МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ УКРАИНЫ
НАЦИОНАЛЬНЫЙ АВИАЦИОННЫЙ УНИВЕРСИТЕТ
Кафедра инженерии программного обеспечения
Институт заочного и дистанционного обучения

Расчетно-графическая работа

по дисциплине «Операционные системы»

на тему

«Операционные системы реального времени»

Выполнил: студент группы ПІ-401з

Бабич В.А.

Преподаватель: Конрад Т.И.

Киев – 2016

Содержание

1. Введение в СРВ

1.1.Основные понятия

Существует несколько определений систем реального времени (СРВ), большинство из которых противоречат друг другу. Приведем несколько из них, чтобы продемонстрировать различные взгляды на назначение и основные задачи СРВ.

Системой реального времени называется система, в которой успешность работы любой программы зависит не только от ее логической правильности, но от времени, за которое она получила результат. Если временные ограничения не удовлетворены, то фиксируется сбой в работе системы.

Таким образом, временные ограничения должны быть гарантировано удовлетворены. Это требует от системы быть предсказуемой, т.е. вне зависимости от своего текущего состояния и загруженности выдавать нужный результат за требуемое время. При этом желательно, чтобы система обеспечивала как можно больший процент использования имеющихся ресурсов.

Стандарт POSIX 1003.1 определяет СРВ следующим образом: «Реальное время в операционных системах – это способность операционной системы обеспечить требуемый уровень сервиса в заданный промежуток времени».

Иногда системами реального времени называется системы постоянной готовности (on-line системы), или «интерактивные системы с достаточным временем реакции». Обычно это делают по маркетинговым соображениям. Действительно, если интерактивную программу называют работающей в «реальном времени», то это просто означает, что она успевает обработать запросы от человека, для которого задержка в сотни миллисекунд даже незаметна.

Иногда понятие «система реального времени» отождествляют с понятием «быстрая система». Это не всегда правильно. Время задержки СРВ на событие не так уж важно (оно может достигать нескольких секунд). Главное, чтобы это время было достаточно для рассматриваемого приложения и гарантировано. Очень часто алгоритм с гарантированным временем работы менее эффективен, чем алгоритм, таким действием не обладающий. Например, алгоритм «быстрой» сортировки в среднем работает значительно быстрее других алгоритмов сортировки, но его гарантированная оценка сложности значительно хуже.

Во многих сферах приложения СРВ вводят свои понятия «реального времени». Например, процесс цифровой обработки сигнала называют идущим в «реальном времени», если анализ (при вводе) и/или генерация (при выводе) данных может быть проведен за то же

время, что и анализ и/или генерация тех же данных без цифровой обработки сигнала. Например, если при обработке аудио данных требуется 2.01 секунд для анализа 2.00 секунд звука, то это не процесс реального времени. Если же требуется 1.99 секунд, то это процесс реального времени.

Назовем системой реального времени аппаратно-программный комплекс, реагирующий в предсказуемые времена на непредсказуемый поток внешних событий.

Это определение означает, что:

- Система должна успеть отреагировать на событие, произошедшее на объекте, в течение времени, критического для этого события (meet deadline). Величина критического времени для каждого события определяется объектом и самим событием, и, естественно, может быть разной, но время реакции системы должно быть предсказано (вычислено) при создании системы. Отсутствие реакции в предсказанное время считается ошибкой для систем реального времени.
- Система должна успевать реагировать на одновременно происходящие события. Даже если два или больше внешних событий происходят одновременно, система должна успеть среагировать на каждое из них в течение интервалов времени, критического для этих событий.

Хорошим примером задачи, где требуется СРВ, является управление роботом, берущим деталь с ленты конвейера. Деталь движется, и робот имеет лишь маленькое временное окно, когда он может ее взять. Если он опоздает, то деталь уже не будет на нужном участке конвейера, и, следовательно, работа не будет сделана, несмотря на то, что робот находится в правильном месте. Если он позиционируется раньше, то деталь еще не успеет подъехать, и робот заблокирует ей путь.

Другим примером может быть самолет, находящийся на автопилоте. Сенсорные серводатчики должны постоянно передавать в управляющий компьютер результаты измерений. Если результат какого-либо измерения будет пропущен, то это может привести к недопустимому несоответствию между реальным состоянием систем самолета и информацией о нем в управляющей программе.

Различают системы реального времени двух типов - системы жесткого реального времени и системы мягкого реального времени.

Системы жесткого реального времени не допускают никаких задержек реакции системы ни при каких условиях, так как:

- результаты могут оказаться бесполезны в случае опоздания,
- может произойти катастрофа в случае задержки реакции,
- стоимость опоздания может оказаться бесконечно велика.

Примеры систем жесткого реального времени - бортовые системы управления, системы аварийной защиты, регистраторы аварийных событий.

Системы мягкого реального времени характеризуются тем, что задержка реакции не критична, хотя и может привести к увеличению стоимости результатов и снижению производительности системы в целом.

Пример - работа сети. Если система не успела обработать очередной принятый пакет, это приведет к таймауту на передающей стороне и повторной посылке (в зависимости от протокола, конечно). Данные при этом не теряются, но производительность сети снижается.

Основное отличие между системами жесткого и мягкого реального времени можно выразить так: система жесткого реального времени никогда не опоздает с реакцией на событие, система мягкого реального времени - не должна опаздывать с реакцией на событие.

1.2.Структура СРВ.

Любая система реального масштаба времени может быть описана как состоящая из трех основных подсистем.

Управляемая (контролируемая) подсистема (например, индустриальный завод, управляемое компьютером транспортное средство), диктует требования в реальном масштабе времени; подсистема контроля (контролирующая) управляет некоторыми вычислениями и связью с оборудованием для использования от управляемой подсистемы; подсистема оператора (операционная) контролирует полную деятельность системы. Интерфейс между управляемыми и подсистемами контроля состоит из таких устройств как датчики и приводы. Интерфейс между управляющей подсистемой и оператором связывает человека с машинной.

Управляемая подсистема представлена задачами (в дальнейшем называемыми прикладными задачами) которые используют оборудование, управляемое подсистемой контроля. Эта последняя подсистема может быть построена из очень большого количества процессоров, управляющими такими местными ресурсами, как память и устройства хранения, и доступ к локальной сети в реальном масштабе времени. Эти процессоры и ресурсы управляются системой программного обеспечения, которую мы называем операционной системой реального масштаба времени (RTOS – real time operating system).

1.3.ОСРВ и ОС общего назначения

Назовем операционной системой реального времени такую систему, которая может быть использована для построения систем жесткого реального времени.

ОС общего назначения, особенно многопользовательские, такие как UNIX, ориентированы на оптимальное распределение ресурсов компьютера между пользователями и задачами (системы разделения времени), В операционных системах реального времени подобная задача отходит на второй план - все отступает перед главной задачей - успеть среагировать на события, происходящие на объекте.

Другое отличие - применение операционной системы реального времени всегда связано с аппаратурой, с объектом, с событиями, происходящими на объекте. Система реального времени, как аппаратно-программный комплекс, включает в себя датчики, регистрирующие события на объекте, модули ввода-вывода, преобразующие показания датчиков в цифровой вид, пригодный для обработки этих показаний на компьютере, и, наконец, компьютер с программой, реагирующей на события, происходящие на объекте. Операционная система реального времени ориентирована на обработку внешних событий. Именно это приводит к коренным отличиям (по сравнению с ОС общего назначения) в структуре системы, в функциях ядра, в построении системы ввода-вывода. Операционная система реального времени может быть похожа по пользовательскому интерфейсу на ОС общего назначения (к этому, кстати, стремятся почти все производители операционных системах реального времени), однако устроена она совершенно иначе - об этом речь впереди.

1.4. Характеристики ОСРВ

Как было сказано ранее, сердцем системы реального времени является ОСРВ (операционная система реального времени). В связи со специфичность решаемых задач, ОСРВ должна обладать определенными свойствами.

Время реакции системы на внешние события. Согласно определению, ОСРВ должна обеспечить требуемый уровень сервиса в заданный промежуток времени. Этот промежуток времени задается обычно периодичностью и скоростью процессов, которым управляет система.

Видно, что времена очень разнятся и накладывают различные требования на вычислительную установку, на которой работает ОСРВ.

Интервал времени - от события на объекте и до выполнения первой инструкции в

программе обработки этого события и является временем реакции системы на события. В ОСРВ заложен параллелизм, возможность одновременной обработки нескольких событий, поэтому все операционные системы реального времени являются многозадачными (многопроцессными, многонитиевыми).

Время перезагрузки системы. Этот параметр кажется второстепенным, однако были свидетелями случаи, когда именно этот параметр целиком определял выбор дорогой операционной системы (время перезагрузки не должно было превышать 1 секунду). Этот параметр важен для систем, от которых требуется непрерывная работа; в этих случаях ставятся ловушки, отслеживающие зависание системы или приложений, и, если таковое произошло, автоматически перезагружающие систему (такие ловушки необходимы в системах повышенной надежности, т.к. от ошибок, по крайней мере, в приложениях никто не застрахован). В таких случая важным является такое свойство системы как ее живучесть при незапланированных перезагрузках. Большинство операционных систем реального времени устойчивы к перезагрузкам и могут быть прерваны и перезагружены в любое время.

Размеры системы. Для систем реального времени важным параметром является размер системы исполнения, а именно суммарный размер минимально необходимого для работы приложения системного набора (ядро, системные модули, драйверы и т. д.). Хотя, надо признать, что с течением времени значение этого параметра уменьшается, тем не менее он остается важным и производители систем реального времени стремятся к тому, чтобы размеры ядра и обслуживающих модулей системы были невелики.

Возможность исполнения системы из ПЗУ (ROM). Система должна иметь возможность осуществлять загрузку из ПЗУ. Для экономии места в ПЗУ часть системы может храниться в сжатом виде и загружаться в ОЗУ по мере необходимости. Часто система позволяет исполнять код как в ПЗУ, так и в ОЗУ. При наличии свободного места в ОЗУ система может копировать себя из медленного ПЗУ в более быстрое ОЗУ.

К дополнительным свойствам ОСРВ можно отнести следующие:

Наличие необходимых драйверов устройств. Если разрабатываемая система имеет обширную периферию, то наличие уже готовых драйверов может оказать большое влияние на выбор операционной системы.

Поддержка процессоров различной архитектуры. В связи с тем, что в промышленных компьютерах, серверах, встраиваемых системах широко распространены процессоры разной архитектуры с различной системой команд, ОСРВ по возможности должна поддерживать как можно более широкий ряд процессоров.

1.5.Механизмы реального времени

Важнейшими характеристиками ОСРВ являются заложенные в операционную систему механизмы реального времени.

Система приоритетов и алгоритмы диспетчеризации. Базовыми инструментами разработки сценария работы системы являются система приоритетов процессов (задач) и алгоритмы планирования (диспетчеризации) операционных системах реального времени.

В многозадачных ОС общего назначения используются, как правило, различные модификации алгоритма круговой диспетчеризации, основанные на понятии непрерывного кванта времени ("time slice"), предоставляемого процессу для работы. Планировщик по истечении каждого кванта времени просматривает очередь активных процессов и принимает решение, кому передать управление, основываясь на приоритетах процессов (численных значениях, им присвоенных). Приоритеты могут быть фиксированными или меняться со временем - это зависит от алгоритмов планирования в данной ОС, но рано или поздно процессорное время получат все процессы в системе.

Алгоритмы круговой диспетчеризации неприменимы в чистом виде в операционных системах реального времени. Основной недостаток - непрерывный квант времени, в течение которого процессором владеет только один процесс. Планировщики же операционных систем реального времени имеют возможность сменить процесс до истечения "time slice", если в этом возникла необходимость. Один из возможных алгоритмов планирования при этом "приоритетный с вытеснением".

Механизмы межзадачного взаимодействия. Другой набор механизмов реального времени относится к средствам синхронизации процессов и передачи данных между ними. Для операционных систем реального времени характерна развитость этих механизмов. К таким механизмам относятся: семафоры, мьютексы, события, сигналы, средства для работы с разделяемой памятью, каналы данных (pipes), очереди сообщений. Многие из подобных механизмов используются и в ОС общего назначения, но их реализация в операционных системах реального времени имеет свои особенности - время исполнения системных вызовов почти не зависит от состояния системы, и в каждой операционной системе реального времени есть по крайней мере один быстрый механизм передачи данных от процесса к процессу.

Средства для работы с таймерами. Такие инструменты, как средства работы с таймерами, необходимы для систем с жестким временным регламентом, поэтому развитость средств работы с таймерами - необходимый атрибут операционных систем реального времени.

2. Архитектура ОСРВ. Классы ОСРВ.

2.1.Архитектура ОСРВ

Все ОСРВ сегодня являются многозадачными системами. Задачи делят между собой ресурсы вычислительной системы, в том числе и процессорное время.

Четкой границы между ядром (KERNEL) и операционной системой нет. Различают их, как правило, по набору функциональных возможностей. Ядра предоставляют пользователю такие базовые функции, как планирование синхронизация задач, межзадачная коммуникация, управление памятью, устройствами ввода-вывода и т. п. Операционные системы в дополнение к этому имеют файловую систему, сетевую поддержку, интерфейс с оператором и другие средства высокого уровня. Они обеспечивают также взаимодействие системы и управляющего/управляемого оборудования.

ОСРВ с монолитной архитектурой можно представить в виде прикладного уровня: состоит из работающих прикладных процессов; системного уровня: состоит из монолитного ядра операционной системы.

Модульная архитектура (на основе микроядра) появилась как попытка убрать узкое место – API и облегчить модернизацию системы и перенос ее на новые процессоры.

API в модульной архитектуре играет только одну роль: обеспечивает связь прикладных процессов и специального модуля – менеджера процессов. Однако, теперь микроядро играет двойную роль:

1) управление взаимодействием частей системы (например, менеджеров процессов и файлов) ;
2) обеспечение непрерывности выполнения кода системы (т.е. отсутствие переключения задач во время исполнения микроядра).

Недостатки у модульной архитектуры фактически те же, что и у монолитной. Проблемы перешли с уровня API на уровень микроядра. Системный интерфейс по-прежнему не допускает переключения задач во время работы микроядра, только сократилось время пребывания в этом состоянии. API по-прежнему может быть реализован только на ассемблере, проблемы с переносимостью микроядра уменьшились (в связи с сокращением его размера), но остались.

Объектная архитектура на основе объектов-микроядер. В этой архитектуре API отсутствует вообще. Взаимодействие между компонентами системы (микроядрами) и пользовательскими процессами осуществляется посредством обычного вызова функций, поскольку и система, и приложения написаны на одном языке. Это обеспечивает

максимальную скорость системных вызовов.

Фактическое равноправие всех компонент системы обеспечивает возможность переключения задач в любое время.

Объектно-ориентированный подход обеспечивает модульность, безопасность, легкость модернизации и повторного использования кода.

Роль API играет компилятор и динамический редактор объектных связей (linker). При старте приложения динамический linker загружает нужные ему микроядра (т.е., в отличие от предыдущих систем, не все компоненты самой операционной системы должны быть загружены в оперативную память). Если микроядро уже загружено для другого приложения, оно повторно не загружается, а использует код и данные уже имеющегося ядра. Это позволяет уменьшить объем требуемой памяти.

Поскольку разные приложения разделяют одни микроядра, то они должны работать в одном адресном пространстве. Следовательно, система не может использовать виртуальную память и тем самым работает быстрее (так как исключаются задержки на трансляцию виртуального адреса в физический).

Микроядра по своим характеристикам напоминают структуры, используемые в других операционных системах. Однако есть и свои различия.

2.2.Классы ОСРВ

1-й класс: исполнительные СРВ. Это программы для программируемых микропроцессоров, встраиваемых в различные устройства, очень небольшие и обычно написаны на языке низкого уровня типа ассемблера или PLM. Внутрисхемные эмуляторы пригодны для отладки, но высокоуровневые средства разработки и отладки программ не применимы. Операционная среда обычно недоступна.

Признаки систем этого типа - различные платформы для систем разработки и исполнения. Как правило, приложение реального времени - это одна задача и параллелизм здесь достигается с помощью нитей (threads).

Системы этого типа обладают рядом достоинств, среди которых главное - скорость и реактивность системы. Главная причина высокой реактивности систем этого типа - наличие только нитей(потоков) и, следовательно, маленькое время переключения контекста между ними (в отличие от процессов).

2-й класс: минимальное ядро системы реального времени. На более высоком уровне находятся системы реального времени, обеспечивающие минимальную среду исполнения.

Предусмотрены лишь основные функции, а управление памятью и диспетчер часто недоступны. Ядро представляет собой набор программ, выполняющих типичные, необходимые для встроенных систем низкого уровня функции, такие, как операции с плавающей запятой и минимальный сервис ввода/вывода.

В этот класс входят системы с монолитным ядром, где и содержится реализация всех механизмов реального времени этих операционных систем.

3-й класс: ядро системы реального времени и инструментальная среда. Этот класс систем обладает многими чертами ОС с полным сервисом. Разработка ведется в инструментальной среде, а исполнение - на целевых системах. Этот тип систем обеспечивает гораздо более высокий уровень сервиса для разработчика прикладной программы. Сюда включены такие средства, как дистанционный символьный отладчик, протокол ошибок и другие средства CASE. Часто доступно параллельное выполнение программ.

4-й класс: ОС с полным сервисом. Такие ОС могут быть применены для любых приложений реального времени. Разработка и исполнение прикладных программ ведутся в рамках одной и той же системы.

Область применения расширений реального времени - большие системы реального времени, где требуется визуализация, работа с базами данных, доступ в интернет и пр.

К таким системам можно отнести ОСРВ, написанные на основе UNIX и расширения Windows.

3. Планирование задач и процессов ОСРВ.

3.1.Функции ядра операционных систем.

Как было указано ранее, основой любой среды исполнения в реальном времени является ядро. Все современные ОСРВ являются многозадачными. Таким образом, все программное обеспечение, включая также часть операционной системы, организуется в виде набора последовательных процессов. Исходя из этого, ядро может обеспечивать сервис пяти типов:

Синхронизация ресурсов. Метод синхронизации требует ограничить доступ к общим ресурсам (данным и внешним устройствам). Наиболее распространенный тип примитивной синхронизации - двоичный семафор, обеспечивающий избирательный доступ к общим ресурсам. Так, процесс, требующий защищенного семафором ресурса, вынужден ожидать до тех пор, пока семафор не станет доступным, что свидетельствует об освобождении ожидаемого ресурса, и, захватив ресурс, установить семафор. В свою очередь, другие процессы также будут ожидать доступа к ресурсу вплоть до того момента, когда семафор возвратит соответствующий ресурс системе распределения ресурсов. Системы, обладающие большей ошибкоустойчивостью, могут иметь счетный семафор. Этот вид семафора разрешает одновременный доступ к ресурсу лишь определенному количеству процессов.

Межзадачный обмен. Часто необходимо обеспечить передачу данных между программами внутри одной и той же системы. Кроме того, во многих приложениях возникает необходимость взаимодействия с другими системами через сеть. Внутренняя связь может быть осуществлена через систему передачи сообщений. Внешнюю связь можно организовать либо через датаграмму (наилучший способ доставки), либо по линиям связи (гарантированная доставка). Выбор того или иного способа зависит от протокола связи.

Разделение данных. В прикладных программах, работающих в реальном времени, наиболее длительным является сбор данных. Данные часто необходимы для работы других программ или нужны системе для выполнения каких-либо своих функций. Во многих системах предусмотрен доступ к общим разделам памяти. Широко распространена организация очереди данных. Применяется много типов очередей, каждый из которых обладает собственными достоинствами.

Обработка запросов внешних устройств. Каждая прикладная программа в реальном времени связана с внешним устройством определенного типа. Ядро должно обеспечивать службы ввода/вывода, позволяющие прикладным программам осуществлять чтение с этих

устройств и запись на них. Для приложений реального времени обычным является наличие специфического для данного приложения внешнего устройства. Ядро должно предоставлять сервис, облегчающий работу с драйверами устройств.

Обработка особых ситуаций. Особая ситуация представляет собой событие, возникающее во время выполнения программы. Она может быть синхронной, если ее возникновение предсказуемо, как, например, деление на нуль. А может быть и асинхронной, если возникает непредсказуемо, как, например, падение напряжения. Предоставление возможности обрабатывать события такого типа позволяет прикладным программам реального времени быстро и предсказуемо отвечать на внутренние и внешние события.

3.2.Процессы и потоки. Схемы назначения приоритетов

Рассмотрим подробнее, что такое процесс. Процесс – это динамическая сущность программы, код в процессе своего выполнения. Имеет:

- собственные области памяти под код и данные, включая значения регистров и счетчика команд;
- собственный стек;
- собственное отображение виртуальной памяти (в системах с виртуальной памятью) на физическую;
- собственное состояние.

Каждый процесс имеет свой жизненный цикл, состоящий из 4 стадий:

1. создание;
2. загрузка;
3. выполнение;
4. завершение.

Создание процесса обычно состоит из присвоения новому процессу идентификатора процесса и подготовки информации, которая определяет окружение процесса.

Загрузка процесса означает загрузку в память кода процесса.

После того, как код программы загружен, процесс готов к выполнению. Он начинает конкурировать с другими процессами за ресурсы процессора. Процесс может выполняться, а может блокироваться по тем или иным причинам.

Завершение процесса означает освобождение всех ресурсов, выделенных процессу – файловых дескрипторов, памяти и т.д.

Модель потока базируется на двух независимых концепциях: группировании ресурсов

и выполнении программы. С одной стороны, процесс можно рассматривать как способ группирования родственных ресурсов в одну группу. У процесса есть адресное пространство, содержащее текст программы и данные, а также другие ресурсы. Ресурсами могут быть открытые файлы, обработчики сигналов, учетная информация и многое другое. С другой стороны, процесс можно рассматривать как поток исполняемых команд, или просто поток. У потока есть счетчик команд, отслеживающий порядок выполнения действий. У него есть регистры, в которых хранятся текущие переменные. У него есть стек, содержащий протокол выполнения процесса, где на каждую процедуру, вызванную, но еще не вернувшуюся, отведен отдельный фрейм. У процесса есть свое состояние. Потоки одного процесса выполняются в одном адресном пространстве, используют одни и те же глобальные переменные, ресурсы.. Таким образом, процессы используются для группирования ресурсов, а потоки являются объектами, поочередно выполняющимися на процессоре.

Каждому процессу и каждому потоку в ОСРВ приписывается некоторое число, называемое приоритетом. Чем больше это число, тем важнее процесс или поток.

3.3.Планирование задач.

3.3.3. Алгоритмы и задачи планирования в ОСРВ

Необходимость планирования задач появляется, как только в очереди активных (готовых) задач появляются более одной задачи (в многопроцессорных системах - более числа имеющихся процессоров). Алгоритм планирования задач является основным отличием систем реального времени от "обычных" операционных систем. В последних целью планирования является обеспечение выполнения всех задач из очереди готовых задач, обеспечивая иллюзию их параллельной работы и не допуская монополизацию процессора какой-либо из задач. В ОСРВ же целью планирования является обеспечение выполнения каждой готовой задачи к определенному моменту времени, при этом часто "параллельность" работы задач не допускается, поскольку тогда время исполнения задачи будет зависеть от наличия других задач.

Важнейшим требованием при планировании задач в ОСРВ является предсказуемость времени работы задачи. Это время не должно зависеть от текущей загруженности системы, количества задач в очередях ожидания (процессора, семафора, события, ...) и т.д. При этом желательно, чтобы длина этих очередей не была бы ограничена (т.е. ограничена только объемом памяти, доступной системе).

Определение. Планировщик задач (scheduler) - это модуль (программа), отвечающий

за разделение времени имеющихся процессоров между выполняющимися задачами. Отвечает за коммутацию задач из состояния блокировки в состояние готовности, и за выбор задачи (задач - по числу процессоров) из числа готовых для исполнения процессором (ами).

Ключевым вопросом планирования является выбор момента принятия решения: Во-первых, когда создается новый процесс, необходимо решить, какой процесс запустить, родительский или дочерний. Поскольку оба процесса находятся в состоянии готовности, эта ситуация не выходит за рамки обычного и планировщик может запустить любой из двух процессов.

Во-вторых, планирование необходимо, когда процесс завершает работу. Этот процесс уже не существует, следовательно, необходимо из набора готовых процессов выбрать и запустить следующий.

В-третьих, когда процесс блокируется по какой-либо причине, необходимо выбрать и запустить другой процесс.

В-четвертых, решение по диспетчеризации должно приниматься после разблокировки процесса.

В-пятых, планировщик может принимать решение по истечении кванта времени, отпущенному процессу.

Алгоритмы планирования можно разделить на две категории согласно их поведению после прерываний. Алгоритмы планирования без переключений, иногда называемого также неприоритетным планированием, выбирают процесс и позволяют ему работать вплоть до блокировки либо вплоть до того момента, когда процесс сам не отдаст процессор. Процесс не будет прерван, даже если он работает часами. Соответственно, решения планирования не принимаются по прерываниям от таймера. После обработки прерывания таймера управление всегда возвращается приостановленному процессу.

Напротив, алгоритмы планирования с переключениями, называемого также приоритетным планированием, выбирают процесс и позволяют ему работать некоторое максимально возможное время. Если к концу заданного интервала времени процесс все еще работает, он приостанавливается и управление переходит к другому процессу. Приоритетное планирование требует прерываний по таймеру, происходящих в конце отведенного периода времени (решения планирования могут, например, приниматься при каждом прерывании по таймеру, или при каждом k-ом прерывании), чтобы передать управление планировщику.

Напомним, что приоритетом называется число, приписанное операционной системой (а именно, планировщиком задач) каждому процессу и задаче. Существуют несколько схем назначения приоритетов.

- Фиксированные приоритеты - приоритет задаче назначается при ее создании и не меняется в течение ее жизни. Эта схема с различными дополнениями

применяется в большинстве систем реального времени. В схемах планирования ОСРВ часто требуется, чтобы приоритет каждой задачи был уникальным, поэтому часто ОСРВ имеют большое число приоритетов (обычно 255 и более).

- Турнирное определение приоритета - приоритет последней исполнявшейся задачи понижается.
- Определение приоритета по алгоритму round robin - приоритет задачи определяется ее начальным приоритетом и временем ее обслуживания. Чем больше задача обслуживается процессором, тем меньше ее приоритет (но не опускается ниже некоторого порогового значения). Эта схема в том или ином виде применяется в большинстве UNIX систем.

Отметим, что в разных системах различные алгоритмы планирования задач могут вводить новые схемы изменения приоритетов. Например, в системе OS-9 приоритеты ожидающих задач увеличиваются для избегания слишком больших времен ожидания.

«Наименьшее оставшееся время выполнения». Является версией предыдущего алгоритма с переключениями. В соответствии с этим алгоритмом планировщик каждый раз выбирает процесс с наименьшим оставшимся временем выполнения. В этом случае также необходимо знать заранее время выполнения каждого процесса. Когда поступает новый процесс, его полное время выполнения сравнивается с оставшимся временем выполнения текущего процесса. Если время выполнения нового процесса меньше, текущий процесс приостанавливается и управление передается новому процессу. Эта схема позволяет быстро обслуживать короткие процессы.

«Карусельная диспетчеризация (циклическое планирование)». При карусельной диспетчеризации процесс продолжает выполнение, пока не наступит момент, когда он:

- добровольно уступает управление (т.е. блокируется);
- вытесняется процессом с более высоким приоритетом;
- использовал свой квант времени (timeslice). После того, как процесс использовал свой квант времени, управление передается следующему процессу, который находится в состоянии готовности и имеет такой же уровень приоритета.

«Адаптивная диспетчеризация». При адаптивной диспетчеризации процесс ведет себя следующим образом:

- Если процесс использовал свой квант времени (т.е. он не блокировался), то его приоритет уменьшается на 1. Это получило название снижение приоритета (priority decay). "Пониженный" процесс не будет продолжать "снижаться", даже если он использовал еще один квант времени и не блокировался - он снизится только на один уровень ниже своего исходного приоритета.

- Если процесс блокируется, то ему возвращается первоначальное значение приоритета.

В системах реального времени наиболее распространенными являются алгоритмы FIFO, адаптивной диспетчеризации, карусельной диспетчеризации и их разновидности.

3.3.2. Планирование периодических процессов

Проблемы планирования задач в ОСРВ

Классификация задач реального времени:

- периодические (входят в состояние выполнения периодически);
- апериодические (выполняются при возникновении определенных событий, характеризуются мягким реальным временем);

- спорадические (выполняются при возникновении определенных событий, характеризуются жестким реальным временем).

Общие принципы планирования задач РВ:

1) Целью планирование является выполнение задач за заранее определенный интервал времени. Поэтому алгоритм планирования должен определить последовательность выполнения задач в соответствии с их требованиями к ресурсам и ко времени исполнения.
2) Алгоритмы планирования должны быть вытесняющими.
3) Алгоритмы планирования могут быть статическими и динамическими.

Внешние события, на которые система реального времени должна реагировать, можно разделить на периодические (возникающие через регулярные промежутки времени) и непериодические (возникающие непредсказуемо). Возможно наличие нескольких потоков событий, которые система должна обрабатывать. В зависимости от времени, затрачиваемого на обработку каждого из событий, может оказаться, что система не в состоянии своевременно обработать все события.

Классическим примером статического алгоритма планирования реального времени для прерываемых периодических процессов является алгоритм RMS (Rate Monotonic Scheduling – планирование с приоритетом, пропорциональным частоте). Этот алгоритм может использоваться для процессов, удовлетворяющих следующим условиям:

1) Каждый периодический процесс должен быть завершен за время его периода.

2) Ни один процесс не должен зависеть от любого другого процесса.
3) Каждому процессу требуется одинаковое процессорное время на каждом интервале.
4) У непериодических процессов нет жестких сроков
5) Прерывание процесса происходит мгновенно, без накладных расходов.

Требование 2 не вполне реалистично, так как на практике разные процессы могут обращаться к одному разделяемому ресурсу и, соответственно, блокироваться. Последнее требование нереально, но вводится для упрощения модели системы.Во время работы планировщик всегда запускает готовый к работе процесс с наивысшим приоритетом, прерывая при необходимости работающий процесс с меньшим приоритетом. Таким образом, в нашем примере процесс A может прервать процессы B и C, процесс B может прервать C. Процесс C всегда вынужден ждать, пока процессор не освободится.

Другим популярным алгоритмом планирования является алгоритм EDF (Earliest Deadline First– процесс с ближайшим сроком завершения в первую очередь). Алгоритм EDF представляет собой динамический алгоритм, не требующий от процессов периодичности. Он также не требует и постоянства временных интервалов использования процессора. Каждый раз, когда процессу требуется процессорное время, он объявляет о своем присутствии и о своем сроке выполнения задания. Планировщик хранит список процессов, сортированный по срокам выполнения заданий. Алгоритм запускает первый процесс в списке, то есть тот, у которого самый близкий по времени срок выполнения. Когда новый процесс переходит в состояние готовности, система сравнивает его срок выполнения со сроком выполнения текущего процесса. Если у нового процесса график более жесткий, он прерывает работу текущего процесса.

Алгоритм EDF работает с любым набором процессов, для которых возможно планирование. Платой за это является использование более сложного алгоритма.

4. Межпроцессное взаимодействие.

4.1.Виды межпроцессного взаимодействия

Хотя каждая задача (процесс) в системе, как правило, выполняет какую-либо отдельную функцию, часто возникает необходимость в согласовании (синхронизации) действий, выполняемых различными задачами. Такая синхронизация необходима, в основном в следующих случаях:

1. Функции, выполняемые различными задачами, связаны друг с другом. Например, если одна задача подготавливает исходные данные для другой, то последняя не выполняется до тех пор, пока не получит от первой задачи соответствующего сообщения. Одна из вариаций в этом случае - это когда задача при определенных условиях порождает одну или несколько новых задач.
2. Необходимо упорядочить доступ нескольких задач к разделяемому ресурсу.
3. Необходима синхронизация задачи с внешними событиями. Как правило, для этого используется механизм прерываний.
4. Необходима синхронизация задачи по времени. Диапазон различных вариантов в этом случае достаточно широк, от привязки момента выдачи какого-либо воздействия к точному астрономическому времени до простой задержки выполнения задачи на определенный интервал времени. Для решения этих вопросов в конечном счете используются специальные аппаратные средства, называемые таймером.

Межпроцессное взаимодействие – это тот или иной способ передачи информации из одного процесса в другой. Наиболее распространенными формами взаимодействия являются:

1) Разделяемая память – два или более процесса могут иметь доступ к одному и тому же блоку памяти. В системах с виртуальной памятью организация такого вида взаимодействия требует поддержки со стороны операционной системы, поскольку необходимо отобразить соответствующие блоки виртуальной памяти на один и тот же блок физической памяти.
2) Семафоры – два и более процесса имеют доступ к одной переменной, принимающей значение 0 или 1. Сама переменная часто находится в области данных операционной системы и доступ к ней организуется с помощью специальных функций.
3) Сигналы – это сообщения, доставляемые посредством операционной системы процессу. Процесс должен зарегистрировать обработчик этого сообщения у операционной системы, чтобы получить возможность реагировать на него. Часто операционная система извещает процесс сигналом о наступлении какого-либо сбоя, например, делении на 0, или о каком-либо аппаратном прерывании, например,

прерывании таймера.

4) Почтовые ящики – это очередь сообщений (обычно – тех или иных структур данных), которые помещаются в почтовый ящик процессами и/или операционной системой.

Несколько процессов могут ждать поступления сообщения в почтовый ящик и активизироваться по поступлении сообщения. Требует поддержки со стороны операционной системы.

Событие – это оповещение процесса со стороны операционной системы о той или иной форме межпроцессного взаимодействия, например, о принятии семафором нужного значения, наличии сигнала, о поступлении сообщения в почтовый ящик.

Взаимное согласование задач с помощью сообщений является одним из важнейших принципов операционных систем реального времени. В дальнейшем будем называть сообщениями любой механизм явной передачи информации от одной задачи к другой (такие объекты, как семафоры, можно отнести к механизму неявной передачи сообщений). Объем информации, передаваемой в сообщении, может меняться от одного бита до всей свободной емкости памяти системы. Во многих ОСРВ компоненты операционной системы, также как и пользовательские задачи, способны принимать и передавать сообщения. Сообщения могут быть асинхронными и синхронными. В первом случае доставка сообщений задаче производится после того, как она в плановом порядке получит управление, а во втором случае циркуляция сообщений оказывает непосредственное влияние на планирование задач. Например, задача, пославшая какое-либо сообщение, немедленно блокируется, если для продолжения работы ей необходимо дождаться ответа, или если низкоприоритетная задача шлет высокоприоритетной задаче сообщение, которого последняя ожидает, то высокоприоритетная задача, если конечно, используется приоритетная многозадачность с вытеснением, немедленно получит управление. Иногда сообщения передаются через буфер определенного размера (почтовый ящик). При этом, как правило, новое сообщение затирает старое, даже если последнее не было обработано.

Однако наиболее часто используется принцип, когда каждая задача имеет свою очередь сообщений, в конец которой ставится всякое вновь полученное сообщение. Стандартный принцип обработки очереди сообщений по принципу FIFO не всегда оптимально соответствует поставленной задаче. В некоторых ОСРВ предусматривается такая возможность, когда сообщение от высокоприоритетной задачи обрабатывается в первую очередь (в этом случае говорят, что сообщение наследует приоритет пославшей его задачи).

Иногда полезным оказывается непосредственное управление приоритетом сообщений. Представим, что задача послала серверу (драйверу) принтера несколько сообщений, содержащих данные для печати. Если теперь задача хочет отменить всю печать, ей надо послать соответствующее сообщение с более высоким приоритетом, чтобы оно встало в

очередь впереди всех посланных ранее заданий на печать.

Сообщение может содержать как сами данные, предназначенные для передачи, так и указатель на такие данные. В последнем случае обмен может производиться с помощью разделяемых областей памяти, разделяемых файлов и т.п.

4.2.Ресурсы и их характеристики

Ресурс - это общий термин, описывающий объект (физическое устройство, область памяти), который может одновременно использоваться только одной задачей.

По своим характеристикам ресурсы разделяют на:

- активные – способны изменить информацию (процессор) ;
- пассивные – способны хранить информацию;
- локальные – принадлежат одному процессу;
- время жизни совпадает с временем жизни процесса
- разделяемые – могут быть использованы несколькими процессами; существуют, пока есть хоть один процесс, который их использует ;
- постоянные – используются посредством операций «захватить» и «освободить» ;
- временные – используются посредством «создать» и «удалить».

Разделяемые ресурсы бывают:

- не критичные – могут быть использованы одновременно несколькими процессами (например, жесткий диск)
- критичные – могут быть использованы только одним процессом, и пока этот процесс не завершит работу с ресурсом, последний не доступен другим процессам (например, разделяемая память, доступная на запись).
- По типу взаимодействия различают:
- сотрудничающие процессы:

  – процессы, разделяющие только коммуникационный канал, по которому один передает данные, а другой получает их;

  – процессы, осуществляющие взаимную синхронизацию: когда работает один, другой ждет окончания его работы (типично для программ, управляющих рядом технологических процессов)
- конкурирующие процессы:

- процессы, использующие совместно разделяемый ресурс;
- процессы, использующие критические секции;
- процессы, использующие взаимные исключения.

4.3.Проблемы взаимодействия процессов

Представим, например, что несколько процессов пытаются одновременно выводить данные на принтер. Если процессу требуется вывести на печать файл, он помещает его имя в специальный каталог спулера. Другой процесс, демон печати, периодически проверяет наличие файлов, которые нужно печатать, печатает файл и удаляет его имя из каталога. Пусть каталог спулера состоит из большого числа пронумерованных сегментов, в каждом из которых может храниться имя файла. Также есть две совместно используемые (и доступные всем процессам) переменные – out, указывающая на следующий файл для печати, и in, указывающая на следующий свободный сегмент. Процессы А и В одновременно пытаются поставить файлы в очередь на печать. Процесс А считывает значение переменной in и сохраняет его в локальной переменной. После этого происходит прерывание по таймеру, и процессор переключается на процесс В. Он, в свою очередь, также считывает значение переменной in и сохраняет его в своей локальной переменной. Процесс В сохраняет в каталоге спулера имя файла и увеличивает на 1 значение переменной in. После того как управление перейдет к процессу А, он продолжит выполнение с того места, где был прерван. Он обращается к своей локальной переменной, хранящей старое значение переменной in, считывает ее значение и записывает имя файла в тот же сегмент, что и процесс В, а затем изменяет значение in. Структура каталога не нарушена, но файл процесса В не будет напечатан. Ситуация, в которой два процесса считывают или записывают данные одновременно и конечный результат зависит от того, какой процесс был первым, называется состоянием состязания. Основным способом предотвращения состояния состязания является запрет использования совместно используемых данных одновременно нескольким процессам.

Критическая секция (области) – это участок программы, в котором есть обращение к совместно используемым данным. На этом участке запрещается переключение задач для обеспечения исключительного использования ресурсов процессом. Все ОСРВ предоставляют вызовы «войти в критическую секцию» и «выйти из критической секции».

Взаимное исключение (mutual exclusion, mutex) – это способ синхронизации параллельно работающих процессов, использующих разделяемый постоянный критичный

ресурс. Если ресурс занят, то системный вызов «захватить ресурс» переводит процесс из состояния выполнения в состояние ожидания. Когда ресурс будет освобожден посредством системного вызова «освободить ресурс», то этот процесс вернется в состояние выполнения и продолжит работу. Ресурс при этом перейдет в состояние «занят».

При взаимодействии процессов возможны следующие проблемы:

Смертельный захват, тупик (Deadlock). Обычно побочные эффекты этой ситуации называются более прозаично – « зацикливание» или «зависание». А причина этого может быть достаточно проста – « задачи не поделили ресурсы». Пусть, например, Задача А захватила ресурс клавиатуры и ждет, когда освободится ресурс дисплея, а в это время Задача В, успев захватить ресурс дисплея, ждет теперь, когда освободится клавиатура. В таких случаях рекомендуется придерживаться тактики «или все, или ничего». Другими словами, если задача не смогла получить все необходимые для дальнейшей работы ресурсы, она должна освободить все, что уже захвачено, и повторить попытку только через определенное время. Другим решением, которое уже упоминалось, является использование серверов ресурсов.

Инверсия приоритетов (Priority inversion). Представим, что у нас есть высокоприоритетная Задача А, среднеприоритетная Задача В и низкоприоритетная Задача С. Пусть в начальный момент времени Задачи А и В блокированы в ожидании какого-либо внешнего события. Допустим, получившая в результате этого управление Задача С захватила Ресурс А, но не успела его отдать, как была прервана Задачей А. В свою очередь, Задача А при попытке захватить Ресурс А будет блокирована, так как этот ресурс уже захвачен Задачей С. Если к этому времени Задача В находится в состоянии готовности, то управление после этого получит именно она, как имеющая более высокий, чем у Задачи С, приоритет. Теперь Задача В может занимать процессорное время, пока ей не надоест, а мы получаем ситуацию, когда высокоприоритетная Задача А не может функционировать из-за того, что необходимый ей ресурс занят низкоприоритетной Задачей С.

Блокировка (Lockout). Процесс ожидает ресурс, который никогда не освободится. Голодовка (Starvation). Процесс монополизировал процессор.

4.4.Семафоры. Мьютексы. Критические секции.

Семафор – это объект синхронизации, задающий количество пользователей (задач, процессов), имеющих одновременный доступ к некоторому ресурсу. С каждым семафором связаны счетчик (значение семафора) и очередь ожидания (процессов, задач, ожидающих

принятие счетчиком определенного значения). Различают:

- двоичные (булевские) семафоры – это механизм взаимного исключения для защиты критичного разделяемого ресурса;

- счетные семафоры – это механизм взаимного исключения для защиты ресурса, который может быть использован не более чем ограниченным фиксированным числом задач n.

Список литературы

1. Зыль С.Н. Проектирование, разработка и анализ программного обеспечения систем реального времени (+ CD-ROM) – СПб.: БХВ-Петербург, 2010.
2. Цилюрик О., Горошко Е. QNX/UNIX. Анатомия параллелизма. М.: Символ-Плюс, 2006 г.
3. Зыль С.Н. QNX Momentics: основы применения. – СПб.: БХВ-Петербург, 2005. – 225 с.: ил.
4. Гома Х. UML Проектирование систем реального времени, распределенных и параллельных приложений. М.: ДМК Пресс, 2011 г.
5. Алексеев Д. и др. Практика работы с QNX. М.: Издательский дом «КомБук», 2004. – 432 с.: ил.
6. Зыль С.Н. Операционная система реального времени QNX: от теории к практике. – 2-е изд., перераб. и доп. – СПб.: БХВ-Петербург, 2004. – 192 с.: ил.
7. Асотов Ю. Операционная система реального времени QNX Neutrino 6.3. Системная архитектура. – СПб.: БХВ-Петербург, 2006. – 336 с.
8. Стивенс У. UNIX: взаимодействие процессов. С-Пб., Питер, 2002 г