Лекция 3. Программирование в ОС UNIX. Язык С.

1. Средства программирования ОС UNIX.

1.1. Какой язык программирования использовать?

ОС UNIX поддерживает множество языков программирования. Это и ***компилируемые языки***, и ***интерпретаторы***, и ***генераторы машинных кодов***. Поэтому необходимо сначала ответить на следующие вопросы:

- Какова область применения программы?
- Каков круг пользователей?
- Какова сущность задачи, которую программируем?
- Какова сложность алгоритма?
- Имеет ли задача много независимых частей?
- Можно ли поделить программу на отдельные части (модули) или это будет один модуль?
- К какому сроку программа должна быть готова?
- Нужно ли позаботиться об эффективности программы?
- Будет ли программа переноситься на другие системы?
- Как долго будет эксплуатироваться программа?

1.2. Поддерживаемые языки программирования

Вместе с ОС UNIX/Linux из дистрибутива можно установить большой спектр языков программирования.

1.2.1. Язык С

Язык С тесно связан с ОС UNIX, так как первоначально был разработан именно для реализации ядра операционной системы. Поэтому он очень удобен для программирования задач, использующих системные вызовы операционной системы, например, для организации низкоуровнего ввода/вывода, управления памятью или физическими устройствами, организации связи между процессами и т.д. Кроме того, язык С может успешно применяться и для реализации программ, не требующих такого непосредственного взаимодействия с операционной системой. При выборе языка программирования следует иметь в виду следующие характеристики языка С:

- базовые типы данных: символьный, логический, целый (различной длины), вещественный (различной точности);
- возможность определения производных типов данных, таких как массивы, функции, указатели, структуры и объединения;
- наличие низкоуровневых возможностей;
- наличие многомерных массивов;
- возможность определения указателей на данные конкретного типа и выполнения арифметических действий над указателями с учетом типа адресуемых ими данных;
- побитовые операции;
- множество управляющих конструкций: if, if-else, switch, while, do-while и for;
- высокая степень мобильности программ.

Язык С естественным образом ориентирован на структурное программирование. Большие программы подразделяются на функции, которые можно считать отдельно компилируемыми единицами. Кроме облегчения внесения изменений в программы, при таком подходе в наибольшей степени реализуется идеология программирования в ОС UNIX: стараться в максимальной степени использовать уже имеющиеся программы.

Язык С довольно труден в изучении. Чтобы научиться программировать на С, используя все его возможности, необходимо несколько месяцев интенсивной практики. Поэтому, если Вы программируете лишь эпизодически, лучше выбрать какой-нибудь другой, более простой язык.

1.2.3. Фортран

Фортран является старейшим из языков высокого уровня и успешно применяется для математических расчетов. На Фортране удобно программировать задачи статистического анализа и многие другие научные приложения. Основной целью при проектировании языка было достижение высокой эффективности при выполнении расчетных программ. Цель была достигнута, но для этого пришлось в некоторой степени пожертвовать гибкостью языка и невозможностью использования для реализации

функций ядра ОС. Также тексты программ должны иметь довольно жесткий формат. Фортран является одним из официальных языков программирования ГК РосАтом, РосКосмос.

1.2.4. Паскаль

Паскаль первоначально проектировался как язык для обучения программированию. Поэтому он прост в обращении и получил широкое признание. К другим достоинствам языка можно отнести высокую степень структурированности программ и возможность обращения к системным вызовам (аналогично языку С). Удобнее всего использовать Паскаль для небольших программ, что естественно объясняется его первоначальным предназначением. Недостатками языка являются, например, отсутствие возможности инициализации переменных и недостаточные средства для работы с файлами.

1.2.5. Ассемблер

Ассемблер наиболее близок к аппаратуре и является машинно-зависимым и не всегда переносимым. Необходимость программирования на ассемблере возникает, как правило, при невозможности воспользоваться языками высокого уровня, например при программировании отдельных веток низкоуровневых функций ядра ОС и драйверов устройств.

1.2.6. Специализированные языки программирования

Кроме описанных языков программирования, в ОС UNIX имеются специализированные языки, например, интерпретаторы – shell, awk, lex, yacc, python, perl, ... В этой группе стоит обратить внимание на shell, awk и python. Первые два являются мощными средствами для создания и выполнения сценариев в терминальном или пакетном режиме. Python является мощным средством для визуального и сетевого программирования.

2. Язык С. Введение. Базовые элементы.

2.1. Понятие алгоритма и программы на языке С

Алгоритм – конечная последовательность действий, которую необходимо выполнить для достижения какого-либо результата. Если алгоритм выполняет человек его придумавший (разработчик), то специального оформления алгоритма не требуется. Если алгоритм передается другому человеку (пользователю), то требуется описание алгоритма на каком-либо языке понятном обоим. Если алгоритм предназначается для технической системы (устройство, компьютер, робот), то требуется его описание либо в машинных кодах, либо на специальном алгоритмическом языке.

Язык С (Си) – алгоритмический язык высокого уровня абстракции, принадлежащий к группе компилируемых языков программирования и предназначенный для написания алгоритмов для компьютера, то есть написания компьютерных программ.

Программа на языке С (Си) – файл, содержащий последовательность текстовых строк, описывающих некоторый алгоритм на языке С (Си).

2.2. Элементы языка С

Под элементами языка понимаются его базовые конструкции, используемые при написании программ. Элементами языка Си являются:

– алфавит (набор символов, которые допускает использовать стандарт языка и компилятор);

– константы (объекты, сохраняющие свое значение в процессе выполнения программы, в качестве значений выступают фиксированные значения базовых типов – символьных, целых, вещественных и логических);

– переменные (объекты, изменяющие свое значение в процессе выполнения программы);

– идентификаторы (наименования ячеек памяти, в которых хранятся значения переменных, а также имена функций);

– ключевые слова (зарезервированные символьные конструкции для обозначения операторов или их частей);

– комментарии (конструкции, допускающие написание комментария, в том числе с помощью символов, не входящих в алфавит языка С, игнорируемые компилятором);

– объявления (описание объектов или типов объектов, которые используются в программе или функции);

– операции (элементарные действия над переменными);

– операторы (сложные действия над группами переменных);

– блоки (фрагменты кода, заключенные в фигурные скобки);
– функции (именованные блоки программы, имеющие как правило входные и выходные параметры);
– специальные определения (указания компилятору для преобразования текста программы перед компиляцией);
– макросы (указания компилятору для преобразования текста программы перед компиляцией, оформленный по аналогии с функциями);
– законченные программы (полные тексты программ, преобразующиеся компилятором в исполняемый машинный код).

2.3. Общая структура программы

```
Специальные определения вида:
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// Специальные определения:
#define Pi 3.1415926535897932384
#define myint long long int

// Макросы:
#define TRANSLATE_BOX_COORDS_01(bx0, by0, bz0, bx1, by1, bz1) \
{ \
  bx1 = (bx0) - 1; \
  by1 = (by0) - 1; \
  bz1 = (bz0) - 1; \
}

// Объявления новых типов:
typedef struct tag_int_Ar_1D_t {
  int len;
  int max;
  int *data;
} int_Ar_1D_t;

// Объявления глобальных переменных:
double a = 3.14;
double b = 1.57;

// Объявления функций:
double f1(double x);
double f2(double x);

// Определения функций:
double f1(double x)
{
 return 4.0/(1.0+a*x+x*x);
}

// Обязательная главная функция:
int main()
{
  // Объявления переменных
  int i, n=100;
  myint A; // подставляется long long int A;
  float C = Pi; // подставляется float C = 3.1415926535897932384;
  double a, b = 1.5, h, s;
  float *Adr1;
  double *Adr2;
  double aa[10], bb[10];
  double *cc, *dd;

  int_Ar_1D_t MyStr;

  MyStr.len = 0;
  MyStr.max = 0;
  MyStr.data = NULL;
```

```
  MyStr.data = (int *)(malloc(sizeof(int)*50));
  MyStr.max = 50;

  for(i=0; i<25; i++) MyStr.data[i] = i;
  MyStr.len = 25;

  Adr1 = &C;
  Adr2 = &b;

  // Операторы:
  A = (myint) C;
  printf("A=%ld\n",A);

  a = f1(b);
  b = f2(a);
  printf("a=%lf b=%lf\n",a,b);

  s = 0;
  for (i=0; i<10; i++) {
    aa[i] = 0.1*i + 0.05; /* a_i = 0.1*i+0.05 */
    bb[i] = f1(aa[i]);    /* b_i = 4/(1+a_i*a_i) */
    s = s + bb[i];        /* s = sum_{i=0}^{i=9} b_i */
  }
  printf("s=%16.14lf Pi=%16.14lf\n",s,Pi);

  n = 10000;

// Выделение дополнительной памяти:
  cc = (double*)(malloc(sizeof(double)*n));
  dd = (double*)(malloc(sizeof(double)*n));

  h = 1.0 / n;
  s = 0;
  for (i=0; i<n; i++) {
    cc[i] = h*i + 0.5*h;  /* c_i = 0.1*i+0.05 */
    dd[i] = f1(cc[i]);    /* d_i = 4/(1+c_i*c_i) */
    s = s + dd[i];        /* s = sum_{i=0}^{i=n-1} d_i */
  }
  printf("s=%16.14lf Pi=%16.14lf\n",s,Pi);

  return 0;
}

// Определения функций:
double f2(double x)
{
 return 4.0/(1.0+b*x*x*x);
}
```

2.4. Типы данных.

2.4.1. Базовые типы данных

Базовые типы данных – целые числа, вещественные числа, символы, логические величины, специальные.

Целые типы	Вещественные типы	Символьные типы	Логический тип	Специальные
int **unsigned int** **short int** **unsigned short int** **long int** **unsigned long int** **long long int** **unsigned long long int**	**float** **double** **long double**	**char** **unsigned char**	**bool**	**void** **enum** **int*** **float*** **double*** **char*** **bool***

2.4.2. Производные типы данных

Производные типы данных – массивы, структуры, объединения.

Массив – именованный набор данных, содержащий элементы одного из базовых типов (вектор, матрица, тензор).

Примеры массивов:

```
// Объявление:
char str[10]; // строка из 10 символов
float V[20];     // вектор
double M[3][5];   // матрица 3x5
double T[3][3][3]; // тензор

// Использование:
str[5] = 'A';
V[15] = V[0] + 3.14;
M[2][1] = sqrt(3.0);
T[0][0][0] = cos(Pi);
```

Структура – именованный набор данных, содержащий элементы разных типов.

Пример структуры:

```
// Объявление:
typedef struct tag_MyStruct_t {
  int len;
  double data[10];
} MyStruct_t;
MyStruct_t MySt;

// Использование:
MySt.len = 1;
MySt.data[0] = 1.2345;
```

Объединение – именованный набор данных, содержащий в одних и тех же ячейках памяти вариативно элементы разных типов.

Пример объединения:

```
// Объявление:
typedef union MyUnion_t {
  char      datac[80];
  short int datas[40];
  int       datai[20];
  double    datar[10];
} MyUnion_t;
MyUnion_t MyUn;

// Использование:
MyUN.datar[0] = 3.14;
MyUN.datai[2] = 123;
MyUN.datas[6] = 456;
MyUN.datac[14] = 'Z';
```

Литература

1. С.О. Бочков, Д.М. Субботин. Язык программирования Си для персонального компьютера. – М.: СП "Диалог", Радио и связь, 1990.
2. Б. Керниган, Д. Ритчи. Язык программирования Си. – Санкт-Петербург: Невский диалект, 2000.
3. Б. Страуструп. Язык программирования C++. – СПб.: Невский диалект, спец. изд., 2008.
4. Г. Буч. Объектно-ориентированный анализ и проектирование с примерами приложений на C++. СПб.: Невский диалект, 2-е изд., 2001.
5. С. Мейерс. Эффективное использование C++. – М.: ДМК, 2000.
6. С. Мейерс. Наиболее эффективное использование C++. – М.: ДМК, 2000.
7. С. Мейерс С. Эффективное использование STL. Библиотека программиста. СПб: Питер, 2002.
8. Саттер Г. Решение сложных задач на C++. – М.: Вильямс, 2003.
9. Саттер Г. Новые сложные задачи на C++. – М.: Вильямс, 2005.
10. Брайн Керниган, Роб Пайк. UNIX. Программное окружение. – М.: Символ-Плюс, 2003.
11. Майкл К. Джонсон, Эрик В. Троан. Разработка приложений в среде Linux. Программирование для Linux. – М.: Вильямс, 2007.
12. Иванов Н.Н. Программирование в Linux. Самоучитель. 2007.
13. Н. Мэтью, Р. Стоунс. Основы программирования в Linux (Четвертое издание). 2009.
14. Мартин Р. Чистый код. Создание, анализ и рефакторинг. 2010.
15. Шлее М. Qt4.5. Профессиональное программирование на C++. 2010.
16. Цилюрик О. Инструменты в Linux для программистов из Windows. 2011.
17. Прохоренок Н.А. - PyQt. Создание оконных приложений на Python3. 2011.