Департамент образования США

Академия WOOHP

Кафедра «Информационная Безопасность»

РЕФЕРАТ

По теме «Операционные системы»

Работу выполнил:

Студент группы 3-ИТ Alex Jonson \_\_\_\_\_\_\_\_

Работу проверил:

Зав. Кафедры Mark Links\_\_\_\_\_\_\_\_\_\_\_\_

**ВВЕДЕНИЕ**

Операционные системы реального времени (ОСРВ) предназначены для обеспечения интерфейса к ресурсам критических по времени систем реального времени. Основной задачей в таких системах является своевременность (timeliness) выполнения обработки данных. В качестве основного требования к ОСРВ выдвигается требование обеспечения предсказуемости или детерминированности поведения системы в наихудших внешних условиях, что резко отличается от требований к производительности и быстродействию универсальных ОС. Хорошая ОСРВ имеет предсказуемое поведение при всех сценариях системной загрузки (одновременные прерывания и выполнение потоков). Существует некое различие между системами реального времени и встроенными системами. От встроенной системы не всегда требуется, чтобы она имела предсказуемое поведение, и в таком случае она не является системой реального времени. Однако даже беглый взгляд на возможные встроенные системы позволяет утверждать, что большинство встроенных систем нуждается в предсказуемом поведении, по крайней мере, для некоторой функциональности, и таким образом, эти системы можно отнести к системам реального времени. Принято различать системы мягкого (soft) и жесткого (hard) реального времени. В системах жесткого реального времени неспособность обеспечить реакцию на какие-либо события в заданное время ведет к отказам и невозможности выполнения поставленной задачи. В большинстве русскоязычной литературы такие системы называют системами с детерминированным временем. При практическом применении время реакции должно быть минимальным. Системами мягкого реального времени называются системы, не попадающие под определение "жесткие", т.к. в литературе четкого определения для них пока нет. Системы мягкого реального времени могут не успевать решать задачу, но это не приводит к отказу системы в целом. В системах реального времени необходимо введение некоторого директивного срока (в англоязычной литературе – deadline), до истечения которого задача должна обязательно (для систем мягкого реального времени – желательно) выполниться. Этот директивный срок используется планировщиком задач как для назначения приоритета задачи при ее запуске, так и при выборе задачи на выполнение. Мартин Тиммерман сформулировал следующие необходимые требования для ОСРВ [DEDSYS]:

ОС должна быть многозадачной и допускающей вытеснение (preemptable),

ОС должна обладать понятием приоритета для потоков,

ОС должна поддерживать предсказуемые механизмы синхронизации,

ОС должна обеспечивать механизм наследования приоритетов,

поведение ОС должно быть известным и предсказуемым (задержки обработки прерываний, задержки переключения задач, задержки драйверов и т.д.); это значит, что во всех сценариях рабочей нагрузки системы должно быть определено максимальное время отклика. В течение последних 25-30 лет структура операционных систем эволюционировала от монолитной к многослойной структуре ОС и далее к архитектуре клиент-сервер. При монолитной структуре ОС состоит из набора модулей, и изменения одного модуля влияют на другие модули. Чем больше модулей, тем больше хаоса при эксплуатации такой системы. Кроме того, невозможно распределить ОС в многопроцессорной системе. В многослойной структуре изменения одного слоя влияют на соседние слои; кроме того, обращение через слой невозможно. Для систем реального времени должно быть обеспечено прямое обращение к каждому слою ОС, а иногда напрямую к аппаратуре. Основной идеей клиент-серверной технологии в ОС является сведение базиса ОС к минимуму (планировщик и примитивы синхронизации). Вся остальная функциональность выносится на другой уровень и реализуется через потоки или задачи. Совокупность таких серверных задач отвечает за системные вызовы. Приложения являются клиентами, которые запрашивают сервисы через системные вызовы. Клиент-серверная технология позволяет создавать масштабируемые ОС и упрощает распределение в многопроцессорной системе. При эксплуатации системы замена одного модуля не вызывает эффекта “снежного кома”; кроме того, сбой модуля не всегда влечет за собой отказ системы в целом. Появилась возможность динамической загрузки и отгрузки модулей. Главной проблемой в этой модели является защита памяти, поскольку серверные процессы должны быть защищены. При каждом запросе сервиса система должна переключаться с контекста приложения на контекст сервера. При поддержке защиты памяти время переключения с одного процесса на другой увеличивается. Как правило, большинство современных ОСРВ построено на основе микроядра (kernel или nucleus), которое обеспечивает планирование и диспетчеризацию задач, а также осуществляет их взаимодействие. Несмотря на сведение к минимуму в ядре абстракций ОС, микроядро все же должно иметь представление об абстракции процесса. Все остальные концептуальные абстракции операционных систем вынесены за пределы ядра, вызываются по запросу и выполняются как приложения.

* 1. **ОСНОВНЫЕ ПОНЯТИЯ**

Существуют две группы определений ОС: «совокупность программ, управляющих оборудованием» и «совокупность программ, управляющих другими программами». Обе они имеют свой точный технический смысл, который, однако, становится ясен только при более детальном рассмотрении вопроса о том, зачем вообще нужны операционные системы. Есть приложения вычислительной техники, для которых ОС излишни. Напр., встроенные микрокомпьютеры содержатся сегодня во многих бытовых приборах, автомобилях (иногда по десятку в каждом), сотовых телефонах и т. п. Зачастую такой компьютер постоянно исполняет лишь одну программу, запускающуюся по включении. И простые игровые приставки — также представляющие собой специализированные микрокомпьютеры — могут обходиться без ОС, запуская при включении программу, записанную на вставленном в устройство «картридже» или компакт-диске. (Многие встроенные компьютеры и даже некоторые игровые приставки на самом деле работают под управлением своих ОС). Операционные системы, в свою очередь, нужны, если:

вычислительная система используется для различных задач, причём программы, исполняющие эти задачи, нуждаются в сохранении данных и обмене ими. Из этого следует необходимость универсального механизма сохранения данных; в подавляющем большинстве случаев ОС отвечает на неё реализацией файловой системы. Современные ОС, кроме того, предоставляют возможность непосредственно «связать» вывод одной программы с вводом другой, минуя относительно медленные дисковые операции;

различные программы нуждаются в выполнении одних и тех же рутинных действий. Напр., простой ввод символа с клавиатуры и отображение его на экране может потребовать исполнения сотен машинных команд, а дисковая операция — тысяч. Чтобы не программировать их каждый раз заново, ОС предоставляют системные библиотеки часто используемых подпрограмм (функций);

между программами и пользователями системы необходимо распределять полномочия, чтобы пользователи могли защищать свои данные от чужого взора, а возможная ошибка в программе не вызывала тотальных неприятностей;

необходима возможность имитации «одновременного» исполнения нескольких программ на одном компьютере (даже содержащем лишь один процессор), осуществляемой с помощью приёма, известного как «разделение времени». При этом специальный компонент, называемый планировщиком, «нарезает» процессорное время на короткие отрезки и предоставляет их поочередно различным исполняющимся программам (процессам) наконец, оператор должен иметь возможность, так или иначе, управлять процессами выполнения отдельных программ. Для этого служат операционные среды, одна из которых — оболочка и набор стандартных утилит — является частью ОС (прочие, такие, как графическая операционная среда, образуют независимые от ОС прикладные платформы). Таким образом, современные универсальные ОС можно охарактеризовать прежде всего как

использующие файловые системы (с универсальным механизмом доступа к данным),

многопользовательские (с разделением полномочий),

многозадачные (с разделением времени).Многозадачность и распределение полномочий требуют определённой иерархии привилегий компонентов самой ОС. В составе ОС различают три группы компонентов:

ядро, содержащее планировщик; драйверы устройств, непосредственно управляющие оборудованием; сетевую подсистему, файловую систему;

системные библиотеки и оболочку с утилитами. Большинство программ, как системных (входящих в ОС), так и прикладных, исполняются в непривилегированном («пользовательском») режиме работы процессора и получают доступ к оборудованию (и, при необходимости, к другим ядерным ресурсам, а также ресурсам иных программ) только посредством системных вызовов. Ядро исполняется в привилегированном режиме: именно в этом смысле говорят, что ОС (точнее, её ядро) управляет оборудованием. В определении состава ОС значение имеет критерий операциональной целостности (замкнутости): система должна позволять полноценно использовать (включая модификацию) свои компоненты. Поэтому в полный состав ОС включают и набор инструментальных средств (от текстовых редакторов до компиляторов, отладчиков и компоновщиков).

* 1. **ФУНКЦИИ**

Основные функции (простейшие ОС):

Загрузка приложений в оперативную память и их выполнение;

Стандартизованный доступ к периферийным устройствам (устройства ввода-вывода);

Управление оперативной памятью (распределение между процессами, виртуальная память);

Управление доступом к данным на энергонезависимых носителях (таких как Жёсткий диск, Компакт-диск и т. д.), как правило, с помощью файловой системы;

Пользовательский интерфейс;

Сетевые операции, поддержка стека протоколов

Дополнительные функции:

Параллельное или псевдопараллельное выполнение задач (многозадачность);

Взаимодействие между процессами: обмен данными, взаимная синхронизация;

Защита самой системы, а также пользовательских данных и программ от злонамеренных действий пользователей или приложений;

Разграничение прав доступа и многопользовательский режим работы (аутентификация, авторизация).

* 1. **ПАМЯТЬ**

Как уже упоминалось выше, задержка 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 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. В системах жесткого реального времени обычно применяется статическое распределение памяти. В системах мягкого реального времени, возможно динамическое распределение памяти, без виртуальной памяти и без уплотнения.

* 1. **Стандарты безопасности**

В связи со стандартами для ОСРВ стоит отметить широко известный стандарт критериев оценки пригодности компьютерных систем (Trusted Computer System Evaluation Criteria – TCSEC) [DoD85]. Этот стандарт разработан Министерством обороны США и известен также под названием "Оранжевая книга" (Orange Book – из-за цвета обложки). В ряде других стран были разработаны аналогичные критерии, на основе которых был создан международный стандарт “Общие критерии оценки безопасности информационных технологий” (далее просто – Общие критерии) (Common Criteria for IT Security Evaluation, ISO/IEC 15408) [CC99].

В "Оранжевой книге" перечислены семь уровней защиты:

А1 – верифицированная разработка. Этот уровень требует, чтобы защиту секретной и другой критичной информации средствами управления безопасностью гарантировали методы формальной верификации.

В3 – домены безопасности. Этот уровень предназначен для защиты систем от опытных программистов.

В2 – структурированная защита. В систему с этим уровнем защиты нельзя допустить проникновение хакеров.

В1 – мандатный контроль доступа. Защиту этого уровня, возможно, удастся преодолеть опытному хакеру, но никак не рядовым пользователям.

С2 – дискреционный контроль доступа. Уровень С2 обеспечивает защиту процедур входа, позволяет производить контроль за событиями, имеющими отношение к безопасности, а также изолировать ресурсы.

С1 – избирательная защита. Этот уровень дает пользователям возможность защитить личные данные или информацию о проекте, установив средства управления доступом.

D – минимальная защита. Этот нижний уровень защиты оставлен для систем, которые проходили тестирование, но не смогли удовлетворить требованиям более высокого класса.

Что касается Общих критериев, то в них введены похожие требования обеспечения безопасности в виде оценочных уровней (Evaluation Assurance Levels – EAL). Их также семь:

EAL7 – самый высокий уровень предполагает формальную верификацию модели объекта оценки. Он применим к системам очень высокого риска.

EAL6 определяется, как полуформально верифицированный и протестированный. На уровне EAL6 реализация должна быть представлена в структурированном виде, анализ соответствия распространяется на проект нижнего уровня, проводится строгий анализ покрытия, анализ и тестирование небезопасных состояний.

EAL5 определяется, как полуформально спроектированный и протестированный. Он предусматривает создание полуформальной функциональной спецификации и проекта высокого уровня с демонстрацией соответствия между ними, формальной модели политики безопасности, стандартизованной модели жизненного цикла, а также проведение анализа скрытых каналов.

EAL4 определяется, как методически спроектированный, протестированный и пересмотренный. Он предполагает наличие автоматизации управления конфигурацией, полной спецификации интерфейсов, описательного проекта нижнего уровня, подмножества реализаций функций безопасности, неформальной модели политики безопасности, модели жизненного цикла, анализ валидации, независимый анализ уязвимостей. По всей вероятности, это самый высокий уровень, которого можно достичь на данном этапе развития технологии программирования с приемлемыми затратами.

EAL3 определяется, как методически протестированный и проверенный. На уровне EAL3 осуществляется более полное, чем на уровне EAL2, тестирование покрытия функций безопасности, а также контроль среды разработки и управление конфигурацией объекта оценки.

EAL2 определяется, как структурно протестированный. Он предусматривает создание описательного проекта верхнего уровня объекта оценки, описание процедур инсталляции и поставки, руководств администратора и пользователя, функциональное и независимое тестирование, оценку прочности функций безопасности, анализ уязвимостей разработчиками.

EAL1 определяется, как функционально протестированный. Он обеспечивает анализ функций безопасности с использованием функциональной спецификации и спецификации интерфейсов, руководящей документации, а также независимое тестирование. На этом уровне угрозы не рассматриваются как серьезные.

В соответствии с требованиями Общих критериев, продукты определенного класса (например, операционные системы) оцениваются на соответствие ряду функциональных критериев и критериев доверия – профилей защиты. Существуют различные определения профилей защиты в отношении операционных систем, брандмауэров, смарт-карт и прочих продуктов, которые должны соответствовать определенным требованиям в области безопасности. Например, профиль защиты систем с разграничением доступа (Controlled Access Protection Profile) действует в отношении операционных систем и призван заменить старый уровень защиты С2, определявшийся в соответствии с американским стандартом TCSEC. В соответствии с оценочными уровнями доверия сертификация на соответствие более высокому уровню означает более высокую степень уверенности в том, что система защиты продукта работает правильно и эффективно, и, согласно условиям Общих критериев, уровни 5-7 рассчитаны на тестирование продуктов, созданных с применением специализированных технологий безопасности. Следует отметить, что большинство усилий по оценке продуктов безопасности сосредоточены на уровне 4 стандарта Общих критериев и ниже, что говорит об ограниченном применении формальных методов в этой области. С точки зрения программиста Общие критерии можно рассматривать как набор библиотек, с помощью которых пишутся задания по безопасности, типовые профили защиты и т.п. Следует отметить, что требования могут быть параметризованы.

**1.5. Процессы, потоки, задачи**

Концепция многозадачности (псевдопараллелизм) является существенной для системы реального времени с одним процессором, приложения которой должны быть способны обрабатывать многочисленные внешние события, происходящие практически одновременно. 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). Потоки существуют в одном контексте процесса, поэтому переключение между потоками происходит очень быстро, а вопросы безопасности не принимаются во внимание. Потоки являются легковесными, потому что их регистровый контекст меньше, т.е. их управляющие блоки намного компактнее. Уменьшаются накладные расходы, вызванные сохранением и восстановлением управляющих блоков прерываемых потоков. Объем управляющих блоков зависит от конфигурации памяти. Если потоки выполняются в разных адресных пространствах, система должна поддерживать отображение памяти для каждого набора потоков.

Итак, в системах реального времени процесс распадается на задачи или потоки. В любом случае каждый процесс рассматривается как приложение. Между этими приложениями не должно быть слишком много взаимодействий, и в большинстве случаев они имеют различную природу – жесткого реального времени, мягкого реального времени, не реального времени.

К концу 1960-х гг. отраслью и научно-образовательным сообществом был создан целый ряд ОС, реализующих все или часть очерченных выше функций. К ним относятся «Atlas» (Манчестерский университет), «CTTS» и «ITSS» (Массачусетский технологический институт (МТИ)), «THE» (Эйндховенский технологический университет), «RS4000» (Университет Орхуса) и др. (всего эксплуатировалось более сотни различных ОС).Наиболее развитые ОС, такие как «OS/360» (компания «IBM»), «SCOPE» (компания «CDC») и завершённый уже в 1970-х годах «MULTICS» (МТИ и компания «Bell Labs»), предусматривали возможность исполнения на многопроцессорных компьютерах. Эклектичный характер разработки ОС привёл к нарастанию кризисных явлений, прежде всего, связанных с чрезмерными сложностью и размерами создаваемых систем. ОС были плохо масштабируемыми (более простые не могли использовать все возможности крупных вычислительных систем; более развитые неоптимальное исполнялись на малых или не могли исполняться на них вовсе) и тотально несовместимыми между собой, их разработка и совершенствование затягивалась. Задуманная и реализованная в 1969 году Кеном Томпсоном при участии нескольких коллег (включая Денниса Ричи и Брайана Кернигана), ОС «Unix» («Unix»; первоначально «UNICS», что обыгрывало название «MULTICS») вобрала в себя многие черты более ранних ОС, но обладала целым рядом свойств, отличающих её от большинства предшественниц: простая метафорика (два ключевых понятия: вычислительный процесс и файл); компонентная архитектура: принцип «одна программа — одна функция» плюс мощные средства связывания различных программ для решения возникающих задач («оболочка»); минимизация ядра (кода, выполняющегося в «реальном» («привилегированном») режиме процессора) и количества системных вызовов;независимость от аппаратной архитектуры и реализация на машинно-независимом языке программирования (язык программирования «Си» стал «побочным продуктом» разработки «Unix»); унификация файлов. «Unix», благодаря своему удобству прежде всего в качестве инструментальной среды (среды разработки), была тепло принята сначала в университетах, а затем и в отрасли, получившей прототип единой ОС, которая могла использоваться на самых разных вычислительных системах и, более того, могла быть быстро и с минимальными усилиями перенесена на любую вновь разработанную аппаратную архитектуру. В конце 70-х годов XX века сотрудники Калифорнийского университета в Беркли внесли ряд усовершенствований в исходные коды UNIX, включая работу с протоколами TCP/IP. Их разработка стала известна под именем BSD — «Berkeley Software Distribution». Задачу разработать независимую (от авторских прав «Bell Labs») реализацию той же архитектуры поставил и Ричард Столлмен, основатель проекта «GNU». Благодаря конкурентности реализаций архитектура ОС «Unix» стала вначале фактическим отраслевым стандартом, а затем обрела статус и стандарта юридического — ISO/IEC 9945[1].

ОС, следующие стандарту или опирающиеся на него, называют «POSIX-совместимыми» (чаще встречается словоупотребление «Unix подобные» или «семейство Unix», но оно противоречит статусу торгового знака «Unix», принадлежащего консорциуму «The Open Group» и зарезервированному для обозначения ОС, строго следующих стандарту) благодаря названию стандарта — POSIX. Сертификация на совместимость со стандартом стоит некоторых денег, из-за чего некотрые системы не проходили этот процесс, однако считаются POSIX-совместимыми, просто потому что это так. К Unix-подобным ОС относятся системы, базирующиеся на последней версии «Unix», выпущенной «Bell Labs» («System V»), на разработках Университета Беркли («FreeBSD», «OpenBSD», «NetBSD»), а также ОС «GNU/Linux», разработанная в части утилит и библиотек проектом «GNU» и в части ядра — сообществом, возглавляемым Линусом Торвальдсом. Стандартизация ОС гарантирует возможность безболезненной замены самой ОС и/или оборудования при развитии вычислительной системы или сети и дешёвого переноса прикладного программного обеспечения (строгое следование стандарту предполагает полную совместимость программ на уровне исходного текста; из-за профилирования стандарта и его развития некоторые изменения бывают всё же необходимы, но перенос программы между POSIX-совместимыми системами обходится на порядки дешевле, чем между альтернативными), а также преемственность опыта пользователей. Самым заметным эффектом существования этого стандарта стало эффективное разворачивание Интернета в 90-х годах.

**Список используемой Литературы:**

Вильям Столлингс «Операционные системы» = Operating Systems: Internals and Design Principles. — М.: «Вильямс», 2004. — С. 848. — ISBN 0-13-031999-6

Деннинг П.Дж., Браун Р. Л. «Операционные системы.» В сб.: «Современный компьютер». — М.: 1986.

Керниган Брайан и Пайк Роб. «UNIX — универсальная среда программирования». — М., 1992 (классическое введение в открытые ОС, по большей части сохранившее актуальность).

Отставнов Максим. «Свободные программы и системы в школе». — М., 2003.

Э. Таненбаум, А. Вудхалл. «Операционные системы: Разработка и реализация.» — СПб.: 2006. — ISBN 5-469-00148-2

Э. Таненбаум. «Современные операционные системы. 2-е изд.» — СПб.: Питер, 2005. — 1038 с.: ил. ISBN 5-318-00299-4

Дмитрий Иртегов. «Введение в операционные системы 2-е. изд.» — BHV-СПб, 2007. ISBN 978-5-94157-695-1

А. Гордеев. «Операционные системы» — СПб.: Питер, 2007. ISBN 978-5-94723-632-3 (учебник для ВУЗов)

Raymond Eric S. The Art of Unix Programming. — 2003.

Sobell Mark G. Unix System V. A Practical Guide. 3rd ed. — 1995.

Шоу А. "Логическое проектирование операционных систем": Пер. с англ. - М.Мир, 1981. - 360 с., ил.

Интернет.