# 1. Введение в системное программирование

Разделы:

- Системное , промежуточное и прикладное ПО
- Классификация системных программ
- Модульная разработка программ
- Обязанности системного программиста
- Особенности процесса дальнейшего изучения системного программирования
- Автоматизация модульного тестирования на примере CUnit

# Классы программного обеспечения

1. Системное.
2. Промежуточное (*middleware*).
3. Прикладное.

- Объекты нашего изучения относятся пп. 1 и 2.

# Системные программы

- Ассемблеры
- Загрузчики
- Компоновщики (*linkers*)
- Макропроцессоры
- Текстовые редакторы
- Трансляторы
- Операционные системы
- Средства отладки и профилирования программ
- (Иногда) системы управления версиями исходного кода программ
- (Иногда) СУБД

# Модульная организация

- При выборе модульной структуры должны учитываться следующие основные соображения:
  - Функциональность
  - Несвязность
  - Специфицируемость

- Исходные модули
- Объектные модули
- Загрузочные модули

# Системные программы

- Системы, обеспечивающие подготовку программ в среде, отличной от целевой - кросс-системы

- В кросс-системе может выполняться вся подготовка программы или ее отдельные этапы:
  - Макрообработка и трансляция
  - Редактирование связей
  - Отладка

# Обязанности системных программистов

- Из профессионального стандарта:
  - «создание модулей системного программного обеспечения»
  - «создание встраиваемого программного обеспечения и драйверов устройств»
  - «создание системных утилит и компонент операционных систем»
  - «создание средств разработки программ»
  - …

# GNU Toolchain

- Рабочая среда - Debian Linux as Guest OS under Oracle VirtualBox (или альтернативные варианты)
- Язык командной оболочки – *bash* (возможны альтернативы)
- Toolchain:
  - заголовочные файлы ядра Linux
  - *binutils* (компоновщик *ld*, ассемблер *as* и другие программы)
  - *GNU Compiler Collection* (он же *gcc*) – набор компиляторов
  - стандартная библиотека языка *C* – *GNU/libc* или другая (например *uClibc* или *dietlibc*)
  - *GNU make*
  - *autotools* (*autoconf*,...)

# See also

- Oracle VM VirtualBox - https://www.virtualbox.org/
- Debian – Универсальная Операционная Система - http://www.debian.org/
- Операционная система GNU - http://www.gnu.org/
- GNU toolchain - http://ru.wikipedia.org/wiki/GNU_toolchain
- Кузнецов, А. С. Операционные системы и системное программное обеспечение: учеб. пособие / А. С. Кузнецов, И. В. Ковалев. – Красноярск: ИПЦ КГТУ, 2005. – 302 с.
- Курячий, Г.В. Операционная система Linux: Курс лекций. Учебное пособие / Г.В.Курячий, К.А. Маслинский. М.: ДМК-Пресс, 2010. – 348 с. – Доступ через электронную библиотечную систему Издательства «Лань» - http://e.lanbook.com/
- Профессиональный стандарт. Системный программист - http://www.apkit.ru/committees/education/PS_SP_4.0.pdf

# Автоматическое тестирование

- Известные разновидности автоматического тестирования:
  - **Системное тестирование** – это тестирование «черного ящика» с помощью сценариев, близких к сценариям использования системы в реальной работе
  - **Модульное тестирование** (*unit testing*) – это тестирование «белого ящика», подразумевающего проверку внутренних функций и методов, непосредственно реализующих систему
    - Реализуется в виде набора тестов, описывающих сценарии вызова функций и сравнивающих фактические результаты работы с ожидаемыми при помощи набора **проверок** (*assert*)
- Широко распространенная практика разработки – TDD
- В основе лежит реализация и/или доработка функциональности программного обеспечения минимальными частями

# Автоматическое тестирование

- Преимущества:
  - Инкрементально-пополняемый набор тестов снижает риск внесения ошибки при модификации кода
  - Автоматическое тестирование позволяет уменьшить издержки по сравнению с «ручным» тестированием
  - Наличие набора тестов положительно влияет на психологическое состояние программистов, модифицирующих код
  - Идея «сначала тест, потом код» позволяет спроектировать программные интерфейсы новой функциональности на раннем этапе цикла разработки
  - Использование модульных тестов мотивирует к разумной декомпозиции кода и уменьшению связности между его частями

# Полуавтоматическое тестирование

- Для реализации модульного тестирования можно использовать функцию *assert*(*cond*)
- Она аварийно завершает выполнение программы при невыполнении условия *cond*, т.е. при его нулевом значении
- Выдает сообщение, подобное следующему:

```
<Название программы> <имя_файла> <номер_строки> <название
    функции> Assertion <выражение> failed.
Aborted
```

```
#include <assert.h>
int main()
{
  int a = 5;
  assert(a < 0);
  int b = -5;
  assert(b < 0);
  return 0;
}
```

# Автоматическое тестирование

- В CUnit тест является функцией языка C, не имеющей параметров и возвращаемого значения
- Каждый тест является листом иерархии:
  - **Реестр тестов** (*test registry*)
    - **Набор тестов** (*test suite*)
      - **Тест** (*test*)
- Для тестов, входящих в набор, могут быть заданы функция **установки** и функция **очистки** контекста
- Функция установки вызывается единовременно до запуска всех тестов набора, функция очистки также один раз после работы тестов

# Реестры CUnit

- Функции:
  - *CU_initialize_registry*
  - *CU_cleanup_registry*
  - *CU_create_new_registry*
  - *CU_get_registry*
  - *CU_set_registry*
  - *CU_destroy_existing_registry*

# Наборы CUnit

- Функции:

```
CU_pSuite CU_add_suite
   (const char* strName,
   CU_InitializeFunc pInit,
   CU_CleanupFunc pClean
   );

CU_ErrorCode CU_get_error(void);

const char* CU_get_error_msg(void);
```

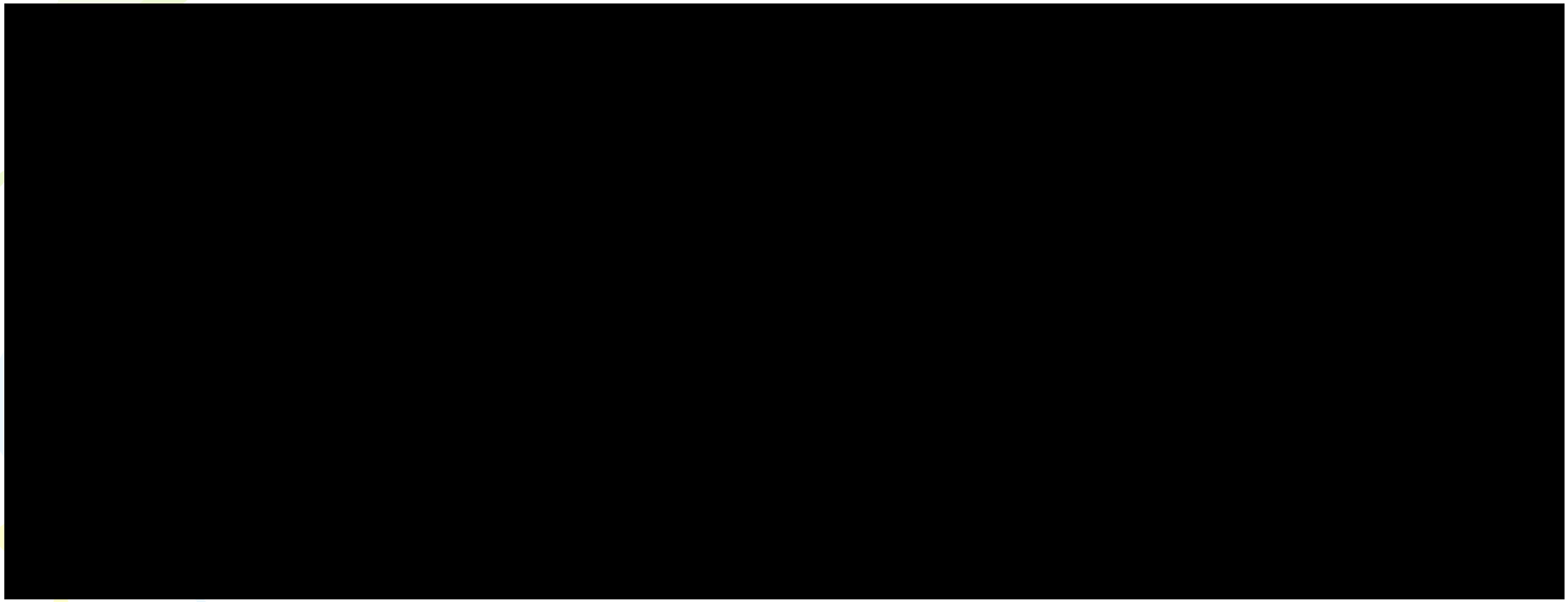
# Тесты CUnit

- Функции и макросы:

```
CU_pTest CU_add_test
   (CU_pSuite pSuite,
   const char* strName,
   CU_TestFunc pTestFunc
   );
typedef void (*CU_TestFunc)(void);


#define CU_ADD_TEST(suite, test)
   (CU_add_test(suite, #test,
   (CU_TestFunc)test))
```

# Проверки



# Запуск тестов

- Функции и макросы:

```
CU_ErrorCode CU_basic_run_tests(void);

CU_ErrorCode CU_basic_run_suite(CU_pSuite
   pSuite);

CU_ErrorCode CU_basic_run_test(CU_pSuite pSuite,
   CU_pTest pTest);

void CU_console_run_tests(void);

const CU_pRunSummary CU_get_run_summary(void);

const CU_pFailureRecord
   CU_get_failure_list(void);
```

# Запуск тестов

```
typedef struct CU_RunSummary
{
  unsigned int nSuitesRun;
  unsigned int nSuitesFailed;
  unsigned int nTestsRun;
  unsigned int nTestsFailed;
  unsigned int nAsserts;
  unsigned int nAssertsFailed;
  unsigned int nFailureRecords;
} CU_RunSummary;
typedef CU_Runsummary* CU_pRunSummary;
const CU_FailureRecord
   CU_get_failure_list(void);
```

# Запуск тестов

```
typedef struct CU_FailureRecord
{
  unsigned int uiLineNumber;
  char* strFileName;
  char* strCondition;
  CU_pTest pTest;
  CU_pSuite pSuite;
  struct CU_FailureRecord* pNext;
  struct CU_FailureRecord* pPrev;
} CU_FailureRecord;
typedef CU_FailureRecord* CU_pFailureRecord;
```

# Запуск тестов

```
#include <stdlib.h>

struct _Set
{
    int* data;
    int size;
};
typedef struct _Set Set;

#define SUCCESS 1
#define FAIL 0

typedef int Result;
int contains(Set* set, int elem);
```

# Запуск тестов

```
Result insert(Set* set, int elem);

Result Remove(Set* set, int elem);

int _bsearch (int* array, int len, int elem, int* posBefore)
{
    int low = 0, high = len - 1;
    while(low <= high)
    {
        int mid = (low + high) / 2;
        if (elem == array[mid])
            return mid;
        else if (elem < array[mid])

            high = mid - 1;
        else
            low = mid + 1;
    }
    if(NULL != posBefore)
        *posBefore = high;
    return -1;
}
```

# Запуск тестов

```
int contains(Set* set, int elem)
{
    return indexOf(set, elem) != -1;
}
int indexOf(Set* set, int elem)
{
    return _bsearch(set->data, set->size - 1, elem, NULL);
}
Result insert(Set* set, int elem)
{
    int index, posBefore;
    index = _bsearch(set->data, set->size, elem, &posBefore);
    if (index != -1)
        return FAIL;
    set->data = realloc(set->data, sizeof(int) * (set->size + 1));
    memmove(set->data + posBefore + 2, set->data + posBefore + 1,
        sizeof(int) * (set->size - posBefore - 1));
    set->data[posBefore + 1] = elem;
    ++set->size;
    return SUCCESS;
}
```

# Запуск тестов

```
Result Remove(Set* set, int elem)
{
    int index, posBefore;
    index = _bsearch(set->data, set->size, elem, &posBefore);
    if (index == -1)
        return FAIL;
    set->data = realloc(set->data, sizeof(int) * (set->size - 1));
    memmove(set->data + index, set->data + index + 1,
            sizeof(int) * (set->size - index - 1));
    --set->size;
    return SUCCESS;
}
```

# Запуск тестов

```
void test1(void)
{
    Set s = {0};
    CU_ASSERT_EQUAL(s.size, 0);
    insert(&s, 3);
    CU_ASSERT_EQUAL(s.size, 1);
    insert(&s, 1);
    CU_ASSERT_EQUAL(s.size, 2);
    CU_ASSERT_EQUAL(s.data[0], 1);
    CU_ASSERT_EQUAL(s.data[1], 3);
}
void test2(void)
{
    Set s = {0};
    CU_ASSERT_EQUAL(insert(&s, 1), SUCCESS);
    CU_ASSERT_EQUAL(insert(&s, 1), FAIL);
}
void test3(void)
{
    Set s = {0}; insert(&s, 1);
    CU_ASSERT_EQUAL(Remove(&s, 2), FAIL);
    CU_ASSERT_EQUAL(Remove(&s, 1), SUCCESS);
}
```

# Запуск тестов

```
void test4()
{
    int array[] = {1, 3, 5, 11};
    int size = sizeof(array) / sizeof(int);
    int posBefore;

    CU_ASSERT_EQUAL(_bsearch(array, size, 4, &posBefore), -1);
    CU_ASSERT_EQUAL(posBefore, 1);
    CU_ASSERT_EQUAL(_bsearch(array, size, 19, &posBefore), -1);
    CU_ASSERT_EQUAL(posBefore, 3);
    CU_ASSERT_EQUAL(_bsearch(array, size, 5, &posBefore), 2);
}
```

# Запуск тестов

```
#include <CUnit/Basic.h>
int main()
{
    CU_pSuite suite;
    CU_initialize_registry();
    suite = CU_add_suite("main_suite", NULL, NULL);
    CU_ADD_TEST(suite, test1);
    CU_ADD_TEST(suite, test2);
    CU_ADD_TEST(suite, test3);
    CU_ADD_TEST(suite, test4);
    CU_basic_run_tests();

    CU_cleanup_registry();
    return CU_get_error();
}
```

# Запуск тестов

```
#include <CUnit/Console.h>
int main()
{
    CU_pSuite suite;
    CU_initialize_registry();
    suite = CU_add_suite("main_suite", NULL, NULL);
    CU_ADD_TEST(suite, test1);
    CU_ADD_TEST(suite, test2);
    CU_ADD_TEST(suite, test3);
    CU_ADD_TEST(suite, test4);
    CU_console_run_tests();
    CU_cleanup_registry();
    return 0;
}
```

# Некоторые библиотеки для модульного тестирования

- CUnit - A Unit Testing Framework for C - http://cunit.sourceforge.net/
- CuTest: C Unit Testing Framework - http://cutest.sourceforge.net/
- Check: a unit test framework for C - http://check.sourceforge.net/
- JUnit - A programmer-oriented testing framework for Java - http://junit.org/
- CppUnit - C++ port of JUnit - http://apps.sourceforge.net/mediawiki/cppunit/
- List of unit testing frameworks - https://en.wikipedia.org/wiki/List_of_unit_testing_frameworks
- googletest - Google C++ Testing Framework - Google Project Hosting - http://code.google.com/p/googletest/