

Комментарии к Протоколу Объединенного Пространства Памяти

Статус этого документа

Данная записка является документом Internet-Draft и полностью соответствует всем условиям Раздела 10 RFC2026 [1].

Internet-Draft являются рабочими документами IETF, его областей, и его рабочих групп. Другие группы также могут распространять рабочие документы, как Internet-Draft.

Internet-Draft это черновые документы, которые хранятся максимум шесть месяцев и могут быть модифицированы, заменены, или удалены в соответствии с другими документами в любое время. Ссылки на Internet-Draft следует использовать только с пометкой "work in progress."

Список текущих Internet-Draft доступен на <http://www.ietf.org/ID.html>

Список зеркальных сайтов Internet-Draft доступен на <http://www.ietf.org/shadow.html>.

Резюме

Этот документ содержит описание моделей нескольких систем, для которых предназначен Протокол Объединенного Пространства Памяти (Unified Memory Space Protocol - UMSP). Целью документа является обзор задач, которые были сформулированы при разработке спецификации протокола.

1 Введение

Данный документ является дополнением к документу "Unified Memory Space Protocol Specification" [2]. UMSP является сетевым протоколом, который предоставляет возможность непосредственного доступа к памяти удаленных узлов. Он опирается на распределенное 128-разрядное пространство памяти. Приведенные в документе модели были использованы при постановке задачи для разработки UMSP и повлияли на выбор технических решений. Для описанных в документе систем не существует детальных проектов. Также, для них не приведено строгое обоснование возможности реализации.

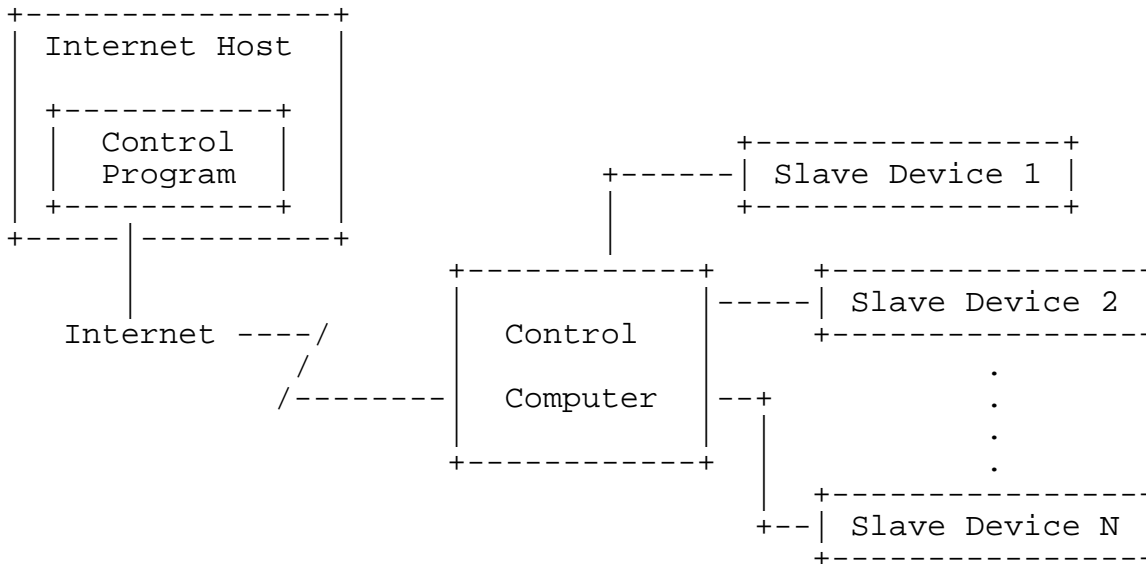
В документе приведены следующие модели:

- О управление простыми устройствами
- О универсальный высокоуровневый протокол
- О распределенное приложение, создаваемые в одиночном процессе компиляции
- О распределенная виртуальная память.

2 Описание моделей

2.1 Управление простыми устройствами

Рассматриваемая в данной секции архитектура ориентирована на удаленное управление простыми устройствами. Логика минимального профиля UMSP относительно проста и может быть реализована в таких устройствах. Базовая архитектура приведена на следующем рисунке:



Управляющий компьютер (Control Computer - CC) может выполнять следующие функции:

- осуществлять функции защиты от несанкционированного доступа,
- выполнять функции шлюза, если коммуникации между CC и подчиненными устройствами (Slave Device - SD) отличны от TCP/IP,
- содержать программу, выполняющую часть логики SD.

Подчиненные устройства (SD) могут иметь простую логику и взаимодействовать с CC по сетевым коммуникациям.

UMSP обеспечивает прямое взаимодействие между управляющей программой (CP) и SD. Возможность реализации приведенной архитектуры обеспечивается следующими функциями протокола:

- Установление сеансового соединения не является обязательным. До начала обмена CP может получить информацию о SD (например о типе устройства или VM) непосредственно считывая данные по определенным адресам памяти устройства.
- Взаимодействие между CP и SD может осуществляться при помощи трех простых инструкций с фиксированной длиной данных: чтения/записи в память и синхронизации (для отслеживания асинхронных событий). Эти инструкции могут даже исполнять устройства, не имеющие микрокод. Все остальные инструкции могут игнорироваться на SD.
- С точки зрения логики UMSP SD может являться:
 - 1) Адресуемым элементом сети, на котором установлена VM определенного типа. Такая VM не имеет собственной системы команд и прямо выполняет инструкции UMSP.
 - 2) Отдельной VM, которая подключена к узлу CC.
 - 3) Программируемым объектом в адресном пространстве CC. В этом случае логика устройства полностью определяется функциональными возможностями CC.

В общем случае, SD могут иметь большие вычислительные ресурсы и реализовывать стек TCP/IP. При этом возможны различные варианты архитектуры, которые в документе не рассматриваются.

2.2 Универсальный высокоуровневый протокол

UMSP может являться последним в стеке и единственным высокоуровневым сетевым протоколом. Все остальные функции могут быть реализованы в виде Application Program Interface (API) на стандартных виртуальных машинах (VM). В пользу сказанного можно привести следующие аргументы:

- Стандартизовать VM проще, чем сетевой протокол. При этом даже на VM с простейшим набором инструкций можно реализовать очень сложное API.
- Стандартизация API является более простой задачей, чем стандартизация сравнимого по функциональности сетевого протокола. Более того, API может быть значительно более сложным и многофункциональным по сравнению с возможностями сетевых протоколов.
- UMSP напрямую конвертирует логику инструкций прикладных программ, работающих с данными в памяти, в логику сетевого обмена. Благодаря этому может быть снижено потребление вычислительных ресурсов, так как отпадает необходимость в функциях уровня представления данных.
- Для разработки программ использовать API проще, чем использовать протокол. Это позволяет значительно снизить трудоемкость разработки распределенных приложений и реализовать функции недоступные при использовании сетевых протоколов.
- Унификация высокоуровневого протокола позволяет осуществлять 100% верификацию пользовательского трафика на шлюзах и обеспечивать эффективную защиту локальных сетей от внешних атак.

Вероятно, в виде API могут быть реализованы даже функции управления сетью, маршрутизацией, DNS и прочие.

2.3 Распределенное приложение, создаваемое в одиночном процессе компиляции

Основное удобство для разработчика заключается в существенном упрощении создания распределенных приложений. Для того чтобы это показать, в этой секции приведены примеры программ с несуществующими на сегодняшний день расширениями языков. Эти примеры являются только иллюстрацией.

Примеры содержат минимально необходимый код. Возможность непосредственно адресовать память на удаленной системе открывает широкие возможности перед разработчиками VM для реализации различных распределенных сервисов и API. Решение этих вопросов не относится к сфере сетевого протокола.

1) Следующий пример написан на языке BASIC с использованием процедурного подхода:

```
' Эту функцию предполагается выполнять на удаленной машине
' Вероятно, код распределенной функции имеет некоторые ограничения,
' по сравнению с обычной функцией. Для того, чтобы компилятор мог
их
' проконтролировать, в определении использован квалификатор
' Relocatable. Кроме этого переменные ссылочного типа в такой
функции
' должны обрабатываться компилятором особым способом.
Relocatable Function dFunc(parm1 As Integer) As String
```

```
' Здесь можно писать обычный код, использующий локальные и
' глобальные переменные или вызывающий функции пользователя или
```

```
' API, предоставляемые VM.

DFunc = "OK"
End Function

' Пример вызова функции на другой машине с использованием
' предварительного установления соединения
Sub RunD()
  On Error GoTo Next

  ' Переменная идентифицирующая удаленный хост
  Dim OtherVM As RelocatableVM
  Dim ForRet As String

  ' Установить соединение с удаленным узлом. Первый параметр
  ' является константой типа адреса (в данном случае десятичный
  ' адрес IP).
  Set OtherVM = ConnectVM(vbTrIP, "10.15.127.15")
  If Err Then
    MsgBox "Ошибка установления соединения"
    Exit Sub
  End If

  ' Вызвать функцию
  ForRet = OtherVM.Call dFunc(10)
  If Err Then
    MsgBox "Ошибка выполнения функции"
    Exit Sub
  End If
End Sub
```

2) Второй пример написан на языке JAVA с использованием объектного подхода:

```
/**
 * Экземпляры этого класса могут быть созданы на удаленном узле
 */
relocatable class dClass
{
  // Здесь могут быть переменные-члены класса

  /**
   * Этот конструктор будет вызван на удаленном узле
   * после выделения памяти.
   */
  dClass()
  {
    // Здесь расположен код инициализации
  }

  /**
   * Функция-член класса
   */
  String dMetod(int Parm1)
  {
    // Код, который здесь находится, может выполнять обычные
    // вычисления, вызывать функции API, создавать экземпляры
    // стандартных или специальных классов, а также экземпляры
    // классов, определенных в этой программе и имеющих
    // квалификатор relocatable

    return "OK";
  }
}
```

```
/**
 * Пример создания экземпляра класса на удаленном узле
 */
class dExamp
{
    dFunc(int Parm1)
    {
        try
        {
            // Создаем класс на удаленном узле IP
            dClass dc = new ("10.15.5.120", dClass);

            // Вызываем функцию. Эта функция выполняется на удаленном
            // узле.
            String sRet = dc.dMetod(15);

            // Удаляем экземпляр
            dc = null;
        }
        catch (Exception e)
        {
            // Обработка исключений
        }
    }
}
```

Из приведенных примеров видно, что для разработчика программ основным усложнением по сравнению с нераспределенным приложением является обязательная обработка ошибок. Это объясняется тем, что исключительное условие может вызвать любой оператор.

2.4 Распределенная виртуальная память

Создание распределенной между узлами Internet виртуальной памяти является сложной с задачей с неоднозначным решением. Традиционная подкачка страниц вероятно нецелесообразна, так как приведет к большому трафику ненужных данных. Кроме этого, алгоритмы блокировки страниц в сетевой среде являются неочевидными. По этим причинам UMSP в явном виде не поддерживает эти функции.

Единственное требование к протоколу, которое было сформулировано на основе анализа подобных систем, это 128-битный адрес памяти фиксированной длины. Вероятно, именно такой адрес оптимален с точки зрения аппаратной реализации и одновременно удобен при программной реализации. Также, данная длина адреса может обеспечить решение любых задачи на любых платформах в настоящее время и в предсказуемом будущем.

3 Использованные сокращения

API Application Programming Interface.

CC Control Computer

CP Control Program

SD Slave Device

UMSP Unified Memory Space Protocol

VM Virtual Machine

4 ССЫЛКИ

- [1] S. Bradner, "The Internet Standards Process -- Revision 3", BCP 9, RFC 2026, October 1996.
- [2] A. Bogdanov, "Unified Memory Space Protocol Specification", RFC3018, December 2000.

5 Адрес автора

Email: a_bogdanov@iname.ru

Авторские права

Copyright (C) the Internet Society (2000). Все права зарезервированы.