武汉大学国际软件学院 2016_-2017 学年第一学期期末考试试卷

B 卷 【标准答案与改卷要点】

课程名称：《 系统程序设计 》
年级：_____专业：_____ 专业方向：_____ 层次：本科
姓名：_____学号：_____考分：_____

说明：1、答案一律书写在答题纸上，书写在试卷上或其他地方一律无效。
2、请准确规范书写姓名和学号，否则作废。

一、 名词解释（共 20 分）

1. 请用括号“（ ）”括出语句 “k = 1, 2;”的优先级顺序：_____。（1 分）
答案：(k = 1), 2;
唯一答案

2. C 语言中，通常一个符号可能有多重含义，要正确理解它们，需要上下文的帮助。比如，“*”用于声明时可以理解为_____，而用于数学表达时，则理解为_____。（各 1 分）
答案：指针操作符，或者，简介指针引用
乘法操作符
下划线部分为关键词

3. 在对堆内存的操作函数中，malloc() 与 calloc() 的主要差异是_____，而 sbrk(0)返回的是_____（各 2 分）当前堆的 brk 指针的地址。
答案：malloc()只对堆进行分配操作，而 calloc()除了堆堆进行分配外，还进行了初始化操作
当前的 brk 指针的内存地址

4. 在线程存储模型中，每个线程拥有独立的线程上下文，其组成包括：**线程 ID**、堆栈、堆栈指针、程序计数器、条件代码、**通用寄存器值**（各 1 分）
答案：如上，唯一答案

5. 在 32 位系统下，对于某个位于 k+1 层的 Cache 结构，假设其为直接映射型（Direct Mapped Cache），如果已知其容量为 1024bytes，Cache Line 块大小为 4bytes，那么，此层的 Cache 有_____个 sets，其对应的上层 k 的地址构成中，t、s、b 分别为 22 位长、_____长位，以及_____位长。（各 1 分）
答案：256、8、 2

改卷注意：只要后两个数值相加等于 10，就给 1 分，全对给 2 分

6. 在现代计算机内存结构体系中，通常运行时堆栈位于内存布局的_____地址位，而运行时堆与之相比位则于内存布局的_____地址位。（各 1 分）
答案：高　、　低

7. IEEE 规定的三种浮点数格式分别为：_____格式、_____格式，以及扩展双精度格式。（各 1 分）
答案：单精度　、双精度

8. 用英语填充以下对声明“char *(*c[10]) (int **p); ”解释的缺失部分：c is an array[0..9] of pointer to a function returning a pointer-to-char（各 1 分）
答案：写 array 也对
唯一答案 （如写中文，给一半分）

9. 假设在 linux 操作系统中，使用 gcc 作为编译器。对于标准 C 程序 hello.c 源码，经过预处理过程后，将转变为 modified source program 形式 hello.i，经过编译器过程后，将转化为 assembly program 形式 hello.s，经过汇编过程后，将转变为 relocatable object program 形式 hello.o，经过链接过程后，将转变为最终的 executable object program 形式 hello。（各 1 分）
答案：如上
注意：写中文给一半分

二、 简答题（每题 5 分，共 25 分）

1、解释未初始化的局部变量与已初始化局部变量在内存表现上有何相同点与不同点？

相同点：未初始化的局部变量和已初始化的局部变量均存储在堆栈的区域。（2.5 分）

不同点：未初始化的局部变量采用统一的标识（Windows 系统中），或随机化的数值（非 Windows 系统）【写任意一个都对】；已初始化的具备变量即为自身的初始化值。（2.5 分）

2、在标准 C 语言中，如果在文件 1 中声明: int p[ ]; 同时试图在文件 2 中引用: extern int *p; 那么运行时会出现异常。请问这种现象背后的原理是什么？

数组的声明直接分配的是符号的地址，int p[]中分配的就是 p 的地址。而 extern int *p 中，首分配的是 p 的地址，而引用时则是该地址指向的地址内容。因此，如果如题目中引用，后者引用的是声明 p 中指向的值作为地址，故出错。

如果写“不可更改的左值”作为原理也对。

3、现有值“-1”，预先以 short 型声明，那么（1）、在计算机内部，该值以 2 的补码形态表达，其二进制表达是怎样的？（2）、若将该值转换为无符号的整数（2 字节），那么其十进制表达是多少？【答题提示：请写出每步操作，并说明操作原理】

（1）（3 分）

-1 的 1 的补码：11111111，11111110

-1 在计算机中 2 的补码是：11111111，11111111【直接写也对】

（2）（2 分）

-1 的 2 的补码转换为无符号类型，则直接为 11111111，11111111

换算为 10 进制，为 65535

4、为什么说（1）、stack 机制是隐式内存分配，而 heap 机制是显式内存分配？（2）、但是二者又都是动态内存分配？

（1）、stack 内存分配是由系统控制的，且回收也是由系统处理的，故称为“隐式内存分配”。但是，heap 的内存分配是由应用程序发起的，内存的回收也是由应用程序控制的，故称为“显式内存分配”（2 分）

（2） 、但是，二者均根据实际的需要完成分配，因而是动态内存分配。（3 分）

5、现有如下代码，（1）、请写出其运行结果；（2）、这种现象在 C 语言中被称为什么？

```
Switch (2) {
    case 1: printf (“case 1 \n”);
    case 2: printf (“case 2 \n”);
    case 3: printf (“case 3 \n”);
    case 4: printf (“case 4 \n”);
    default: printf (“default \n”);
}
```

（1） 一运行结果为：

```
case 2
```

```
case 3
case 4
default
```

（答错扣 3 分）

（2） 一这种现象在 C 语言中称之为“Fall Through”现象。中文可翻译为“跌落”。（答错扣 2 分。改卷注意：写成中文，只要意思对也给分）

三、 实践题（每题 10 分，共 30 分）

1、 写出 Amdahl's Law 的形式化表达式，注明各个符号的含义，并阐明其对代码性能优化作业的指导意义。

$$T_{New} = (1-\alpha)T_{Old} + (\alpha T_{Old})/k$$

或

$$S = \frac{T_{Old}}{T_{New}} = \frac{1}{(1-\alpha)+\alpha/k}$$

（2 分）

其中：

T_{new} 代表优化后的应用程序运行时间

T_{old} 代表应用程序优化前的运行时间

a 代表应用程序优化的比例

k 代表优化部分性能提高的倍数

S 代表应用程序优化前后性能整体提升的比例（可以不写）

（共 4 分，缺 1 个扣 1 分）

指导意义：

1、所有的应用程序都存在优化的空间（可能性）

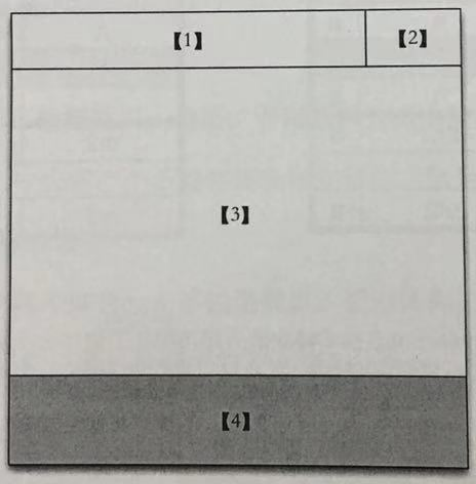
2、对应用程序的少量的优化可以换来性能的大幅度提升

3、如果应用程序仅仅只有少量的部分可以优化，那么，未优化部分的比例决定了应用程序整体的性能提升的幅度

4、知道如何优化是一个重要的技能；但是，知道何时应该停止优化同样是非常重要的

（以上各 1 分）

2、 请填充下图中堆（heap）分配时其块结构的部件名称（从【1】~【4】）。并说明（1）、图中底部灰色部分在什么情况下会填充值？（2）、为什么块的头部可以用来表达块的状态？如何表达？（3）、为什么函数 free(*p)不需要指定释放的内存堆大小？



【1】:块容量；【2】:块状态；【3】:有效载荷（payload）；【4】:结构补齐（或填充）（每个 1 分，共 4 分）

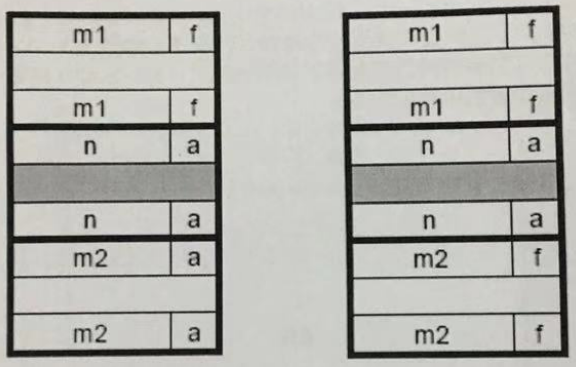
（1）、底部灰色只有在有效载荷不是 8 的倍数时才会填充（2 分）

（2）、块的头部采用 32 比特表示块的大小，但是由于块的大小总是 8 的倍数，故最后 3 为必为 000.因此，可以通过将最后一位取 1 表

示已分配，0 表示未分配。但是计算块的大小是仍然需要将最后 3 位 0 追加到前边的 29 位之后。（2 分）

（3）、由于指针 p 指向的地址内容即为 32 位的整数，该整数表示的就是该块的大小，故 free 函数不需要指定销毁的大小，只需要地址即可。（2 分）

3、 为什么说 Boundary Tag 技术可以有效提高 coalescing 的效率？如下图所示，请分别画出合并后的块的状态，并标明合并后的块的大小、状态。



（a） （b）

（1）、由于 Boundary Tag 技术将块的头尾均标注了块的大小和状态，因此既可以正向遍历，也可以反向遍历，而且可以通过观察相邻的块的状态决定是否 coalescing，故可以提高 coalescing 的效率。（5 分）

（2）对于 a 而言：如果中间的块改为释放状态，那么其上部的块可以与它合并，合并之后的块的大小为：m1+n，状态为 f。（2.5 分）

对于 b 而言，如果中间的块改为 free 状态，那么其上、下的块均可以与它合并。合并之后的块的大小为 m1+m2+n，状态为 f。（2.5 分）

如果认为当前状态不可合并，也对。

四、 综合题（共 25 分）

1、阅读如下代码，回答：

（1）、该代码的功能是什么？（2 分）

（2）、该代码使用的逻辑操作符的英文名称是什么？（2 分）

（3）、若将参数类型转换为 float，该代码还正确吗？理由是什么？（2 分）
（4）、若要保障该代码的安全调用，需设定哪些限制条件？（至少正确回答 2 个）（4 分）

```
1  void swap ( int *x, int *y)
2  {
3          *x = *x ^ *y;
4          *y = *x ^ *y;
5          *x = *x ^ *y;
6  }
```

（1）、该代码对内存中 x、y 指向地址中的内容进行了互换

扣分原则：划线的未表达各扣 2 分

（2）、exclusive OR （2 分）

（3）、不准确。因为位操作只能在整数之间进行。（2 分）

（4）、1）、x y 不能为 NULL

2）、x y 不能指向同一地址，否则返回的两个结果均为 0

3）、x y 中任何一个不能存储在寄存器中，因为无法取得寄存器的地址

4）、x y 其中如何一个不能是数组，否则同样错误

（以上 4 点，只要回答任意两个即可，每个 2 分）

2、现有如下代码。为简单起见，假设采用直接映射型缓存结构，缓存块的大小为 16bytes（即可以存放 4 个浮点数），而且缓存有两个 set 构成，即共有 32bytes 的缓存空间。假设变量 i 的存储在 register 中：

（1） 填写表格中的地址偏移量（从 0 开始）、缓存分组；（4 分）
（2） 逐一标明 a[]、b[]命中（以符号 h 表示）和不命中（以符号 m 表示）的状态；（4 分）
（3） 请改写该段代码，提高该段代码的内存命中率（3 分）
（4） 计算改写前、后的该段代码的内存命中率。（4 分）

```
1  int a[8], b[8];
2  int  main ( )
3  {
4  int i, sum;
5  for (i = 0; i < 8; i++)
6  sum += b[i] * a[i];
7  return sum;
8  }
```

(1)、

单元	地址偏移量	缓存分组	单元	地址偏移量	缓存分组
a[0]	0	0	b[0]	32	0
a[1]	4	0	b[1]	36	0
a[2]	8	0	b[2]	40	0
a[3]	12	0	b[3]	44	0
a[4]	16	1	b[4]	48	1
a[5]	20	1	b[5]	52	1
a[6]	24	1	b[6]	56	1
a[7]	28	1	b[7]	60	1

(2)、

单元	命中状态	单元	命中状态
a[0]	m	b[0]	m
a[1]	m	b[1]	m
a[2]	m	b[2]	m
a[3]	m	b[3]	m
a[4]	m	b[4]	m
a[5]	m	b[5]	m
a[6]	m	b[6]	m
a[7]	m	b[7]	m

(3)、

```
int a[12], b[8];  //只有这种解决方案
void   main ( )
{
   int i;
   for (i = 0; i < 8; i++)
   sum += b[i]*a[i];
   return sum;
}
```

(4)、改写前的内存命中率为：0 / 16 = 0%

改写后的内存命中状态为：（无下表的扣2分）

单元	命中状态	单元	命中状态
a[0]	m	b[0]	m
a[1]	h	b[1]	h
a[2]	h	b[2]	h
a[3]	h	b[3]	h
a[4]	m	b[4]	m
a[5]	h	b[5]	h
a[6]	h	b[6]	h
a[7]	h	b[7]	h
a[8]			
a[9]			
a[10]			
a[11]			

改写后的内存命中率为：

12/16 = 75%