 200 МГц - неприменима для построения систем реального времени, поскольку одно приложение может навсегда захватить управление и заблокировать все остальные приложения.

Первое требование состоит в том, чтобы такая ОС была многонитие-вой или многозадачной и, кроме того, планировщик должен иметь возможность вытеснять любую нить (задачу) и передавать управление той нити (задаче), которая больше всего в этом нуждается. Для обеспечения вытеснения на уровне прерываний структура обслуживания прерываний (в том числе и аппаратная архитектура) должна быть многоуровневой.

Требование 2: Должно существовать понятие приоритета нити (задачи). Как найти нить (задачу), которая нуждается в ресурсах больше всего? В идеальном случае ОСРВ предоставляет ресурсы той задаче или драйверу, у которых осталось меньше всего времени до истечения срока реакции на событие (назовем такую ОС – ОС управляющей критическими сроками). Однако для реализации этого механизма нужно уметь прогнозировать, сколько времени понадобится задаче для завершения своей работы и сколько времени понадобится другим задачам для того, чтобы они успели к своим критическим срокам. Подобная ОСРВ пока еще не создана из-за сложности реализации. Поэтому разработчики ОС используют другой метод: они вводят концепцию приоритетов для нитей (задач).

При построении конкретной системы реального времени разработчик должен выстроить приоритеты задач таким образом, чтобы каждая из них успела с реакцией к своему критическому сроку, то есть он должен трансформировать базовое требование реального времени "успеть с реакцией к нужному моменту" в комбинацию приоритетов и в сценарий их динамического изменения. Очевидно, что при этой трансформации возможны ошибки, приводящие к неправильной работе системы. Для решения этого вопроса используют различные теории, такие как, теорию монотонного планирования или различные методы и средства моделирования. Однако, эти методы оказываются не всегда эффективными. Как бы то ни было, во всех современных ОСРВ приходится использовать механизм приоритетов как один из инструментов предсказуемости поведения системы. На сегодняшний день не имеется другого решения, понятие приоритета потока для систем реального времени неизбежно.

Требование 3: ОС должна поддерживать предсказуемые механизмы синхронизации нитей (задач). Все нити (задачи) разделяют данные (ресурсы) и должны обмениваться между собой информацией, поэтому необходимы механизмы межзадачного (межнитиевого) взаимодействия.

Требование 4: Должен существовать механизм наследования приоритетов (система должна быть защищена от инверсии приоритетов). Под инверсией приоритетов будем понимать изменение их обычного порядка. На самом деле именно эти механизмы синхронизации и тот факт, что разные нити выполняются в одном и том же пространстве памяти, и определяют различие между нитями и процессами. Процессы не разделяют одно и то же пространство памяти.

Комбинации приоритетов нитей и разделение между ними ресурсов приводит к классической проблеме инверсии приоритетов. Для создания условия инверсии приоритетов должно быть задействовано как минимум три нити. Если нить с самым низким приоритетом заблокировала ресурс (который она делит с самой высокоприоритетной нитью), в то время как работает нить с промежуточным приоритетом, возникает следующий эффект: нить с наивысшим приоритетом ожидает освобождения ресурса, нить с промежуточным приоритетом вытесняет низкоприоритетную нить и работает, пока не завершится; управление получает

низкоприоритетная нить, которая освобождает ресурс, и только после этого нить с высоким приоритетом может продолжить свою работу. В этом случае время, необходимое для завершения нити с наивысшим приоритетом, зависит от времени работы нити с более низким приоритетом – это и есть инверсия приоритетов. Очевидно, что в такой ситуации высокоприоритетная нить может "прозевать" критическое событие.

Чтобы избежать таких ситуаций, ОСРВ должна быть снабжена механизмом наследования приоритетов, то есть блокирующая нить должна наследовать приоритет нити, которую она блокирует (конечно, только, в том случае, если заблокированная нить имеет более высокий приоритет). Поведение ОС должно быть предсказуемо. Наследование означает, что блокирующий ресурс тред наследует приоритет треда, который он блокирует (это справедливо лишь в том случае, если блокируемый тред имеет более высокий приоритет). Здесь есть еще одна проблема: количество возможных приоритетов очень мало. Большинство современных ОСРВ допускают использование как минимум 256 приоритетов. В чем суть проблемы? Ответ очевиден: чем больше приоритетов в распоряжении проектировщика, тем более предсказуемую систему можно создать. При оптимальном проектировании системы различным нитям присваиваются различные приоритеты.

Рассмотрим временные требования к операционным системам. Разработчик должен знать все времена выполнения системных вызовов и уметь предсказывать поведение системы в любых ситуациях. Поэтому производитель ОСРВ обязательно должен давать информацию о следующих временных характеристиках системы:

- задержке прерывания (interrupt latency) - то есть время от момента появления запроса на прерывание до начала его обработки;

- максимальном времени исполнения каждого системного вызова. Оно должно быть предсказуемым и не зависеть от количества объектов в системе;

- максимальном времени, на которое ОС и драйверы могут блокировать прерывания.

Разработчик также должен знать и учитывать следующее:

- уровни системных прерываний;

- уровни прерываний устройств, максимальное время, которое занимают программы обработки прерываний, и т.д.

Если все перечисленные выше времена известны, то имеются все предпосылки для создания системы жесткого реального времени. При этом требования к производительности разрабатываемой системы должны быть согласованы с характеристиками выбранной ОСРВ и аппаратуры.Те места в программах, в которых происходит обращение к критическим ресурсам, называются критическими секциями. Решение этой проблемы заключается в организации такого доступа к критическому ресурсу, когда только одному процессу разрешается входить в критическую секцию.

Ресурсы, которые не допускают одновременного использования несколькими процессами, называются критическими. Если нескольким вычислительным ресурсам

необходимо пользоваться критическим ресурсом в режиме разделения, им следует синхронизировать свои действия таким образом, чтобы ресурс всегда находился в распоряжении не более чем одного из процессов.

Любая система реального времени взаимодействует с внешним миром через аппаратуру компьютера. Внешние события преобразуются в прерывания и обрабатываются драйвером устройства.

Доступ к аппаратуре имеют только драйверы. Поскольку приложения реального времени часто работают со специфическими внешними устройствами, требующими и специфического управления, разработчик системы реального времени должен уметь разрабатывать драйверы устройств.

В ОСРВ разработчик в первую очередь узнает, на каких приоритетах работают драйверы

других устройств. Здесь обычно существует свободное пространство для прерываний с приоритетами, которые выше приоритетов стандартных драйверов.

Требование 5: Политика управления памятью в ОСРВ. При проектировании системы реального времени необходимо рассмотреть и другой важный вопрос: как строится политика управления памятью в ОСРВ? От решения этой проблемы во многом зависит быстродействие проектируемой системы.

Требования, накладываемые на вычислительную установку реального времени, формулируются следующим образом:

1. В зависимости от сложности программы управления, требование «реального времени» накладывает различные условия на вычислительную мощность процессора для СРВ.

2. Внешние события становятся известны системе посредством прерываний (interrupt requests (IRQ)) (т.е. запросов на обслуживание со стороны внешних устройств). Поэтому часто для ОСРВ более важна не мощность процессора, а характеристики компьютера, связанные с подсистемой прерываний. Желательными являются:

- наличие как можно большего количества уровней прерываний (IRQ levels) (т.е. аппаратного или/и программного декодирования источника запроса);

- как можно меньшее время реакции на прерывание (т.е. как можно меньшее время между поступлением запроса на обслуживание и началом выполнения обслуживающей программы).

3. СРВ часто сама является инициатором периодических процессов, которыми управляет (например, движением космического аппарата или луча радара). Поэтому необходимо иметь в наличии один или несколько таймеров (аппаратных устройств, выдающих прерывание через заданные промежутки времени), которые могут работать в периодическом или ждущем режиме.

4. Ввиду того, что СРВ часто управляет ответственными промышленными процессами, данное обстоятельство выдвигает очень жесткие требования к надежности используемого оборудования.

В течение длительного времени основными потребителями СРВ были военная и космическая области. Сейчас ситуация кардинально изменилась и СРВ можно встретить даже в товарах широкого потребления.

Рассмотрим основные области применения СРВ.

5. Основные области применения систем реального времени Военная и космическая области:

- бортовое и встраиваемое оборудование;

- системы измерения и управления, радары;

- цифровые видеосистемы, симуляторы;

- ракеты, системы определения положения и привязки к местности.

Промышленность:

- автоматические системы управления производством (АСУП), автоматические системы управления технологическим процессом (АСУТП);

- автомобилестроение: симуляторы, системы управления двигателем, автоматическое сцепление, системы антиблокировки колес и т.д.;

- энергетика: сбор информации, управление данными и оборудованием;

- телекоммуникации: коммуникационное оборудование, сетевые коммутаторы, телефонные станции и т.д.;

- банковское оборудование (например, во многих банкоматах работает СРВ QNX).

Товары широкого потребления:

- мобильные телефоны (например, в телефонах стандарта GSM работает СРВ pSOS);

- цифровые телевизионные декодеры;

- цифровое телевидение (мультимедиа, видеосерверы);

- компьютерное и офисное оборудование (принтеры, копиры), например, в факсах применяется СРВ VxWorks, в устройствах чтения компакт-дисков – СРВ VRTX32.

1.6. Тема: « Примеры ОСРВ. Система QNX»

1.6.1. План:

1) Коммерческие ОСРВ.

2) Операционная система QNX

1.6.2. Краткое содержание вопросов:

Операционная система QNX является мощной операционной системой, позволяющей проектировать сложные программные системы, работающие в реальном времени как на одном компьютере, так и в локальной вычислительной сети. Встроенные средства операционной системы QNX обеспечивают поддержку многозадачного режима на одном компьютере и взаимодействие параллельно

выполняемых задач на разных компьютерах, работающих в среде локальной вычислительной сети. Основным языком программирования в системе является язык С. Основная операционная среда соответствует стандартам POSIX-интерфейса. Это позволяет с небольшими доработками перенести необходимое накопленное программное обеспечение в среду операционной системы QNX для организации их работы в среде распределенной обработки.

ОС QNX является сетевой, мультизадачной, многопользовательской (многотерминальной) и масштабируемой. С точки зрения пользовательского интерфейса и API она похожа на UNIX. Однако QNX - это не версия UNIX. QNX была разработана канадской фирмой QNX Software Systems Limited в 1989 году по заказу Министерства обороны США. Причем эта система построена на совершенно других архитектурных принципах, отличных от принципов, использованных при создании ОС UNIX. QNX была первой коммерческой ОС, построенной на принципах микроядра и обмена сообщениями. Система реализована в виде совокупности независимых (но взаимодействующих через обмен сообщениями) процессов различного уровня (менеджеры и драйверы), каждый из которых реализует определенный вид сервиса. Эти идеи позволили добиться нескольких важнейших преимуществ.

Предсказуемость, означающая ее применимость к задачам жесткого реального времени. QNX является операционной системой, которая дает полную гарантию в том, что процесс с наивысшим приоритетом начнет выполняться практически немедленно, и что критическое событие (например, сигнал тревоги) никогда не будет потеряно. Ни одна версия UNIX не может достичь подобного качества, поскольку код ядра слишком велик. Любой системный вызов из обработчика прерывания в UNIX может привести к непредсказуемой задержке (то же самое касается Windows NT).

Масштабируемость и эффективность, достигаемые оптимальным использованием ресурсов и означающие ее применимость для встроенных систем. Драйверы и менеджеры можно запускать и удалять (кроме файловой системы) динамически, из командной строки. Вы можете иметь только тот сервис, который вам реально нужен, причем это не требует серьезных усилий и не порождает проблем.

Расширяемость и надежность одновременно, поскольку написанный вами драйвер не нужно компилировать в ядро. Менеджеры ресурсов (сервис логического уровня) работают в кольце защиты 3, и возможно добавлять свои, не опасаясь за систему. Драйверы работают в кольце с уровнем привилегий 1 и могут вызвать проблемы, но не фатального характера. Кроме того, их достаточно просто писать и отлаживать.

Быстрый сетевой протокол FLEET, прозрачный для обмена сообщениями, автоматически обеспечивающий отказоустойчивость, балансирование нагрузки и маршрутизацию между альтернативными путями доступа.

Компактная графическая подсистема Photon, построенная на тех же принципах модульности, что и сама ОС, позволяет получить полнофункциональный графический интерфейс пользователя GUI, работающий вместе с POSIX-совместимой ОС всего в 4 Мбайт памяти, начиная с i80386 процессора.

1.7. Тема: « Особенности программирования для СРВ»

1.7.1. План:

1) Требования к языкам программирования для СРВ

2) Понятие параллельного программирования.

1.7.2. Краткое содержание вопросов:

1. Последовательное программирование и программирование задач реального времени Программа представляет собой описание объектов - констант и переменных - и операций, совершаемых над ними. Таким образом, программа -это чистая информация. Ее можно записать на какой-либо носитель, например на бумагу или на дискету.Программы можно создавать и анализировать на нескольких уровнях абстракции (детализации) с помощью соответствующих приемов формального описания переменных и операций, выполняемых на каждом уровне. На самом нижнем уровне используются непосредственное описание - для каждой переменной указывается ее размер и адрес в памяти. На более высоких уровнях переменные имеют абстрактные имена, а операции сгруппированы в функции или процедуры. Программист, работающий на высоком уровне абстракции, не должен думать о том, по каким реальным адресам памяти хранятся переменные, и о машинных командах, генерируемых компилятором. Последовательное программирование (sequential programming) является наиболее распространенным способом написания программ. Понятие "последовательно" подразумевает, что операторы программы выполняются в известной последовательности один за другим.

Целью последовательной программы является преобразование входных данных, заданных в определенной форме, в выходные данные, имеющие другую форму, в соответствии с некоторым алгоритмом - методом решения Программирование в реальном времени (real-time programming) отличается от последовательного программирования - разработчик программы должен постоянно иметь в виду среду, в которой работает программа, будь то контроллер микроволновой печи или устройство управления манипулятором робота. В системах реального времени внешние сигналы, как правило, требуют немедленной реакции процессора. В сущности, одной из наиболее важных особенностей систем реального времени является время реакции на входные сигналы, которое должно удовлетворять заданным ограничениям. Специальные требования к программированию в реальном времени, в частности необходимость быстро реагировать на внешние запросы, нельзя адекватно реализовать с помощью обычных приемов последовательного программирования. Насильственное последовательное расположение блоков программы, которые должны выполняться параллельно, приводит к неестественной запутанности результирующего кода и вынуждает связывать между собой функции, которые, по сути, являются самостоятельными. В большинстве случаев применение обычных приемов последовательного программирования не позволяет построить систему реального времени. В таких системах независимые программные модули или задачи должны быть активными одновременно, то есть работать параллельно, при этом каждая задача выполняет свои специфические функции. Такая техника известна под названием параллельного программирования (concurrent programming). В названии делается упор на взаимодействие между отдельными программными модулями. Параллельное

исполнение может осуществляться на одной или нескольких ЭВМ, связанных распределенной сетью.

1.8. Тема: « Программирование на C++ для СРВ»

1.8.1. План:

1) Система Microsoft Visual Studio 2010

2) Программирование на C++.

1.8.2. Краткое содержание вопросов:

В силу преимуществ объектно-ориентированного подхода приложения создаются на его основе, используя один из языков программирования, наилучшим образом поддерживающий этот подход. Архитектуры же классических операционных систем реального времени основаны на архитектурах UNIX систем и используют традиционный процедурный подход к программированию. Сочетание объектно-ориентированных приложений и процедурных операционных систем имеет ряд недостатков:

1. Происходит разрыв парадигмы программирования: в едином работающем комплексе (приложение + ОСРВ) разные компоненты используют разные подходы к разработке программного обеспечения.

2. Не используются все возможности объектно-ориентированного подхода.

3. Возникают некоторые потери производительности из-за разного типа интерфейсов в ОСРВ и приложении.

Естественно, возникает идея строить саму СРВ, используя объектно-риентированный подход. При этом:

- как приложение, так и операционная система полностью объектно-ориентированны и используют все преимущества этого подхода;

- приложение и ОСРВ могут быть полностью интегрированы, поскольку используют один объектно-ориентированный язык программирования;

- обеспечивается согласование интерфейсов ОСРВ и приложения;

- приложение может «моделировать» ОСРВ для своих потребностей, заказывая нужные ему объекты;

- единый комплекс (приложение + ОСРВ) является модульным и легко модернизируемым. Идея реализована в ОСРВ SoftKernel, целиком написанной на C++. ОСРВ с монолитной архитектурой можно представить в виде:

- прикладного уровня: состоящего из работающих прикладных процессов;

- системного уровня: состоящего из монолитного ядра операционной системы, в котором можно выделить следующие части

а) интерфейс между приложениями и ядром (API);

б) собственно ядро системы;

в) интерфейс между ядром и оборудованием (драйверы устройств). API в таких системах играет двойную роль:

1) управляет взаимодействием прикладных процессов и системы;

2) обеспечивает непрерывность выполнения кода системы (отсутствие переключения задач во время исполнения кода системы).

Основным преимуществом монолитной архитектуры является ее относительная быстрота работы по сравнению с другими архитектурами. Однако, достигается это, в основном, за счет написания значительных частей системы на ассемблере.

1.9. Тема: « Процессы, потоки, задачи»

1.9.1. План:

1) Модели состояний процессов.

2) Понятие потока.

3) Преимущества многопоточности.

1.9.2. Краткое содержание вопросов:

Выполнение — это активное состояние, во время которого процесс обладает всеми необходимыми ему ресурсами. В этом состоянии процесс непосредственно выполняется процессором.

Ожидание — это пассивное состояние, во время которого процесс заблокирован и не может быть выполнен, потому что ожидает какое-то событие, например, ввода данных или освобождения нужного ему устройства.

Готовность — это тоже пассивное состояние, процесс тоже заблокирован, но в отличие от состояния ожидания, он заблокирован не по внутренним причинам (ведь ожидание ввода данных — это внутренняя, «личная» проблема процесса — он может ведь и не ожидать ввода данных и свободно выполняться — никто ему не мешает), а по внешним, независящим от процесса, причинам.

Когда процесс может перейти в состояние готовности? Предположим, что наш процесс выполнялся до ввода данных. До этого момента он был в состоянии выполнения, потом перешел в состояние ожидания — ему нужно подождать, пока мы введем нужную для работы процесса информацию. Затем процесс хотел уже перейти в состояние выполнения, так как все необходимые ему данные уже введены, но не тут-то было: так как он не единственный процесс в системе, пока он был в состоянии ожидания, его «место под солнцем» занято — процессор выполняет другой процесс. Тогда нашему процессу ничего не остается как перейти в состояние готовности: ждать ему нечего, а выполняться он тоже не может.

Из состояния готовности процесс может перейти только в состояние выполнения. В состоянии выполнения может находится только один процесс на один

процессор. Если у вас n – процессорная машина, у вас одновременно в состоянии выполнения могут быть n процессов.

Из состояния выполнения процесс может перейти либо в состояние ожидания, либо в состояние готовности. Почему процесс может оказаться в состоянии ожидания, мы уже знаем — ему просто нужны дополнительные данные или он ожидает освобождения какого-нибудь ресурса, например, устройства или файла. В состояние готовности процесс может перейти, если во время его выполнения квант времени выполнения «вышел». Другими словами, в операционной системе есть специальная программа — планировщик, которая следит за тем, чтобы все процессы выполнялись отведенное им время. Например, у нас есть три процесса. Один из них находится в состоянии выполнения. Два других — в состоянии готовности.

Планировщик следит за временем выполнения первого процесса, если «время вышло», планировщик переводит процесс 1 в состояние готовности, а процесс 2 — в состояние выполнения. Затем, когда, время отведенное, на выполнение процесса 2, закончится, процесс 2 перейдет в состояние готовности, а процесс 3 — в состояние выполнения.

Более сложная модель — это модель, состоящая из пяти состояний. В этой моделипоявилось два дополнительных состояния: рождение процесса и смерть процесса. Рождение процесса — это пассивное состояние, когда самого процесса еще нет, но уже готова структура для появления процесса. Смерть процесса — самого процесса уже нет, но может случиться, что его «место", то есть структура данных осталась в списке процессов. Такие процессы называются зобми.

Потоки концепция процесса, пришедшая из мира UNIX, плохо реализуется в многозадачной системе, поскольку процесс имеет тяжелый контекст. Возникает понятие потока (thread), который понимается как подпроцесс, или легковесный процесс (light-weight process), выполняющийся в контексте полноценного процесса. С помощью процессов можно организовать параллельное выполнение программ. Для этого процессы клонируются вызовами fork() или exec(), а затем между ними организуется взаимодействие средствами IPC. Это довольно дорогостоящий в отношении ресурсов способ. С другой стороны, для организации параллельного выполнения и взаимодействия процессов можно использовать механизм многопоточности. Основной единицей здесь является поток, который представляет собои облегченную версию процесса. Чтобы понять, в чем состоит его особенность, необходимо вспомнить основные характеристики процесса. Процесс располагает определенными ресурсами. Он размещен в некотором виртуальном адресно м пространстве, содержашем образ этого процесса. Кроме того, процесс управляет другими ресурсами (фаилы, устроиства ввода/вывода и т.д.). Процесс подвержен диспетчеризации. Он определяет порядок выполнения одной или нескольких программ, при этом выполнение может перекрываться другими процессами. Каждый процесс имеет состояние выполнения и приоритет диспетчеризации.

Если рассматривать эти характеристики независимо друг от друга (как это принято в современной теории ОС), то:

владельцу ресурса, обычно называемому процессом или задачей, присущи:

виртуальное адресное пространство;

индивидуальный доступ к процессору, другим процессам, файлам, и ресурсам ввода — вывода.

Модулю для диспетчеризации, обычно называемому потоком или облегченным процессом, присущи

- состояние выполнения (активное, готовность и т.д.);
- сохранение контекста потока в неактивном состоянии;
- стек выполнения и некоторая статическая память для локальных переменных;
- доступ к пространству памяти и ресурсам своего процесса.

Все потоки процесса разделяют общие ресурсы. Изменения, вызванные одним потоком, становятся немедленно доступны другим. При корректной реализации потоки имеют определенные преимущества перед процессами.

Им требуется

- меньше времени для создания нового потока, поскольку создаваемый поток использует адресное пространство текущего процесса;
- меньше времени для завершения потока;
- меньше времени для переключения между двумя потоками в пределах процесса;
- меньше коммуникационных расходов, поскольку потоки разделяют все ресурсы, и в частности адресное пространство.

Данные, продуцируемые одним из потоков, немедленно становятся доступными всем другим потокам.

Преимущества многопоточности

Если операционная система поддерживает концепции потоков в рамках одного процесса, она называется многопоточной. Многопоточные приложения имеют ряд преимуществ:

Улучшенная реакция приложения — любая программа, содержащая много не зависящих друг от друга действий, может быть перепроектирована так, чтобы каждое действие выполнялось в отдельном потоке. Например, пользователь многопоточного интерфейса не должен ждать завершения одной задачи, чтобы начать выполнение другой.

Более эффективное использование мультипроцессирования — как правило, приложения, реализующие параллелизм через потоки, не должны учитывать число доступных процессоров.

Производительность приложения равномерно увеличивается при наличии дополнительных процессоров. Численные алгоритмы и приложения с высокой степенью параллелизма, например перемножение матриц, могут выполняться намного быстрее. Улучшенная структура программы — некоторые программы более эффективно представляются в виде нескольких независимых или полуавтономных единиц, нем в виде единой монолитной программы. Многопоточные программы легче адаптировать к изменениям требований пользователя.

Эффективное использование ресурсов системы — программы, использующие два или более процессов, которые имеют доступ к общим данным через разделяемую

память, содержат более одного потока управления. При этом каждый процесс имеет полное адресное пространство и состояние в операционной системе. Стоимость создания и поддержания большого количества служебной информации делает каждый процесс более затратным, чем поток. Кроме того, разделение работы между процессами может потребовать от программиста значительных усилий, чтобы обеспечить связь между потоками в различных процессах или синхронизировать их действия.

Задачи

Как уже говорилось, СРВ — это программно-аппаратный комплекс, осуществляющий мониторинг какого-то объекта и/или управление им в условиях временнЫх ограничений.

Возникающие на объекте события подлежат обработке в СРВ. Будем сопоставлять каждому типу события задачу.

ЗАДАЧА (TASK) — блок программного кода, ответственный за обработку тех или иных событий, возникающих на объекте управления.

Задача может быть «оформлена» в виде:

- Отдельного процесса Потока управления внутри процесса (нити, легковесного процесса)
- Обработчика прерывания/подпрограммы

РАБОТА ЗАДАЧИ (JOB) — процесс исполнения блока программного кода в ходе

обработки события.

Каждая работа задачи характеризуется следующими временнЫми параметрами:

r (Release Time) — момент времени, когда задача становится готовой к исполнению (например, процесс переходит в состояние готовности)

d (Absolute Deadline) — абсолютный крайний срок, момент времени, к которому задача должна завершить очередную работу.

s (Start Time) — момент времени, когда задача начала исполняться на процессоре

c (Compltion Time) — момент времени, когда задача закончила работу, обработав событие

D (Relative Deadline) — относительный крайний срок. D = d — r

e (Execution Time) — время исполнения задачи при выполнении ею очередной работы. e

= c — s

R (Response Time) — время отклика. R = c — r

1.10. Тема: « Управление процессами. Диспетчеризация Windows»

1.10.1. Вопросы лекции:

1) Общие принципы управления процессами.

2) Алгоритмы планирования.

3) Управление процессами в Windows.

1.10.2. Краткое содержание вопросов:

В операционной системе UNIX традиционно поддерживается классическая схема мультипрограммирования. Система поддерживает возможность параллельного (или квази-параллельного в случае наличия только одного аппаратного процессора) выполнения нескольких пользовательских программ. Каждому такому выполнению соответствует процесс операционной системы. Каждый процесс выполняется в собственной виртуальной памяти, и, тем самым, процессы защищены один от другого, т.е один процесс не в состоянии неконтроллируемым образом прочитать что-либо из памяти другого процесса или записать в нее. Однако контролируемые взаимодействия процессов допускаются системой, в том числе за счет возможности разделения одного сегмента памяти между виртуальной памятью нескольких процессов.

Конечно, не менее важно (а на самом деле, существенно более важно) защищать саму операционную систему от возможности ее повреждения каким бы то ни было пользовательским процессом. В ОС UNIX это достигается за счет того, что ядро системы работает в собственном "ядерном" виртуальном пространстве, к которому не может иметь доступа ни один пользовательский процесс.

Ядро системы предоставляет возможности (набор системных вызовов) для порождения новых процессов, отслеживания окончания порожденных процессов и т.д. С другой стороны, в ОС UNIX ядро системы - это полностью пассивный набор программ и данных. Любая программа ядра может начать работать только по инициативе некоторого пользовательского процесса (при выполнении системного вызова), либо по причине внутреннего или внешнего прерывания (примером внутреннего прерывания может быть прерывание из-за отсутствия в основной памяти требуемой страницы виртуальной памяти пользовательского процесса; примером внешнего прерывания является любое прерывание процессора по инициативе внешнего устройства). В любом случае считается, что выполняется ядерная часть обратившегося или прерванного процесса, т.е. ядро всегда работает в контексте некоторого процесса.

В последние годы в связи с широким распространением так называемых симметричных мультипроцессорных архитектур компьютеров (Symmetric Multiprocessor Architectures - SMP) в ОС UNIX был внедрен механизм легковесных процессов (light-weight processes), или нитей, или потоков управления (threads). Говоря по-простому, нить - это процесс, выполняющийся в виртуальной памяти, используемой совместно с другими нитями того же "тяжеловесного" (т.е. обладающего отдельной виртуальной памятью) процесса. В принципе, легковесные процессы использовались в операционных системах много лет назад. Уже тогда стало ясно, что программирование с неконтролируемым использованием общей памяти приносит больше хлопот и неприятностей, чем пользы, по причине необходимости использования явных примитивов синхронизации.

Однако, до настоящего времени в практику программистов так и не были внедрены более безопасные методы параллельного программирования, а реальные возможности мультипроцессорных архитектур для обеспечения распараллеливания нужно было как-то использовать. Поэтому опять в обиход вошли легковесные процессы, которые теперь получили название threads (нити). Наиболее важно (с нашей точки зрения) то, что для внедрения механизма нитей потребовалась существенная переделка ядра. Разные производители аппаратуры и программного обеспечения стремились как можно быстрее выставить на рынок продукт, пригодный для эффективного использования на SMP-платформах. Поэтому версии ОС UNIX опять несколько разошлись.

1.11. Тема: « Управление памятью в ОСРВ»

1.11.1. План:

1) Требования к управлению памятью в ОСРВ.

2) Модели защиты памяти.

3) Часы и таймеры

1.11.2. Краткое содержание вопросов:

Время задержки на переключение контекста потока напрямую зависит от конфигурации памяти, т.е. от модели защиты памяти. Рассмотрим четыре наиболее распространенных в ОСРВ модели защиты памяти.

Модель без защиты

- системное и пользовательское адресные пространства не защищены друг от друга, используется два сегмента памяти: для кода и для данных; при этом от системы не требуется никакого управления памятью, не требуется MMU (memory management unit – специальное аппаратное устройство для поддержки управления виртуальной памятью).

Модель защиты "система/пользователь"

– системное адресное пространство защищено от адресного пространства пользователя, системные и пользовательские процессы выполняются в общем виртуальном адресном пространстве, при этом требуется MMU. Защита обеспечивается страничным механизмом защиты.

Различаются системные и пользовательские страницы. Пользовательские приложения никак не защищены друг от друга. Процессор находится в режиме супервизора, если текущий сегмент имеет уровень 0, 1 или 2. Если уровень сегмента – 3, то процессор находится в пользовательском режиме. В этой модели необходимы четыре сегмента – два сегмента на уровне 0 (для кода и данных) и два сегмента на уровне 3.

Механизм страничной защиты не добавляет накладных расходов, т.к. защита проверяется одновременно с преобразованием адреса, которое выполняет MMU, при этом ОС не нуждается в управлении памятью.

Модель защиты "пользователь/пользователь"

– к модели система/пользователь добавляется защита между пользовательскими процессами; требуется MMU.

Как и в предыдущей модели, используется механизм страничной защиты. Все страницы помечаются как привилегированные, за исключением страниц текущего процесса, которые помечаются как пользовательские. Таким образом, выполняющийся поток не может обратиться за пределы своего адресного пространства.

ОС отвечает за обновление флага привилегированности для конкретной страницы в таблице страниц при переключении процесса. Как и в предыдущей модели используются четыре сегмента.

Модель защиты виртуальной памяти

- каждый процесс выполняется в своей собственной виртуальной памяти, требуется MMU.

У каждого процесса имеются свои собственные сегменты и, следовательно, своя таблица описателей.

ОС несет ответственность за поддержку таблиц описателей. Адресуемое пространство может превышать размеры физической памяти, если используется страничная организация памяти совместно с подкачкой. Однако в системах реального времени подкачка обычно не применяется из-за ее непредсказуемости. Для решения этой проблемы доступная память разбивается на фиксированное число логических адресных пространств равного размера. Число одновременно выполняющихся процессов в системе становится ограниченным.

Фундаментальное требование к памяти в системе реального времени заключается в том, что время доступа к ней (памяти) должно быть ограничено (или, другими словами, предсказуемо). Прямым следствием становится запрет на использование для процессов реального времени техники вызова страниц по запросу (подкачка с диска). Поэтому системы, обеспечивающие механизм виртуальной памяти, должны уметь блокировать процесс в оперативной памяти, не допуская подкачки. Итак, подкачка недопустима в ОСРВ, потому что непредсказуема.

Если поддерживается страничная организация памяти (paging), соответствующее отображение страниц в физические адреса должно быть частью контекста процесса. Иначе опять появляется непредсказуемость, неприемлемая для ОСРВ.

Для процессов, не являющихся процессами жесткого реального времени, возможно использование механизма динамического распределения памяти, однако при этом ОСРВ должна поддерживать обработку таймаута на запрос памяти, т.е. ограничение на предсказуемое время ожидания.

В обычных ОС при использовании механизма сегментации памяти для борьбы с фрагментацией применяется процедура уплотнения после сборки мусора. Однако такой подход неприменим в среде реального времени, т.к. во время уплотнения перемещаемые задачи не могут выполняться, что ведет к непредсказуемости системы. В этом состоит основная проблема применимости объектно-ориентированного подхода к системам реального времени. До тех пор, пока проблема уплотнения не будет решена,

C++ и JAVA останутся не самым лучшим выбором для систем жесткого реального времени.

1.12. Тема: « Методы и средства обработки событий»

1.12.1. План:

1) Средства межпроцессного взаимодействия.

2) Проблемы, возникающие при организации параллельных вычислений.

3) Виды каналов связи процессов.

1.12.2. Краткое содержание вопросов:

1. Обобщенная функциональная структура информационного тракта СРВ и устройства связи с объектом

Из всего состава функциональных устройств СРВ, образующих информационный тракт системы, рассмотрим только те, которые осуществляют функции сбора, предварительной обработки, представления, передачи и обработки информации. Блок-схема обобщенной функциональной структуры информационного тракта и устройства связи с объектом представлены на рис. 1.

На вход системы поступает в общем случае аналоговый сигнал S(t), формированный информационным устройством (или датчиком), являющимся источником данных. Сигнал S(t) рассматривается как реализация случайного процесса. Цепь преобразования данных одного устройства (или датчика) в многоканальной системе образует измерительный канал. В блоке подготовки сигнал подвергается предварительной аналоговой обработке – согласованию, усилению (приведение амплитуды к динамическому диапазону устройством выборки и хранения – УВХ), полосовой фильтрации (ограничение полосы частот сигналов для корректной оцифровки).

Поскольку подсистема обработки является цифровой системой, то каждый сигнал подвергается процедуре аналого-цифрового преобразования в модуле АЦП. Последовательность отсчетов от различных измерительных каналов объединяется в общий поток для последующего ввода в компьютер или передачи по каналу связи. В ряде случаев могут применяться устройства сжатия данных (либо сжатие осуществляется после ввода данных в компьютер – программные методы сжатия). Состав и последовательность расположения функциональных устройств в различных СРВ может отличаться от приведенной в блок-схеме. Но, характерным является наличие данных устройств, как типовых в системах различного назначения и технического воплощения.

Подсистема передачи включает кодер и декодер канала связи, передающее и приемное устройства и собственно канал связи (среда с антенными устройствами). Кодер и декодер осуществляют помехоустойчивое кодирование и декодирование сигналов с целью дополни дополнительной защиты передаваемых сообщений от помех в канале связи и могут отсутствовать при наличии качественного канала.

Восстановление исходного аналогового сообщения по цифровым отсчетам с допустимой погрешностью производится на приемной стороне. В современных системах восстановление непрерывного сообщения, как правило, не выполняется,

поскольку регистрация, хранение и обработка информации выполняются в цифровом виде, но принципиальная возможность восстановления предусматривается.

Одна из задач подсистемы цифровой обработки, которая выполняется с использованием ресурсов компьютера и специализированных процессоров цифровой обработки – сортировка информации и отбраковка аномальных результатов наблюдений. Отбраковка является частным случаем более общей задачи – фильтрации сигналов от помех или использования методов распознавания образов. Другими задачами подсистемы обработки являются:

предварительная обработка данных (сглаживание, удаление тренда);

статистическая обработка сигналов (применяются различные алгоритмы в зависимости от назначения СРВ);

спектральная обработка;

формирование моделей процессов и явлений;

представление результатов предварительной обработки или анализа;

хранение данных.

Исходная информация для последующего анализа исследуемого явления (или объекта)формируется с помощью средств проведения эксперимента, представляюших собой совокупность средств измерений различных типов (измерительных устройств, преобразователей, датчиков и принадлежностей к ним), каналов передачи информации и вспомогательных устройств для обеспечения условий проведения эксперимента. В различных предметных областях совокупность средств для проведения эксперимента может называться по-разному (например, экспериментальная установка, информационно-измерительная система, измерительная система). В дальнейшем будем пользоваться термином "измерительная система"(ИС).

В зависимости от целей эксперимента иногда различают измерительные информационные (исследование), измерительные контролирующие (контроль, испытание) и измерительные управляющие (управление, оптимизация) системы, которые различаются в общем случае как составом оборудования, так и сложностью обработки экспериментальных данных.

2. МЕТОДИЧЕСКИЕ УКАЗАНИЯ
ПО ПРОВЕДЕНИЮ ПРАКТИЧЕСКИХ ЗАНЯТИЙ

2.1. Практическое занятие №1

Тема: «Понятие о системах реального времени».

2.1.1. Задание для работы:

Определение системы реального времени.

Системы постоянной готовности.

Системы жесткого и мягкого реального времени.

2.1.2. Краткое описание проводимого занятия:

2.1.2.1. Ответы на вопросы семинарского (практического) занятия.

2.1.2.2. Проведение текущего контроля успеваемости

Задания для проведения текущего контроля успеваемости

1. В предсказуемом поведении нуждается большинство:

+ a) встроенных систем

b) независимых систем

c) экономических систем

d) динамических систем

2. Принято различать системы:

+ a) мягкого и жесткого реального времени

b) длинного и короткого реального времени

c) одиночного и коллективного реального времени

d) белого и черного реального времени

3. Как в системах реального времени называется срок, до истечения которого задача должна обязательно выполниться.

+ a) Deadline

b) Bloodline

c) Redline

d) Lifeline

2.1.3 Результаты и выводы: в результате выполнения работы были изучены основные понятия систем реального времени.

2.2. Практическое занятие № 2

Тема: «Требования к системам реального времени».

2.2.1. Задание для работы

Требования к жестким системам реального времени.

Требования к мягким системам реального времени.

Составляющие элементы систем реального времени.

2.2.2. Краткое описание проводимого занятия:

2.2.2.1. Ответы на вопросы семинарского (практического) занятия.

2.2.2.2. Проведение текущего контроля успеваемости

Задания для проведения текущего контроля успеваемости

1. Ошибкой в системах реального времени является

+ отсутствие реакции в предсказуемое время

реакция в непредсказуемое время

невозможность взаимодействия со специальной аппаратурой

небезопасность работы

2. Цена опоздания в системах жесткого реального времени считается

+ бесконечно большой

больше чем цена самой системы

больше чем цена используемого оборудования

равной цене вышедшего из строя оборудования

3. Обработка аудио данных считается процессом реального времени, если для анализа

2,00 секунд звука требуется

+ 1,99 секунды

2,01 секунды

ровно 2,00 секунды

4,00 секунды

2.2.3 Результаты и выводы: в результате выполнения работы были изучены требования к системам реального времени.

2.3. Практическое занятие № 3

Тема: «Аппаратурная среда систем реального времени».

2.3.1. Задание для работы:

Виды вычислительных установок систем реального времени.

Требования к вычислительным установкам систем реального времени.

Особенности аппаратного обеспечения систем реального времени.

2.3.2. Краткое описание проводимого занятия:

2.3.2.1. Ответы на вопросы семинарского (практического) занятия.

2.3.2.2. Проведение текущего контроля успеваемости

Задания для проведения текущего контроля успеваемости

1. Подкачка недопустима в ОСРВ, потому что она:

+ a) непредсказуема

b) предсказуема

c) ограничена

d) неограниченна

2. Для отсчета временных интервалов на основе часов реального времени создаются:

+ a) таймеры

b) секундомеры

c) механические часы

d) электронные часы

3. Компактные и дешевые устройства, предназначенные для решения несложных типовых задач автоматизации управления во встроенных и бортовых системах:

+ a) микроконтроллеры

b) контроллеры

c) адаптеры

d) датчики

2.3.3 Результаты и выводы: в результате выполнения работы была изучена аппаратурная среда систем реального времени.

2.4. Практическое занятие № 4 .

Тема: «Понятие операционных систем реального времени».

2.4.1. Задание для работы:

Характеристики компьютерных операционных систем

Операционные системы общего назначения.

2.4.2. Краткое описание проводимого занятия:

2.4.2.1. Ответы на вопросы семинарского (практического) занятия.

2.4.2.2. Проведение текущего контроля успеваемости

Задания для проведения текущего контроля успеваемости

1. Существуют ли операционные системы жесткого и мягкого реального времени

+ не существуют

существуют

только жесткого времени

только мягкого времени

2. Понятие многонитевая операционная система означает

+ многозадачная

многопроцессорная

многоядерная

многоприоритетная

3. Метод диспетчеризации QNX

+ FIFO

Round Robin

первым пришел последним обслужен

адаптивный

2.4.3 Результаты и выводы: в результате выполнения работы изучено понятие операционных систем реального времени.

2.5. Практическое занятие № 5 .

Тема: «Обязательные требования к ОСРВ».

2.5.1. Задание для работы:

Требования многозадачности.

Требования к работе с памятью.

2.5.2. Краткое описание проводимого занятия:

2.5.2.1. Ответы на вопросы семинарского (практического) занятия.

2.5.2.2. Проведение текущего контроля успеваемости

Задания для проведения текущего контроля успеваемости

1. Операционные системы реального времени (ОСРВ) предназначены для:

+ a) обеспечения интерфейса к ресурсам критических по времени систем реального времени

b) поддержания систем реального времени

c) жесткого ограничения времени

d) выполнение требований по времени

2. В качестве основного требования к ОСРВ выдвигается требование обеспечения

+ a) предсказуемости

b) быстродействия

c) эффективности

d) непрерывности

3. Хорошая ОСРВ имеет предсказуемое поведение:

+ a) при всех сценариях системной загрузки

b) при нескольких сценариях системной загрузки

c) в одном случае системной загрузки

d) никогда

2.5.3 Результаты и выводы: в результате выполнения работы изучены обязательные требования к ОСРВ

2.6. Практическое занятие № 6.

Тема: «Примеры ОСРВ. Система QNX».

2.6.1. Задание для работы:

Коммерческие ОСРВ

Операционная система QNX.

2.6.2. Краткое описание проводимого занятия:

2.6.2.1. Ответы на вопросы семинарского (практического) занятия.

2.6.2.2. Проведение текущего контроля успеваемости

Задания для проведения текущего контроля успеваемости

1. Операционная система QNX Neutrino Realtime Operating System (RTOS) корпорации QNX Software Systems является:

+ a) микроядерной ОС

b) наноядерной ОС

c) экзоядерной ОС

d) ОС с монолитным ядром

2. Ядро QNX Neutrino RTOS выполняется на уровне:

+ a) 0

b) 1

c) 2

d) 3

3. Система RTEMS была создана по заказу министерства обороны США для использования в системах управления:

+ a) ракетными комплексами

b) производством удобрений

c) системой GPS

d) воздушным движением

2.6.3 Результаты и выводы: в результате выполнения работы были изучены примеры ОСРВ. Система QNX.

2.7. Практическое занятие № 7

Тема: «Особенности программирования для СРВ».

2.7.1. Задание для работы:

Требования к языкам программирования для СРВ.

Понятие параллельного программирования.

2.7.2. Краткое описание проводимого занятия:

2.7.2.1. Ответы на вопросы семинарского (практического) занятия.

2.7.2.2. Проведение текущего контроля успеваемости

Задания для проведения текущего контроля успеваемости

1. Основные классы архитектур программных систем:

+ a) цельная, слоистая, комплекс автономно-выполняемых программ, коллектив параллельно выполняемых программ

b) цельная, слоистая, комплекс автономно-выполняемых программ

c) цельная, слоистая, комплекс автономно-выполняемых программ, коллектив автономно выполняемых программ

d) цельная, слоистая, комплекс параллельно-выполняемых программ, коллектив параллельно выполняемых программ

2. При разработке ПП этот принцип реализуют путем разработки большой программы по частям, которые называют программными модулями, а сам такой метод разработки программ

называют:

a) программирование с использованием множеств

+ b) модульное программирование

c) структурное программирование

d) иерархичное программирование

3. Последовательность сменяющих друг друга состояний некоторой информационной среды…

a) алгоритм

b) поток

+ c) процесс

d) транслятор

2.7.3 Результаты и выводы: в результате выполнения работы были изучены особенности программирования для СРВ.

2.8. Практическое занятие № 8

Тема: «Программирование на С++ для СРВ».

2.8.1. Задание для работы:

Система Microsoft Visual Studio 2010.

Программирование на С++

2.8.2. Краткое описание проводимого занятия:

2.8.2.1. Ответы на вопросы семинарского (практического) занятия.

2.8.2.2. Проведение текущего контроля успеваемости

Задания для проведения текущего контроля успеваемости

1. Фрагмент описания процесса, оформляемый как самостоятельный программный продукт это …

+ программный модуль

файл

[illegible]

подпрограмма

2. Консольное приложение отличается …

+ работой в командной строке

наименьшими размерами

отсутствием интерфейса

работой в DOS эмуляторе

3. Заголовочный файл С++ имеет расширение …

+ используется без расширения

.h

cpp

main

2.8.3 Результаты и выводы: в результате выполнения работы изучено программирование на C++ для СРВ.

2.9. Практическое занятие № 9 .

Тема: «Процессы, потоки, задачи».

2.9.1. Задание для работы:

Модели состояний процессов.

Понятие потока.

Преимущества многопоточности.

2.9.2. Краткое описание проводимого занятия:

2.9.2.1. Ответы на вопросы семинарского (практического) занятия.

2.9.2.2. Проведение текущего контроля успеваемости

Задания для проведения текущего контроля успеваемости

1. Модель трех состояний процессов включает в себя

+ выполнение, ожидание, готовность

выполнение, ожидание, загрузку

загрузку, ожидание, готовность

выполнение, загрузку, готовность

2. Поток является

+ легковесным процессом

тяжеловесным процессом

совокупностью процессов

информацией, передающейся между процессами

3. FIFO расшифровывается как

+ first in first out

first in final out

first input first output

final in first out

2.9.3 Результаты и выводы:

2.10. Практическое занятие № 10

Тема: «Управление процессами. Диспетчеризация Windows».

2.10.1. Задание для работы:

Общие принципы управления процессами.

Алгоритмы планирования.

Управление процессами в Windows.

2.10.2. Краткое описание проводимого занятия:

2.10.2.1. Ответы на вопросы семинарского (практического) занятия.

2.10.2.2. Проведение текущего контроля успеваемости

Задания для проведения текущего контроля успеваемости

1. Модуль (программа), отвечающий за разделение времени имеющихся процессоров между выполняющимися задачами:

+ a) планировщик задач

b) организационный модуль

c) редактор

d) кластер

2. Определяют план выполнения задач по их априорным характеристикам:

+ a) статические алгоритмы

b) динамические алгоритмы

c) вытесняющие алгоритмы

d) простые алгоритмы

3. Модифицируют план во время исполнения задач:

+ a) динамические алгоритмы

b) статические алгоритмы

c) вытесняющие алгоритмы

d) простые алгоритмы

2.10.3 Результаты и выводы: в результате выполнения работы изучено управление процессами. Диспетчеризация Windows.

2.11. Практическое занятие № 11 .

Тема: «Управление памятью в ОСРВ».

2.11.1. Задание для работы:

Требования к управлению памятью в ОСРВ.

Модели защиты памяти.

Часы и таймеры.

2.11.2. Краткое описание проводимого занятия

2.11.2.1. Ответы на вопросы семинарского (практического) занятия

2.11.2.2. Проведение текущего контроля успеваемости

Задания для проведения текущего контроля успеваемости

1. Системное и пользовательское адресные пространства не защищены друг от друга, используется два сегмента памяти для кода и для данных; при этом от системы не

требуется никакого управления памятью:

+ a) модель без защиты

b) модель защиты система/пользователь

c) модель защиты пользователь/пользователь

d) модель защиты виртуальной памяти

2. Каждый процесс выполняется в своей собственной виртуальной памяти, у каждого процесса имеются свои собственные сегменты и, следовательно, своя таблица

описателей, ОС несет ответственность за поддержку таблиц описателей:

+ a) модель защиты виртуальной памяти

b) модель защиты система/пользователь

c) модель защиты пользователь/пользователь

d) модель без защиты

3. Фундаментальное требование к памяти в системе реального времени заключается в том, что время доступа к ней:

+ a) ограничено

b) определяется системой

c) неограниченно

d) максимально для приложений

2.11.3 Результаты и выводы: в результате выполнения работы изучено управление памятью в ОСРВ.

2.12. Практическое занятие № 12 .

Тема: «Методы и средства обработки событий».

2.12.1. Задание для работы:

Средства межпроцессного взаимодействия.

Проблемы, возникающие при организации параллельных вычислений.

Виды каналов связи процессов

2.12.2. Краткое описание проводимого занятия:

2.12.2.1 Ответы на вопросы семинарского (практического) занятия.

2.12.2.2. Проведение текущего контроля успеваемости

Задания для проведения текущего контроля успеваемости

1. Способ передачи информации из одного процесса в другой

+ a) межпроцессное взаимодействие

b) копирование

c) замещение

d) совмещение

2. Оповещение процесса со стороны операционной системы о той или иной форме межпроцессного взаимодействия – это …

+ a) событие

b) сигнал

c) мероприятие

d) способ

3. Механизм, предназначенный для синхронизации потоков внутри одного процесса, - это …

+ a) критическая секция

b) счетчик

c) семафор

d) мьютекст

2.12.3 Результаты и выводы: в результате выполнения работы изучены методы и средства обработки событий.

ЛИТЕРАТУРА

1. Солонина А И , Улахович Д.А., Яковлев Л.А. Алгоритмы и процессорьI обработки сигналов - СПБ, «БХВ-СПБ», 2001 г., 464 с.
2. Жуанов А А Операционные системы реального времени - Москва, журнал "PC Week", № 8, 1999г.
3. 0синов Л.А Обработка сигналов на цифровых процессорах - Москва, изд. «Телеком», 2001 г., 112 с, 5 Никамин В.А Аналог- цифровые и цифро - аналоговые преобразователи Москва, «Альтекс», 2005 г., 222 с.
4. Стивенс У. UNIX: взаимодействие процессов. С-Пб., Питер, 2002
5. Кондукова Е., Операционные системы реального времени QNX Neutrino.3 Системная архитектура., БХВ-Петербург. 2005
6. Немеет Э., Снайдер Г., Сибас С., Хейн Т. UNIX: руководство системного администратора. С-Пб., Питер, 2003
7. Жданов А.А. Операционные системы реального времени. М. , "PCWeek", N 8, 1999
8. Ларионов А.М. и др. Вычислительные комплексы, системы и сети, 1987
9. Хетагуров Я А., Древс Ю.Т. Проектирование информационно-вычислительных комплексов, 1987
10. *Олифер В.Г., Олифер Н. А. Сетевые операционные системы — СПб.: Питер, 2003*
11. *Синельников Е. А.* Курс. Системы реального времени. — 2010.

Дополнительны литературы

1. Айфигер Э.С Джервис Б.У. Цифровая обработка сигналов практический подход. - Пер, с англ. - М., Вильямс, 2004 г., 9^2 с.
2. Сергиенио А.Б. Цифровая обработка сигналов - СПБ, Питер, 2002 г., 608 с
3. Интернет сайты hlt://172/19/130/171 8080/ факультеты, www.edu.uz
4. Валов О.П.Учебное пособие по дисциплине системы реального времени. Казанский Государственный Технический Университет им.А.Н.Туполева, Казан(Россия), 2003г.,142 с.
5. Таненбаум, Эндрю С. Современные операционные системы. 2-е изд. — СПб. Питер, 2007.
6. Богачёв К. Ю. Основы параллельного программирования — М.: БИНОМ. Лаб. знаний, 2010.
7. Карпов Ю. Г. Верификация параллельных и распределённых программных систем. — СПб: БХВ-Петербург, 2010.