Принципы построения новой операционной системы: **проще, дешевле, безопаснее**

Станислав Братанов

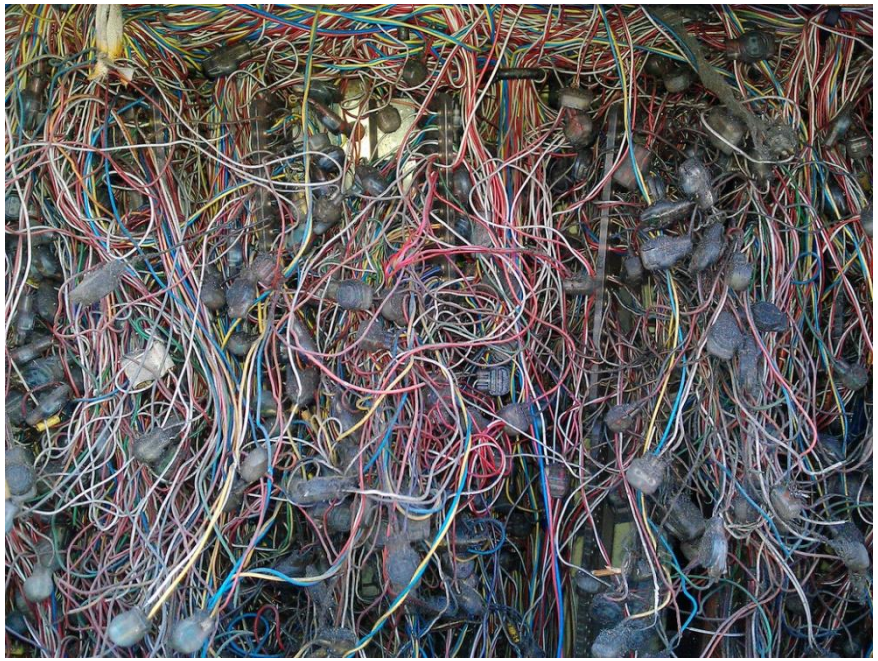
Stanislav.Bratanov@intel.com

24 мая 2017

Содержание

- Задачи новой операционной системы
- Базовый принцип и его свойства
- Сопутствующие технические решения
- Заключение и дальнейшие шаги

Цели и задачи



- **Упрощение**, **удешевление** и **унификация** поддержки современных и будущих вычислительных систем
- Формирование программной платформы, **масштабируемой** на широкий спектр технических решений
- Стимулирование программных решений, поддерживающих вышеперечисленные принципы
- Формулирование требований к развитию аппаратуры

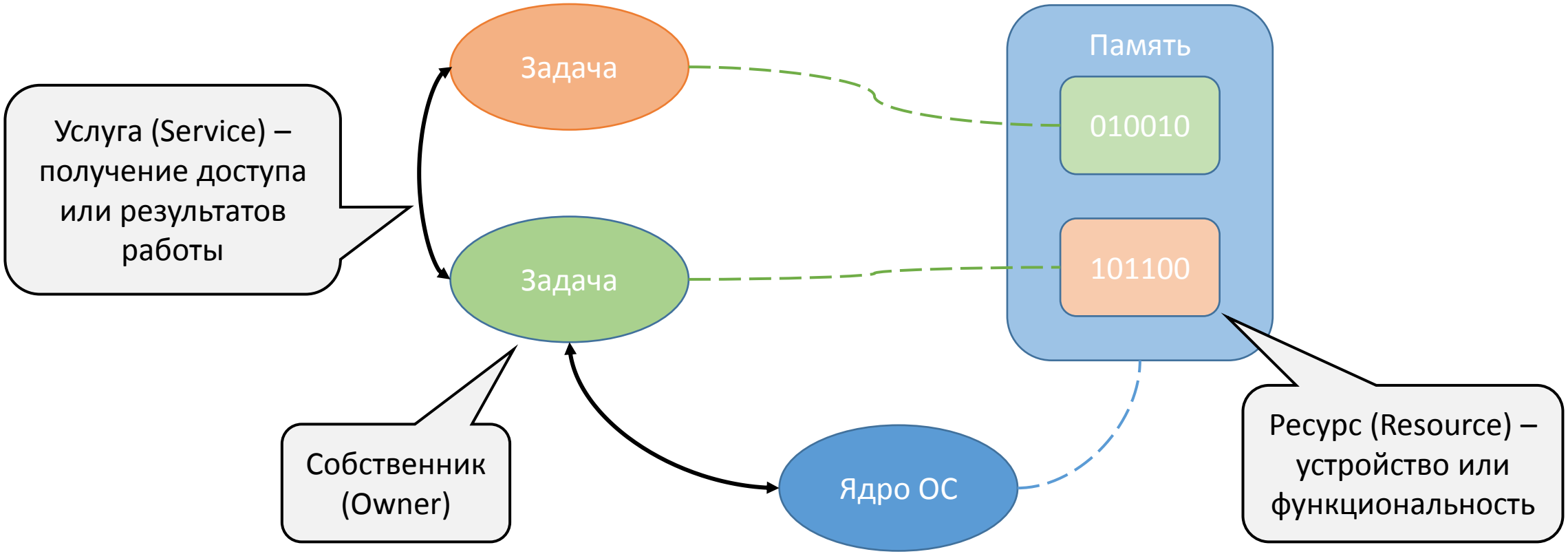
Масштабируемость как шаг к упрощению

- Экономия при переносе на новую платформу – важнейший шаг к упрощению и удешевлению

Минимальный платформенный базис?

- Разное – от набора инструкций до организации памяти
- Общее – необходимость организации параллельного исполнения кода и обмена данными
- Что если предложить модель, обеспечивающую поддержку общего и минимизирующую зависимость от отличий?

ROS-модель


Задача
Задача
Услуга (Service) – получение доступа или результатов работы
Память
010010
101100
Собственник (Owner)
Ядро ОС
Ресурс (Resource) – устройство или функциональность

Материализация услуги

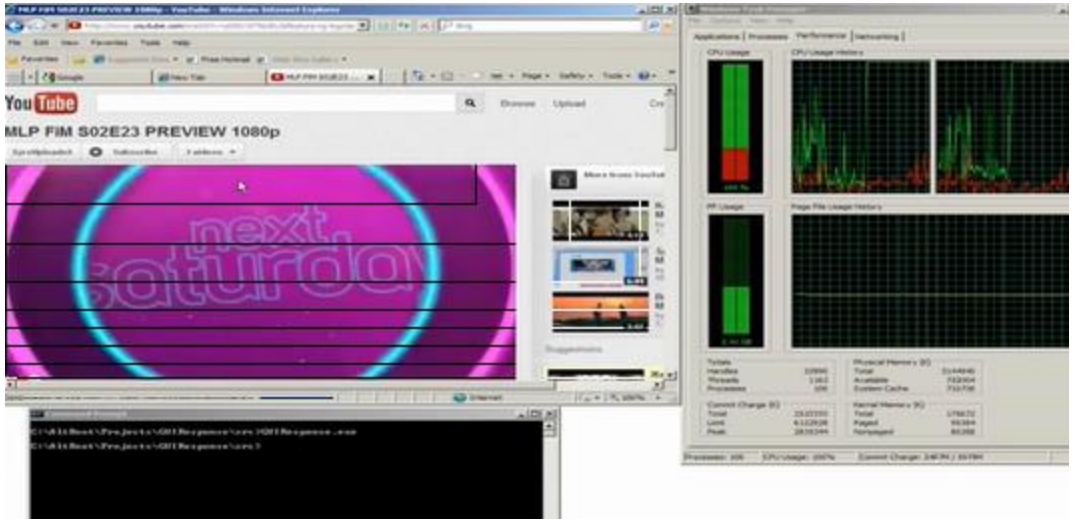
- Услуга – это не только интерфейс обмена данными, а коммуникационная и синхронизационная сущность – канал
- Канал нужно сначала материализовать:
  - Запросить у ОС ресурс-память под канал
- Потом «договориться» с поставщиком услуги и установить коммуникацию
  - Топологии каналов подробно рассматриваются в статье

Следствия для организации задач

- Все сводится к взаимодействию через канал
- Следовательно, и передачу управления надо сделать через каналы
- Схема Yield-To:
  - вместо традиционных объектов синхронизации

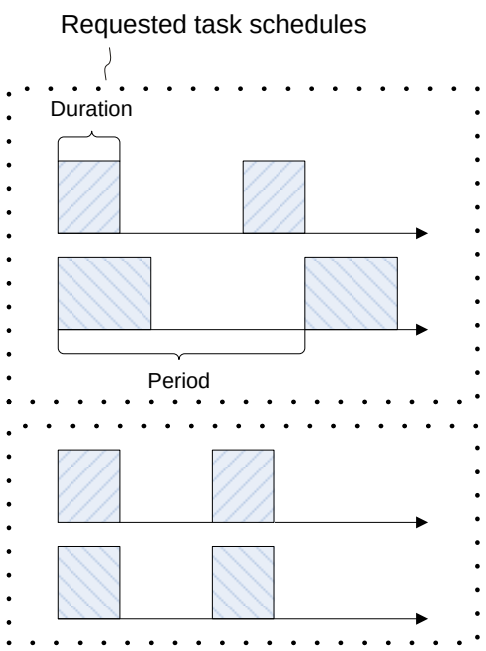
Следствия продолжаются

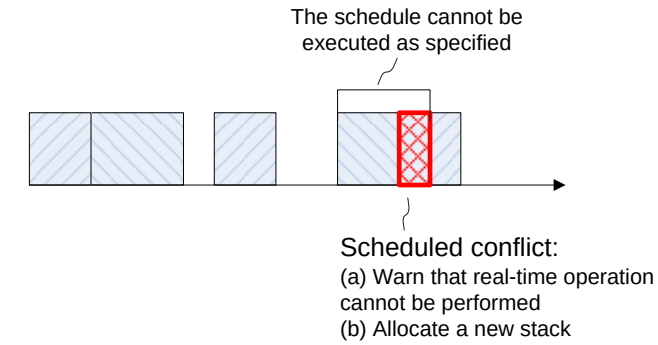
- Задачи не знают друг о друге, а только об услугах
- Следовательно, их работу нужно организовать по принципу качества обслуживания (*см. след. слайд*)

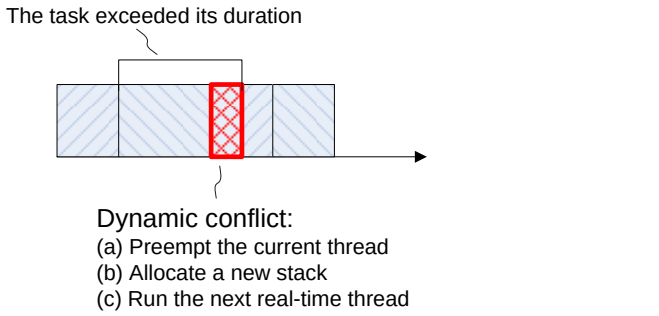


Скоординированное вытеснение

- ***Период*** и ***скважность*** вместо приоритетов
- ОС может:
  - «предвидеть» конфликты
  - «предсказывать» сбои реального времени
  - «изолировать» нарушителей
  - экономить на сохранении контекста «добросовестных» задач







Следствия для программирования

- Каналы могут содержать и код, и данные
  - нужны конструкции, позволяющие компилятору объединять код и данные
  - код, инвариантный к адресам загрузки
  - адресация данных относительно кода
  - определение стандартной асинхронной функции канала:


```
/// in-channel functions are advised to have the following prototype, wherein:
///   chanfunc may be:
///     task start function - with (arg0, sys, arg1) semantics
///     channel initialization - returning a pointer to local data,
///     or any service function contained in a channel
///   chan is the self-pointer to the channel
///   sys is the pointer to the system channel
///   loc is the pointer to local data or an optional parameter
void* chanfunc(chan, sys, loc);
```

```
/// declarations
channel class A    /// a new type modifier
{
  int x;
  virtual void f();
  void g();
} *a, *b, *c, *d;

/// system channel - to make system calls
syschan_t* sys;
bool multi = false;             /// or true for multi-channels

/// allocations
a = new A;                      /// combines x, f, g, and vft
b = new A;                      /// import space allocation

/// support functions
c = export(a, sys, multi);      /// export a channel of type A
if(c != a){ delete a; a = c; }  /// the system moved the channel

d = import(b, sys, multi);      /// import a channel of type A
if(d != b){ delete b; b = d; }  /// the system moved the channel

int i = self(a, sys);           /// get in-channel client's index

disconnect(a, sys);             /// disconnect the exported channel
disconnect(b, sys);             /// disconnect the imported channel
                                /// OS may free channel descriptors now
```

Проще и дешевле – обсудили, а безопаснее?

- Изоляция задач через каналы и «дробление ответственности»
  - В случае сбоя задача не получит качественную услугу, но сможет перейти к другому поставщику
- Система может заменять/перезапускать задачи «прозрачно» для их контрагентов
  - Помехоустойчивость и «горячее» обновление
- Переносимость
  - «Прозрачная» трансляция кода при импорте услуги с системы с несовместимой архитектурой
  - Перенос задачи «целиком» на другую систему (виртуализация без виртуализации)

Заключение и дальнейшие шаги

- Перспективная модель для построения масштабируемой системы
- Разрабатывается с 1999 как среда для предсказуемого измерения производительности
- В 2009 доработана для сенсорных сетей

- Сейчас нужно найти правильную нишу для продвижения ОС

- Статья: http://www.hoopoesnest.com/texts/chansys.pdf
- Перевод: https://habrahabr.ru/company/intel/blog/300884/

Спасибо за внимание!