2. Структура и основные особенности операционной системы OpenVMS

Содержание

2.1. Введение

OpenVMS - это структурированная операционная система, обеспечивающая многоуровневые средства планирования и контроля, управления памятью и ввода-вывода. Для взаимодействия с системой на высшем уровне используется командный язык, поставляемый Digital или определяемый пользователем. На уровне прикладного программирования доступ к средствам OpenVMS осуществляется через системные вызовы и библиотеки процедур. Для всех средств системы существует расширенный On-line Help.

OpenVMS обладает рядом возможностбей для выполнения задач реального времени и средствами управления операционной средой и системными ресурсами. Многие из этих средств недоступны обычным пользователям, работающим с разделением времени, но легко могут быть доступны привилегированным пользователям, реализующим под OpenVMS прикладные программы реального времени.

В дополнение к средствам реального времени OpenVMS обеспечивает богатую инструментальную среду, позволяющую реализовывать проект с минимальными затратами в кратчайшее время. Сетевые возможности OpenVMS позволяют совместно использовать информацию и распределят рабочую нагрузку в внутри организации, оптимизируя использование вычислительных ресурсов.

Многозадачные средства

Под управлением OpenVMS процессы реального времени планируются на основе назначаемых пользователем приоритетов. На назначенные приоритеты реального времени (16-31) не влияет ни планирование квантов времени, ни автоматическое назначение приоритетов, как при разделении времени. С разрешения пользователя процессы реального времени могут с помощью различных системных средств синхронизироваться от системного таймера.

Средства управления памятью

OpenVMS пользуется всеми преимуществами виртуальной адресации в архитектуре VAX. Каждому процессу доступен 1 Гб виртуального адресного пространства. Преобразование виртуального адреса в физический полностью прозрачно для прикладного программиста.

Важнейшие части кода или данных могут быть сделаны резидентными в физической памяти, чтобы гарантировать их доступность и уменьшить затраты связанные с листанием. Привилегированный пользователь полностью контролирует ресурсы физической памяти.

OpenVMS обеспечивает средства для динамического создания, распределения и удаления разделяемых глобальных секций - областей физической памяти, содержащихся в виртуальном адресном пространстве нескольких процессов. Это позволяет нескольким процессам совместно использовать физическую память для общих пулов данных и обмена сообщениями.

Инструментальная среда OpenVMS

OpenVMS поддерживает обширную инструментальную среду, включающую языковые компиляторы, компоновщик, редакторы, ориентированные на разные языки, и средства символьной отладки. Дополнительные средства OpenVMS и сторонних поставщиков обеспечивают управление кодом/конфигурацией и автоматический контроль параметров.

Средства ввода-вывода OpenVMS

Программа пользователя может взаимодействовать с подсистемой ввода-вывода OpenVMS на любом из нескольких уровней. Поддержка языков высокого уровня обеспечивается с помощью VAX Common Runtime Procedures Library. Программы могут также выполнять операции ввода-вывода, используя VAX Record Management System (RMS) или Queued I/O (QIO). Эти средства могут использоваться как для синхронного, так и для асинхронного ввода-вывода.

Для доступа в реальном времени дисковый ввод-вывод может быть оптимизирован за счет создания файлов в логически непрерывных блоках или в определенных участках дискового тома, что уменьшает время доступа к данным.

Симметричная мультиобработка

Для многопроцессорных систем OpenVMS VAX обеспечивает поддержку симметричной мультиобработки (Symmetric Multiprocessing - SMP). Конфигурация OpenVMS SMP состоит из нескольких центральных процессоров, выполняющих код из единого адресного пространства. Все процессоры используют одну копию операционной системы OpenVMS.

Совместное использование структур данных несколькими процессорами осуществляется за счет взаимной блокировки в системном виртуальном адресном пространстве. Любой процессор, захвативший данную блокировку, может исполнять базовый вариант кода и модифицировать любые структуры данных связанные с данной блокировкой.

Алгоритмы планирования SMP обеспечивают процессам реального времени заданный приоритет доступа ко всем процессорам системы. То есть, n процессов реального времени с высшим приоритетом могут приоритетно планироваться в n-процессорной системе. Если процессы реального времени отсутствуют, планируется только один процесс (не реального времени) с высшим приоритетом.

Многие утилиты и структурированные продукты OpenVMS имеют стандартные расширения для поддержки SMP. Прозрачная поддержка SMP обеспечивается во многих средствах VAXset, включая языковые компиляторы и MP Debugger. Для параллельной реализации программ в системе OpenVMS используется OpenVMS Parallel Processing Run-Time Library (PPL$).

2.2. Основные части системы OpenVMS

2.2.1. Планировщик и контроль процессов

Назначение

- Передает процессор вычислительному процессу с наивысшим приоритетом.
- Обслуживает переходные состояния процессов.
- Облегчает синхронизацию выполнения процессов.
- Выполняет проверки и действия через определенные промежутки времени.

Программный код и данные

- Программа обслуживания прерываний Планировщика.
- Программный код системы сообщения о событиях.
- Набор программ обслуживания прерываний аппаратных часов и программного таймера.
- Системное обслуживание ($WAKE)

2.2.2. Обслуживание памяти

Назначение

- Перевод мнимых (виртуальных) адресов в адреса физической памяти.
- Распределение и возможность разделения физической памяти между процессами.
- Защита информации процессов от несанкционированного доступа.

Программный код и данные

- Программа обслуживания исключения потери страниц и процесс обмена (SWAPPER PROCESS)
- База данных PFN[1], таблицы страниц памяти.
- Системное обслуживание ($CRETVA)

2.2.3. Подсистема Ввода/Вывода

Назначение

- Чтение/Запись с устройств по программным запросам
- Обслуживание прерываний от устройств.
- Журнал ошибок устройств.

Программный код и данные

- Драйверы устройств и программы не зависящие от типа устройства.
- Структуры данных ввода/вывода.
- Системное обслужиывание ($QIO)

[1] PFN - Таблица страниц памяти (Page Frame Number)

Пример 2-1. Команда SHOW SYSTEM

```
  Pid     Process Name    State  Pri  I/O       CPU         Page flts  Ph.Mem
00000081  SWAPPER          HIB     16    0   0 00:00:00.53       0        0
00000085  IPCACP           HIB     10    7   0 00:00:00.02      97       78
00000086  ERRFMT           HIB      8 1859   0 00:00:00.59     159      193
00000087  OPCOM            HIB      8  312   0 00:00:00.27     369      176
00000088  AUDIT_SERVER     HIB     10   70   0 00:00:00.19     546      590
00000089  JOB_CONTROL      HIB      8   85   0 00:00:00.10     229      271
0000008A  QUEUE_MANAGER    HIB      8   47   0 00:00:00.16     593      901
0000008B  SECURITY_SERVER  HIB     10   54   0 00:00:00.40    2346      955
0000008C  TP_SERVER        HIB     10   19   0 00:00:00.06     340      143
0000008D  LATACP           HIB     14    9   0 00:00:00.04     355      131
0000008E  SYMBIONT_1       HIB      4   21   0 00:00:00.09     340      276
0000008F  NETACP           HIB      9 2649   0 00:00:01.29     718      260
00000090  EVL              HIB      6  138   0 00:00:00.11     374      130   N
00000091  REMACP           HIB      9   17   0 00:00:00.04      91       67
00000092  RDMS_MONITOR     LEF     15  171   0 00:00:00.55    9074       73
00000093  EPC$REGISTRAR    HIB     10  585   0 00:00:02.14    5848      392
00000094  RPC$SWL          HIB      9   22   0 00:00:00.02     139      215
00000095  ACMS_SWL         HIB      9   23   0 00:00:00.05     147      425
00000097  DECW$SERVER_0    HIB      7  473   0 00:00:38.71    9403     2990
00000098  DECW$SESSION     LEF      6 1317   0 00:00:18.71   22777     5150
0000009D  DECW$MWM         LEF      4  164   0 00:00:16.51    3978      982   S
0000009F  VUE$RYBAKOV_1    LEF      6  200   0 00:00:01.41    4609      316   S
000000A2  VUE$RYBAKOV_4    LEF      4  348   0 00:00:05.48   11434      561   S
000000AF  VUE$RYBAKOV_6    LEF      5  493   0 00:00:08.64    8505      416
000000B1  VUE$RYBAKOV_2    LEF      9  596   0 00:00:00.57    1163      384   S
000000C1  GRIBKOV          CUR      4   88   0 00:00:00.25     661      448

OpenVMS VAX V6.1 on node SHARK 01-DEC-1995 00:53:14.09 Uptime 2 10:15:08
```

Пример 2-1 - замечания.

- Команда показывает список процессов, работающих в системе.
- Образы, работающие как процессы.
- Только "верхний уровень".
- Отметим отсутствие:
  - Программ - планировщиков.
  - Программ управления вводом/выводом.
  - Программ системных обслуживаний.

2.3. Процессы

Обзор

Процесс - это среда, в которой выполняется образ задачи. Вы можете создавать процессы и управлять ими для выполнения ниже перечисленных программных задач:

- Разработка приложения для выполнения одной функции одним процессом.
- Процессы предназначенные для выполнения команд DCL.
- Параллельное выполнение несколько процессов, когда один процесс выполняет одну часть программы одновременно с другим, который выполняет другую часть программы.
- Выполнение прикладных программы, где один процесс контролирует и координирует работу других процессов.
- Планировка работы программ.
- Разделение (изоляция) кода по следующим соображениям:
  - Устранение логических ошибок в программе.
  - Выполнение привилегированного кода.
  - Исполнение чувствительного кода.

2.3.1. Создание процессов

Обзор

Создаваемые процессы могут быть двух видов - *порожденные*(spawned) и *отсоединенные* (detached).

Типы процессов

Порожденные подпроцессы зависят от создавшего их процесса (родительский процесс) и удаляются, если родительский процесс заканчивает работу. Отсоединенные процессы не зависят процесса, создавшего их. Если вы хотите создать процесс, продолжающий работать после выхода из родительского процесса, используйте отсоединенный процесс. Вы можете так же использовать отсоединенный процесс для вывода на терминал другого процесса (используя системный сервис SYS$BREAKTHRU)

Режимы выполнения

Процессы выполняются в одном из следующих режимов:

- *Интерактивный* (Interactive) - принимает ввод с устройств ориентированных на работу с записями (record oriented), например терминал или почтовый ящик.
- *Пакетный* (Batch) - создается контроллером заданий (job controller) и не интерактивный.
- *Сетевой* (Network) - создается сетевой программой контроля доступа (network ACP).
- *Прочие* - не подходящие под вышеперечисленные определения (к примеру порожденный подпроцесс, который получает ввод из командной процедуры).

Таблица 2-1 содержит обзор характеристик отсоединенных процессов и подпроцессов.

Таблица 2-1. Отсоединенные процессы и порожденные подпроцессы

Характеристика	*Подпроцесс*	*Отсоединенный процесс*
Привилегии (Privileges).	От создающего процесса.	Задаются создающим процессом.
Квоты и лимиты (Quotas and limits).	Делит с создающим процессом.	Задаются создающим процессом, но не разделяются с ним.
Файл авторизации пользователей (UAF[2]).	Используется для получения информации не переданной создающим процессом.	Большинство параметров не передающихся создающим процессом.
Код идентификации пользователя (UIC[3]).	Код родительского процесса.	Задается создающим процессом.
Ограничения (Restrictions).	Существует, пока работает родительский процесс.	Нет.
Как создается.	С используйте системные функции SYS$CREPRC или LIB$SPAWN из другого процесса.	При помощи функции LIB$SPAWN.
Когда уничтожается.	Когда закончит работу или когда закончит работу родительский процесс.	Когда закончит работу.
Присутствие интерпретатора команд.	Обычно нет.	Обычно нет.

2.3.2. Управление процессами

Обзор

Управление процессами включает в себя следующие задачи:

- Получение информации о процессе.
- Установка привилегий для процесса.
- Установка приоритета процесса.
- Приостановка выполнения процесса.
- Установка имени процесса.
- Удаление процесса.
- Синхронизация выполнения процесса.

Для решения этих задач Вы можете использовать функции системы или команды DCL. Таблица 2-2 показывает использование функций и команд DCL. Вы можете использовать команды DCL в файлах командных процедур, которые будут использоваться в создаваемом подпроцессе (или отсоединенном процессе).

[2] UAF - User Authorizaton File
[3] UIC - User Identification Code

Таблица 2-2. Функции и команды используемые для управления процессами

Системная функция	*Команда DCL*	*Назначение*
LIB$GETJPI SYS$GETJPI SYS$GETJPIW	SHOW POROCESS	Возвращают информацию о процессе
SYS$SETPRV	SET PROCESS	Установка привилегий процесса
SYS$SETPRI	SET PROCESS	Установка приоритета процесса
SYS$HIBER SYS$SUSPND SYS$RESUME	SET PROCESS	Приостановка выполнения процесса.
SYS$SETPRN	SET PROCESS	Задать имя процесса.
SYS$EXIT SYS$FORCEX SYS$DELPRC	EXIT и STOP	Удалить процесс

2.4. Связь между процессами

Обзор

Операционная система OpenVMS процессам общаться с самим собой, другими процессами, с операционной системой и с другими операционными системами.

2.4.1. Связь внутри процесса

Связь внутри процесса, между компонентами задачи, может осуществляться несколькими способами при помощи:

- Локальных флагов событий.
- Логических имен (в режиме супервизора).
- Глобальных символьных имен (символов).
- Общих блоков.

Локальные флаги событий, логические имена, символы и общие блоки одинаково применимы для "последовательных" задач, т.е. задача, которая читает информацию выполняется непосредственно после задачи поместившей эту информацию. Только общие блоки дают возможность надежно передавать данные от задачи к задаче если в промежутке выполнялась другая задача. Поэтому локальные флаги событий, логические имена, символы с уверенностью можно использовать для общения внутри одной задачи. Кроме того флаги событий могут использоваться для синхронизации выполнения внутри задачи.

Поскольку постоянные почтовые (permanent mailbox) ящики или постоянные глобальные секции не уничтожаются после окончания работы создавшей их задачи, они тоже могут использоваться для передачи данных задачам, которые будут работать после завершения выполнения данной задачи.

Локальные флаги событий

Флаги событий - это посылка набора бит для использования в программах, поддерживаемая системой OpenVMS. Программа может выставлять, читать и снимать флаги событий. Установкой и снятием флага событий одна часть программы может сигнализировать другой части, о начале и окончании события. Другой компонент программы проверкой флага события может определить, что событие происходит или окончено.

Логические имена

Логические имена могут хранить до 255 байт информации. Если вы хотите передать данные от одной программы к другой внутри одного процесса, вы можете присвоить логическому имени определенное значение и другая программа будет иметь доступ к этой информации.

Символы интерпретатора командного языка

Символьные имена, которые вы можете создавать и читать - это символы интерпретатора командного языка (CLI[4]). Все символы хранятся в таблице логических символов, поддерживаемых DCL. По умолчанию итерпретатором командного языка является DCL. Символ может хранить до 255 байт информации. Вы можете использовать символы только в процессах используюших интерпретатор DCL.

Когда используются глобальные символы

Если вы хототе передавать данные внутри одного процесса, то вы можете присвоить данные глобальному символу, в процессе его создания. Тогда другие программы могут иметь доступ к содержанию глобального символа.

Когда используются локальные символы

Локальные символы можно использовать только на уровне интерпретатора команд DCL.

Использование общих блоков

Общие блоки используются для сохранения данных от одной задачи для следующей задачи. Общие блоки не могут быть повреждены от того момента когда одна задача записывает информацию, до того момента когда следующая задача сцитывает информацию.

2.4.2. Связь между процесами

Средствами передачи информации между процессами являются:

- Общие флаги событий.
- Глобальные секции.
- Почтовые ящики.
- Блокировки ресурсов.
- Разделяемые файлы.

Когда общие флаги событий и блокировки ресурсов устанавливают связь, они еще используются для синхронизации событий.

[4] CLI - Command Language Interpreter

2.5. Синхронизация

Техника синхронизации зависит от того, в каких программных модулях вы хотите использовать синхронизацию: в одном программном модуле, в разных программах в рамках одного процесса или в разнах программах из разных процессов. Если приложение требует выполнения двух или более программ вы можете выполнять программы: последовательно в рамках одного процесса, последовательно используя несколько процессов или параллельно в нескольких процессах.

Синхронизация программ может выполнятся с использованием следующих возможностей системы OpenVMS:

- Местные и общие флаги событий.
- Асинхронные системные прерывания (AST[5]).
- Таймеры.
- Синхронные процедуры.
- Блокировки ресурсов.
- Системные библиотеки параллельных вычислений (PPL$).
- Передача управления от одной программы к другой.

2.5.1. Синхронизация при помощи флагов событий

Обзор

Для синхронизации событий при помощи флагов событий, программа устанавливает некоторые биты флагов события после выполнения некоторой части своего кода. Другая программа проверяет значения битов флага события. Если биты флага событи я установлена, вторая программа может продолжать выполнение.

Типы флагов событий

Флаги событий делятся на четыре ветви - две для локальных флагов событий и две для общих флагов. Локальные флаги событий характерны для синхронизации событий внутри одной программы или для передачи информации от одной программы ко второй, которая выполняется сразу после первой в том же процессе. Общие флаги событи используются для синхронизации событий между задачами выполняемыми в разных процессах (предоставляются процессам выполняемым в одной группе.

Кличество и использование флагов событий описывает Таблица 2-3.

Таблица 2-3. Флаги событий

Номер ветви	*Номер флага*	*Тип*	*Использование*
0	0	Локальный	Флаг, который используют по умолчанию системные процедуры.
0	с 1 по 23	Локальный	Могут использоваться в системных процедурах. Призапросе флага события его значение недоступно до тех пор, пока он специальным

[5] AST - Asynchronius system traps

Номер ветви	*Номер флага*	*Тип*	*Использование*
			образом не освобожден.
0	с 24 по 31	Локальный	Зарезервированы DIGITAL.
1	с 32 по 63	Локальный	Доступен для общего пользования
2	с 64 по 95	Общий	Доступен для общего пользования
3	с 96 по 127	Общий	Доступен для общего пользования

2.6. Разделяемые ресурсы

Обзор

Операционная система OpenVMS предоставляет следующие возможности для разделения данных и программного кода между программами:

- Логические символы и логические имена DCL.
- Библиотеки.
- Разделяемые образы.
- Глобальные секции.
- Общие блоки встроенные в разделяемые образы.
- Разделяемые файлы OpenVMS RMS.

Символы и логические имена также используются для взаимодействия внутра процесса и между процессаи и описываются в разделе 2.4. Общие блоки так же могут использоваться для общения между процессами.

2.6.1. Разделение программного кода

Вы можете использовать следующие ресурся OpenVMS для разделения кода между программами:

- Текстовые, Макро или Обьектные библиотеки, хранящие части кода. Текстовые и Макро библиотеки хранят исходные тексты (исходный код); обьектные библиотеки хранят обьектный код. Для поддержки и управления библиотек используется команда LIBRARIAN. Как использовать команду LIBRARIAN вы можете прочесть в документации от операционной системы (*Librarian Utility Manual*).
- Разделяемые образы - это образы задач которые откомпилированны (comriled), скомпонованы (linked), но не могут выполнятся независимо. Эти образы можно хранить в библиотеках.

Обьектные библиотеки

Объектные баблиотеки используются для хранения часто используемых процедур. Это позволяет:

- Уменьшить количество перекомпиляций.
- Уменишить количество файлов, которые надо администрировать.
- Упростить процесс компоновки задач.

Исходный текст хранимых в библиотеке объектных модулей может быть написан на любом языке программирования который поддерживает Ваша система, а модули хранимые в библиотеке могут компоноваться с другими модулями, написанными на любом языке программирования из числа поддерживаемых OpenVMS.

Для объектных библиотек всегда используйте расширение файла OLB. Все модули, хранящиеся в объектной библиотеке должны иметь расширение OBJ.

Текстовые и Макро библиотеки

Любые часто используемые процедуры могут храниться в библиотеках в иходных текстах. Если вы хотите воспользоваться такой процедурой, вы можете затребовать ее из своей программы.

Мдули исходных кодов хранятся в текстовых библиотеках. Расшрение файлов у текстовых библиотек TLB.

Если вы используете ассемблер VAX MACRO, вы можете хранить любые исходные тексты в Макро библиотеке. Расширение файла у макро библиотеке MLB. Все исходные тексты, хранимые в Макро библиотеке, должны иметь расширение MAR.

Разделяемые образы

Разделяемый образ не может выполнятся самостоятельно, но может быть скомпонован вместе с выполняемым образом. Если у вас есть программная елиница, которая используется больше чем в одной программе, вы можете сделать ее разделяемым образом для того чтобы:

- *Сохранить дисковое пространство* - Выполняемый образ, который скомпонован с разделяемым образом, физически не содержит разделяемого образа. Существует только одна копия разделяемого образа.
- *Упрощают поддержку задачи* - Есил вы используете векторы передачи и опцию GSMATCH[6]. Вы можете перекомпилировать и перекомпоновать разделяемый образ без перекомпоновки выполняемого образа.

Разделяемый образ может позволить улучшить использование памяти, если он установлен как разделяемый образ.

[6] О векторах передачи и опции GSMATCH читайте в Guide to VMS Programming Resouse