**2020年Linux 操作系统分析考试题**

1. 单项选择题（2×5=10分）

1、字符设备文件类型的标志是( B )。  
 A. p B. c C. s D. l  
2、下面关于i节点描述错误的是( A ) 。（inode是一种数据结构，vfs中描述文件的相关参数）

A． i节点和文件是一一对应的 B． i节点能描述文件占用的块数

C． i节点描述了文件大小和指向数据块的指针 D． 通过i节点实现文件的逻辑结构和物理结构的转换

3、下列关于链接描述，错误的是 ( B ) 。

A 硬链接就是让链接文件的i节点号指向被链接文件的i节点 B 硬链接和符号连接都是产生一个新的i节点

C 链接分为硬链接和符号链接 D 硬连接不能链接目录文件

4、不需要编译内核的情况是 ( D ) 。

A 删除系统不用的设备驱动程序时 B 升级内核时 C 添加新硬件时 D 将网卡激活

5、在日常管理中，通常CPU会影响系统性能的情况是： ( A ) 。

A CPU已满负荷地运转 B CPU的运行效率为30%

C CPU的运行效率为50% D CPU的运行效率为80%

1. 填空题 （2×10=20分）

1、 在Linux系统中，以 文件 方式访问设备 。  
2、 Linux文件系统中每个文件用 索引节点 来标识。  
3、 安装Linux系统对硬盘分区时，必须有两种分区类型： 根分区 和 交换分区 。

4、在超级用户下显示Linux系统中正在运行的全部进程，应使用的命令及参数是 ps -aux 。

5、将前一个命令的标准输出作为后一个命令的标准输入，称之为 管道 。

6、在Linux系统下，第二个IDE通道的硬盘（从盘）被标识为 hdb 。

7、 当系统管理员需升级内核版本和改变系统硬件配置时，应 重新编译内核 。

8、在i节点表中的磁盘地址表中，若一个文件的长度是从磁盘地址表的第1块到第11块，则该文件共占有 266 块号。

9、linux的内存管理采取的是 分页存取 机制.

三、问答题（6×5=30分）

1、在打开文件的过程中，i节点从磁盘中被读出，然后放入内存中的i节点表里。这个表中有些域在磁盘中没有。其中一个是计数器，用来记录i节点已经被打开的次数。为什么需要这个域？

答：当文件关闭时，内存中i节点的计数器递减。如果大于零，则无法从表中删除i节点，因为该文件在某些进程中仍然处于打开状态。只有当计数器变为0时才能删除i节点。没有这个域，系统将不知道何时从表中删除i节点。每次打开文件时，单独复制一个i节点将不起作用，因为在一个副本中所做的更改在其他副本中不可见。

2、在使用线程的系统中，若使用用户级线程，是每个线程一个堆栈还是每个进程一个堆栈？如果使用内核级线程情况又如何呢？请给予解释。

答：每个线程都是自己调用例程，因此它必须有其自己的堆栈以保存局部变量、返回地址等等。这一点用户级线程和内核级线程是一样的。

3、假设一个机器有48位的虚拟地址和32位的物理地址。

a)假设页面大小是4KB，如果只有一级页表，那么在页表里有多少页表项？ 请解释。

b)假设同一系统有32个快表表项，并且假设一个程序的指令正好能放入一个页，并且该程序顺序地从有数千个页的数组中读取长整型元素。在这种情况下快表的效果如何？

答：a)避免把全部页表保留在内存中。

b)2^38/2^14 = 2^24，故页面长度为2^24，需要24位偏移量，而二级页表的表项为4字节，故PT2 = 2 ，所以 PT1 = 38 - 2 - 24 = 12，因此，对第一级页表域分配12位，对第二级页表分配域2位。

4、什么是僵死进程，如何产生的？

答：僵尸进程是指一个已经终止、但是其父进程尚未对其进行善后处理获取终止进程的有关信息的进程，这个进程被称为“僵尸进程”。

产生方式：一个进程结束了，但是他的父进程没有等待(调用wait / waitpid)他， 那么他将变成一个僵尸进程。

5、 Linux中的段是如何定义的？为什么这样定义？

答：linux创建4个段描述符：特权级为0的代码段和数据段、特权级为3的代码段和数据段。段的基地址全部为0x00000000；段的上限全部为0xffff；段长单位为4KB；四个段的方位都为32位指令；四个段都在内存。

这样定义段是为了实现一个纯的分页，并且分页机制会提供我们需要的保护。

6、Linux中创建一个新进程时，PCB存放在何处？其内存是如何分配的？

答:创建新进程时，pcb存放在内核栈中，pcb是存储器上的一组数据结构，需要的资源都分配在pcb上。

四、**程序题（10+10+20=40分）**

**1、请分析以下代码，说明内存泄漏的原因。代码如下：**

void MyFunction(int nSize， bool mark)

{

char\* p= new char[nSize];

if( mark &&!GetStringFrom(p，nSize) ){

MessageBox("Error");

return;

}

//using the string pointed by p;

delete p;

答：当if判定为真时,函数MyFunction结束而指针指向的内存却没有被释放,此时便出现了内存泄漏。在程序段入口处分配内存,在出口处释放内存,但是函数可以在任何地方退出,所以一旦有某个出口处没有释放应该释放的内存,就很容易发生内存泄漏。

**2、阅读程序并指出其中的错误和错误原因。**

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

char \*upcase(char \*string, int length);

int main(void)

{

char \*buf, \*newstr;

char oldstr1[ ]={“abcdefg”};

char oldstr2[ ]={“xyz”};

int counter;

buf=malloc(15);

strcpy(buf, ”CHANGE STRINGS.”);

fprintf(stdout, “%s\n”,buf);

free(buf);

for(counter=0;counter<sizeof(oldstr1); counter++)

{

putchar(oldstr1[counter]);

putchar(oldstr2[counter]);

}

newstr=upcase(oldstr1,sizeof(oldstr1));

fprintf(stdout, “NEW STRING 1:%s.\n”,newstr);

newstr=upcase(oldstr2,sizeof(oldstr2));

fpritf(stdout,”NEW STRING 2:%s.\n”,newstr);

free(newstr);

strcpy(buf, “THE END.”);

fprintf(stdout, “%s\n”, buf);

free(buf);

}

char \*upcase(char \*string, int length)

{

char \*newstring;

char temp;

int counter;

newstring=calloc(length,sizeof(char));

for(counter=0; counter<length; counter++)

{

temp=\*(string+counter);

if(temp>=97&&temp<=122)

\*(newstring+ counter)=temp-32;

else

\*(newstring+ counter)=temp;

}

return newstring;

}

3、编写一个内核模块，打印系统中各进程的PID和进程名字，同时统计系统中进程个数。（提示：使用list\_for\_each（）和list\_entry（）内核函数）

//module.c

#include <linux/init.h>

#include <linux/module.h>

#include <linux/kernel.h>

#include <linux/sched.h>

#include <linux/init\_task.h>

// 初始化函数

# include<linux/module.h>

# include<linux/kernel.h>

# include<linux/init.h>

# include<linux/list.h>

# include<linux/sched.h>

static int print\_pid(void)

{

struct task\_struct \* task, \* p;

struct list\_head \* pos;

int count = 0;

printk("Hello World enter begin:\n");

task =& init\_task;

list\_for\_each(pos, &task->tasks)

{

p = list\_entry(pos, struct task\_struct, tasks);

count++;

printk("%d---------->%s\n", p->pid, p->comm);

}

printk("the number of process is: %d\n", count);

return 0;

}

static int \_\_init lkp\_init(void)

{

printk("<1>Hello, World! from the kernel space...\n");

print\_pid;

return 0;

}

static void \_\_exit lkp\_cleanup(void)

{

printk("<1>Good Bye, World! leaving kernel space..\n");

}

module\_init(lkp\_init);

module\_exit(lkp\_cleanup);

MODULE\_LICENSE("GPL");

//makefile

obj-m:=module.o

KDIR:= /lib/modules/$(shell uname -r)/build

PWD:= $(shell pwd)

default:

$(MAKE) -C $(KDIR) M=$(PWD) modules

clean:

$(MAKE) -C $(KDIR) M=$(PWD) clean