Linux Kernel

1. Каким образом пользовательское приложение может обратиться к ядру?

   Ответ: для взаимодействия пользовательского приложения я ядра существует механизм системных вызовов

2. Каковы, по Вашему мнению, основные функции ОС общего назначения, без выполнения которых, она не может называться таковой?

   Ответ:

   - Возможность взаимодействия с системой пользователя (интерактивность)
   - Единообразная система приложений
   - Позволяет проводить эффективную разработку и тестирование ПО

3. Что вы знаете об отличительных особенностях программирования ядерных модулей?

   Ответ:

   - Требуется больше внимательности при разработке и написании кода
   - Нет стандартной защиты
   - Нет стандартной библиотеки

4. Перечислите, какие примитивы синхронизации Вам знакомы?

   Ответ: мьютексы, спинлоки, семафоры

5. Что Вы знаете о дедлоках и о том, почему они возникают?

   Ответ: дедлок — ситуация, в которой разные процессы запращивают ресурсы системы, занятые друг другом. При этом ни один из процессов не может продолжить работу, пока другой процесс не освободит требуемые ресурсы, т. е. процессы блокируют взаимно блокируют свою работу. Процессы не могут продолжиться, что бы ни делал планировщик задач

6. Может ли иметь смысл реализовывать многопоточное приложение для исполнения в среде с единственным ядром процессора?

   Ответ: если приложение будет исполняться на одном ядре (без виртуализации), то смысла реализовывать многопоточное приложение нет

7. Имеет ли смысл использовать спинлоки при исполнении в однопроцессорной среде?

   Ответ: Использование спинлока (в чистом виде) в однопроцессорной среде неэффективно, т.к. ожидание освобождения спинлока приводит к активному ожиданию в цикле → впустую расходуются вычислительные ресурсы.

8. Дан код:

```
u32 f(u32 a, u32 b)
{
    b = DIV_ROUND_UP(a, b);
    b = DIV_ROIND_UP(a, b);
    return b;
}
/* <...> */
u32 x = f(y, z);
```

   Каково будет, для $0 < y, z <= 10^9$, соотношение между x и z в результате выполнения кода? Определение макроса DIV_ROUND_UP посмотрите в коде ядра Linux.

   - z<=x
   - z==x
   - z>=x
   - В общем случае никакого соотношения не будет

   Ответ: в общем случае никакого соотношения не будет, т.к. при некоторых значениях y и z программа аварийно завершится. Макрос DIV_ROUND_UP определеятся как:

```
...
#define __KERNEL_DIV_ROUND_UP(n, d) (((n) + (d) - 1) / (d))
...
#define DIV_ROUND_UP __KERNEL_DIV_ROUND_UP
...
```

Данный макрос не проверяет делитель на равенство нулю, не следит за переполнением. Из-за отсутствия данных проверок при некоторых входных данных результат суммы y и z становится больше максимального значения uint32 и происходит переполнение. В результате переполнения сумма может оказатся меньше делителя и результатом действия макроса станет 0. При последующем использовании нуля как делителя произойдет ошибка деления на 0 и программа аварийно завершится.

9. Пусть дана функция:

```
struct my_struct
{
        char tag;
        char const *value;
};

int f(void)
{
        struct my_struct a, b;
        char const *value = "value";

        a.tag = 'T';
        a.value = value;

        b.tag = 'T';
        b.value = value;

        if (memcmp(&a, &b, sizeof(struct my_struct)))
                return 1;
        return 0;
        }
```

Какое значение вернет эта функция? Почему?

Ответ: функция **в большинстве** случаев вернет значение 1. Результат будет таким, потому что memcmp(...) сравнивает участки памяти, а не поля структуры. Размер сравниваемых участков памяти зависит от размера структуры, а размер структуры – от компилятора. Компилятор может оптимизировать время обращения к каждому элементу: он располагает данные так, чтобы любое поле можно было прочитать за минимальное количество канальных циклов.

Так в результате компиляции в большинстве случаев размер будет равняться 8 (или 16 - в зависимости от разрядности) байт, часть из которых заполнена “мусором”. Вероятность совпадения “мусора” крайне низкая, поэтому функция в подавляющем большинстве случаев вернет 1.

10. Даны две функции, обрабатывающие двумерный массив чисел:

```
uint32_t A(uint32_t **a, size_t width, size_t h)
{
    uint32_t sum = 0;
    for (size_t i = 0;  i < width;  i++)
     for (size_t j = 0;  j < h;  j++)
       sum += a[j][i];
    return sum;
}

uint32_t B(uint32_t **a, size_t w, size_t height)
{
    uint32_t sum = 0;
    for (size_t j = 0;  j < height;  j++)
     for (size_t i = 0;  i < w;  i++)
       sum += a[j][i];
    return sum;
}
```

Будет ли разница в их производительности? Если да, то какая из функций будет быстрее? Ответ обоснуйте.

Ответ: Разницы в производительности не будет. Мат. Модели вычисления отличаются лишь порядком обхода матрицы, что не влияет на производетльность. Реализация же после компиляции будет

идентичной, т.к. матрица в памяти располагается линейно и непрерывно, а индексы вычисляются по формуле: адрес_элемента_i_j = адрес_начала_массива + i*M*sizeof(T) + j*sizeof(T).

Т.е. сложность обхода не изменится.

User Space Python Development

Задание 1

Что и почему выведет код на Python:

```
x = 1
y = 1
print(x is y) # => ???

y = pow(10, 30, 10**30-1) # => 1
print(x, y, x is y) # => ???
```

Вывод:
True
1 1 False
В первом случае оператор is вернул True, потому что x и y ссылались на один и тот же объект. Во втором случае y стал ссылаться на другой объект. Их значения равны, но местанохождения в памяти – разные. Поэтому оператор is вернул False

Задание 2

Как и почему исполнится код на Python:

```
from contextlib import contextmanager

@contextmanager
def nullresource(*args, **kwargs):
    try:
        yield None
    except:
        pass

class NullResource():
    def __enter__(self):
        return None
    def __exit__(self, *args,
    **kwargs):
        return True

with nullresource() as r:
    r.some_method()

with NullResource() as r:
    r.some_method()
```

Задание 3

Как и почему завершится код на Python:

```
import signal
import time

if not signal.SIGALRM:
    exit(0)

def handler(*args, **kwargs):
    raise InterruptedError()

signal.signal(signal.SIGALRM, handler)

signal.alarm(5)
time.sleep(4.991)
```

signal.signal функция биндит обработчик, когда вызывается соответствующий сигнал. В приведенном коде это SIGALRM.

При использовании функции signal.alarm устанавливается таймер для отправки сигнала. Значение таймера - 5

Затем исполняется time.sleep, чтобы процесс повис. time.sleep имеет значение 4.991, что меньше 5.

Когда time.sleep исполнится, python-процесс будет уничтоже и сигнал не будет отправлен.

Если установить время sleep больше, чем у таймера, будет вызван обработчик и выброшено исключение

QA Engineer

Задание 1

Вам необходимо написать тест-кейсы для прлиожения, которое получает на вход целые положительные числа от нуля до 10 миллионов, и отадет на выход числетельные русского языка.

Ответ: так как внутренняя реализация, включая методы, ЯП исходных кодов и т.п., неизвестны, то пришлось написать небольшую "обертку"для тест-кейсов. В "обертке"происходит обращение к приложению «blackbox_app». Приложение «blackbox_app» получает один параметр - число в десятичной системе счисление и выдает числительное русского языка.

Представленный код:

1. Формирует команду для запуска приложения blackbox_app на основе названия приложения и числа, переданного тест-кейсом.
2. Сохраняет результат работы приложения
3. Сравнивает полученную строку со строкой, переданной в тест-кейсе
4. При совпадении и отсутствии инверсии результата Или несовпадении и инверсии результата тест считается пройденным ("Test №i accepted"), иначе - проваленным ("Test №i failed");

```
#include<stdio.h>
#include<cmath>
#include<string.h>
#include<stdlib.h>
#include<ctype.h>
#define BLACKBOX_APP_LEN 14
#define DATA_BUFF_LEN 200
#define TESTS_NUMBR 16
class Test;
typedef int(Test::*ARR_PTR)(void);
class Test{
public:
  Test();
  ~Test();
  void test_all();
private:
  int test_base(const char *value,int value_len, const char *str, int len);
  int test_0();
  int test_1();
  int test_2();
  int test_3();
  int test_4();
  int test_5();
  int test_6();
  int test_7();
  int test_8();
  int test_9();
  int test_10();
  int test_11();
  int test_12();
  int test_13();
  int test_14();
  int test_15();
};
Test::Test(){}
Test::~Test(){}
int Test::test_base(const char *value, int value_len, const char* str, int len){

  char *data = (char*)calloc(DATA_BUFF_LEN,sizeof(char));
  char* exec_command = (char*)calloc(( BLACKBOX_APP_LEN + 1 + value_len + 1 ),sizeof(char));
  int res;
  sprintf(exec_command,"");
  exec_command = strcat(exec_command, "./blackbox_app ");
  exec_command = strcat(exec_command,value);
  exec_command[BLACKBOX_APP_LEN + 1 + value_len] = '\0';
  FILE*bb_app = popen(exec_command,"r");
  int lastchar = fread(data, 1, DATA_BUFF_LEN, bb_app);
  data[lastchar] = '\0';
  for(int data_cur = 0; data_cur < DATA_BUFF_LEN; data_cur++){
    data[data_cur] = tolower(data[data_cur]);
  }
```

```
52    res = strcmp(data,str);
53    pclose(bb_app);
54    free(data);
55    free(exec_command);
56    return res;
57  }
58  void Test::test_all(){
59    ARR_PTR f[] = {&Test::test_0,&Test::test_1,&Test::test_2,&Test::test_3,&Test::test_4,
60    &Test::test_5, &Test::test_6, &Test::test_7, &Test::test_8, &Test::test_9,&Test::test_10,
61    &Test::test_11,&Test::test_12,&Test::test_13,&Test::test_14,&Test::test_15,};
62    for(int i = 0; i<TESTS_NUMBR; i++){
63      if((this->*f[i])()==0){
64        printf("Test %d accepted\n",i);
65      }
66      else{
67        printf("Test %d failed\n",i);
68      }
69    }
70    return;
71  }
72  int Test::test_0(void){
73    return !test_base("1",1,"динf",4);
74  }
75  int Test::test_1(){
76    return test_base("10",2,"десять", 6);
77  }
78  int Test::test_2(){
79    return test_base("3",1,"три",3);
80  }
81  int Test::test_3(){
82    return test_base("2589",4,"две тысячи пятьсот восемьдесят девять",37);
83  }
84  int Test::test_4(){
85    return !test_base("18",2,"один восемь",11);
86  }
87  int Test::test_5(){
88    return test_base("333333",6, "триста тридцать три тысячи триста тридцать три",46);
89  }
90  int Test::test_6(){
91    return test_base("13",2,"тринадцать",10);
92  }
93  int Test::test_7(){
94    return !test_base("13",2, "тренадцать",10);
95  }
96  int Test::test_8(){
97    return test_base("9876543",7,"девять миллионов восемьсот семьдесят шесть тысяч пятьсот сорок три",65);
98  }
99  int Test::test_9(){
100   return test_base("1345678",7, "один миллион триста сорок пять тысяч шестьсот семьдесят восемь",62);
101 }
102 int Test::test_10(){
103   return test_base("781",3,"семьсот восемьдесят один",24);
104 }
105 int Test::test_11(){
106   return test_base("10000000",8,"десять миллионов",16);
107 }
108 int Test::test_12(){
109   return test_base("0",1,"нуль",4)||test_base("0",1,"ноль",4);
110 }
111 int Test::test_13(){
112   return test_base("14",2,"четырнадцать",12);
113 }
114 int Test::test_14(){
115   return !test_base("один",4,"1",1);
116 }
117 int Test::test_15(){
118   return test_base("76",2,"семьдесят шесть",15);
119 }
120 int main(int argc, char *argv[]){
121   Test test;
122   test.test_all();
123   return 0;
124 }
```

Задание 2

Имеется 6 жестких дисков Seagate ST3600057SS
Необходимо дать грубую оценку для RAID-массивов, построенных из данных дисков:

- Производительность потокового чтения и записи крупными блоками в МБ/s при объединении этих дисков в RAID-массив уровня 1+0;
- Производетельность случайного (random) чтения и записи блоками 8kB в IOps (Количество операций ввода-вывода в секунду) при объединении этих дисков в RAID-массив уровня 1+0;
- Полученный полезный объем при объединении данных дисков в RAID-массив 6-го уровня

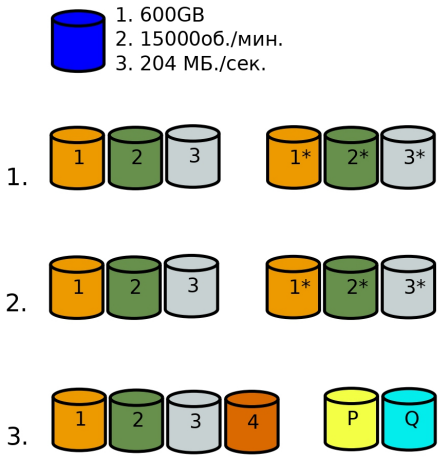

Рис. 1: Схемы разделения дисков в RAID-массивы

- При потоковом чтении временем на позиционирование считывающей головки можно пренебречь. При RAID-массиве уровня 1+0, диски располагаются так, как показано на Рис.1 в строке 1. Диске $1-3$ хранят информацию, диски 1^*-3^* хранят копию информации.

  Скорость потоковых чтения/записи при этом будет составлять три скорости чтения/записи одного диска, т.е. $\approx$ 612МБ/с.

- При случайном чтении/записи считывающая головка будет находить нужный сектор в среднем за $\frac{60}{7500}$ секунд, т.е. время нахождения требуемого участка диска $\approx$ 0.008с.. Время чтения/записи 8кБ:

$$\frac{8\text{кБ}}{204\text{МБ/с.}} = \frac{8\text{кБ}}{208896\text{кБ/с.}} \approx 3.8{\cdot}10^{-5}c.$$

  - сильно меньше времени позиционирования.

  Тогда суммарное время позиционирования и чтения/записи: $\approx$0.008с. $\rightarrow$ IOps = $\approx$ 125

- При создании RAID-массива уровня 6 (Рис. 1, строка 3) два диска следует отвести под контрольные суммы. Оставшиеся 4 диска будут хранить полезный объем:

$$600\text{GB}{\cdot}4 = 2400\text{GB}$$

или, если учитывать преамбулы, ECC-коды и межсекторные интервалы:

$$\approx 2400\text{GB} - 2400 \cdot 0.15 = 2040\text{GB}$$

Задание 3

Необходимо написать shell-скрипт, выполняющий следующие действия в операционной системе GNU/Linux:

- Проверяет, смонтировано ли удаленное блочное хранилище в точку /var/backup;
- Выполняет архивирование файлов в папке /home/somename/work и сохраняет архив в /var/backup с присвоением архиву имени, соответствующему текущей дате;

- Удаляет архивы давности $>$ 1 месяца

Ответ:

```
#!/bin/bash
is_mounted=mount | grep /var/backup
if [[ -z is_mounted ]]; then
  echo "Mounted"
else
  echo "Not mounted"
fi
cd ~/../home/somename/work/
tar -cf archive.tar.gz ./*

cur_date=$(date -I)
mv archive.tar.gz ./"${cur_date}".tar.gz
mv ./*.gz /var/backup/
find /var/backup -name "*.gz" -type f -mtime +30 -delete
```