ПОВЫШЕНИЕ ОТКАЗОУСТОЙЧИВОСТИ СЕТЕВЫХ ПРИЛОЖЕНИЙ РЕАЛЬНОГО ВРЕМЕНИ

С. Н. Зыль

Как бывает досадно, когда нужная Интернет-страница оказывается недоступна в самый неподходящий момент! Но это вполне можно пережить. Гораздо хуже, если не удается изменить опасное направление движения транспортного средства из-за того, что программа управления рулевым механизмом не может принять управляющую директиву с компьютера диспетчера. Способность сервиса быть доступным для приема и обработки клиентских запросов измеряют такой величиной, как коэффициент готовности (availability).

В классической микроядерной ОС реального времени Neutrino, разработанной компанией QNX, базовый системный модуль имеет размер 375 Кбайт для платформы Intel x86 или 697 Кбайт для PowerPC

Проблема обеспечения высокого коэффициента готовности достаточно сложна и имеет разные пути решения, их можно свести к двум: увеличение времени безотказной работы сервиса и сокращение времени восстановления работоспособного состояния сервиса после сбоя. Сложность реализации программной поддержки требуемого коэффициента готовности заключается в том, что большинство серверных ОС построены на базе монолитных ядер, поэтому дополнительные системные механизмы, как правило, являются надстройкой над стандартными сервисами таких ОС, а это не может не сказаться на эксплуатационных характеристиках управляемых ими конечных изделий.

В другом классе операционных систем — реального времени — бóльшую часть составляют системы с «плоской» архитектурой, в которой компоненты ОС и прикладной код составляют, по сути дела, один исполняемый модуль. Про такие ОС говорят, что каждый их компонент является единой точкой сбоя (Single Point of Failure — SPoF), поскольку ошибка в любом из компонентов может привести к сбою всей ОС.

Особую роль в современных системах с повышенными требованиями к коэффициенту готовности играют микроядерные ОС, реализующие полную изоляцию адресных пространств как системных, так и прикладных компонентов, что сужает SPoF до размеров базового системного модуля, содержащего микроядро. В классической микроядерной ОС реального времени Neutrino, разработанной компанией QNX, такой базовый модуль имеет размер 375 Кбайт для платформы Intel x86 или 697 Кбайт для PowerPC.

Микроядерная архитектура Neutrino предоставляет разработчику достаточно широкое пространство для творчества, поскольку, чтобы добавить к ОС новый сервис, достаточно написать менеджер ресурса — прикладную программу, использующую простой, хорошо документированный интерфейс сообщений и пространства путевых имен. Менеджеры ресурсов не расширяют SPoF системы, если не содержат функций — обработчиков прерываний (ISR). Поэтому разработчики стремятся либо минимизировать ISR, либо использовать для обработки прерываний такие механизмы извещения, имеющиеся в ОС, как импульсы или сигналы.

Механизм пространства путевых имен в сочетании с Qnet* — технологией прозрачного доступа к ресурсам локаль-

** Qnet — уникальный сетевой протокол ОС Neutrino, позволяющий на уровне функций ядра ОС обеспечить прозрачный доступ к ресурсам вычислительной сети.*

*** Префиксы — элементы «пространства путевых имен», или «дерева префиксов», — специального механизма ОС QNX Neutrino, обеспечивающего унифицированный доступ к ресурсам компьютера, сервисам ОС и серверным приложениям. По своей нотации аналогичен файловому пространству имен в других ОС. При использовании протокола Qnet служит для обеспечения установления соединения между клиентским и серверным приложениями.*

ной сети — позволяет обеспечить как распределение нагрузки, так и «горячее» резервирование серверных программ посредством перезагрузки префиксов** или автоматического перенаправления запросов к другим префиксам. Это позволяет повысить коэффициент готовности путем увеличения времени безотказной работы.

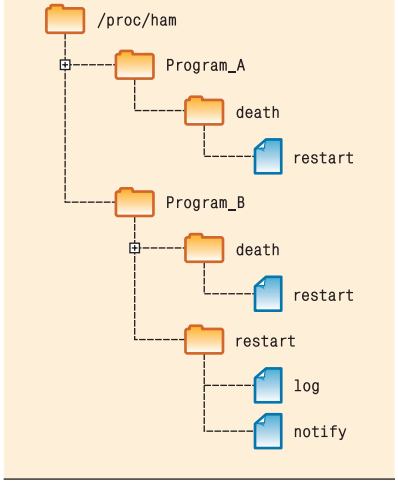
Для повышения коэффициента готовности путем снижения среднего времени восстановления работоспособного состояния в Neutrino можно использовать менеджер ресурсов, выполняющий функции монитора ключевых процессов (High Availability Manager — HAM).

Основное назначение монитора ключевых процессов — максимально быстрое определение факта сбоя серверного приложения и принятие мер для восстановления нормальной работы сервиса.

Чтобы выполнять свои задачи, менеджер HAM для начала клонирует сам себя, т. е. порождает дублирующий процесс, которому, используя механизм разделяемой памяти, предоставляет полную информацию о мониторинге системы. HAM и его дублер строго следят друг за другом, чтобы при сбое одного из них второй процесс мог немедленно породить новый процесс-дублер. Информация о ходе мониторинга доступна также и для оператора с помощью механизма пространства путевых имен: HAM регистрирует префикс с именем /proc/ham, имеющий тип «каталог». Содержимое этого виртуального каталога дает достаточно наглядное представление о возможностях монитора ключевых процессов (см. рисунок).

Заполняется «каталог» /proc/ham из кода прикладной программы с помощью функций HAM API. Для этого сначала задаются объекты мониторинга, т. е. указывается, какие программы необходимо контролировать. Затем для каждого из объектов предписывается, на какое событие, происшедшее с объектом, следует прореагировать. И наконец, для каждого из событий определяется реакция.

Обеспечить высокий коэффициент готовности серверного приложения — это только полдела, ведь, как известно, в сетевой среде самым уязвимым местом является именно сеть. Для восстановления логического соединения клиента в ОС Neutrino предусмотрена специальная библиотека



Структура информации, отображаемой монитором ключевых процессов через префикс /proc/ham

На рисунке показана конфигурация виртуального каталога /proc/ham, которая в случае сбоя предопределяет перезапуск серверной программы Program_A, затем перезапуск серверной программы Program_B, после чего делает запись в журнале, а также посылает извещение об этом факте.

Однако обеспечить высокий коэффициент готовности серверного приложения — это только полдела. Ведь, как известно, в сетевой среде самым уязвимым местом является именно сеть. Значит, важное клиентское приложение может не выполнить задачу не из-за того, что произошел сбой сервера, а, например, из-за временной неисправности на линии. Для того чтобы помочь разработчикам систем на базе Neutrino решать проблемы такого рода, существует библиотека восстановления логического соединения клиента (Client Recovery Library).

Идея восстановления логического соединения клиента, как и все остальное в Neutrino, предельно проста. Ведь, несмотря на то что программист для организации взаимодействия клиента с удаленным сервером, как правило, использует POSIX-функции файлового ввода-вывода, реально на нижнем уровне используется механизм сообщений микроядра. Если во время обработки клиентского сообщения-запроса произошел сбой сервера, низкоуровневая клиентская функция, инициировавшая запрос, завершится с ошибкой, имеющей определенный код. Следовательно, для восстановления работоспособности необходимо «отлавливать» коды ошибок, возникающие при разрывах соединений,

и принимать меры к восстановлению соединений. Именно для этого предназначена библиотека Client Recovery Library.

Рассмотрим пример кода клиентского приложения, которое получает информацию от серверного приложения, выполняющегося на узле host3 сети Qnet:

```
fd = open ("/net/host3/
dev/myserver", O_RDONLY);
read (fd, &buffer, size-
of(buffer));
```

В этих двух строках клиент сначала устанавливает соединение с серверным приложением, используя зарегистрированный сервером префикс `/dev/myserver`. Дескриптор `fd`, возвращаемый функцией `open()`, используется клиентом для получения информации от сервера с помощью функции `read()`. Если при получении информации от сервера произойдет, например, разрыв кабеля, то функция `open()` завершится со значением –1.

Теперь посмотрим, как те же действия выполняются с использованием Client Recovery Library:

```
fd = ha_open ("/net/
host3/dev/myserver",
O_RDONLY, vosstanovim,
"/net/host3/dev/myserver",
0);
read (fd, &buffer, size-
of(buffer));
```

В приведенном примере видно, что вместо POSIX-функции `open()` используется функция `ha_open()`, которой передаются дополнительные параметры — имя функции-восстановителя, еще одно имя файла и 0.

Указанная в параметрах `ha_open()` функция-восстановитель (в примере — `vosstanovim()`) будет вызвана автоматически при разрыве соединения, идентифицируемого дескриптором `fd`, при этом функция `vosstanovim()` получит в качестве параметров вызова значение файлового дескриптора, подлежащего восстановлению, и имя файла, который следует открывать для восстановления соединения (это имя — второе имя файла в аргументах функции `ha_open()`). Возвращает функция-восстановитель новый файловый дескриптор, который должен быть таким же, как и подлежащий восстановлению, чтобы функция `read()` не заметила подмены. Посмотрим на код функции-восстановителя:

```
int vosstanovim ( int fd,
void *prefix )
{
/* Здесь мы можем, напри-
мер, включить сирену и по-
дождать минуту-другую,
чтобы эксплуатирующий пер-
сонал успел восстановить
физическое соединение */
return ha_reopen( fd, (char
*) prefix, O_RDONLY );
}
```

Клиентский и серверный механизмы повышения коэффициента готовности можно сочетать. Например, при запуске клиент может, используя HAM API, запросить у соответствующего монитора ключевых процессов извещение о перезапусках серверного приложения при сбоях. Затем в теле функции-восстановителя можно ожидать это извещение (лучше, конечно, само ожидание выполнять в отдельном потоке и при получении извещения уведомлять функцию-восстановитель с помощью такого POSIX-механизма синхронизации потоков, как условные переменные). Разумеется, разработчик серверного приложения должен позаботиться о надлежащем «горячем» резервировании, чтобы при восстановлении соединения клиент мог продолжить работу с того места, на котором она приостановилась. Неплохим образцом эффективного «горячего» резервирования является сам менеджер HAM.

Другими словами, при использовании микроядерных сетевых технологий возможности по поддержанию требуемого коэффициента готовности устройств, управляемых ОС Neutrino, ограничены, по сути дела, только аппаратурой и фантазией разработчиков.

В заключение остается только добавить, что исходные тексты HAM, HAM API и Client Recovery Library входят в состав комплекта для разработки средств мониторинга ключевых процессов (Critical Process Monitoring Technology Development Kit — CPM TDK), поэтому программисту при создании отказоустойчивых систем нет необходимости изобретать велосипед.

При использовании микроядерных сетевых технологий возможности по поддержанию требуемого коэффициента готовности устройств, управляемых ОС Neutrino, ограничены только аппаратурой и фантазией разработчиков

Об авторе
Зыль Сергей Николаевич,
преподаватель
компании SWD Software
Телефон: (812) 102-0833
E-mail: s.zyl@swd.ru