блохбук

© Dmitry Ponyatov <dponyatov@gmail.com>

1 марта 2016 г.

Оглавление

Об этом сборнике

В этот сборник я пишу отдельные статьи, сортированные только по общей тематике, и добавляю их, когда у меняв очередной раз зачешется LaTeX.

Это сборник черновых материалов, которые мне лень компоновать в отдельные книги, и которые пишутся просто по желанию "чтобы было". Заказчиков на подготовку учебных материалов подобного типа нет, бо̀льшая часть только на этапе освоения мной самим, просто хочется иметь некое слабоупорядоченное хранилище наработок, на которое можно дать кому-то ссылку.

Часть I

Язык bI

AST: Абстрактный Символьный Тип /Sym/

Использование класса **Sym** и виртуально наследованных от него классов, позволяет реализовать на C_{++}^{+} хранение и обработку **любых** данных в виде деревьев[1]. Прежде всего этот ***символьный тип*** применяется при разборе текстовых форматов данных, и текстов программ. **Язык** *bI* **построен как интерпретатор AST, примерно так же как язык** *Lisp* **использует списки**.

```
1 // ============================================== ABSTRACT SYMBOLIC TYPE (AST)
2 struct Sym {
```

тип (класс) и значение элемента данных

```
1 // ------------------------------------------------------------------------
2     string tag;                          // data type / class
3     string val;                          // symbol value
```

конструкторы (токен используется в лексере)

```
1 // ------------------------------------------------------------ constructors
2     Sym(string,string);                  // <T:V>
3     Sym(string);                         // token
```

Хранение вложенных элементов реализовано через указатели на базовый тип **Sym**. Благодаря виртуальному наследованию и использованию RTTI, этими указателями можно пользоваться для работы с любыми другими наследованными типами данных[2]

[1] в этом ACT близок к традиционной аббревиатуре AST: Abstract Syntax Tree

[2] числа, списки, высокоуровневые и скомпилированные функции, элементы GUI,..

AST может хранить (и обрабатывать) вложенные элементы

```
// ------------------------------------------------------------ nest[]ed elements
    vector<Sym*> nest;
    void push(Sym*);
    void pop();
```

параметры (и поля класса)

```
// ------------------------------------------------------------ par{}ameters
    map<string,Sym*> pars;
    void par(Sym*);                          // add parameter
```

вывод дампа объекта в текстовом формате

```
// ------------------------------------------------------------ dumping
    virtual string dump(int depth=0);    // dump symbol object as text
    virtual string tagval();             // <T:V> header string
    string tagstr();                     // <T:'V'> Str-like header string
    string pad(int);                     // padding with tree decorators
```

Операции над *символами* выполняются через использование набора *операторов*:

вычисление объекта

```
// ------------------------------------------------------------ compute (evaluate)
    virtual Sym* eval();
```

операторы

```
// ------------------------------------------------------------ operators
    virtual Sym* str();                      // str(A)    string representation
```

```
3       virtual Sym* eq(Sym*);              // A = B     assignment
4       virtual Sym* inher(Sym*);           // A : B     inheritance
5       virtual Sym* member(Sym*);          // A % B,C   named member (class slot)
6       virtual Sym* at(Sym*);              // A @ B     apply
7       virtual Sym* add(Sym*);             // A + B     add
8       virtual Sym* div(Sym*);             // A / B     div
9       virtual Sym* ins(Sym*);             // A += B    insert
10  };
```

Часть II

em*Linux* для встраиваемых систем

Структура встраиваемого микро*Linux*а

syslinux Загрузчик

em*Linux* поставляется в виде двух файлов:

1. ядро `$(HW)$(APP).kernel`
2. сжатый образ корневой файловой системы `$(HW)$(APP).rootfs`

Загрузчик считывает их с одного из носителей данных, который поддерживается загрузчиком[3], распаковывает в память, включив защищенный режим процессора, и передает управление ядру *Linux*.

Для работы em*Linux* не требуются какие-либо носители данных: вся (виртуальная) файловая система располагается в ОЗУ. При необходимости к любому из каталогов корневой ФС может быть *подмонтирована* любая существующая дисковая или сетевая файловая система (FAT,NTFS,Samba,NFS,..), причем можно явно запретить возможность записи на нее, защитив данные от разрушения.

Использование rootfs в ОЗУ позволяет гарантировать защиту базовой ОС и пользовательских исполняемых файлов от внезапных выключений питания и ошибочной записи на диск. Еще большую защиту даст отключение драйверов загрузочного носителя в ядре. Если отключить драйвера SATA/IDE и грузиться с USB флешки, практически невозможно испортить основную установку ОС и пользовательские файлы на чужом компьютере.

kernel Ядро *Linux* 3.13.xx

[3] IDE/SATA HDD, CDROM, USB флешка, сетевая загрузка с BOOTP-сервера по Ethernet

ulibc Базовая библиотека языка Си

busybox Ядро командной среды UNIX, базовые сетевые сервера

дополнительные библиотеки

zlib сжатие/распаковка gzip
??? библиотека помехозащищенного кодирования
png библиотека чтения/записи графического формата .png
freetype рендер векторных шрифтов (TTF)
SDL полноэкранная (игровая) графика, аудио, джойстик

кодеки аудио/видео форматов: ogg vorbis, mp3, mpeg, ffmpeg/gsm

К базовой системе добавляются пользовательские программы ***/usr/bin***
и динамические библиотеки ***/usr/lib***.

Процедура сборки

Глава 1

clock: коридорные электронные часы = контроллер умного дурдома

Глава 2

gambox: игровая приставка

Часть III

Микроконтроллеры Cortex-Mx

Часть IV

Технологии

Часть V

Сетевое обучения

Часть VI

Прочее

Литература

[1] **Dragon Book: Компиляторы**

Ахо, Сети, Ульман

[2] **SICP: Структура и интерпретация компьютерных программ**