

СОДЕРЖАНИЕ

Об авторе	26
Предисловие	27
Новое в девятом издании	27
Цели	28
Примеры систем	28
Поддержка курса ACM/IEEE computer science curricula 2013	29
План книги	29
Материалы для преподавателей	31
Проекты и упражнения для студентов	32
OS/161	32
Моделирование	33
Анимации	33
Программные проекты	33
Онлайн-документация и видеопримечания для студентов	34
Благодарности	34
Ждем ваших отзывов!	36
Часть I. Основы	37
Глава 1. Обзор компьютерной системы	39
1.1. Основные элементы	40
1.2. Эволюция микропроцессоров	42
1.3. Выполнение команд	43
1.4. Прерывания	46
Прерывания и цикл команды	48
Обработка прерывания	50
Множественные прерывания	54
1.5. Иерархия памяти	57
1.6. Кеш	60
Обоснование	61
Принципы работы кеша	61
Внутреннее устройство кеша	64
1.7. Прямой доступ к памяти	65
1.8. Организация многопроцессорных и многоядерных систем	66
Симметричная многопроцессорность	67
Многоядерные компьютеры	69
1.9. Ключевые термины, контрольные вопросы и задачи	71
Ключевые термины	71
Контрольные вопросы	71
Задачи	72
Приложение 1.A. Характеристики производительности двухуровневой памяти	75
Локальность	75
Функционирование двухуровневой памяти	78
Производительность	78

Глава 2. Обзор операционных систем	83
2.1. Цели и функции операционных систем	85
Операционная система как интерфейс между пользователем и компьютером	86
Операционная система как диспетчер ресурсов	88
Простота развития операционной системы	89
2.2. Эволюция операционных систем	90
Последовательная обработка	90
Простые пакетные системы	91
Многозадачные пакетные системы	95
Системы, работающие в режиме разделения времени	98
2.3. Основные достижения	100
Процессы	101
Управление памятью	105
Защита информации и безопасность	108
Планирование и управление ресурсами	108
2.4. Разработки, ведущие к современным операционным системам	110
2.5. Отказоустойчивость	114
Фундаментальные концепции	114
Отказы	116
Механизмы операционных систем	117
2.6. Вопросы проектирования операционных систем для многопроцессорных и многоядерных систем	117
Операционные системы для SMP	117
Вопросы проектирования операционных систем для многоядерных систем	119
2.7. Обзор операционной системы Microsoft Windows	121
Основы	121
Архитектура	121
Модель “клиент/сервер”	125
Потоки и симметричная многопроцессорность	126
Объекты Windows	127
2.8. Традиционные системы UNIX	129
Историческая справка	129
Описание	130
2.9. Современные системы UNIX	132
System V Release 4 (SVR4)	133
BSD	134
Solaris 11	134
2.10. Linux	134
История	134
Модульная структура	136
Компоненты ядра	138
2.11. Android	141
Программная архитектура Android	142
Система времени выполнения Android	144
Системная архитектура Android	147
Операции	149
Управление электропитанием	149

2.12. Ключевые термины, контрольные вопросы и задачи	150
Ключевые термины	150
Контрольные вопросы	150
Задачи	151
Часть II. Процессы	153
Глава 3. Описание процессов и управление ими	155
3.1. Что такое процесс	157
Основы	157
Процессы и управляющие блоки процессов	158
3.2. Состояния процесса	159
Модель процесса с двумя состояниями	162
Создание и завершение процессов	163
Модель с пятью состояниями	166
Приостановленные процессы	170
3.3. Описание процессов	177
Управляющие структуры операционной системы	177
Структуры управления процессами	179
3.4. Управление процессами	188
Режимы выполнения	188
Создание процессов	189
Переключение процессов	190
3.5. Выполнение кода операционной системы	194
Ядро вне процессов	194
Выполнение в составе пользовательских процессов	195
Операционная система на основе процессов	197
3.6. Управление процессами в операционной системе UNIX SVR4	198
Состояния процессов	198
Описание процессов	200
Управление процессами	203
3.7. Резюме	204
3.8. Ключевые термины, контрольные вопросы и задачи	205
Ключевые термины	205
Контрольные вопросы	205
Задачи	206
Глава 4. Потoki	211
4.1. Процессы и потоки	213
Многопоточность	214
Функциональность потоков	218
4.2. Типы потоков	220
Потоки на пользовательском уровне и на уровне ядра	220
Другие схемы	226
4.3. Многоядерность и многопоточность	228
Производительность программного обеспечения в многоядерных системах	228
Пример приложения: игровые программы Valve	231

4.4. Управление процессами и потоками в Windows	233
Управление фоновыми задачами и жизненным циклом приложений	234
Процессы в Windows	236
Объекты процессов и потоков	237
Многопоточность	238
Состояния потоков	239
Поддержка подсистем операционной системы	240
4.5. Управление потоками и SMP в Solaris	241
Многопоточная архитектура	241
Мотивация	242
Структура процессов	242
Выполнение потоков	244
Прерывания в роли потоков	245
4.6. Управление процессами и потоками в Linux	246
Задания Linux	246
Потоки Linux	248
Пространства имен Linux	249
4.7. Управление процессами и потоками в Android	252
Приложения Android	252
Операции	253
Процессы и потоки	255
4.8. Mac OS X Grand Central Dispatch	256
4.9. Резюме	259
4.10. Ключевые термины, контрольные вопросы и задачи	259
Ключевые термины	259
Контрольные вопросы	259
Задачи	260
Глава 5. Параллельные вычисления: взаимоисключения и многозадачность	265
5.1. Взаимоисключения: программный подход	269
Алгоритм Деккера	269
Алгоритм Петерсона	275
5.2. Принципы параллельных вычислений	276
Простой пример	277
Состояние гонки	279
Участие операционной системы	279
Взаимодействие процессов	280
Требования к взаимным исключениям	285
5.3. Взаимоисключения: аппаратная поддержка	285
Отключение прерываний	285
Специальные машинные команды	286
5.4. Семафоры	289
Задача производителя/потребителя	296
Реализация семафоров	302
5.5. Мониторы	304
Мониторы с сигналами	305
Мониторы с оповещением и широковещанием	308

5.6. Передача сообщений	311
Синхронизация	312
Адресация	314
Формат сообщения	315
Принцип работы очереди	316
Взаимные исключения	316
5.7. Задача читателей/писателей	318
Приоритетное чтение	319
Приоритетная запись	319
5.8. Резюме	324
5.9. Ключевые термины, контрольные вопросы и задачи	325
Ключевые термины	325
Контрольные вопросы	325
Задачи	326
Глава 6. Параллельные вычисления: взаимоблокировка и голодание	341
6.1. Принципы взаимного блокирования	343
Повторно используемые ресурсы	347
Расходуемые ресурсы	349
Графы распределения ресурсов	349
Условия возникновения взаимоблокировок	351
6.2. Предотвращение взаимоблокировок	352
Взаимоисключения	352
Удержание и ожидание	353
Отсутствие перераспределения	353
Циклическое ожидание	353
6.3. Устранение взаимоблокировок	354
Запрещение запуска процесса	354
Запрет выделения ресурса	355
6.4. Обнаружение взаимоблокировок	360
Алгоритм обнаружения взаимоблокировки	360
Восстановление	361
6.5. Интегрированные стратегии разрешения взаимоблокировок	362
6.6. Задача об обедающих философах	363
Решение с использованием семафоров	364
Решение с использованием монитора	364
6.7. Механизмы параллельных вычислений в UNIX	367
Каналы	367
Сообщения	368
Совместно используемая память	368
Семафоры	368
Сигналы	369
6.8. Механизмы параллельных вычислений ядра Linux	370
Атомарные операции	371
Циклические блокировки	373
Семафоры	375
Барьеры	377

6.9. Прimitives синхронизации потоков Solaris	379
Блокировки взаимного исключения	380
Семафоры	381
Блокировки “читатели/писатель”	381
Условные переменные	382
6.10. Механизмы параллельных вычислений в Windows	382
Функции ожидания	382
Объекты диспетчера	383
Критические участки	384
Гибкие блокировки читателя/писателя и условные переменные	385
Синхронизация без участия блокировок	385
6.11. Межпроцессное взаимодействие в Android	386
6.12. Резюме	388
6.13. Ключевые термины, контрольные вопросы и задачи	388
Ключевые термины	388
Контрольные вопросы	389
Задачи	389
Часть III. Память	397
Глава 7. Управление памятью	399
7.1. Требования к управлению памятью	401
Перемещение	401
Защита	401
Совместное использование	403
Логическая организация	403
Физическая организация	403
7.2. Распределение памяти	404
Фиксированное распределение	404
Динамическое распределение	408
Система двойников	412
Перемещение	414
7.3. Страничная организация памяти	416
7.4. Сегментация	419
7.5. Резюме	421
7.6. Ключевые термины, контрольные вопросы и задачи	422
Ключевые термины	422
Контрольные вопросы	422
Задачи	423
Приложение 7.A. Загрузка и связывание	426
Загрузка	426
Компоновка	430
Глава 8. Виртуальная память	433
8.1. Аппаратное обеспечение и управляющие структуры	436
Локальность и виртуальная память	437
Страничная организация	439

Сегментация	451
Комбинация сегментации и страничной организации	453
Защита и совместное использование	454
8.2. Программное обеспечение операционной системы	455
Стратегия выборки	457
Стратегия размещения	457
Стратегия замещения	458
Управление резидентным множеством	465
Стратегия очистки	473
Управление загрузкой	473
8.3. Управление памятью в UNIX и Solaris	476
Страничная система	476
Распределение памяти ядра	480
8.4. Управление памятью в Linux	481
Виртуальная память Linux	483
Распределение памяти ядра	485
8.5. Управление памятью в Windows	486
Карта виртуальных адресов Windows	486
Страничная организация Windows	488
Свопинг в Windows	489
8.6. Управление памятью в Android	489
8.7. Резюме	490
8.8. Ключевые термины, контрольные вопросы и задачи	491
Ключевые термины	491
Контрольные вопросы	491
Задачи	492
Часть IV. Планирование	497
Глава 9. Однопроцессорное планирование	499
9.1. Типы планирования процессора	501
Долгосрочное планирование	502
Среднесрочное планирование	503
Краткосрочное планирование	504
9.2. Алгоритмы планирования	504
Критерии краткосрочного планирования	504
Использование приоритетов	506
Альтернативные стратегии планирования	507
Сравнение производительности	521
Справедливое планирование	526
9.3. Традиционное планирование UNIX	528
9.4. Резюме	530
9.5. Ключевые термины, контрольные вопросы и задачи	532
Ключевые термины	532
Контрольные вопросы	532
Задачи	533

Глава 10. Многопроцессорное планирование и планирование реального времени	539
10.1. Многопроцессорное и многоядерное планирование	541
Зернистость	542
Вопросы проектирования	544
Планирование процессов	546
Планирование потоков	546
Планирование потоков в многоядерных системах	554
10.2. Планирование реального времени	555
Введение	555
Характеристики операционных систем реального времени	556
Планирование реального времени	560
Планирование с предельными сроками	562
Частотно-монотонное планирование	566
Инверсия приоритета	569
10.3. Планирование в Linux	572
Планирование реального времени	572
Обычное планирование	574
10.4. Планирование в UNIX SVR4	576
10.5. Планирование в UNIX FreeBSD	578
Классы приоритетов	578
Поддержка SMP и многоядерности	579
10.6. Планирование в Windows	581
Приоритеты процессов и потоков	582
Многопроцессорное планирование	584
10.7. Резюме	585
10.8. Ключевые термины, контрольные вопросы и задачи	585
Ключевые термины	585
Контрольные вопросы	586
Задачи	586
Часть V. Ввод-вывод и файлы	591
Глава 11. Управление вводом-выводом и планирование дисковых операций	593
11.1. Устройства ввода-вывода	595
11.2. Организация функций ввода-вывода	597
Эволюция функций ввода-вывода	598
Прямой доступ к памяти	599
11.3. Вопросы проектирования операционных систем	601
Цели проектирования	601
Логическая структура функций ввода-вывода	602
11.4. Буферизация операций ввода-вывода	604
Двойной буфер	607
Циклический буфер	607
Использование буферизации	607
11.5. Дисковое планирование	608
Параметры производительности диска	608
Стратегии дискового планирования	611

11.6. RAID	617
RAID 0	618
RAID 1	622
RAID 2	623
RAID 3	623
RAID 4	624
RAID 5	625
RAID 6	625
11.7. Дисковый кеш	626
Вопросы разработки	626
Вопросы производительности	628
11.8. Ввод-вывод в UNIX SVR4	630
Буфер кеша	630
Очередь символов	632
Небуферизованный ввод-вывод	632
Устройства UNIX	632
11.9. Ввод-вывод в Linux	633
Дисковое планирование	633
Страничный кеш Linux	637
11.10. Ввод-вывод в Windows	638
Основные средства ввода-вывода	638
Асинхронный и синхронный ввод-вывод	639
Программное обеспечение RAID	640
Теневые копии тома	640
Шифрование тома	640
11.11. Резюме	641
11.12. Ключевые термины, контрольные вопросы и задачи	642
Ключевые термины	642
Контрольные вопросы	642
Задачи	643
Глава 12. Управление файлами	645
12.1. Обзор	647
Файлы и файловые системы	647
Структура файла	648
Системы управления файлами	650
Функции управления файлами	652
12.2. Организация файлов и доступ к ним	654
Смешанный файл	654
Последовательный файл	656
Индексно-последовательный файл	657
Индексированный файл	658
Файл прямого доступа	658
12.3. В-деревья	659
12.4. Каталоги файлов	662
Содержимое	662
Структура	663
Именованное	665

12.5. Совместное использование файлов	667
Права доступа	667
Одновременный доступ	668
12.6. Записи и блоки	668
12.7. Управление вторичной памятью	670
Размещение файлов	670
Управление свободным пространством	676
Тома	678
Надежность	679
12.8. Управление файлами в UNIX	679
Индексные узлы	680
Размещение файлов	682
Каталоги	683
Структура тома	684
12.9. Виртуальная файловая система Linux	684
Суперблок	686
Индексный узел	687
Запись каталога	687
Файл	687
Кеши	688
12.10. Файловая система Windows	688
Ключевые возможности NTFS	688
Том NTFS и файловая структура	689
Способность восстановления данных	692
12.11. Управление файлами в Android	693
Файловая система	693
SQLite	695
12.12. Резюме	695
12.13. Ключевые термины, контрольные вопросы и задачи	696
Ключевые термины	696
Контрольные вопросы	696
Задачи	697
Часть VI. Дополнительные темы	699
Глава 13. Встроенные операционные системы	701
13.1. Встроенные системы	703
Концепции встроенных систем	703
Прикладные и специализированные процессоры	705
Микропроцессоры	705
Микроконтроллеры	707
Глубоко встроенные системы	708
13.2. Характеристики встроенных операционных систем	709
Исходные и целевые среды	710
Подходы к разработке	712
Адаптация существующей коммерческой операционной системы	713
Специально разработанная встроенная операционная система	713

13.3. Встроенная система Linux	714
Характеристики встроенной системы Linux	714
Файловые системы встроенного Linux	716
Преимущества встроенных систем Linux	717
μClinux	718
Android	720
13.4. TinyOS	721
Беспроводные сети датчиков	722
Цели TinyOS	723
Компоненты TinyOS	724
Планировщик в TinyOS	727
Пример конфигурации	728
Интерфейс ресурсов TinyOS	730
13.5. Ключевые термины, контрольные вопросы и задачи	732
Ключевые термины	732
Контрольные вопросы	732
Задачи	733
Глава 14. Виртуальные машины	735
14.1. Концепции виртуальных машин	736
14.2. Гипервизоры	740
Назначение гипервизоров	740
Паравиртуализация	743
Аппаратно поддерживаемая виртуализация	744
Виртуальное устройство	744
14.3. Контейнерная виртуализация	745
Группы управления ядром	745
Концепции контейнеров	746
Файловая система контейнера	750
Микрослужбы	750
Docker	752
14.4. Вопросы виртуализации на уровне процессоров	753
14.5. Управление памятью	755
14.6. Управление вводом-выводом	757
14.7. VMware ESXi	760
14.8. Варианты Hyper-V и Xen от корпорации Microsoft	762
14.9. Java VM	764
14.10. Архитектура виртуальной машины Linux VServer	765
Архитектура	765
Планирование процессов	767
14.11. Резюме	769
14.12. Ключевые термины, контрольные вопросы и задачи	769
Ключевые термины	769
Контрольные вопросы	769
Задачи	770

Глава 15. Безопасность операционных систем	771
15.1. Злоумышленники и зловредные программы	773
Угрозы системного доступа	773
Контрмеры	775
15.2. Переполнение буфера	778
Атаки типа переполнения буфера	778
Защита времени компиляции	782
Защита времени выполнения	785
15.3. Управление доступом	787
Управление доступом к файловой системе	787
Стратегии управления доступом	790
15.4. Управление доступом в UNIX	797
Традиционное управление доступом к файлам в UNIX	797
Списки управления доступом в UNIX	799
15.5. Усиление защиты операционных систем	800
Установка операционной системы и применение обновлений	801
Удаление ненужных служб, приложений и протоколов	802
Конфигурирование пользователей, групп и аутентификации	802
Конфигурирование средств управления ресурсами	803
Установка дополнительных средств управления защитой	804
Тестирование защиты системы	805
15.6. Поддержание безопасности	805
Протоколирование	805
Резервное копирование и архивирование данных	806
15.7. Безопасность Windows	806
Схема управления доступом	807
Токен доступа	808
Дескрипторы безопасности	809
15.8. Резюме	813
15.9. Ключевые термины, контрольные вопросы и задачи	814
Ключевые термины	814
Контрольные вопросы	814
Задачи	814
Глава 16. Облачные операционные системы и операционные системы Интернета вещей	819
16.1. Облачные вычисления	821
Элементы облачных вычислений	821
Модели предоставления услуг облачных вычислений	823
Модели развертывания облака	824
Эталонная архитектура облачных вычислений	828
16.2. Облачные операционные системы	831
Инфраструктура как служба	832
Требования к облачной операционной системе	834
Общая архитектура облачной операционной системы	835
OpenStack	842

16.3. Интернет вещей	851
Вещи в Интернете вещей	851
Эволюция	852
Компоненты IoT-устройств	852
Интернет вещей в контексте облака	853
Границы	853
Туманные вычисления	853
Базовая сеть	855
Облачная сеть	855
16.4. Операционные системы для Интернета вещей	856
Устройства с ограниченными ресурсами	857
Требования к операционным системам для Интернета вещей	858
Архитектура операционной системы для Интернета вещей	860
Операционная система RIOT	862
16.5. Ключевые термины и контрольные вопросы	865
Ключевые термины	865
Контрольные вопросы	865
Глава 17. Сетевые протоколы	867
17.1. Потребность в архитектуре протоколов	869
17.2. Архитектура протоколов TCP/IP	872
Уровни протоколов TCP/IP	872
Протоколы TCP и UDP	873
Протоколы IP и IPv6	875
Принцип действия протоколов TCP/IP	875
Приложения протокола TCP/IP	879
17.3. Сокеты	879
Сокет	880
Вызовы интерфейса Socket	881
17.4. Организация сетей в Linux	884
Передача данных	885
Прием данных	885
17.5. Резюме	886
17.6. Ключевые термины, контрольные вопросы и задачи	887
Ключевые термины	887
Контрольные вопросы	887
Задачи	887
Приложение 17.A. Простой протокол передачи файлов	891
Введение в протокол TFTP	891
Пакеты TFTP	891
Краткий обзор передачи данных	893
Ошибки и задержки	894
Синтаксис, семантика и синхронизация	895

Глава 18. Распределенная обработка, вычисления “клиент/сервер” и кластеры	897
18.1. Вычисления “клиент/сервер”	899
Что такое вычисления “клиент/сервер”	899
Приложения “клиент/сервер”	901
Промежуточное программное обеспечение	909
18.2. Распределенный обмен сообщениями	912
Надежность и ненадежность	914
Блокировка и неблокирующее выполнение	915
18.3. Вызов удаленных процедур	915
Передача параметров	917
Представление параметров	917
Привязка к архитектуре “клиент/сервер”	917
Синхронность и асинхронность	918
Объектно-ориентированные механизмы	918
18.4. Кластеры	919
Конфигурации кластеров	920
Вопросы проектирования операционных систем	922
Архитектура кластерных вычислительных систем	924
Кластеры в сравнении с симметричной многопроцессорной обработкой	925
18.5. Кластерный сервер Windows	926
18.6. Кластеры Beowulf и Linux	928
Функциональные средства Beowulf	928
Программное обеспечение Beowulf	929
18.7. Резюме	930
18.8. Ключевые термины, контрольные вопросы и задачи	931
Ключевые термины	931
Контрольные вопросы	931
Задачи	931
Глава 19. Управление распределенными процессами	933
19.1. Перенос процессов	934
Побудительные причины	934
Механизмы переноса процессов	935
Согласования переноса процессов	939
Выселение	941
Вытесняющие переносы в сравнении с невытесняющими	942
19.2. Распределенные глобальные состояния	942
Глобальные состояния и распределенные моментальные снимки	942
Алгоритм распределенных моментальных снимков	945
19.3. Распределенное взаимное исключение	947
Принципы распределенного взаимного исключения	948
Упорядочение событий в распределенной системе	950
Распределенная очередь	953
Метод передачи эстафеты	957
19.4. Распределенная взаимоблокировка	958
Взаимоблокировка при распределении ресурсов	960
Взаимоблокировка при обмене сообщениями	968

19.5. Резюме	972
19.6. Ключевые термины, контрольные вопросы и задачи	972
Ключевые термины	972
Контрольные вопросы	972
Задачи	973
Глава 20. Обзор вероятности и стохастических процессов	975
20.1. Основы теории вероятности	976
Определения вероятности	976
Условная вероятность и независимость	979
Теорема Байеса	980
20.2. Случайные переменные	981
Функции распределения и плотности	982
Важные виды распределений	983
Множество случайных переменных	985
20.3. Элементарные понятия стохастических процессов	987
Статистика первого и второго порядка	988
Стационарные стохастические процессы	989
Спектральная плотность	990
Независимые приращения	991
Эргодичность	996
20.4. Задачи	997
Глава 21. Анализ очередей	1001
21.1. Простой пример поведения очередей	1003
21.2. Цель анализа очередей	1008
21.3. Модели очередей	1011
Одноканальная система массового обслуживания	1011
Многоканальная система массового обслуживания	1015
Основные соотношения из теории массового обслуживания	1016
Предположения	1017
21.4. Одноканальные системы массового обслуживания	1018
21.5. Многоканальные системы массового обслуживания	1022
21.6. Примеры	1022
Сервер базы данных	1022
Вычисление процентилей	1024
Сильно связанный мультипроцессор	1025
Задача построения многоканальной системы массового обслуживания	1027
21.7. Очереди с приоритетами	1029
21.8. Сети очередей	1031
Разделение и объединение потоков трафика	1031
Последовательные очереди	1031
Теорема Джексона	1032
Применение теоремы Джексона в сети с коммутацией пакетов	1033
21.9. Другие модели систем массового обслуживания	1035

21.10. Оценка параметров модели	1035
Выборка	1036
Ошибки выборки	1038
21.11. Задачи	1038
Приложение А. Вопросы параллельности	1043
A.1. Состояния гонки и семафоры	1044
Постановка задачи	1044
Первая попытка	1044
Вторая попытка	1046
Третья попытка	1047
Четвертая попытка	1049
Правильное решение	1050
A.2. Задача о парикмахерской	1052
Неполное решение задачи о парикмахерской	1052
Полное решение задачи о парикмахерской	1055
A.3. Задачи	1057
Приложение Б. Проекты в области программирования и операционных систем	1059
B1. Программный проект 1 — разработка оболочки	1060
Требования к проекту	1061
Представление проекта	1062
Требуемая документация	1063
B2. Программный проект 2 — диспетчер процессов HOST	1063
Диспетчер процессов с четырехуровневым приоритетом	1063
Ограничения ресурсов	1064
Выделение памяти	1065
Процессы	1066
Список диспетчеризации	1067
Требования к проекту	1068
Практические результаты	1069
Представление проекта	1069
Приложение В. Дополнительные вопросы параллельности	1071
V.1. Регистры процессора	1072
Регистры, доступные пользователю	1072
Управляющие регистры и регистры состояния	1073
V.2. Выполнение команд функций ввода-вывода	1074
V.3. Технологии ввода-вывода	1075
Программируемый ввод-вывод	1075
Ввод-вывод, управляемый прерываниями	1076
Прямой доступ к памяти	1077
V.4. Вопросы аппаратной производительности в многоядерных системах	1078
Увеличение степени параллелизма	1078
Энергопотребление	1080

Приложение Г. Объектно-ориентированное проектирование	1083
Г.1. Мотивация	1084
Г.2. Объектно-ориентированные концепции	1085
Структура объектов	1086
Классы объектов	1087
Включение	1090
Г.3. Преимущества объектно-ориентированного подхода	1090
Г.4. CORBA	1091
Г.5. Дополнительные материалы	1095
Приложение Д. Закон Амдала	1097
Приложение Е. Хеш-таблицы	1099
Приложение Ж. Время отклика	1103
Приложение З. Концепции теории массового обслуживания	1107
З.1. Зачем нужна теория массового обслуживания	1107
З.2. Очередь в случае одного сервера	1109
З.3. Многоканальная очередь	1111
З.4. Пуассонова скорость поступления	1113
Приложение И. Сложность алгоритмов	1115
Приложение К. Дисковые устройства хранения	1119
К.1. Магнитные диски	1119
Организация данных и форматирование	1119
Физические характеристики	1122
К.2. Оптическая память	1126
CD-ROM	1126
CD с возможностью записи	1128
CD-R с возможностью перезаписи	1128
DVD	1129
Оптические диски высокой четкости	1129
Приложение Л. Криптографические алгоритмы	1131
Л.1. Симметричное шифрование	1131
DES	1133
AES	1134
Л.2. Шифрование с открытым ключом	1134
Алгоритм Ривеста–Шамира–Адлемана (RSA)	1137
Л.3. Аутентификация сообщений и хеш-функции	1137
Аутентификация с использованием симметричного шифрования	1138
Аутентификация без шифрования	1138
Код аутентификации сообщения	1139
Функция одностороннего хеширования	1140
Л.4. Безопасные хеш-функции	1142

Приложение М. Введение в программирование сокетов	1143
М.1. Сокеты, дескрипторы, порты и соединения	1145
М.2. Модель “клиент/сервер”	1146
Запуск программы с использованием сокетов на компьютере под управлением Windows, не подключенном к сети	1146
Запуск программы с использованием сокетов на компьютере под управлением Windows, подключенном к сети, когда и сервер, и клиент находятся на одной машине	1148
М.3. Работа с сокетами	1148
Создание сокета	1148
Адрес сокета	1148
Привязка к локальному порту	1149
Представление данных и порядок байтов	1150
Подключение сокета	1151
Функция <code>gethostbyname()</code>	1152
Прослушивание входящих соединений	1154
Прием соединения от клиента	1154
Отправка и получение сообщения через сокет	1155
Заккрытие сокета	1156
Сообщение об ошибках	1157
Пример клиентской программы TCP/IP (инициация соединения)	1158
Пример серверной программы TCP/IP (пассивное ожидание соединения)	1159
М.4. Сокеты потоков и дейтаграмм	1161
Пример клиентской программы UDP (инициация соединения)	1162
Пример серверной программы UDP (пассивное ожидание соединения)	1164
М.5. Управление программой времени выполнения	1165
Неблокирующие вызовы сокетов	1165
Асинхронный ввод-вывод (ввод-вывод, управляемый сигналом)	1166
М.6. Удаленное выполнение консольного приложения Windows	1169
Локальный код	1169
Удаленный код	1172
Приложение Н. Международный справочный алфавит	1175
Приложение О. Параллельная система программирования BACI	1179
О.1. Введение	1180
О.2. BACI	1180
Обзор системы	1180
Параллельные конструкции BACI	1181
Как получить BACI	1183
О.3. Примеры программ BACI	1183
О.4. Проекты BACI	1188
Реализация примитивов синхронизации	1188
Семафоры, мониторы и их реализации	1188
О.5. Усовершенствования системы BACI	1190

Приложение П. Управление процедурами	1193
П.1. Реализация стека	1193
П.2. Вызов процедуры и возврат из нее	1194
П.3. Реентерабельные процедуры	1196
Приложение Р. eCos	1199
Р.1. Настраиваемость	1200
Р.2. Компоненты eCos	1202
Уровень аппаратных абстракций	1203
Ядро eCos	1204
Система ввода-вывода	1205
Стандартные библиотеки C	1208
Р.3. Планировщик eCos	1209
Планировщик битовой карты	1209
Планировщик многоуровневой очереди	1209
Р.4. Синхронизация потоков eCos	1211
Мьютексы	1211
Семафоры	1211
Условные переменные	1212
Флаги событий	1215
Почтовые ящики	1215
Циклические блокировки	1216
Глоссарий	1217
Сокращения	1235
Список литературы	1236
Предметный указатель	1251

ВСТРОЕННЫЕ ОПЕРАЦИОННЫЕ СИСТЕМЫ

В ЭТОЙ ГЛАВЕ...

13.1. Встроенные системы

- Концепции встроенных систем
- Прикладные и специализированные процессоры
- Микропроцессоры
- Микроконтроллеры
- Глубоко встроенные системы

13.2. Характеристики встроенных операционных систем

- Исходные и целевые среды
 - Начальный загрузчик
 - Ядро
 - Корневая файловая система
- Подходы к разработке
- Адаптация существующей коммерческой операционной системы
- Специально разработанная встроенная операционная система

13.3. Встроенная система Linux

- Характеристики встроенной системы Linux
 - Размер ядра
 - Объем памяти
 - Прочие характеристики
- Файловые системы встроенного Linux
- Преимущества встроенных систем Linux
- µClinux
 - Сравнение с полномасштабной системой Linux
 - µClibc
- Android

13.4. TinyOS

Беспроводные сети датчиков

Цели TinyOS

Компоненты TinyOS

Планировщик в TinyOS

Пример конфигурации

Интерфейс ресурсов TinyOS

13.5. Ключевые термины, контрольные вопросы и задачи

Ключевые термины

Контрольные вопросы

Задачи

УЧЕБНЫЕ ЦЕЛИ

- Разъяснить понятие встроенной системы.
- Уяснить характеристики встроенных операционных систем.
- Объяснить отличия обычной ОС Linux от встроенной.
- Описать архитектуру и основные функциональные возможности операционной системы TinyOS.

В этой главе рассматриваются встроенные операционные системы — одна из самых важных и широко употребляемых категорий операционных систем. Среда встроенной системы накладывает на операционную систему особые жесткие требования; кроме того, предусматриваются совсем иные стратегии проектирования, чем в случае обычных операционных систем.

Начнем эту главу с краткого обзора понятия встроенных систем, а затем перейдем к рассмотрению принципов действия встроенных операционных систем. И наконец в этой главе будут представлены два самых разных подхода к проектированию встроенных операционных систем: встроенной операционной системы Linux и операционной системы TinyOS. Еще один подход к проектированию встроенных операционных систем рассматривается в приложении P, “eCos”, на примере eCos — еще одной очень важной встроенной ОС.

13.1. ВСТРОЕННЫЕ СИСТЕМЫ

В этом разделе представлено понятие встроенной системы. Для понимания этой концепции требуется также пояснить, чем микропроцессор отличается от микроконтроллера.

Концепции встроенных систем

Термин *встроенная система* означает применение в конкретном продукте аппаратных и программных средств, выполняющих особую функцию или ряд функций, в отличие от универсального (например, переносного или настольного) компьютера. Встроенную систему можно также определить как любое устройство, в состав которого входит компьютерная микросхема, но не являющееся универсальной рабочей станцией, настольным или переносным компьютером. С одной стороны, ежегодно продаются сотни миллионов компьютеров, в том числе переносных и персональных, рабочих станций, серверов, больших и суперЭВМ. С другой стороны, ежегодно производятся десятки миллиардов микроконтроллеров, встраиваемых в более крупные устройства. Ныне большинство устройств, работающих на электрической энергии, содержат встроенную вычислительную систему. И, вероятнее всего, в ближайшем будущем практически все подобные устройства будут иметь в своем составе встроенные вычислительные системы.

Типы устройств со встроенными системами слишком многочисленны, чтобы пытаться их перечислить. К их числу относятся сотовые телефоны, цифровые фото- и видеокамеры, калькуляторы, микроволновые печи, домашние системы безопасности, стиральные и посудомоечные машины, системы освещения, термостаты, печатающие ус-

тройства, различные автомобильные системы (например, управления коробкой передач, регулирования скорости движения, впрыскивания топлива, незаклинивающих тормозов и подвесок), теннисные ракетки, зубные щетки и многочисленные типы датчиков и исполнительных механизмов в автоматизированных системах.

Нередко встроенные системы тесно связаны со своим окружением. А это может привести к ограничениям реального времени, накладываемым в связи с необходимостью взаимодействовать с окружающей средой. И такие ограничения (например, накладываемые скоростью движения, точностью измерения или временными промежутками) требуют синхронизации программных операций. Если же под контролем одновременно должны выполняться многие действия, то это накладывает еще более сложные ограничения — реального времени.

На рис. 13.1 наглядно показана организация встроенных систем с использованием наиболее общей терминологии.



Рис. 13.1. Возможная организация встроенной системы

Типичный настольный компьютер отличается от переносного не только процессором и оперативной памятью, но и, как поясняется ниже, рядом других элементов.

- Для целей измерения, манипулирования или иного взаимодействия с внешним окружением в системе могут присутствовать самые разные интерфейсы. Встроенные системы нередко взаимодействуют (реагируют, манипулируют и сообщаются) с внешним миром через датчики и исполнительные механизмы, а следовательно, они, как правило, являются реагирующими системами. Реагирующая система находится в постоянном взаимодействии со своим окружением, выполняя действия в темпе, который задается данным окружением.
- Интерфейс пользователя может быть как простым (наподобие сигнальных лампочек), так и сложным (типа робототехнического зрения реального времени). Зачастую интерфейс с пользователем может просто отсутствовать.

- Диагностический порт может быть использован для диагностирования контролируемой системы (а не просто для работы компьютера).
- Для повышения производительности или надежности может использоваться интегральная схема специального назначения — программируемая (FPGA) или созданная для конкретного приложения (ASIC) или даже нецифровая аппаратура.
- Программное обеспечение нередко выполняет фиксированную функцию и предназначено для конкретного приложения.
- Первостепенное значение для встроенных систем имеет эффективность. Такие системы оптимизированы по потребляемой энергии, объему кода, времени выполнения, весу, габаритам и стоимости.

Имеется несколько заслуживающих внимания областей применения встроенных систем, сходных с универсальными вычислительными системами, как описывается ниже.

- Несмотря на фиксированную функциональность программного обеспечения, возможность обновления для устранения программных ошибок, повышения безопасности и расширения функциональных возможностей стала очень важным свойством встроенных систем, и не только в бытовой технике.
- Одной относительно недавней разработкой встроенных систем стали платформы, поддерживающие обширный ряд приложений. Характерными тому примерами служат смартфоны и аудиовизуальные устройства вроде “интеллектуальных” телевизоров.

Прикладные и специализированные процессоры

Прикладные процессоры (application processors) определяются как способные выполнять сложные операционные системы, подобные Linux, Android и Chrome. Таким образом, прикладной процессор имеет универсальный характер. Соответствующая встроенная система разработана для поддержки многочисленных приложений и выполнения обширного ряда функций.

В большинстве встроенных систем применяется **специализированный процессор** (dedicated processor), который, как подразумевает его название, специально предназначен для решения одной конкретной задачи или небольшого числа подобных задач по требованию главного устройства. А поскольку такая встроенная система изначально предназначена для решения конкретной задачи или нескольких задач, то процессор и связанные с ним компоненты могут быть спроектированы специальным образом, позволяющим сократить его габариты и стоимость.

Микропроцессоры

Микропроцессор — это процессор, элементы которого сведены в одну или несколько интегральных схем (ИС). Первоначально процессорные ИС включали в себя регистры, арифметико-логическое устройство (АЛУ) и некоторый блок управления или логику обработки команд. По мере увеличения плотности размещения транзисторов на кристалле появилась возможность усложнить архитектуру набора команд, а в конечном счете — добавить больше оперативной памяти и процессоров. Современные микропроцессорные ИС содержат несколько процессоров, иначе называемых *ядрами*, а также значительный

объем сверхоперативной или так называемой *кеш-памяти*. Но, как показано на рис. 13.2, микропроцессорная ИС включает в себя лишь некоторые элементы, образующие вычислительную систему.

Большинство компьютеров, включая встроенные компьютеры в смартфонах и планшетах, а также персональные и переносные компьютеры и рабочие станции, располагаются на материнской плате. Но, прежде чем рассматривать это размещение, необходимо определить ряд терминов. В частности, **печатная плата** (printed circuit board — РСВ) представляет собой жесткую монтажную плоскость для установки и соединения ИС и прочих электронных компонентов. Такая плата состоит из нескольких слоев (как правило, от двух до десяти), соединяющих электронные компоненты с помощью медных дорожек, вытравленных на плате.

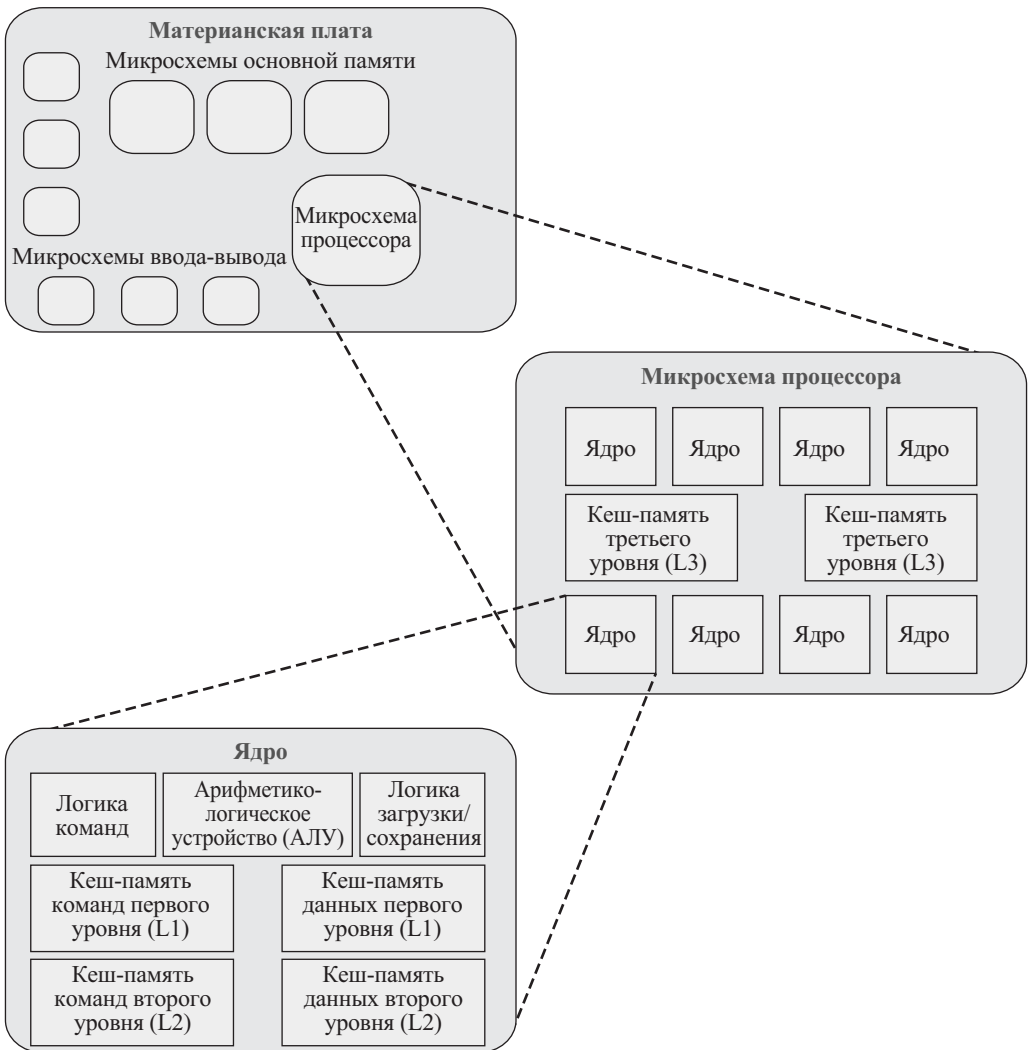


Рис. 13.2. Упрощенное схематическое представление основных элементов многоядерного компьютера

Основная печатная плата в компьютере называется **системной** или **материнской**, тогда как более мелкие печатные платы, вставляемые в разъемы на основной плате, называются платами расширения.

Самыми заметными элементами на системной плате являются микросхемы. **Микросхема** (чип) — это единое полупроводниковое (как правило, кремниевое) изделие с электронными и логическими схемами. Получающееся в конечном итоге изделие называется *интегральной схемой* (ИС).

На системной плате находится разъем или панель для установки процессора, который обычно состоит из нескольких отдельных ядер, и поэтому он называется *многоядерным процессором*. Там же находятся разъемы для установки интегральных схем оперативной памяти, контроллеров ввода-вывода и других базовых компонентов компьютера. В соответствующие разъемы системных плат настольных компьютеров можно установить дополнительные компоненты и платы расширения. Таким образом, современная материнская плата соединяет лишь некоторые отдельные микросхемы, причем каждая из них содержит от нескольких тысяч до сотен миллионов транзисторов.

Микроконтроллеры

Микроконтроллер — это однокристалльная интегральная схема, состоящая из процессора, постоянной памяти для хранения программы (ПЗУ или флеш-памяти), кратковременной памяти (ОЗУ) для ввода-вывода данных, генератора тактовых сигналов и блока управления вводом-выводом. Иногда она называется “однокристалльным компьютером”. Схемы микроконтроллеров используют доступное логическое пространство совершенно иначе. На рис. 13.3 представлена общая терминология для обозначения тех элементов, которые обычно находятся в интегральной схеме микроконтроллера. В частности, процессорная часть микроконтроллера занимает намного меньше места на кристалле, чем у других микропроцессоров, и обладает намного более высокой энергетической эффективностью.

Ежегодно миллиарды микроконтроллерных устройств встраиваются в миллиарды изделий: от игрушек до бытовых приборов и автомобилей. Например, в одном автомобиле может быть использовано 70 и более микроконтроллеров. Как правило, микроконтроллеры (особенно более мелкие и менее дорогие их разновидности) применяются в качестве специализированных процессоров, предназначенных для решения конкретных задач. Например, микроконтроллеры широко применяются в автоматизации производственных процессов. Обеспечивая простую реакцию на ввод, они способны управлять механизмами, включать и выключать вентиляторы, открывать и закрывать клапаны и т.д. Они являются неотъемлемой частью современной промышленной технологии и относятся к числу самых недорогих средств производства механизмов, способных выполнять чрезвычайно сложные функции.

Микроконтроллеры отличаются самыми разными габаритами и производительностью, а процессоры — архитектурами: от 4- до 32-разрядных. Микроконтроллеры обычно работают намного медленнее, чем микропроцессоры: первые, как правило, — в диапазоне тактовых частот порядка мегагерц, тогда как вторые — в диапазоне тактовых частот порядка гигагерц. Для микроконтроллеров также характерно, что они не обеспечивают взаимодействие с человеком. Микроконтроллер, встраиваемый в устройство, программируется для решения конкретной задачи, выполняемой данным устройством, а следовательно, вступает в действие, когда в этом возникает потребность.

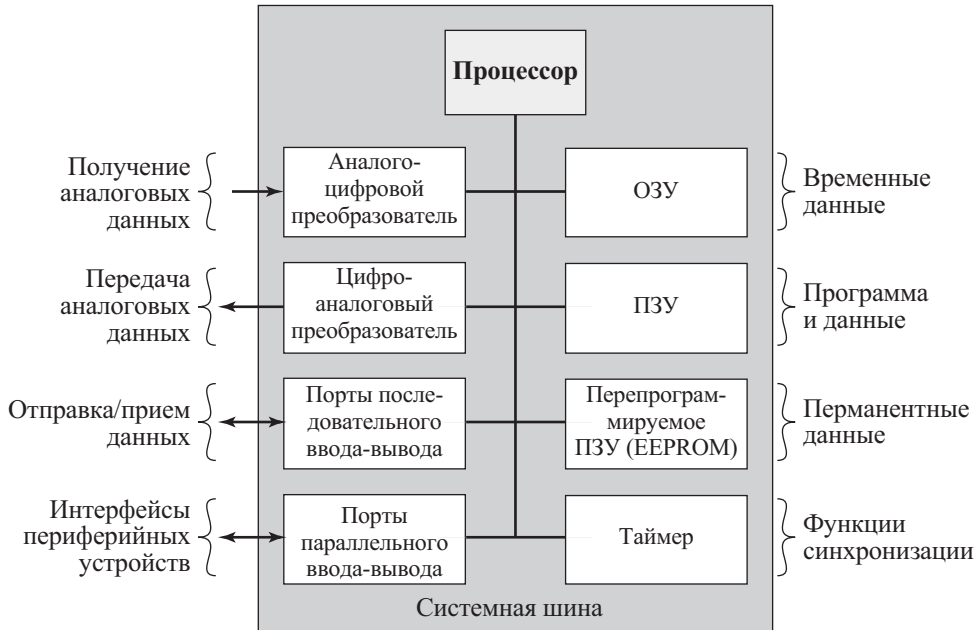


Рис. 13.3. Типичные элементы в ИС микроконтроллера

Глубоко встроенные системы

Большая часть из общего числа встроенных систем называется **глубоко встроенными системами**. И хотя этот термин широко употребляется в технической и коммерческой литературе, искать ясное его определение в Интернете бесполезно (по крайней мере, автору этих строк найти такое определение не удалось). Как правило, можно сказать, что глубоко встроенная система содержит процессор, поведение которого трудно наблюдать как программисту, так и пользователю. В глубоко встроенной системе применяется микроконтроллер, а не микропроцессор, она программируется лишь один раз, когда программная логика работы конкретного устройства “зашивается” в ПЗУ, т.е. в доступную только для чтения память, а кроме того, она никак не взаимодействует с пользователем.

Глубоко встроенные системы являются специализированными устройствами узкого назначения, обнаруживающими нечто в своем окружении, выполняющими обработку сначала на элементарном уровне, а затем и получаемых результатов тем или иным образом. Глубоко встроенные системы нередко обладают возможностями беспроводной связи и присутствуют в таких сетевых конфигурациях, как сети датчиков, разворачиваемые на большой площади (например, на фабрике или сельскохозяйственном поле). В частности, Интернет вещей сильно зависит от глубоко встроенных систем. Как правило, в глубоко встроенных системах накладываются крайне жесткие ограничения на доступные ресурсы, включая оперативную память, габариты процессора и потребляемую мощность.

13.2. ХАРАКТЕРИСТИКИ ВСТРОЕННЫХ ОПЕРАЦИОННЫХ СИСТЕМ

Управлять простой встроенной системой, обладающей несложной функциональностью, можно из одной специализированной программы или ряда программ, не прибегая к услугам другого программного обеспечения. Как правило, более сложные встроенные системы включают операционную систему. И хотя для встроенной системы в принципе можно воспользоваться универсальной операционной системой (например, Linux), ограничения, накладываемые на объем оперативной памяти и потребляемую мощность, а также требования к режиму работы в реальном времени обычно предписывают пользоваться специализированной операционной системой, предназначенной для применения в окружении конкретной встроенной системы.

Ниже перечислен ряд особых характеристик и требований к проектированию встроенных операционных систем.

- **Работа в реальном времени.** Во многих встроенных системах правильность вычисления отчасти зависит от времени получения его результата. Зачастую ограничения реального времени диктуются требованиями к внешнему интерфейсу и устойчивости управления.
- **Реагирующее действие.** Встроенное программное обеспечение может выполняться в ответ на внешние события. Если эти события не наступают периодически или через прогнозируемые промежутки времени, встроенное программное обеспечение, возможно, должно принять во внимание наихудшие условия и задать приоритеты выполнения процедур.
- **Конфигурируемость.** Вследствие большого разнообразия встроенных систем к функциональным возможностям встроенной операционной системы предъявляются самые разные (как количественные, так и качественные) требования. Следовательно, встраиваемая операционная система, предназначенная для применения в разнообразных встроенных системах, должна допускать гибкое конфигурирование, чтобы предоставлять лишь те функциональные возможности, которые требуются для конкретного приложения или комплекта оборудования. Характерные тому примеры приведены в [164]: применение функций компоновки и загрузки для выбора только тех модулей операционной системы, которые требуется загрузить; условная компиляция; определение соответствующих подклассов, если применяется объектно-ориентированная структура. Но при проектировании встроенных систем с большим количеством производных специально настраиваемых операционных систем могут возникнуть трудности, связанные с верификацией. На подобную потенциальную трудность для eCos указывает Такада в [250].
- **Гибкость устройств ввода-вывода.** Практически не существует такого устройства ввода-вывода, которое требовалось бы поддерживать во всех версиях операционных систем, и имеется большое разнообразие таких устройств. Поэтому в [164] предлагается поддержка относительно медленных устройств ввода-вывода (например, жестких дисков и сетевых интерфейсов) с помощью специальных задач вместо того, чтобы интегрировать их драйверы в ядро операционной системы.

- **Рационализованные механизмы защиты.** Как правило, встроенные системы предназначены для выполнения ограниченных, вполне определенных функций. Непротестированные программы редко внедряются в программное обеспечение. Следовательно, после настройки и тестирования программное обеспечение можно рассматривать как надежное. Таким образом, кроме мер безопасности, у встроенных систем могут быть ограниченные механизмы защиты. Например, команды ввода-вывода не обязаны быть привилегированными, чтобы операционная система могла перехватывать управление; задачи могут самостоятельно выполнять свой ввод-вывод. Аналогично могут быть сокращены до минимума механизмы защиты памяти. В [164] приведен следующий тому пример: пусть `switch` соответствует отображаемому в память адресу ввода-вывода значения, которое требуется проверить в ходе операции ввода-вывода. Мы можем позволить программе ввода-вывода выполнить отдельную команду наподобие загрузки этого `switch` в регистр для определения текущего значения. Такой подход предпочтителен, чем использование вызова служб операционной системы, которые приводят к накладным расходам на сохранение и восстановление контекста задачи.
- **Непосредственное применение прерываний.** В универсальных операционных системах любому пользовательскому процессу обычно не разрешается пользоваться прерываниями непосредственно. В [164] перечислены причины, по которым прерывания могут непосредственно запускать и останавливать задачи (например, сохраняя адрес запуска задачи в таблице адресов векторов прерываний) вместо того, чтобы проходить обычные для операционных систем процедуры обслуживания прерываний: 1) встроенные системы можно рассматривать как тщательно проверенные, с весьма редкими модификациями операционной системы или прикладного кода; 2) защита не является обязательной, как пояснялось в предыдущем абзаце; 3) требуется эффективное управление самыми разнообразными устройствами.

Исходные и целевые среды

Главное отличие настольных и серверных дистрибутивов операционной системы Linux от встроенных заключается в том, что настольное и серверное программное обеспечение, как правило, компилируется или конфигурируется на той платформе, на которой оно будет выполняться, тогда как встроенные дистрибутивы Linux обычно компилируются или конфигурируются на одной платформе, называемой исходной (*host*), а предназначаются для выполнения на другой платформе, называемой целевой (*target*, рис. 13.4). Основными компонентами, которые сначала разрабатываются на исходной платформе, а затем переносятся в целевую систему, являются начальный загрузчик, ядро и корневая файловая система.

Начальный загрузчик

Это небольшая программа, загружающая операционную систему в оперативную память после включения электропитания. Она отвечает за процесс первоначальной загрузки системы и загрузку ее ядра в основную память.



Рис. 13.4. Исходная и целевая платформы

Ниже приведена типичная последовательность начальной загрузки во встроенной системе.

1. Процессор во встроенной системе выполняет код, хранящийся в ПЗУ, чтобы загрузить начальный загрузчик первой стадии из внутренней флеш-памяти, SD-карты памяти или порта последовательного ввода-вывода.
2. Начальный загрузчик первой стадии инициализирует контроллер памяти и несколько периферийных устройств, а также загружает начальный загрузчик второй стадии в ОЗУ. Никакое взаимодействие с ним невозможно; обычно этот начальный загрузчик предоставляется поставщиком процессора в ПЗУ.
3. Начальный загрузчик второй стадии загружает ядро и корневую файловую систему из флеш-памяти в основную память (ОЗУ). Ядро и корневая файловая система обычно хранятся во флеш-памяти в виде упакованных файлов, так что процесс начальной загрузки отчасти состоит в распаковке файлов в бинарные образы ядра и корневой файловой системы. Затем начальный загрузчик передает управление ядру. Как правило, на второй стадии данного процесса применяется начальный загрузчик с открытым кодом.

Ядро

Полное ядро включает в себя целый ряд отдельных модулей, выполняющих среди прочего следующие функции.

- Управление памятью
- Управление процессами и потоками выполнения
- Межпроцессное взаимодействие; таймеры
- Ввод-вывод, поддержка сети, звука, хранения данных, графики и прочего с помощью драйверов
- Организация различных файловых систем

- Организация сети
- Управление электропитанием

Из программного обеспечения полного ядра данной операционной системы для встроенной системы опускается целый ряд дополнительных компонентов. Так, если аппаратные средства встроенной системы не поддерживают страничный обмен, то подсистему управления памятью можно удалить. Полное ядро будет включать в себя несколько файловых систем, драйверов устройств и так далее, из которых могут потребоваться лишь некоторые.

Как упоминалось ранее, главное отличие настольных и серверных дистрибутивов Linux от встроенных заключается в том, что настольное и серверное программное обеспечение, как правило, компилируется на той платформе, на которой оно будет выполняться, тогда как встраиваемые дистрибутивы Linux обычно компилируются на одной платформе, а предназначаются для выполнения на другой. Программное обеспечение, предназначенное для этой цели, называется *кросскомпилятором* (*межплатформенным компилятором*). Его применение наглядно показано на рис. 13.5.

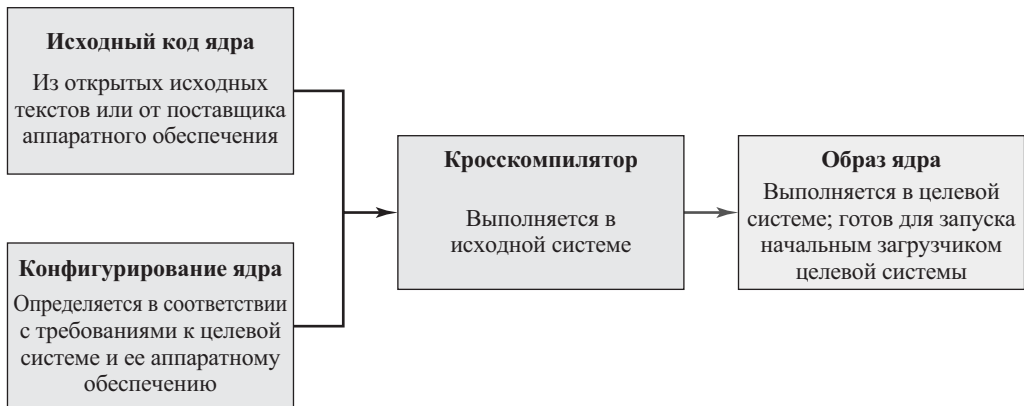


Рис. 13.5. Компиляция ядра

Корневая файловая система

Во встроенной или любой другой операционной системе имеется единая глобальная иерархия каталогов и файлов, предназначенная для представления всех файлов в системе. На вершине такой иерархии (в ее корне) находится корневая файловая система, в которой содержатся все файлы, требующиеся для нормальной работы системы. Корневая файловая система встроенной операционной системы подобна используемой на рабочей станции или сервере, с тем отличием, что она содержит лишь минимальный набор приложений, библиотек и файлов, требующихся для функционирования системы.

Подходы к разработке

Имеются два общих подхода к разработке встроенной операционной системы. Первый подход состоит в том, чтобы взять существующую операционную систему и приспособить ее для применения во встроенном варианте. Другой подход состоит в разработке и реализации операционной системы, предназначенной исключительно для встроенного применения.

Адаптация существующей коммерческой операционной системы

Существующая коммерческая операционная система может быть использована для разработки встроенной системы; для этого необходимо дополнить ее возможностями функционирования в реальном времени, рационализировать ее работу и добавить необходимые функциональные средства. При таком подходе к разработке встроенной системы обычно применяется не только Linux, но и FreeBSD, Windows и другие универсальные операционные системы, хотя зачастую они работают медленнее и менее предсказуемо, чем специализированные встроенные операционные системы. Преимущество такого подхода заключается в том, что встроенная операционная система, производная от коммерческой универсальной операционной системы, основывается на ряде знакомых интерфейсов, что упрощает ее переносимость.

Недостаток же применения универсальной операционной системы для разработки встроенной заключается в том, что она не оптимизирована для применения в реальном времени и во встроенном варианте. Следовательно, для достижения необходимой производительности может потребоваться значительная ее модификация. В частности, типичная операционная система оптимизируется для среднего, а не наихудшего случая планирования заданий, обычно назначая ресурсы по требованию и пренебрегая большинством, если не всеми семантическими сведениями о приложении.

Специально разработанная встроенная операционная система

Значительное количество операционных систем разработаны с самого начала для применения во встроенном варианте. Двумя характерными примерами такого подхода к разработке встроенных систем являются операционные системы eCos и TinyOS, обсуждаемые далее в этой главе.

Ниже перечислены типичные характеристики специализированной встроенной операционной системы.

- Наличие переключателя быстрых и упрощенных процессов или потоков исполнения.
- Стратегия планирования, реализуемая в виде модуля диспетчеризации в реальном времени как часть планировщика, а не отдельного компонента.
- Малые размеры.
- Быстрое реагирование на внешние прерывания. Типичным требованием к встроенной операционной системе является время отклика менее 10 мс.
- Минимизация промежутков времени, в течение которых запрещены прерывания.
- Предоставление для управления памятью разделов фиксированного или переменного размера, а также возможности блокировать код и данные в памяти.
- Предоставление специальных последовательных файлов, в которых можно накапливать данные с большой скоростью.

Чтобы каким-то образом удовлетворять временным ограничениям, ядро

- предоставляет ограниченное время для выполнения большинства примитивов;
- поддерживает часы реального времени;
- выдает специальные предупреждающие сигналы и блокировки по времени;
- поддерживает правила организации очередей (например, первоочередное обслуживание запросов с самым ранним сроком выполнения), а также примитивов для сжатия сообщения в начале очереди;
- предоставляет примитивы для задержки обработки на фиксированное время, а также для приостановки и возобновления процесса выполнения.

Перечисленные выше характеристики имеют немало общего с требованиями к функционированию встроенных операционных систем в реальном времени. Тем не менее в требованиях к сложным встроенным системам основной акцент может быть сделан на предсказуемое, а не на быстрое действие, а для этого придется принимать совсем другие проектные решения, особенно в области планирования заданий.

13.3. ВСТРОЕННАЯ СИСТЕМА LINUX

Термин *встроенная система Linux* просто означает версию Linux, выполняемую во встроенной системе. Как правило, во встроенной системе Linux применяется один из официальных выпусков ядра, несмотря на то что в некоторых системах используется модифицированное ядро, приспособленное к конкретной конфигурации оборудования или для поддержки определенного класса приложений. Прежде всего, ядро встроенной системы Linux отличается от ядра обычной операционной системы Linux, установленной на рабочей станции или сервере, конфигурацией сборки и средой разработки.

В этом разделе сначала выделяются некоторые из основных отличий встроенной Linux от ее версии, работающей на настольном компьютере или сервере, а затем описывается распространенное программное обеспечение под названием “ μ Clinux”.

Характеристики встроенной системы Linux

Размер ядра

Настольные и серверные системы Linux должны поддерживать большое число устройств, поскольку в Linux применяются самые разные конфигурации. Такие системы должны поддерживать и целый ряд протоколов передачи и обмена данными, а следовательно, их можно применять для самых разных целей. Встроенным устройствам, как правило, требуется поддержка определенного ряда конкретных устройств, периферийных устройств и протоколов в зависимости от установленного оборудования и предназначения. К счастью, ядро Linux в высшей степени поддается конфигурированию в плане архитектуры, для которой оно скомпилировано, а также поддерживаемых в нем процессоров и устройств.

Дистрибутив встроенной системы Linux является версией Linux, специально приспособленной под ограничения, накладываемые на размер и оборудование встроенных устройств. В его состав входят программные пакеты, поддерживающие различные службы и приложения, действующие на подобных устройствах. Таким образом, ядро встроенной системы Linux оказывается намного меньше, чем ядро обычной системы Linux.

Объем памяти

Размер встроенной системы Linux классифицируется в [76] по объему доступного ПЗУ или ОЗУ в трех обширных категориях малых, средних и крупных систем. В частности, малые системы характеризуются наличием маломощного процессора, ПЗУ объемом минимум 2 Мбайт и ОЗУ объемом 4 Мбайт. Средние по размеру системы отличаются наличием среднетощного процессора, ПЗУ объемом около 32 Мбайт и ОЗУ объемом 64 Мбайт. А крупные системы характеризуются наличием мощного процессора или даже нескольких процессоров и большими объемами как оперативной, так и постоянной памяти.

В системе без постоянной памяти все ядро Linux должно вмещаться в ОЗУ и ПЗУ, чего нельзя сказать о полномасштабной современной системе Linux. В качестве иллюстрации данного положения на рис. 13.6 показано увеличение размера ядра полномасштабной системы Linux со временем. Безусловно, любую систему Linux можно сконфигурировать лишь с некоторыми компонентами из полного выпуска. Тем не менее на диаграмме, приведенной на рис. 13.6, наглядно показано, что, в особенности для мелких и средних по размеру встроенных систем, могут быть опущены значительные объемы ядра.

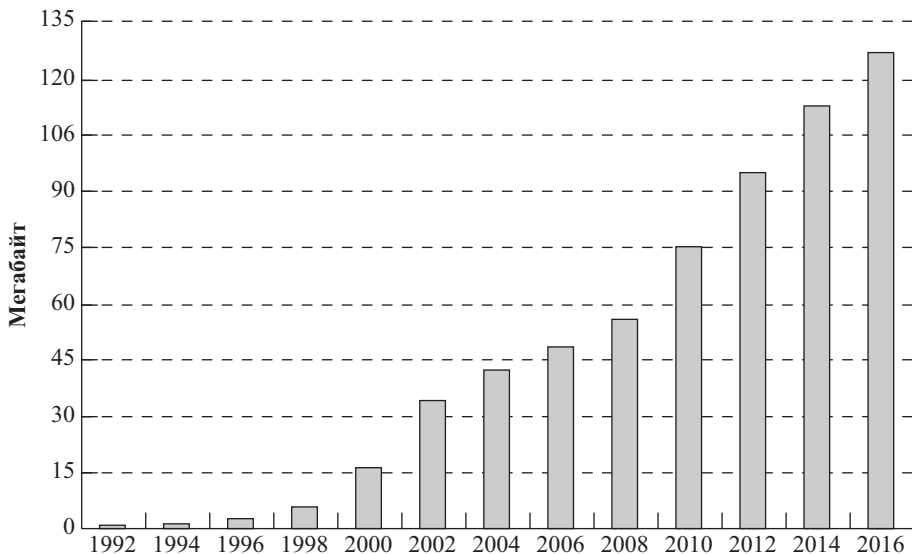


Рис. 13.6. Размер ядра Linux, показанный в сжатом формате файлов GZIP

Прочие характеристики

К другим характеристикам встроенных систем Linux относятся следующие.

- **Временные ограничения.** Жесткие временные ограничения требуют от системы реакции на внешние воздействия в течение указанного промежутка времени. А мягкие временные ограничения пригодны для тех систем, в которых медленная реакция системы является не критичной.

- **Подключаемость к сети.** Означает способность системы работать в сети. Буквально все современные встроенные устройства обладают такой способностью, подключаясь, как правило, к беспроводной сети.
- **Степень взаимодействия с пользователем.** Одни устройства сосредоточены на взаимодействии с пользователем, тогда как другие, в том числе управляющие промышленными процессами, могут предоставлять для взаимодействия с пользователем очень простой интерфейс (например, через светодиоды и кнопки). Имеются и такие устройства, у которых взаимодействие с конечным пользователем отсутствует (например, датчики, действующие в Интернете вещей, собирающие данные и передающие их в облако).

В табл. 13.1, взятой из [76], приведены характеристики некоторых коммерчески доступных встроенных систем, в которых применяется ядро Linux.

Таблица 13.1. Характеристики примеров встроенных систем Linux

Описание	Тип	Размер	Временные ограничения	Подключаемость к сети	Степень взаимодействия с пользователем
Устройства управления акселератором	Управление промышленными процессами	Средний	Жесткие	Да	Низкая
Автоматизированная обучающая система	Авиакосмическая отрасль	Крупный	Жесткие	Нет	Высокая
Устройство типа Bluetooth для доступа к местным данным	Организация сети	Мелкий	Мягкие	Да	Очень низкая
Преобразователь протоколов для управления системой сбора данных	Управление промышленными процессами	Средний	Жесткие	Нет	Очень низкая
Карманный персональный компьютер	Бытовая электроника	Средний	Мягкие	Да	Очень высокая
Устройство управления двигателем, применяемое в системе управления космическим аппаратом	Авиакосмическая отрасль	Крупный	Жесткие	Да	Высокая

Файловые системы встроенного Linux

В некоторых приложениях могут создаваться относительно небольшие файловые системы, которые предназначены для применения только во время работы самого приложения и могут храниться в основной памяти. Но в целом файловая система должна храниться в долговременной памяти (например, во флеш-памяти или на традиционных

дисковых запоминающих устройствах). Впрочем, для большинства встроенных систем внутренний или внешний диск не подходит, и поэтому в качестве долговременной памяти обычно применяется флеш-память.

Что же касается остальных свойств встроенной системы Linux, то ее файловая система должна быть как можно более компактной. Для применения во встроенных системах был разработан целый ряд компактных файловых систем. В качестве примера ниже перечислены некоторые из наиболее употребительных систем данной категории.

- **cramfs.** Упакованная файловая система в ОЗУ (Compressed RAM File System), простая и доступная только для чтения. Она служит для того, чтобы минимизировать размер, максимально эффективно используя емкость исходного запоминающего устройства. В файловых системах cramfs отдельные файлы упаковываются в блоки, совпадающие по размеру со страницами в Linux (как правило, 4096 байт или 4 Мбайт в зависимости от версии ядра и конфигурации), чтобы обеспечить эффективный произвольный доступ к содержимому файлов.
- **squashfs.** Как и cramfs, squashfs является упакованной доступной только для чтения файловой системой, предназначенной для применения в средах с малой или ограниченной емкостью запоминающего устройства (например, во встроенных системах Linux).
- **jffs2.** Вторая версия файловой системы для флеш-памяти с журналированием (Journaling Flash File System). Она предназначена для применения в устройствах флеш-памяти типа NOR и NAND с особым акцентом на такие вопросы эксплуатации флеш-памяти, как выравнивание износа.
- **ubifs.** Файловая система с несортированными образами блоков (Unsorted Block Image File System), которая обычно обеспечивает лучшую производительность, чем файловая система jffs2 в крупных устройствах флеш-памяти. Для дополнительного повышения производительности в ней также поддерживается кеширование записи.
- **yaffs2.** Еще одна версия файловой системы (Yet another Flash File System) для флеш-памяти, обеспечивающая быстрое и надежное хранение файлов во флеш-памяти. Для хранения данных состояния файловой системы yaffs2 требуется меньше места в ОЗУ, чем в таких файловых системах, как jffs2. И, как правило, она обеспечивает лучшую производительность, если запись данных в файловой системе выполняется слишком часто.

Преимущества встроенных систем Linux

Встроенные версии Linux стали появляться еще в 1999 году. В целом ряде компаний были разработаны свои версии, предназначенные для конкретных платформ. Ниже перечислены преимущества применения Linux в качестве основания для разработки встроенной операционной системы.

- **Независимость поставщиков.** Поставщик платформы не зависит от поставщика конкретной встроенной системы, чтобы предоставить необходимые функциональные средства и уложиться в крайние сроки развертывания.

- **Разнообразная аппаратная поддержка.** В системе Linux поддерживается обширный ряд архитектур процессоров и периферийных устройств, и благодаря этому она оказывается вполне пригодной для разработки буквально каждой встроенной системы.
- **Малые затраты.** Применение Linux позволяет свести к минимуму затраты на разработку и обучение.
- **Открытость исходного кода.** Применение Linux дает все преимущества программного обеспечения с открытым исходным кодом.

µClinux

µClinux (т.е. микроконтроллерная Linux) является весьма распространенной разновидностью ядра Linux с открытым исходным кодом, предназначенного для микроконтроллеров и прочих очень малых встроенных систем. В силу модульного характера Linux совсем не трудно сократить операционную среду, исключив из нее служебные программы, инструментальные средства и прочие системные службы, которые не нужны во встроенной среде. В этом, собственно, состоит основной принцип проектирования µClinux.

Чтобы дать ясное представление о размере загружаемого образа µClinux (ядра и корневой файловой системы), обратимся к опыту компании EmCraft Systems, разрабатывающей системы на уровне печатных плат с помощью микроконтроллеров Cortex-M и микропроцессоров Cortex-A [74]. Это едва ли не самые малые встроенные системы, в которых применяется µClinux. Так, минимальная конфигурация µClinux может занимать не больше 0,5 Мбайт, хотя поставщик посчитал вполне практичным размер загружаемого образа, включая Ethernet, TCP/IP, приемлемый набор инструментальных средств из пользовательского пространства и сконфигурированных приложений, в пределах от 1,5 до 2 Мбайт. А объем памяти, требующийся для µClinux во время выполнения, должен находиться в пределах от 8 до 32 Мбайт. Эти числовые показатели значительно меньше, чем у типичной системы Linux.

Сравнение с полномасштабной системой Linux

Главные отличия µClinux от Linux для крупных систем (подробнее об этом см. в [165]) заключаются в следующем.

- Linux является многопользовательской операционной системой, основанной на UNIX, а µClinux — версией Linux, предназначенной для разработки встроенных систем (как правило, без взаимодействия с пользователем).
- В отличие от Linux, в µClinux не поддерживается управление памятью. Следовательно, виртуальные адресные пространства в µClinux не требуются, а приложения должны быть привязаны к абсолютным адресам.
- В ядре Linux поддерживается отдельное пространство виртуальных адресов для каждого процесса, а в µClinux — единое адресное пространство, общее для всех процессов.
- Адресное пространство в Linux восстанавливается при переключении контекста, чего нельзя сказать о µClinux.

- В отличие от Linux, в μ Clinux не обеспечивается системный вызов `fork()`, а вместо этого применяется функция `vfork()`. По существу, системный вызов с помощью функции `fork()` приводит к созданию дубликата, во многом похожего на вызывающий процесс (однако при этом копируется не все (например ресурсы в некоторых реализациях ограничены), хотя основной замысел состоит в том, чтобы создать как можно более полную копию). Новый (порожденный) процесс получает другой идентификатор процесса и имеет тот же идентификатор родительского процесса. Главное отличие функции `vfork()` от функции `fork()` заключается в следующем: когда новый процесс создается с помощью функции `vfork()`, родительский процесс временно приостанавливается, а порожденный процесс может позаимствовать адресное пространство своего родителя. И так продолжается до тех пор, пока завершится порожденный процесс или же им будет вызвана функция `execve()`, после чего родительский процесс продолжится.
- μ Clinux модифицирует драйверы устройств для использования локальной системной шины вместо ISA или PCI.

Самое значительное отличие полномасштабной системы Linux от μ Clinux относится к области управления памятью. Отсутствие поддержки управления памятью в μ Clinux может иметь целый ряд последствий, в том числе следующие.

- Основная память, выделяемая для процесса, в общем случае должна быть непрерывной. Если же выполняется свопинг целого ряда процессов, это может привести к фрагментации памяти (см. рис. 7.4). Тем не менее во встроенных системах, как правило, имеется фиксированное множество процессов, загружаемых во время начальной загрузки и работающих до следующего сброса. Следовательно, такая функциональная возможность обычно не требуется.
- μ Clinux не может расширить память для работающих процессов, поскольку с ней могут быть смежными другие процессы. Следовательно, вызовы функций `brk()` и `sbrk()`, динамически изменяющих объем используемого пространства, выделяемого для сегмента данных из вызывающего процесса, недоступны. Но в μ Clinux все же предоставлена реализация функции `malloc()`, предназначенная для выделения блока памяти из глобального пула памяти.
- В μ Clinux отсутствует стек динамических приложений. Это может привести к переполнению стека, а значит, и к повреждению памяти. Во избежание этого на стадии разработки и конфигурирования приложений необходимо принять соответствующие меры.
- В μ Clinux не обеспечивается защита памяти, что представляет риск повреждения одним приложением части другого приложения или даже ядра. Впрочем, этот недостаток все же исправлен в некоторых реализациях. Например, архитектура Cortex-M3/M4 предоставляет механизм защиты памяти под названием “MPU” (Memory Protection Unit — блок защиты памяти). Используя механизм MPU, компания Emcraft Systems внедрила в ядро дополнительное функциональное средство, реализующее взаимную защиту как самих процессов, так и ядра и процессов, сопоставимую с механизмами защиты памяти, реализованными в Linux с помощью блока управления памятью (MMU) [130].

µClibc

Это системная библиотека на C, первоначально разработанная для поддержки µClinux и обычно применяемая вместе с µClinux, хотя библиотекой µClibc можно пользоваться и вместе с другими ядрами Linux. Основное назначение µClibc — представить системную библиотеку, написанную на языке C и пригодную для разработки встроенных систем Linux. Она намного меньше библиотеки GNU C (glibc), широко применяемой в системах Linux, хотя практически все приложения, поддерживаемые в библиотеке glibc, идеально взаимодействуют и с µClibc. Перенос приложений из библиотеки glibc в библиотеку µClibc, как правило, предусматривает лишь перекомпиляцию исходного кода. В библиотеке µClibc поддерживаются также совместно используемые библиотеки и поточная обработка.

В табл. 13.2, составленной по материалам, взятым из [4], сравниваются размеры функций в обеих упомянутых выше библиотеках. Как видите, экономия занимаемого места вполне очевидна. Такая экономия достигается благодаря запрету некоторых функциональных средств по умолчанию и радикальной переделке исходного кода с целью исключить его избыточность. А на рис. 13.7 показана высокоуровневая программная архитектура встроенной системы, построенной с помощью µClinux и µClibc.

ТАБЛИЦА 13.2. РАЗМЕРЫ НЕКОТОРЫХ ФУНКЦИЙ ИЗ БИБЛИОТЕК µCLIBC И GLIBC

Имя функции из glibc	Размер функции glibc, Кбайт	Имя функции из µClibc	Размер функции µClibc, Кбайт
libc-2.3.2.so	1200	libuClibc-0.9.2.7.so	284
ld-2.3.2.so	92	libcrypt-0.9.2.7.so	20
libcrypt-2.3.2.so	20	libdl-0.9.2.7.so	12
libdl-2.3.2.so	12	libm-0.9.2.7.so	8
libm-2.3.2.so	136	libnsl-0.9.2.7so	56
libnsl-2.3.2.so	76	libpthread-0.9.2.7.so	4
libpthread-2.3.2.so	84	libresolv-0.9.2.7.so	84
libresolv-2.3.2.so	68	libutil-0.9.2.7.so	4
libutil-2.3.2.so	8	libcrypt-0.9.2.7.so	8

Android

Как подробно обсуждалось ранее в этой книге, Android является встроенной операционной системой, основанной на ядре Linux. Следовательно, Android можно с полным основанием рассматривать как пример встроенной системы Linux. Тем не менее многие разработчики встроенных систем Linux не считают Android примером системы данной категории [47]. С их точки зрения, у классического встроенного устройства имеется фиксированный набор функций, прошитых изготовителем. Android же в большей степени относится к категории платформенных операционных систем, поддерживающих разнообразные приложения, различающиеся на разных платформах. Кроме того, Android является вертикально интегрированной системой, включая модификации ядра Linux, специально сделанные для Android.

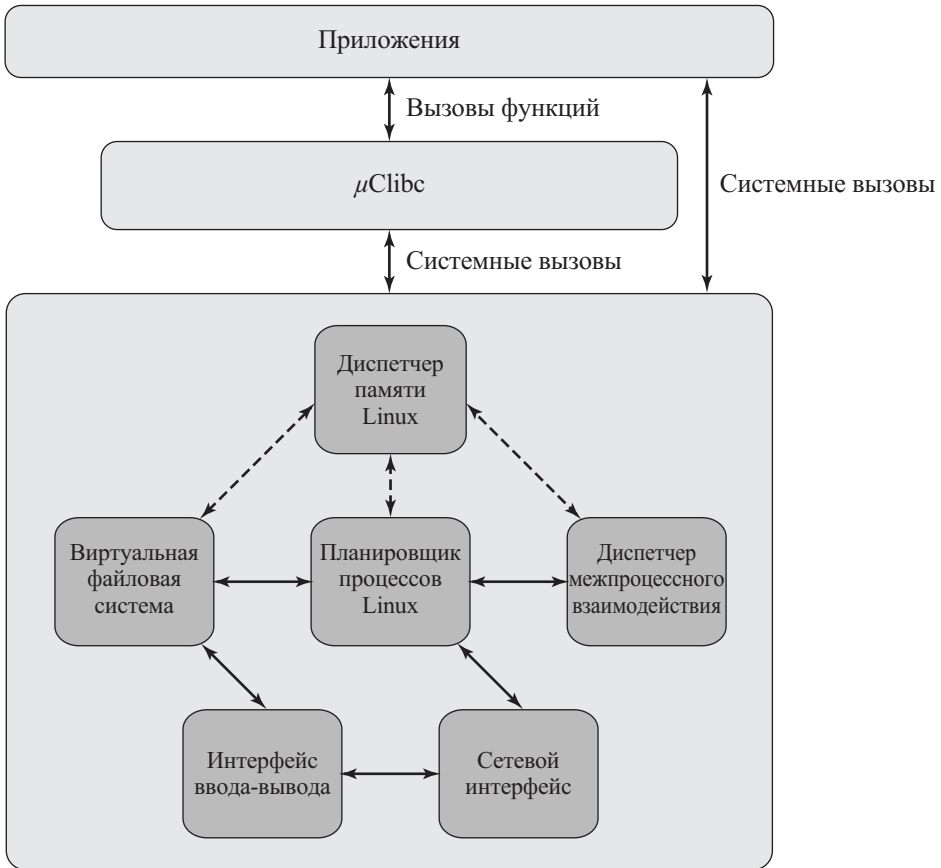


Рис. 13.7. Программная архитектура встроенной системы, построенной на основе *μClinux* и *μClibc*

Основное внимание в Android уделяется вертикальной интеграции ядра Linux и компонентов из пользовательского пространства Android. В конечном счете это вопрос семантики, а “официальное” определение Android как встроенной системы Linux, на которое можно было бы положиться, отсутствует.

13.4. TINYOS

В системе TinyOS представлен более рациональный подход к разработке встроенных операционных систем, чем основанный на коммерческих универсальных операционных системах (например, встроенной версии Linux). Следовательно, TinyOS и аналогичные ей системы в большей степени пригодны для разработки небольших встроенных систем с жесткими требованиями к памяти, времени обработки и реагирования, потребляемой мощности и т.д. Процесс рационализации заходит в TinyOS довольно далеко, а в результате получается самая маленькая операционная система для встроенных систем. Базовой операционной системе требуется всего 400 байт памяти для кода и данных, вместе взятых.