os_exercises

徐永健

Published
with GitBook

目錄

MOOC OS习题集

这里包括从互联网上搜集的操作系统课程相关习题和答案，包括部分考研试题，版权属于各出题单位或个人。由清华大学MOOC OS课的老师和助教撰写的习题和答案的文档版权属于陈渝、向勇，并采用 Creative Commons Attribution/Share-Alike (CC-BY-SA) License.

MOOC OS习题集采用gitbook的方式展现，通过gitbook访问，可进行在线交互式答题。

下面是gitbook提供的试题编写的例子：

This is a quiz:

Here's a quiz about Gitbook

	Good	Bad
What is Gitbook?	(x)	( )

> Gitbook is good

What does Gitbook support?

- [x] Table-based questions with radio buttons
- [x] Table-based questions with checkboxes
- [ ] Telepathy
- [x] List-based questions with checkboxes
- [x] List-based questions with radio buttons
- [ ] Moon-on-a-stick

> Gitbook supports table and list based quiz questions using either radio buttons or checkboxes.
>
> Gitbook is not telepathic and does not give you the moon on a stick.

操作系统概述

单选题

（华中科技大学，2005）程序正在试图读取某个磁盘的第100个逻辑块，使用操作系统提供的（ ）接口

- [x] 系统调用
- [ ] 图形用户
- [ ] 原语
- [ ] 键盘命令

> 操作系统作为用户和计算机硬件系统之间的接口，用户可以通过3种方式使用计算机，命令方式、系统调用方式、图形方式。系统调用按照功能分为进程管理、文件操作、设备管理等，本题描述的是文件操作系统调用相关的执行。

(2009计算机统考)单处理器系统中，可并行的是（ ） 1)进程与进程 2)处理器与设备 3)处理器与通道 4)设备与设备

- [ ] 1 2 3
- [ ] 1 2 4
- [ ] 1 3 4
- [x] 2 3 4

> 并行指同一时刻同时发生，同一时刻单个处理器只能运行一个进程。

(2010统考）下列选项中，操作系统提供给应用程序的接口是（ ）

- [x] 系统调用
- [ ] 中断
- [ ] 库函数
- [ ] 原语

> 解释：略

(2011统考)下列选项中，在用户态执行的是（ ）

- [x] 命令解释程序
- [ ] 缺页处理程序
- [ ] 进程调度程序
- [ ] 时钟中断处理程序

> 后3个选项都属于内核的功能，在内核态。命令解释程序则属于应用程序。

(2013联考)计算机开机后，操作系统最终被加载到（ ）

- [ ] BIOS
- [ ] ROM
- [ ] EPROM
- [x] RAM

> 操作系统被加载到内存（RAM）中

操作系统属于__（单选）

- [ ] 硬件
- [x] 系统软件
- [ ] 通用库
- [ ] 应用软件

> 操作系统是管理计算机硬件与软件资源的计算机程序，例如Windows，Linux，Android，iOS等。应用软件一般是基于操作系统提供的接口，为针对使用者的某种应用目的所撰写的软件，例如Office Word，浏览器，手机游戏等。而通用库，一般是指为了便于程序开发，对常用的程序功能封装后被调用的程序。
>
> 以ucore OS为例，它通过I/O子系统和各种驱动程序直接控制时钟，串口，显示器等计算机硬件外设，并通过系统调用接口给在其上运行的应用软件提供服务，并通过进程管理子系统、CPU调度器、内存管理子系统、文件子系统、I/O子系统来管理应用软件的运行和实现具体的服务。

以下哪个不能用于描述操作系统（单选）

- [ ] 使计算机方便使用
- [ ] 可以管理计算机硬件
- [x] 可以控制应用软件的执行
- [ ] 负责生成应用软件

> 解释：操作系统负责管理计算机的硬件资源，使得用户不需要关心硬件的工作过程，极大地方便了计算机的使用。我们日常使用计算机，往往已经在使用了特定的操作系统，例如Windows，而在操作系统上，会同时运行多个应用软件，例如浏览器，音乐播放器等，为了让一个或者多个软件能够正常使用有限的硬件资源，操作系统需要管理应用程序的执行过程。一般来说，像浏览器，音乐播放器，和其他应用软件，都是由特定的个人和团队开发的，操作系统不负责生成应用软件。 以ucore OS开发为例，有了ucore OS，应用软件访问硬件跟简单了，有统一的文件访问方式来访问磁盘或各种外设。ucore OS 可以通过I/O操作控制硬件和应用软件的运行等。但编写软件是程序员的工作，把基于C语言或汇编语言的程序转换并生成执行代码是编译器（如gcc,gas）、连接器(如link)的工作。操作系统可加载运行应用软件的执行代码。

以下不属于操作系统的功能是（）

- [ ] 进程调度
- [ ] 内存管理
- [x] 视频编辑
- [ ] 设备驱动

> 解释：视频编辑是一个特定的功能，不是系统范围内的共性需求，具体完成这个功能的是视频编辑应用软件。当然，视频编辑应用软件在涉及文件访问时，是需要操作系统中的文件子系统支持；在涉及视频显示方面，需要操作系统的显卡/GPU等设备驱动支持。

操作系统中的多道程序设计方式用于提高__（单选）

- [ ] 稳定性
- [x] 效率
- [ ] 兼容性
- [ ] 可靠性

> 解释：是在计算机内存中同时存放几道相互独立的程序，使它们在管理程序（早期的操作系统）控制之下，相互穿插的运行。两个或两个以上程序在计算机系统中同处于开始到结束之间的状态（这里用进程来表示，在后续课程中会讲解“进程管理”）。这样可以使得几道独立的程序可以并发地共同使用各项硬件资源，提高了资源的利用率。 以ucore OS为例，在lab5中支持了用户进程，从而可以在内存中存放多个程序，并以进程的方式被操作系统管理和调度。

下面对于分时操作系统的说法，正确的是（）

- [ ] 应用程序执行的先后顺序是完全随机的

- [ ] 应用程序按照启动的时间依次执行
- [x] 应用程序可以交替执行
- [ ] 应用程序等待的时间越长，下一次调度被选中的概率一定越大

> 解释：选择3更合适。分时操作系统把多个程序放到内存中，将处理机（CPU）时间按一定的时间间隔（简称时间片）分配给程序运行，这样CPU就可以轮流地切换给各终端用户的交互式程序使用。由于时间片很短，远小于用户的交互响应延迟，用户感觉上好像独占了这个计算机系统。应用程序执行的先后顺序主要是由操作系统的调度算法和应用程序本身的行为特征来确定的。调度算法需要考虑系统的效率、公平性等因素。对于1,2而言，从系统的效率上看不会带来好处；对于4而言，可以照顾到公平性，但“一定”的表述太强了，比如如果调度算法是简单的时间片轮转算法（在后续章节“处理器调度”），则4的要求就不会满足了，且更实际的调度算法其实还需考虑等待的事件等诸多因素。 以ucore OS为例，在lab6中支持实现不同的调度算法。对于分时操作系统而言，体现其特征的一个关键点就是要实现时间片轮转调度算法或多级反馈队列调度算法（在后续章节“处理器调度”）。在ucore OS中，可以比较方便地实现这两种调度算法。

Unix操作系统属于__()

- [x] 分时操作系统
- [ ] 批处理操作系统
- [ ] 实时操作系统
- [ ] 分布式操作系统

> 解释：选择1更合适。Unix操作系统支持交互式应用程序，属于分时操作系统。比早期的批处理操作系统要强大。且它更多地面向桌面和服务器领域，并没有很强的实时调度和实时处理功能，所以一边不划归为实时系统。它虽然有网络支持（如TCP/IP），但实际上它管理的主要还是单个计算机系统让的硬件和应用软件。 以ucore OS为例，它模仿的是Unix操作系统，实现了对应的分时调度算法（时间片轮转、多级反馈队列），所以也算是分时系统。如果ucore实现了实时进程管理、实时调度算法，并支持在内核中的抢占（preempt in kernel），则可以说它也是一个实时系统了。

批处理的主要缺点是__()

- [ ] 效率低
- [x] 失去了交互性
- [ ] 失去了并行性
- [ ] 以上都不是

> 解释：批处理操作系统没有考虑人机交互所需要的分时功能，所以开发人员或操作人员无法及时与计算机进行交互。以ucore OS为例，如果它实现的调度算法是先来先服务调度算法（在后续章节“处理器调度”，相对其他调度算法，具体实现更简单），那它就是一种批处理操作系统了，没有很好的人机交互能力。

多选题

关于操作系统，说法正确的是（）

- [x] 操作系统属于软件
- [x] 操作系统负责资源管理
- [x] 操作系统使计算机的使用更加方便
- [ ] 操作系统必须要有用户程序才能正常启动

> 操作系统是一种软件，特定指是系统软件，其更功能是管理计算机资源，让用户和应用程序更方便高效地使用计算机。 以ucore OS为例，其实没有用户程序，操作系统也可以正常运行。所以选项4是部队的。

设备管理的功能包括__（）

- [x] 设备的分配和回收
- [ ] 进程调度
- [x] 虚拟设备的实现
- [x] 外围设备启动

> 进程调度是属于操作系统的进程管理和处理器调度子系统要完成的工作，与设备管理没有直接关系 以ucore OS为例（lab5以后的实验），与进程调度相关的实现位于kern/process和kern/schedule目录下；与设备管理相关的实现主要位于kern/driver目录下

多道批处理系统主要考虑的是__()

- [ ] 交互性
- [ ] 及时性
- [x] 系统效率
- [x] 吞吐量

> 解释：交互性和及时性是分时系统的主要特征。多道批处理系统主要考虑的是系统效率和系统的吞吐量。 以ucore OS为例（lab6实验）,这主要看你如何设计调度策略了，所以如果实现FCFS(先来想服务)调度算法，这可以更好地为多道批处理系统服务；如果实现时间片轮转（time-slice round robin）调度算法，则可以有比较好的交互性；如果采用多级反馈队列调度算法，则可以兼顾上述4个选项，但交互性用户程序获得CPU的优先级更高。

调查问卷

为帮助老师了解同学们的选课意图，以便更好地安排教学内容和教学要求，请同学们思考并回答如下的问题。我希望同学们尽可能按自己的兴趣或目的来安排操作系统课的学习。这些问题的回答将用作课程内容和要求的调整依据。

学生来源问卷

年龄

- [ ] 19岁以下
- [x] 20~29岁
- [ ] 30~39岁
- [ ] 40岁以上

> 解释：统计学生的年龄组成情况。

性别

- [x] 男
- [ ] 女

> 解释：统计学生的性别组成情况。

所在地区

- [x] 北京
- [ ] 其他地区

> 解释：统计学生的地理位置分布情况。

文化程度

- [ ] 初中
- [ ] 高中
- [x] 本科
- [ ] 硕士
- [ ] 博士
- [ ] 其他

从事专业

- [x] 计算机
- [ ] 信息技术
- [ ] 其他，请注明专业名称

选课问卷

为什么要学这门课？

- [x] 对内容有兴趣
- [ ] 内容与自己的目标相一致，结果有用
- [ ] 由于学分要求，必须选
- [ ] 其他，请注明原因

你打算如何来学这门课？

- [ ] 了解操作系统的基本知识：看视频，不做练习
- [ ] 学习操作系统的基本知识：看视频，做练习，不做实验
- [x] 掌握操作系统的原理：看视频、做练习、做实验
- [ ] 平时上课不多，作业基本不做，考前看两天，考试能过60分就行
- [ ] 平时上课不多，作业会独立完成，不会主动进入深入的学习，考前看几天，考试尽力而为，但对分数不是太关注
- [ ] 平时认真上课，作业独立完成，对有兴趣的内容会主动进行深入的学习，希望取得好的学习成绩
- [ ] 对操作系统十分有兴趣，想做课程设计
- [ ] 其他，请简要说明你的打算

对自己的课程学习要求是什么？

- [x] 没有兴趣，对是否能通过考试都不关心
- [ ] 没有兴趣，但关心考试是否能及格
- [ ] 没有兴趣，但需要一个好成绩，会尽力完成基本的课程要求
- [ ] 有兴趣，会尽力学好，但可能精力有限，会有限的投入条件下，尽力学好
- [ ] 有兴趣，也愿意投入精力，尽力把课程学好 ；
- [ ] 其他，请简要说明。

你对实验抄袭现象如何看？你愿意如实报告是否独立完成实验任务？我问这个问题的目的是，希望对同学实验的情况进行真实的评价，处罚抄袭者，奖励独立完成者。

以前的学习情况？是否有课程进入前10%？如果有，是哪些课程？是否有课程不及格？如果有，是哪些课程？

对计算机专业的看法是什么？自己毕业后的求职意向是什么？自己在毕业后有兴趣从事计算机专业的工作吗？

你希望在操作系统课上学到什么知识和什么能力？老师希望从你的回答中了解到同学们的兴趣点。

操作系统概述思考题

个人思考题

分析你所认识的操作系统（Windows、Linux、FreeBSD、Android、iOS）所具有的独特和共性的功能？

- [x]

请总结你认为操作系统应该具有的特征有什么？并对其特征进行简要阐述。

- [x]

请给出你觉得的更准确的操作系统的定义？

- [x]

你希望从操作系统课学到什么知识？

- [x]

小组讨论题

目前的台式PC机标准配置和价格？

- [x]

你理解的命令行接口和GUI接口具有哪些共性和不同的特征？

- [x]

为什么现在的操作系统基本上用C语言来实现？

- [x]

为什么没有人用python，java来实现操作系统？

- [x]

请评价用C++来实现操作系统的利弊？

- [x]

开放思考题

请评价微内核、单体内核、外核（exo-kernel）架构的操作系统的利弊？

- [x]

请评价用LISP,OCcaml, GO, D，RUST等实现操作系统的利弊？

- [x]

进程切换的可能实现思路？

- [x]

计算机与终端间通过串口通信的可能实现思路？

- [x]

为什么微软的Windows没有在手机终端领域取得领先地位？

- [x]

你认为未来（10年内）的操作系统应该具有什么样的特征和功能？

- [x]

---

lab0在线练习

单选题

清华大学目前的操作系统实验中采用的OS对象是()

- [ ] Linux
- [x] ucore
- [ ] xv6
- [ ] Nachos

> 是参考了xv6, OS161, Linux的教学操作系统ucore OS

在ucore lab的实验环境搭建中，使用的非开源软件是()

- [ ] eclipse CDT
- [x] Scitools Understand
- [ ] gcc
- [ ] qemu

> Scitools Understand 是非开源软件，主要可以用于分析代码，可免费试用一段时间。

在ucore lab的实验环境搭建中，用来模拟一台PC机（即基于Intel 80386 CPU的计算机）的软件是()

- [ ] apt
- [ ] git
- [ ] meld
- [x] qemu

> qemu是一个支持模拟多种CPU的模拟软件

是非题

	是	否

ucore lab实验中8个实验是否可以不按顺序完成|(x)|()|

> 每个实验i依赖前面所有的实验(0～i-1)，即完成了lab i，才能完成lab i+1

	是	否

ucore lab实验中在C语言中采用了面向对象的编程思想，包括函指针表和通用链表结构|(x)|()|

> 是的，这使得可编出更加灵活的操作系统功能模块和数据结构

多选题

x86-32 CPU（即80386）有多种运行模式，ucore lab中碰到和需要处理哪些模式()

- [x] 实模式
- [x] 保护模式
- [ ] SMM模式
- [ ] 虚拟8086模式

> ucore需要碰到和处理16位的实模式和32位的保护模式

lab0 SPOC思考题

个人思考题

能否读懂ucore中的AT&T格式的X86-32汇编语言？请列出你不理解的汇编语言。

- [x]

> http://www.imada.sdu.dk/Courses/DM18/Litteratur/IntelnATT.htm inb一般应用程序用不到的指令等。

虽然学过计算机原理和x86汇编（根据THU-CS的课程设置），但对ucore中涉及的哪些硬件设计或功能细节不够了解？

- [x]

> 中断寄存器和非通用寄存器等。

哪些困难（请分优先级）会阻碍你自主完成lab实验？

- [x]

如何把一个在gdb中或执行过程中出现的物理/线性地址与你写的代码源码位置对应起来？

- [x]

> 1. 在gdb中通过break加行号得到物理地址，list加*物理地址得到行号。
> 2. 用nm, objdump工具可以看到

了解函数调用栈对lab实验有何帮助？

- [x]

> 除了错可以调试 对于函数的调用过程和程序的运行过程有更好的理解。 便于调试以及检查。

你希望从lab中学到什么知识？

- [x]

小组讨论题

搭建好实验环境，请描述碰到的困难和解决的过程。

- [x]

> 困难：在virtualbox中设置虚拟机的时候找不到Linux的64位选项。 解决：需要通过BIOS设置将电脑的虚拟化功能打开（本电脑LenovoY480的VT功能是锁的，需要打开）。 开始时选择了UBUNTU 32位，不能启动，后来换成64位就能顺利运行

熟悉基本的git命令行操作命令，从github上 的 http://www.github.com/chyyuu/ucore_lab 下载 ucore lab实验

- [x]

> clone 仓库 gitclone http://www.github.com/chyyuu/ucore_lab

尝试用qemu+gdb（or ECLIPSE-CDT）调试lab1

- [x]

> 清除文件夹：make clean 编译lab1：make 调出debug命令行：make debug

对于如下的代码段，请说明”：“后面的数字是什么含义

```
/* Gate descriptors for interrupts and traps */
struct gatedesc {
   unsigned gd_off_15_0 : 16;        // low 16 bits of offset in segment
   unsigned gd_ss : 16;            // segment selector
   unsigned gd_args : 5;            // # args, 0 for interrupt/trap gates
   unsigned gd_rsv1 : 3;            // reserved(should be zero I guess)
   unsigned gd_type : 4;            // type(STS_{TG,IG32,TG32})
   unsigned gd_s : 1;               // must be 0 (system)
   unsigned gd_dpl : 2;            // descriptor(meaning new) privilege level
   unsigned gd_p : 1;               // Present
   unsigned gd_off_31_16 : 16;        // high bits of offset in segment
};
```

- [x]

> 每一个filed(域，成员变量)在struct(结构)中所占的位数; 也称“位域”，用于表示这个成员变量占多少位(bit)。

对于如下的代码段，

```
#define SETGATE(gate, istrap, sel, off, dpl) {            \
    (gate).gd_off_15_0 = (uint32_t)(off) & 0xffff;        \
    (gate).gd_ss = (sel);                                 \
    (gate).gd_args = 0;                                     \
    (gate).gd_rsv1 = 0;                                     \
    (gate).gd_type = (istrap) ? STS_TG32 : STS_IG32;    \
    (gate).gd_s = 0;                                    \
    (gate).gd_dpl = (dpl);                                \
    (gate).gd_p = 1;                                    \
    (gate).gd_off_31_16 = (uint32_t)(off) >> 16;        \
}
```

如果在其他代码段中有如下语句，

```
unsigned intr;
intr=8;
SETGATE(intr, 0,1,2,3);
```

请问执行上述指令后， intr的值是多少？

- [x] 0x10002

> https://github.com/chyyuu/ucore_lab/blob/master/related_info/lab0/lab0_ex3.c

请分析 list.h内容中大致的含义，并能include这个文件，利用其结构和功能编写一个数据结构链表操作的小C程序

- [x]

开放思考题

是否愿意挑战大实验（大实验内容来源于你的想法或老师列好的题目，需要与老师协商确定，需完成基本lab，但可不参加闭卷考试），如果有，可直接给老师email或课后面谈。

- [x]

中断、异常和系统调用

单选题

(西北工业大学)CPU执行操作系统代码的时候称为处理机处于（ ）

- [ ] 自由态
- [ ] 目态
- [x] 管态
- [ ] 就绪态

> （知识点：3.4系统调用）内核态又称为管态

(2012统考)下列选项中，不可能在用户态发生的是（ ）

- [ ] 系统调用
- [ ] 外部中断
- [x] 进程切换
- [ ] 缺页

> （知识点：3.3中断、异常和系统调用比较）系统调用是提供给应用程序使用的，由用户态发出，进入内核态执行。外部中断随时可能发生；应用程序执行时可能发生缺页；进程切换完全由内核来控制。

（2012统考）中断处理和子程序调用都需要压栈以保护现场。中断处理一定会保存而子程序调用不需要保存其内容的是（ ）

- [ ] 程序计数器
- [x] 程序状态字寄存器
- [ ] 通用数据寄存器
- [ ] 通用地址寄存器

> （知识点：3.3中断、异常和系统调用比较）程序状态字（PSW）寄存器用于记录当前处理器的状态和控制指令的执行顺序，并且保留与运行程序相关的各种信息，主要作用是实现程序状态的保护和恢复。所以中断处理程序要将PSW保存，子程序调用在进程内部执行，不会更改PSW。

（2013统考）下列选项中，会导致用户进程从用户态切换到内核态的操作是（ ） 1）整数除以0 2）sin()函数调用 3）read系统调用

- [ ] 1、2
- [x] 1、3
- [ ] 2、3
- [ ] 1、2、3

> （知识点：3.4系统调用）函数调用并不会切换到内核态，而除零操作引发中断，中断和系统调用都会切换到内核态进行相应处理。

（华中科技大学）中断向量地址是（ ）

- [ ] 子程序入口地址
- [x] 中断服务例程入口地址
- [ ] 中断服务例程入口地址的地址
- [ ] 例行程序入口地址

> （知识点：3.3中断、异常和系统调用比较）略

下列选项中，__可以执行特权指令？()

- [x] 中断处理例程
- [ ] 普通用户的程序
- [ ] 通用库函数
- [ ] 管理员用户的程序

> （知识点：3.3中断、异常和系统调用比较）中断处理例程（也可称为中断处理程序）需要执行打开中断，关闭中断等特权指令，而这些指令只能在内核态下才能正确执行，所以中断处理例程位于操作系统内核中。而1,3,4都属于用户程序和用于用户程序的程序库。 以ucore OS为例，在lab1中就涉及了中断处理例程，可查看intr_enable，sti，trap等函数完成了啥事情?被谁调用了?

一般来讲，中断来源于__（）

- [x] 外部设备
- [ ] 应用程序主动行为
- [ ] 操作系统主动行为
- [ ] 软件故障

> （知识点：3.3中断、异常和系统调用比较）中断来源与外部设备，外部设备通过中断来通知CPU与外设相关的各种事件。第2选项如表示是应用程序向操作系统发出的主动行为，应该算是系统调用请求。第4选项说的软件故障也可称为软件异常，比如除零錯等。 以ucore OS为例，外设产生的中断典型的是时钟中断、键盘中断、串口中断。在lab1中，具体的中断处理例程在trap.c文件中的trap_dispatch函数中有对应的实现。对软件故障/异常的处理也在trap_dispatch函数中的相关case default的具体实现中完成。在lab1的challenge练习中和lab5中，有具体的系统调用的设计与实现。

系统调用的主要作用是（）

- [ ] 处理硬件问题
- [ ] 应对软件异常
- [x] 给应用程序提供服务接口
- [ ] 管理应用程序

> （知识点：3.4系统调用）应用程序一般无法直接访问硬件，也无法执行特权指令。所以，需要通过操作系统来间接完成相关的工作。而基于安全和可靠性的需求，应用程序运行在用户态，操作系统内核运行在内核态，导致应用程序无法通过函数调用来访问操作系统提供的各种服务，于是通过系统调用的方式就成了应用程序向OS发出请求并获得服务反馈的唯一通道和接口。 以ucore OS为例，在lab1的challenge练习中和lab5中，系统调用机制的初始化也是通过建立中断向量表来完成的（可查看lab1的challenge的答案中在trap.c中idt_init函数的实现），中断向量表描述了但应用程序产生一个用于系统调用的中断号时，对应的中断服务例程的具体虚拟地址在哪里，即建立了系统调用的中断号和中断服务例程的对应关系。这样当应用程序发出类似 “int 0x80”这样的指令时（可查看lab1的challenge的答案中在init.c中lab1_switch_to_kernel函数的实现），操作系统的中断服务例程会被调用，并完成相应的服务（可查看lab1的challenge的答案中在trap.c中trap_dispatch函数有关“case T_SWITCH_TOK:”的实现）。

用户程序通过__向操作系统提出访问外部设备的请求（）

- [ ] I/O指令
- [x] 系统调用
- [ ] 中断
- [ ] 创建新的进程

> （知识点：3.3中断、异常和系统调用比较）具体内容可参见10.的回答。 以ucore OS为例，在lab5中有详细的syscall机制的设计实现。比如用户执行显示输出一个字符的操作，由于涉及向屏幕和串口等外设输出字符，需要向操作系统发出请求，具体过程是应用程序运行在用户态，通过用户程序库函数cputch，会调用sys_putc函数，并进一步调用syscall函数（在usr/libs/syscall.c文件中），而这个函数会执行“int 0x80”来发出系统调用请求。在ucore OS内核中，会

> 接收到这个系统调用号（0x80）的中断（参见 kernel/trap/trap.c中的trap_dispatch函数有关 “case T_SYSCALL:”的实现）,并进一步调用内核syscall函数（参见 kernel/syscall/syscall.c中的实现）来完成用户的请求。内核在内核态（也称特权态）完成后，通过执行“iret”指令（kernel/trap/trapentry.S中的“__trapret:”下面的指令），返回到用户态应用程序发出系统调用的下一条指令继续执行应用程序。

应用程序引发异常的时候，操作系统可能的反应是（）

- [ ] 删除磁盘上的应用程序
- [ ] 重启应用程序
- [x] 杀死应用程序
- [ ] 修复应用程序中的错误

> （知识点：3.3中断、异常和系统调用比较）更合适的答案是3。因为应用程序发生异常说明应用程序有错误或bug，如果应用程序无法应对这样的错误，这时再进一步执行应用程序意义不大。如果应用程序可以应对这样的错误（比如基于当前c++或java的提供的异常处理机制，或者基于操作系统的信号（signal）机制（后续章节“进程间通信”会涉及）），则操作系统会让应用程序转到应用程序的对应处理函数来完成后续的修补工作。 以ucore OS为例，目前的ucore实现在应对应用程序异常时做的更加剧烈一些。在lab5中有有对用户态应用程序访问内存产生错误异常的处理（参见 kernel/trap/trap.c中的trap_dispatch函数有关 “case T_PGFLT: ”的实现），即ucore判断用户态程序在运行过程中发生了内存访问错误异常，这是ucore认为重点是查找错误，所以会调用panic函数，进入kernel的监控器子系统，便于开发者查找和发现问题。这样ucore也就不再做正常工作了。当然，我们可以简单修改ucore当前的实现，不进入内核监控器，而是直接杀死进程即可。你能完成这个修改吗？

下列关于系统调用的说法错误的是（）

- [ ] 系统调用一般有对应的库函数
- [x] 应用程序可以不通过系统调用来直接获得操作系统的服务
- [ ] 应用程序一般使用更高层的库函数而不是直接使用系统调用
- [ ] 系统调用可能执行失败

> （知识点：3.4系统调用）更合适的答案是2。根据对当前操作系统设计与实现的理解，系统调用是应用程序向操作系统发出服务请求并获得操作系统服务的唯一通道和结果。如果操作系统在执行系统调用服务时，产生了错误，就会导致系统调用执行失败。 以ucore OS为例，在用户态的应用程序（lab5,6,7,8中的应用程序）都是通过系统调用来获得操作系统的服务的。为了简化应用程序发出系统调用请求，ucore OS提供了用户态的更高层次的库函数（user/libs/ulib.[ch]和syscall.[ch]），简化了应用程序的编写。如果操作系统在执行系统调用服务时，产生了错误，就会导致系统调用执行失败。

以下关于系统调用和常规调用的说法中，错误的是（）

- [ ] 系统调用一般比常规函数调用的执行开销大
- [ ] 系统调用需要切换堆栈
- [ ] 系统调用可以引起特权级的变化
- [x] 常规函数调用和系统调用都在内核态执行

> （知识点：3.4系统调用）系统调用相对常规函数调用执行开销要大，因为这会涉及到用户态栈和内核态栈的切换开销，特权级变化带来的开销，以及操作系统对用户态程序传来的参数安全性检查等开销。如果发出请求的请求方和应答请求的应答方都在内核态执行，则不用考虑安全问题了，效率还是需要的，直接用常规函数调用就够了。 以ucore OS为例，我们可以看到系统调用的开销在执行“int 0x80”和“iret”带来的用户态栈和内核态栈的切换开销，两种特权级切换带来的执行状态（关注 kern/trap/trap.h中的trapframe数据结构）的保存与恢复等（可参看 kern/trap/trapentry.S的**alltraps**和trapret的实现）。而函数调用使用的是"call"和“ret”指令，只有一个栈，不涉及特权级转变带来的各种开销。如要了解call, ret, int和iret指令的具体功能和实现，可查看“英特尔 64 和 iA-32 架构软件开发人员手册卷 2a's,指令集参考（A-M）”和“英特尔64 和 iA-32 架构软件开发人员手册卷 2B' s,指令集参考（N-Z）”一书中对这些指令的叙述。

操作系统与用户的接口包括__（）

- [x] 系统调用

- [ ] 进程调度
- [ ] 中断处理
- [ ] 程序编译

> （知识点：3.3中断、异常和系统调用比较）解释： 更合适的答案是1。根据对当前操作系统设计与实现的理解，系统调用是应用程序向操作系统发出服务请求并获得操作系统服务的唯一通道和结果。

多选题

操作系统处理中断的流程包括__（）

- [x] 保护当前正在运行程序的现场
- [x] 分析是何种中断，以便转去执行相应的中断处理程序
- [x] 执行相应的中断处理程序
- [x] 恢复被中断程序的现场

> （知识点：3.3中断、异常和系统调用比较）解释：中断是异步产生的，会随时打断应用程序的执行，且在操作系统的管理之下，应用程序感知不到中断的产生。所以操作系统需要保存被打断的应用程序的执行现场，处理具体的中断，然后恢复被打断的应用程序的执行现场，使得应用程序可以继续执行。 以ucore OS为例（lab5实验），产生一个中断XX后，操作系统的执行过程如下： vectorXX(vectors.S)--> **alltraps(trapentry.S)-->trap(trap.c)-->trap_dispatch(trap.c)-->-->……**具体的中断处理**-->**trapret(trapentry.S) 通过查看上述函数的源码，可以对应到答案1-4。另外，需要注意，在ucore中，应用程序的执行现场其实保存在trapframe数据结构中。

下列程序工作在内核态的有__()

- [x] 系统调用的处理程序
- [x] 中断处理程序
- [x] 进程调度
- [x] 内存管理

> （知识点：3.3中断、异常和系统调用比较）这里说的“程序”是一种指称，其实就是一些功能的代码实现。而1-4都是操作系统的主要功能，需要执行相关的特权指令，所以工作在内核态。 以ucore OS为例（lab5实验），系统调用的处理程序在kern/syscall目录下，中断处理程序在kern/trap目录下，进程调度在kern/schedule目录下，内存管理在kern/mm目录下

---

lec 3 SPOC Discussion

第三讲 启动、中断、异常和系统调用-思考题

3.1 BIOS

1. 比较UEFI和BIOS的区别。
2. 描述PXE的大致启动流程。

3.2 系统启动流程

1. 了解NTLDR的启动流程。
2. 了解GRUB的启动流程。
3. 比较NTLDR和GRUB的功能有差异。
4. 了解u-boot的功能。

3.3 中断、异常和系统调用比较

1. 举例说明Linux中有哪些中断，哪些异常？
2. Linux的系统调用有哪些？大致的功能分类有哪些？ (w2l1)

```
+ 采分点：说明了Linux的大致数量（上百个），说明了Linux系统调用的主要分类（文件操作，进程管理，内存管理等）
- 答案没有涉及上述两个要点；（0分）
- 答案对上述两个要点中的某一个要点进行了正确阐述（1分）
- 答案对上述两个要点进行了正确阐述（2分）
- 答案除了对上述两个要点都进行了正确阐述外，还进行了扩展和更丰富的说明（3分）
```

1. 以ucore lab8的answer为例，uCore的系统调用有哪些？大致的功能分类有哪些？(w2l1)

```
+ 采分点：说明了ucore的大致数量（二十几个），说明了ucore系统调用的主要分类（文件操作，进程管理，内存管理等）
- 答案没有涉及上述两个要点；（0分）
- 答案对上述两个要点中的某一个要点进行了正确阐述（1分）
- 答案对上述两个要点进行了正确阐述（2分）
- 答案除了对上述两个要点都进行了正确阐述外，还进行了扩展和更丰富的说明（3分）
```

3.4 linux系统调用分析

1. 通过分析lab1_ex0了解Linux应用的系统调用编写和含义。(w2l1)

```
+ 采分点：说明了objdump，nm，file的大致用途，说明了系统调用的具体含义
- 答案没有涉及上述两个要点；（0分）
- 答案对上述两个要点中的某一个要点进行了正确阐述（1分）
- 答案对上述两个要点进行了正确阐述（2分）
- 答案除了对上述两个要点都进行了正确阐述外，还进行了扩展和更丰富的说明（3分）
```

1. 通过调试lab1_ex1了解Linux应用的系统调用执行过程。(w2l1)

```
+ 采分点：说明了strace的大致用途，说明了系统调用的具体执行过程（包括应用，CPU硬件，操作系统的执行过程）
- 答案没有涉及上述两个要点；（0分）
- 答案对上述两个要点中的某一个要点进行了正确阐述（1分）
- 答案对上述两个要点进行了正确阐述（2分）
- 答案除了对上述两个要点都进行了正确阐述外，还进行了扩展和更丰富的说明（3分）
```

3.5 ucore系统调用分析

1. ucore的系统调用中参数传递代码分析。
2. ucore的系统调用中返回结果的传递代码分析。
3. 以ucore lab8的answer为例，分析ucore 应用的系统调用编写和含义。
4. 以ucore lab8的answer为例，尝试修改并运行ucore OS kernel代码，使其具有类似Linux应用工具 `strace` 的功能，即能够显示出应用程序发出的系统调用，从而可以分析ucore应用的系统调用执行过程。

3.6 请分析函数调用和系统调用的区别

1. 请从代码编写和执行过程来说明。
    i. 说明 `int` 、 `iret` 、 `call` 和 `ret` 的指令准确功能

lab1在线练习

选择题

80386机器加电启动后，CPU立刻跳转到()执行

- [ ] ucore第一条指令
- [ ] bootloader第一条指令
- [x] BIOS的第一条指令
- [ ] GRUB的第一条指令

> 是调到BIOS去执行

应用程序中的C函数调用中不需要用到()指令

- [ ] push
- [ ] ret
- [x] iret
- [ ] call

> iret用于中断返回

GCC内联汇编 `asm("movl %ecx, %eax");` 的含义是()

- [ ] 把 ecx 内容移动到 eax
- [ ] 把 eax 内容移动到 ecx

> 把 ecx 内容移动到 eax

为了让系统正确完成80386的中断处理过程中，操作系统需要正确设置()

- [x] 全局描述符表
- [x] 中断描述符表
- [x] 中断服务例程
- [x] 内核堆栈

> 在ucore处理中，上述几个都是要设置好的。

lab1 SPOC思考题

个人思考题

NOTICE

- 有"w2l2"标记的题是助教要提交到学堂在线上的。
- 有"w2l2"和"spoc"标记的题是要求拿清华学分的同学要在实体课上完成，并按时提交到学生对应的git repo上。
- 有"hard"标记的题有一定难度，鼓励实现。
- 有"easy"标记的题很容易实现，鼓励实现。
- 有"midd"标记的题是一般水平，鼓励实现。

请描述ucore OS配置和驱动外设时钟的准备工作包括哪些步骤？ (w2l2)

```
+ 采分点：说明了ucore OS在让外设时钟正常工作的主要准备工作
- 答案没有涉及如下3点；（0分）
- 描述了对IDT的初始化，包了针时钟中断的中断描述符的设置（1分）
- 除第二点外，进一步描述了对8259中断控制器的初始过程（2分）
- 除上述两点外，进一步描述了对8253时钟外设的初始化，或描述了对EFLAG操作使能中断（3分）
```

- [x]

lab1中完成了对哪些外设的访问？ (w2l2)

```
+ 采分点：说明了ucore OS访问的外设
- 答案没有涉及如下3点；（0分）
- 说明了时钟（1分）
- 除第二点外，进一步说明了串口（2分）
- 除上述两点外，进一步说明了并口，或说明了CGA，或说明了键盘（3分）
```

- [x]

lab1中的cprintf函数最终通过哪些外设完成了对字符串的输出？ (w2l2)

```
+ 采分点：说明了cprintf函数用到的3个外设
- 答案没有涉及如下3点；（0分）
- 说明了串口（1分）
- 除第二点外，进一步说明了并口（2分）
- 除上述两点外，进一步说明了CGA（3分）
```

- [x]

小组思考题

---

lab1中printfmt函数用到了可变参，请参考写一个小的linux应用程序，完成实现定义和调用一个可变参数的函数。(spoc)

- [x]

如果让你来一个阶段一个阶段地从零开始完整实现lab1（不是现在的填空考方式），你的实现步骤是什么？（比如先实现一个可显示字符串的bootloader（描述一下要实现的关键步骤和需要注意的事项），再实现一个可加载ELF格式文件的bootloader（再描述一下进一步要实现的关键步骤和需要注意的事项）...） (spoc)

- [x]

如何能获取一个系统调用的调用次数信息？如何可以获取所有系统调用的调用次数信息？请简要说明可能的思路。(spoc)

- [x]

如何修改lab1, 实现一个可显示字符串"THU LAB1"且依然能够正确加载ucore OS的bootloader？如果不能完成实现，请说明理由。

- [x]

对于ucore_lab中的labcodes/lab1，我们知道如果在qemu中执行，可能会出现各种稀奇古怪的问题，比如reboot，死机，黑屏等等。请通过qemu的分析功能来动态分析并回答lab1是如何执行并最终为什么会出现这种情况？

- [x]

对于ucore_lab中的labcodes/lab1,如果出现了reboot，死机，黑屏等现象，请思考设计有效的调试方法来分析常在现背后的原因。

- [x]

开放思考题

如何修改lab1, 实现在出现除零错误异常时显示一个字符串的异常服务例程的lab1？

- [x]

在lab1/bin目录下，通过 `objcopy -O binary kernel kernel.bin` 可以把elf格式的ucore kernel转变成体积更小巧的binary格式的ucore kernel。为此，需要如何修改lab1的bootloader, 能够实现正确加载binary格式的ucore OS？ (hard)

- [x]

>

GRUB是一个通用的bootloader，被用于加载多种操作系统。如果放弃lab1的bootloader，采用GRUB来加载ucore OS，请问需要如何修改lab1, 能够实现此需求？ (hard)

- [x]

>

如果没有中断，操作系统设计会有哪些问题或困难？在这种情况下，能否完成对外设驱动和对进程的切换等操作系统核心功能？

- [x]

各知识点的小练习

4.1 启动顺序

读入ucore内核的代码？

- [x]

跳转到ucore内核的代码？

- [x]

全局描述符表的初始化代码？

- [x]

GDT内容的设置格式？初始映射的基址和长度？特权级的设置位置？

- [x]

可执行文件格式elf的各个段的数据结构？

- [x]

如果ucore内核的elf是否要求连续存放？为什么？

- [x]

>

4.2 C函数调用的实现

函数调用的stackframe结构？函数调用的参数传递方法有哪几种？

- [x]

系统调用的stackframe结构？系统调用的参数传递方法有哪几种？

- [x]

>

4.3 GCC内联汇编

使用内联汇编的原因？

- [x]

> 特权指令、性能优化

对ucore中的一段内联汇编进行完整的解释？

- [x]

>

4.4 x86中断处理过程

4.4 x86中断处理过程

中断描述符表IDT的结构？

- [x]

中断描述表到中断服务例程的地址计算过程？

- [x]

中断处理中硬件压栈内容？用户态中断和内核态中断的硬件压栈有什么不同？

- [x]

中断处理中硬件保存了哪些寄存器？

- [x]

>

trap类型的中断门与interrupt类型的中断门有啥设置上的差别？如果在设置中断门上不做区分，会有什么可能的后果?

- [x]

>

4.5 练习一 ucore编译过程

gcc编译、ld链接和dd生成两个映像对应的makefile脚本行？

- [x]

在函数print_stackframe中要调用函数print_debuginfo(uintptr_t eip)来打印函数源码位置信息，

```
print_stackframe(void)
 eip = read_eip();
 #option 1
 print_debuginfo(eip - 1);
 #option 2
 print_debuginfo(eip );
```

请问option1和 option2 的结有何区别？请说明。

- [x]

对于如下5条语句执行后得到的5个eip（类型为uint32_t）的结果的数值关系是什么？

```
eip = ((uint32_t *)ebp)[2];
```

```
    eip = ((uint32_t *)ebp)[1];
    eip = ((uint32_t *)(ebp+4);
    eip = ((uint32_t *)(ebp+2);
    eip = ((uint32_t *)(ebp+1);
```

- [x]

4.6 练习二 qemu和gdb的使用

qemu的命令行参数含义解释？

- [x]

gdb命令格式？反汇编、运行、断点设置

- [x]

>

练习三 加载程序

A20的使能代码分析？

- [x]

生成主引导扇区的过程分析？

- [x]

保护模式的切换代码？

- [x]

如何识别elf格式？对应代码分析？

- [x]

跳转到elf的代码？

- [x]

> 函数调用栈获取？

- [x]

>

4.8 练习四和五 ucore内核映像加载和函数调用栈分析

如何识别elf格式？对应代码分析？

- [x]

跳转到elf的代码？

- [x]

函数调用栈获取？

- [x]

函数read_ebp是inline的，而函数read_eip是__noinline的，能否正好相反设置，即设置函数read_ebp是_noinline的，而函数read_eip是inline的？为什么？

- [x]

>

4.9 练习六 完善中断初始化和处理

各种设备的中断初始化？

- [x]

> 中断描述符表IDT的排列顺序？
>
> - [x]
>
> 中断号 CPU加电初始化后中断是使能的吗？为什么？

- [x]

> 中断服务例程的入口地址在什么地方设置的？

- [x]

> alltrap的中断号是在哪写入到trapframe结构中的？

- [x]

> trapframe结构？
>
> - [x]

物理内存管理

单选题

(2009联考)分区分配内存管理方式的主要保护措施是（ ）

- [x] 界地址保护
- [ ] 程序代码保护
- [ ] 数据保护
- [ ] 栈保护

> 为了防止程序的访问地址越界，所以需要进行界地址保护，由硬件负责检查程序访问地址是否合法。
>
> 以ucore为例，在ucore lab1中的，x86保护模式中的分段机制在某种程度上可以看成是一种分区方式的物理内存分配。bootloader在启动后，就要完成分段（分区）工作，即建立两个段，内核代码段和内核数据段，一个段就是一个分区。在表述段属性的段描述符（mmu.h中的segdesc数据结构）中，有两个重要的域（字段，field），起始地址（sd_base_15_0,sd_base_23_16,sd_base_31_24）、段限长(sd_lim_15_0,sd_lim_19_16)。这个段大小就是分区的界地址。80386 CPU在每一次内存寻址过程中，都会比较EIP（即段内偏移）是否大于段限长，如果大于段限长，这会产生一个内存访问错误异常。

(2010联考)某基于动态分区存储管理的计算机，其主存容量为55MB（初始为空），采用最佳适配（Best Fit）算法，分配和释放的顺序为：分配15MB，分配30MB，释放15MB，分配8MB，分配6MB，则此时主存中最大空闲分区的大小是（ ）

- [ ] 7MB
- [x] 9MB
- [ ] 10MB
- [ ] 15MB

> 空闲分区链变化：55（初始）；40（分配15MB后）；10（分配30MB后）；10->15（释放15MB后）；2->15（分配8MB后）；2->9（分配6MB后）。
>
> 以ucore为例，能否在ucore中做个试验完成上述算法？

(2009联考）一个分段存储系统中，地址长度为32位，其中段号占8位，则最大段长为（ ）

- [ ] 2^8字节
- [ ] 2^16字节
- [x] 2^24字节
- [ ] 2^32字节

> 在段访问机制中，如果采用的是单地址方案，则段号的位数+段内偏移的位数=地址长度，所以段内偏移占了32 - 8 = 24 比特。
>
> 以ucore lab1为例，在段访问机制上，80386采用了不同，且更加灵活的段寄存器+地址寄存器方案，(可看OS原理部分的"物理内存管理：第2部分"ppt的第10页)，即CS中的值（称为选择子，selector）是段号，作为索引，指向了一个段描述符表中的一个段描述符。 段描述符中的一个字段是段基址（32位），这个32位段基址+段内偏移（即32位的EIP）形成了最终的线性地址（如果使能页机制，则也就是最终的物理地址了）。 所以，如果是这道题说明了采用80386方式，结果就不一样了。

(2010联考）某计算机采用二级页表的分页存储管理方式，按字节编址，页大小为2^10字节，页表项大小为2字节，逻辑地址结构为“|页目录号|页表|页内偏移量|”逻辑地址空间大小为2^16页，则表示整个逻辑地址空间的页目录表中包含表项的个数至

少为（）

- [ ] 64
- [x] 128
- [ ] 256
- [ ] 512

> 页大小为2^10B，页表项大小为2B，一页可以存放2^9个页表项，逻辑地址空间大小为2^16页，需要2^16个页表项，需要2^16/2^9 = 2^7 = 128个页面保存页表项。所以页目录表中包含的表项至少为128。
>
> 以ucore lab2为例，80386保护模式在使能页机制后，采用的是二级页表，32位地址空间，页大小为2^12字节，页表项为4字节，逻辑地址结构为"|页目录号|页表|页内偏移量|"逻辑地址空间大小为2^20页，则也目录表包含的表项个数为1024. 你觉得对吗？

(武汉理工大学）段式存储管理系统中，一个程序如何分段是在（ ）决定的。

- [ ] 分配主存时
- [x] 程序员编程时
- [ ] 装作业时
- [ ] 程序执行时

> 程序员在编程时，其实已经确定了哪些是代码，哪些是数据。所以编译器可以把代码放到代码段，数据放到数据段，操作系统可以根据编译器的设置把程序加载到内存中的代码段和数据段执行。
>
> 以ucore lab1为例，学员编写文件有代码和数据两部分，gcc把代码放到代码段，数据放到数据段，并形成ELF格式的ucore kernel（这其实也是一个程序）。bootloader把ucore的代码或数据放到它设置好的内核代码段和内核数据段中。ucore lab5以后，ucore也可以加载应用程序到用户代码段和用户数据段中。

一般情况下，____的速度最快

- [x] L1 cache
- [ ] L2 cache
- [ ] Main Memory
- [ ] Disk

> 解释：访问速度上 cache > Main Memory > Disk; cache中 L1 > L2 > L3 ... 越靠近cpu速度越快，容量越小。
>
> 这个在qemu模拟器中无法体现，因为它没有模拟不同存储之间的访问速度。 :(

分页系统中, 逻辑地址到物理地址的变换是由____决定的

- [ ] 段表
- [x] 页表
- [ ] 物理结构
- [ ] 重定位寄存器

> 解释：分页系统中，页表负责转换逻辑地址到物理地址。

分段系统中, 逻辑地址到物理地址的变换是由____决定的

- [x] 段表
- [ ] 页表
- [ ] 物理结构
- [ ] 重定位寄存器

> 解释：看看ucore lab1，了解bootloader和ucore是如何建立段表的。

连续内存分配算法中的First Fit（首次适应）算法，其空闲分区链的顺序为____

- [x] 空闲区首地址递增
- [ ] 空闲区首地址递减
- [ ] 空闲区大小递增
- [ ] 空闲区大小递减

> 解释：First Fit是指按地址来寻找第一个满足要求的空闲块，其空闲分区链的顺序也就是按空闲块首地址递增。
>
> 以ucore为例，能否在ucore中做个试验完成上述算法？

连续内存分配算法中的Best Fit（最佳适应）算法，其空闲分区链的顺序为____

- [ ] 空闲区首地址递增
- [ ] 空闲区首地址递减
- [x] 空闲区大小递增
- [ ] 空闲区大小递减

> 解释：Best Fit是指寻找一个大小最合适的空闲块，要求空闲块按照大小排列，其空闲分区链的顺序为按大小递增。
>
> 以ucore为例，能否在ucore中做个试验完成上述算法？

连续内存分配算法First Fit的缺点是____

- [ ] 算法复杂
- [ ] 大的空闲分区会被分割
- [x] 容易产生外部碎片
- [ ] 分配速度慢

> 解释：First Fit算法非常简单，分配速度也较快。但是First Fit不考虑实际的需求和找到的空闲分区的大小的匹配度，所以容易产生外部碎片。
>
> 以ucore为例，能否在ucore中做个试验完成上述算法并看看其缺点能否重现？

连续内存分配算法Best Fit的缺点是____

- [ ] 算法复杂
- [ ] 大的空闲分区会被分割
- [ ] 分配速度慢
- [x] 回收速度慢

> 解释：Best Fit算法也非常简单，分配速度较快。由于选取的空闲分区大小都很合适，所以基本不会出现大的空闲分区总是被分割的情况。但是在此算法中，内存回收则涉及了很多操作：判断左右邻居是否是空闲分区，如果不是，则插入此空闲分区到合适的地方，如果是则合并空闲块，并把合并后的结果插入到合适地方；但是由于空闲分区链不是按地址排序的，所以上述操作需要遍历几次链表用于查找和插入，速度较慢。

连续内存分配算法Worst Fit的缺点是____

- [ ] 算法复杂
- [x] 大的空闲分区会被分割
- [ ] 分配速度慢
- [ ] 容易产生很小的空闲分区

> 解释：Worst Fit每次使用最大的空闲分区，按照需求分割相应的大小，所以会造成大的空闲分区总是被分割。其算法比较简单，分配速度也很快。

> 以ucore为例，能否在ucore中做个试验完成上述算法并看看其缺点能否重现？

应用程序中的逻辑地址到物理内存中的物理地址的转换机制建立的过程发生____程序过程中

- [ ] 编译
- [ ] 链接
- [x] 加载
- [ ] 运行

> 解释：在编译器编译和链接程序的过程中都只涉及到逻辑地址，跟机器的配置无关，这也是编译链接所生成的可执行文件可以直接在相同系统的其它机器上使用的原因。而在操作系统加载应用程序时，操作系统负责建立应用程序的段表或页表。将逻辑地址和实际物理地址对应起来，之后应用程序在运行过程中CPU才能根据逻辑地址通过段表或页表正确访问到物理地址。
>
> 以ucore lab5为例，在加载应用程序时，将建立应用程序对应的进程，并在建立过程中，把应用程序对应进程的页表也建立好了。

某一作业完成后，系统收回其主存空间，并与相邻空闲区合并，为此需修改空闲区表，如果待回收的空闲区有相邻的低址空闲区,也有相邻的高址空闲区，那么空闲区表将____

- [ ] 项数不变，有一个空闲区的大小变大
- [ ] 项数不变，有一个空闲区的起始地址变小，大小变大
- [ ] 项数增加
- [x] 项数减少

> 解释：合并之后，原本的2个相邻空闲分区和被回收的分区合并成一个分区，所以分区项数变为n - 2 + 1 = n - 1。
>
> 以ucore为例，能否在ucore中做个试验完成上述相邻空闲区合并功能，看看合并的效果如何？

对于分页系统与分段系统,下列说法正确的是( ).

- [ ] 页的大小跟具体执行程序有关
- [x] 都属于非连续分配
- [ ] 段的大小固定且由系统确定
- [ ] 分段技术和分页技术是不能共存在一个系统中的

> 解释：页的大小由CPU硬件规定的规范，并由操作系统进行初始化和管理，跟具体执行程序无关； 段的大小是指程序的数据段、代码段等每段的大小，和具体程序相关；分段技术和分页技术是按照需求进行动态的分配和回收，是非连续分配，它们可以融合使用，也称段页式管理。
>
> 以ucore为例，在lab2之后，就采用基于80386的段页式管理了，但段机制的功能没有太用。

采用段页式管理时，程序按逻辑被划分成____

- [x] 段
- [ ] 页
- [ ] 区域
- [ ] 块

> 解释：程序按逻辑划分各段。而页、区域、块由操作系统负责分配、映射和管理，和程序逻辑没有关系。

采用段页式管理的多道程序环境下，一个应用程序都有对应的____

- [x] 一个段表和一个页表
- [ ] 一个段表和一组页表
- [ ] 一组段表和一个页表

- [ ] 一组段表和一组页表

> 解释：每道程序有一个段表，且还要有一个页表，才能完成段页式的内存管理。
>
> 以ucore为例，在lab5之后，就采用基于80386的段页式管理了，一个程序有一个段表，也有一个页表。

在分页式存储管理系统中时，每次CPU取指令或取操作数，至少要访问____次主存。

- [x] 0
- [ ] 1
- [ ] 2
- [ ] 3

> 解释：0次。因为CPU会有cache和mmu
>
> 以ucore为例，在80386中，如果取的指令和取的操作数都在CPU的cache中，且放在页表中的地址映射关系被缓存在CPU的MMU中，则不需要访问主存。

在分段式存储管理系统中时，每次CPU取指令或取操作数，至少要访问____次主存。

- [x] 0
- [ ] 1
- [ ] 2
- [ ] 3

> 解释：0次。因为CPU会有cache，mmu和对段表项的缓存
>
> 以ucore为例，在80386中，如果取的指令和取的操作数都在CPU的cache中，且放在段表中的地址映射关系被缓存在CPU的段表项缓存中，则不需要访问主存。

在段页式存储管理系统中时，每次CPU取指令或取操作数，至少要访问____次主存。

- [x] 0
- [ ] 1
- [ ] 2
- [ ] 3

> 解释：0次。因为CPU会有cache和对段表项的缓存 以ucore为例，在80386中，如果取的指令和取的操作数都在CPU的cache中，且放在段表中的地址映射关系被缓存在CPU的段表项缓存中，且放在页表中的地址映射关系被缓存在CPU的MMU中，则不需要访问主存。

每道程序能在不受干扰的环境下运行，主要是通过____功能实现的。

- [ ] 内存分配
- [x] 内存保护
- [ ] 内存回收
- [ ] 内存扩充

> 解释：内存访问需要将逻辑地址和重定位寄存器(基址寄存器)进行加运算之后才能访问物理地址，而内存保护主要是使用界地址寄存器来实现对逻辑地址的限制，以免逻辑地址越界而造成物理地址访问越界，进而对别的程序进行干扰。
>
> 以ucore为例，在lab5中，ucore对每个应用程序设置的页表，可以完成对不同程序的地址空间的隔离，从而避免了程序间可能的干扰。

可变分区存储管理方案中____作为存储保护使用。

- [ ] 逻辑地址寄存器
- [x] 长度寄存器
- [ ] 物理地址寄存器
- [ ] 基址寄存器

> 解释：长度寄存器或称界地址寄存器，用于存储保护。
>
> 以ucore为例，在80386中，分段机制可以看成是一种可变分区存储管理方案，其段描述符中的段限长字段类似这里提到的“长度寄存器“

分页系统中的页面对____透明，是____管理的。

- [ ] 程序员、编译器
- [x] 程序员、操作系统
- [ ] 操作系统、编译器
- [ ] 程序员、链接器

> 解释：分页由操作系统控制，用户并不能感知。
>
> 以ucore为例，在lab5以后，程序员写应用程序，确实不需要考虑有分页机制的存在。因为ucore已经为每个应用程序建立好了段表（也叫全局描述符表，简称GDT）和页表。

多选题

关于分段系统和分页系统说法正确有____。

- [x] 页是系统层面的内存管理的单位，分页的目的主要是由于操作系统管理的需要；段是编写程序层面的内存管理的单位,分段的目的主要是为了能更好地满足程序员开发的需要
- [x] 页的大小是固定的，而且由系统确定。段的长度却是不固定的，决定于程序员所编写的程序
- [x] 分段系统会产生外碎片，分页系统会产生内碎片
- [x] 分段可灵活的控制存取访问，可根据各段的特点决定访问权

> 解释：1,2,4解释略。3：分段系统中段的大小是跟程序相关的，分段系统中每次分配的大小就是相应段的真实大小所以没有内部碎片；但是却会产生不满足任何段大小的空闲分区，就是外部碎片。

---

lec5 在线练习

选择题

操作系统中可采用的内存管理方式包括() s1

- [x] 重定位(relocation)
- [x] 分段(segmentation
- [x] 分页(paging)
- [x] 段页式（segmentation+paging）

> 都有

在启动页机制的情况下，在CPU运行的用户进程访问的地址空间是() s2

- [ ] 物理地址空间
- [x] 逻辑地址空间
- [ ] 外设地址空间
- [ ] 都不是

> 用户进程访问的内存地址是虚拟地址

连续内存分配的算法中，会产生外碎片的是() s3

- [x] 最先匹配算法
- [x] 最差匹配算法
- [x] 最佳匹配算法
- [ ] 都不会

> 三种算法都会有外碎片

在使能分页机制的情况下，更合适的外碎片整理方法是() s4

- [ ] 紧凑(compaction)
- [ ] 分区对换(Swapping in/out)
- [x] 都不是

> 分页方式不会有外碎片

描述伙伴系统(Buddy System)特征正确的是() s5

- [x] 多个小空闲空间可合并为大的空闲空间
- [x] 会产生外碎片
- [x] 会产生内碎片
- [ ] 都不对

> 前三个是对的。

lec5 SPOC思考题

NOTICE

- 有"w3l1"标记的题是助教要提交到学堂在线上的。
- 有"w3l1"和"spoc"标记的题是要求拿清华学分的同学要在实体课上完成，并按时提交到学生对应的git repo上。
- 有"hard"标记的题有一定难度，鼓励实现。
- 有"easy"标记的题很容易实现，鼓励实现。
- 有"midd"标记的题是一般水平，鼓励实现。

个人思考题

请简要分析最优匹配，最差匹配，最先匹配，buddy systemm分配算法的优势和劣势，并尝试提出一种更有效的连续内存分配算法 (w3l1)

```
+ 采分点：说明四种算法的优点和缺点
- 答案没有涉及如下3点；（0分）
- 正确描述了二种分配算法的优势和劣势（1分）
- 正确描述了四种分配算法的优势和劣势（2分）
- 除上述两点外，进一步描述了一种更有效的分配算法（3分）
```

- [x]

小组思考题

请参考ucore lab2代码，采用 `struct pmm_manager` 根据你的 `学号 mod 4` 的结果值，选择四种（0:最优匹配，1:最差匹配，2:最先匹配，3:buddy systemm）分配算法中的一种或多种，在应用程序层面(可以 用python,ruby,C++，C，LISP等高语言)来实现，给出你的设思路，并给出测试用例。 (spoc)

```
如何表示空闲块？ 如何表示空闲块列表？
[(start0, size0),(start1,size1)...]
在一次malloc后，如果根据某种顺序查找符合malloc要求的空闲块？如何把一个空闲块改变成另外一个空闲块，或消除这个空闲块？如何更新空闲块列表？
在一次free后，如何把已使用块转变成空闲块，并按照某种顺序（起始地址，块大小）插入到空闲块列表中？考虑需要合并相邻空闲块，形成更大的空闲块？
如果考虑地址对齐（比如按照4字节对齐），应该如何设计？
如果考虑空闲/使用块列表组织中有部分元数据，比如表示链接信息，如何给malloc返回有效可用的空闲块地址而不破坏
元数据信息？
伙伴分配器的一个极简实现
http://coolshell.cn/tag/buddy
```

扩展思考题

阅读slab分配算法，尝试在应用程序中实现slab分配算法，给出设计方案和测试用例。

“连续内存分配”与视频相关的课堂练习

5.1 计算机体系结构和内存层次

MMU的工作机理？

- [x]

> http://en.wikipedia.org/wiki/Memory_management_unit

L1和L2高速缓存有什么区别？

- [x]

> http://superuser.com/questions/196143/where-exactly-l1-l2-and-l3-caches-located-in-computer Where exactly L1, L2 and L3 Caches located in computer?
>
> http://en.wikipedia.org/wiki/CPU_cache CPU cache

5.2 地址空间和地址生成

编译、链接和加载的过程了解？

- [x]

动态链接如何使用？

- [x]

5.3 连续内存分配

什么是内碎片、外碎片？

- [x]

为什么最先匹配会越用越慢？

- [x]

为什么最差匹配会的外碎片少？

- [x]

在几种算法中分区释放后的合并处理如何做？

- [x]

5.4 碎片整理

一个处于等待状态的进程被对换到外存（对换等待状态）后，等待事件出现了。操作系统需要如何响应？

- [x]

5.5 伙伴系统

伙伴系统的空闲块如何组织？

- [x]

伙伴系统的内存分配流程？

- [x]

伙伴系统的内存回收流程？

- [x]

struct list_entry是如何把数据元素组织成链表的？

- [x]

lec6 在线练习

选择题

描述段管理机制正确的是() s2

- [x] 段的大小可以不一致
- [x] 段可以有重叠
- [x] 段可以有特权级
- [x] 段与段之间是可以不连续的

> 都对

描述页管理机制正确的是() s3

- [x] 页表在内存中
- [x] 页可以是只读的
- [x] 页可以有特权级
- [ ] 上诉说法都不对

> 前三个都对

页表项标志位包括() s4

- [x] 存在位(resident bit)
- [x] 修改位(dirty bit)
- [x] 引用位(clock/reference bit)
- [x] 只读位(read only OR read/write bit)

> 都有

可有效应对大地址空间可采用的页表手段是() s7

- [x] 多级页表
- [x] 反置页表
- [ ] 页寄存器方案
- [ ] 单级页表

> 前两个是对的。

lec6 SPOC思考题

NOTICE

- 有"w3l2"标记的题是助教要提交到学堂在线上的。
- 有"w3l2"和"spoc"标记的题是要求拿清华学分的同学要在实体课上完成，并按时提交到学生对应的git repo上。
- 有"hard"标记的题有一定难度，鼓励实现。
- 有"easy"标记的题很容易实现，鼓励实现。
- 有"midd"标记的题是一般水平，鼓励实现。

个人思考题

（1） (w3l2) 请简要分析64bit CPU体系结构下的分页机制是如何实现的

```
+ 采分点：说明64bit CPU架构的分页机制的大致特点和页表执行过程
- 答案没有涉及如下3点；（0分）
- 正确描述了64bit CPU支持的物理内存大小限制（1分）
- 正确描述了64bit CPU下的多级页表的级数和多级页表的结构或反置页表的结构（2分）
- 除上述两点外，进一步描述了在多级页表或反置页表下的虚拟地址-->物理地址的映射过程（3分）
```

- [x]

小组思考题

（1）(spoc) 某系统使用请求分页存储管理，若页在内存中，满足一个内存请求需要150ns (10^-9s)。若缺页率是10%，为使有效访问时间达到0.5us(10^-6s),求不在内存的页面的平均访问时间。请给出计算步骤。

- [x]

> 500=0.9*150+0.1*x

（2）(spoc) 有一台假想的计算机，页大小（page size）为32 Bytes，支持32KB的虚拟地址空间（virtual address space），有4KB的物理内存空间（physical memory），采用二级页表，一个页目录项（page directory entry，PDE）大小为1 Byte，一个页表项（page-table entries PTEs）大小为1 Byte，1个页目录表大小为32 Bytes，1个页表大小为32 Bytes。页目录基址寄存器（page directory base register，PDBR）保存了页目录表的物理地址（按页对齐）。

PTE格式（8 bit）：

```
VALID | PFN6 ... PFN0
```

PDE格式（8 bit）：

```
VALID | PT6 ... PT0
```

其

```
VALID==1表示，表示映射存在；VALID==0表示，表示映射不存在。
PFN6..0:页帧号
PT6..0:页表的物理基址>>5
```

在物理内存模拟数据文件中，给出了4KB物理内存空间的值，请回答下列虚地址是否有合法对应的物理内存，请给出对应的pde index, pde contents, pte index, pte contents。

```
Virtual Address 6c74
Virtual Address 6b22
Virtual Address 03df
Virtual Address 69dc
Virtual Address 317a
Virtual Address 4546
Virtual Address 2c03
Virtual Address 7fd7
Virtual Address 390e
Virtual Address 748b
```

比如答案可以如下表示：

```
Virtual Address 7570:
  --> pde index:0x1d  pde contents:(valid 1, pfn 0x33)
    --> pte index:0xb  pte contents:(valid 0, pfn 0x7f)
      --> Fault (page table entry not valid)

Virtual Address 21e1:
  --> pde index:0x8  pde contents:(valid 0, pfn 0x7f)
      --> Fault (page directory entry not valid)

Virtual Address 7268:
  --> pde index:0x1c  pde contents:(valid 1, pfn 0x5e)
    --> pte index:0x13  pte contents:(valid 1, pfn 0x65)
      --> Translates to Physical Address 0xca8 --> Value: 16
```

（3）请基于你对原理课二级页表的理解，并参考Lab2建页表的过程，设计一个应用程序（可基于python, ruby, C, C++, LISP等）可模拟实现(2)题中描述的抽象OS，可正确完成二级页表转换。

（4）假设你有一台支持反置页表的机器，请问你如何设计操作系统支持这种类型计算机？请给出设计方案。

(5)X86的页面结构

扩展思考题

阅读64bit IBM Powerpc CPU架构是如何实现反置页表，给出分析报告。

lab2在线练习

选择题

80386 CPU保护模式下的特权级个数是() s1

- [ ] 1
- [ ] 2
- [ ] 3
- [x] 4

> ring0-ring3

ucore OS中使用了的80386 CPU保护模式下的特权级的级别包括() s1

- [x] 0
- [ ] 1
- [ ] 2
- [x] 3

> ring 0 for OS， ring3 for application

在ucore OS的管理下，如果CPU在ring3特权级执行访存指令，读属于ring0特权级的数据段中的内存单元，将出现的情况是（） s1

- [ ] 产生外设中断
- [x] 产生访存异常
- [ ] CPU继续正常执行
- [ ] 系统重启

> 将产生General Protection Fault异常

段描述符中与特权级相关的一个组成部分的名称是（） s1

- [x] DPL
- [ ] AVL
- [ ] Base
- [ ] Limit

> 是DPL

CS段寄存器中的最低两位保存的是（） s1

- [ ] DPL
- [x] CPL
- [ ] RPL
- [ ] NPL

> 是CPL

DS段寄存器中的最低两位保存的是（） s1

- [ ] DPL
- [ ] CPL
- [x] RPL
- [ ] NPL

> 是RPL

CPU执行一条指令访问数据段时，硬件要做的特权级检查是（） s1

- [x] MAX(CPL, RPL) <= DPL[数据段]
- [ ] MIN(CPL, RPL) <= DPL[数据段]
- [ ] MAX(CPL, DPL) <= RPL[数据段]
- [ ] MIN(CPL, DPL) <= RPL[数据段]

> 是MAX(CPL, RPL) <= DPL[数据段]

对于Task State Segment（TSS）而言，uCore OS可以利用它做（） s2

- [x] 保存ring 0的SS
- [x] 保存ring 0的ESP
- [ ] 保存中断描述符表的基址
- [ ] 保存全局描述符表的基址

> 是保存ring 0的SS和ESP

页目录表的基址是保存在寄存器（） s3

- [ ] CR0
- [ ] CR1
- [ ] CR2
- [x] CR3

> CR3

在启动页机制后，不可能进行的操作包括（） s3

- [x] 取消段机制，只保留页机制
- [ ] 取消页机制，只保留段机制
- [ ] 取消页机制，也取消段机制
- [ ] 保留页机制，也保留段机制

> 不可能取消段机制，只保留页机制

给定一个虚页地址和物理页地址，在建立二级页表并建立正确虚实映射关系的过程中，需要完成的事务包括() s4

- [x] 给页目录表动态分配空间，给页表分配空间
- [x] 让页基址寄存器的高20位内容为页目录表的高20位物理地址
- [x] 在虚地址高10位的值为index的页目录项中的高20位填写页表的高20位物理地址，设置有效位
- [x] 在虚地址中10位的值为index的页表项中中的高20位填写物理页地址的高20位物理地址，设置有效位

> 都对，还要设置更多的一些属性。

---

lab2 SPOC思考题

NOTICE

- 有"w4l1"标记的题是助教要提交到学堂在线上的。
- 有"w4l1"和"spoc"标记的题是要求拿清华学分的同学要在实体课上完成，并按时提交到学生对应的git repo上。
- 有"hard"标记的题有一定难度，鼓励实现。
- 有"easy"标记的题很容易实现，鼓励实现。
- 有"midd"标记的题是一般水平，鼓励实现。

个人思考题

x86保护模式中权限管理无处不在，下面哪些时候要检查访问权限() (w4l1)

- [x] 内存寻址过程中
- [x] 代码跳转过程中
- [x] 中断处理过程中
- [ ] ALU计算过程中

> 前三个需要。这里假定ALU完成计算所需数据都已经在CPU内部了。

请描述ucore OS建立页机制的准备工作包括哪些步骤？ (w4l1)

```
+ 采分点：说明了ucore OS在让页机制正常工作的主要准备工作
- 答案没有涉及如下3点；（0分）
- 描述了对GDT的初始化,完成了段机制（1分）
- 除第二点外进一步描述了对物理内存的探测和空闲物理内存的管理。（2分）
- 除上述两点外，进一步描述了页表建立初始过程和设置CR0控寄存器某位来使能页（3分）
```

- [x]

小组思考题

（1）（spoc）请用lab1实验的基准代码（即没有修改的需要填空的源代码）来做如下实验： 执行`make qemu`，会得到一个输出结果，请给出合理的解释：为何qemu退出了？【提示】需要对qemu增加一些用于基于执行过的参数，重点是分析其执行的指令和产生的中断或异常。

- [x]

（2）(spoc)假定你已经完成了lab1的实验,接下来是对lab1的中断处理的回顾：请把你的学号对37(十进制)取模，得到一个数x（x的范围是-1<x<37），然后在你的答案的基础上，修init.c中的kern_init函数，在大约36行处，即

```
    intr_enable();              // enable irq interrupt
```

语句之后，加入如下语句(把x替换为你学号 mod 37得的值)：

```
    asm volatile ("int $x");
```

然后，请回答加入这条语句后，执行`make qemu`的输出结果与你没有加入这条语句后执行`make qemu`的输出结果的差异，并解释为什么有差异或没差异？

- [x]

（3）对于lab2的输出信息，请说明数字的含义

```
e820map:
  memory: 0009fc00, [00000000, 0009fbff], type = 1.
  memory: 00000400, [0009fc00, 0009ffff], type = 2.
  memory: 00010000, [000f0000, 000fffff], type = 2.
  memory: 07ee0000, [00100000, 07fdffff], type = 1.
  memory: 00020000, [07fe0000, 07ffffff], type = 2.
  memory: 00040000, [fffc0000, ffffffff], type = 2.
```

修改lab2，让其显示`type="some string"`让人能够读懂，而不是不好理解的数字1,2 (easy)

- [x]

（4）(spoc)有一台只有页机制的简化80386的32bit计算机，有地址范围位0~256MB的物理内存空间（physical memory），可表示大小为256MB，范围为0xC0000000~0xD0000000的虚拟地址空间（virtual address space）,页大小（page size）为4KB，采用二级页表，一个页目录项（page directory entry，PDE）大小为4B,一个页表项（page-table entries PTEs）大小为4B，1个页目录表大小为4KB，1个页表大小为4KB。

```
PTE格式（32 bit） :
  PFN19 ... PFN0|NOUSE9 ... NOUSE0|WRITABLE|VALID
PDE格式（32 bit） :
  PT19 ... PT0|NOUSE9 ... NOUSE0|WRITABLE|VALID

其中：
NOUSE9 ... NOUSE0为保留位，要求固定为0
WRITABLE：1表示可写，0表示只读
VLAID：1表示有效，0表示无效
```

假设ucore OS已经为此机器设置好了针对如下虚拟地址<-->物理地址映射的二级页表，设置了页目录基址寄存器（page directory base register，PDBR）保存了页目录表的物理地址（按页对齐），其值为0。已经建立好了从物理地址0x1000~0x41000的二级页表，且页目录表的index为0x300~0x363的页目录项的(PT19 ... PT0)的值=(index-0x300+1)。请写出一个translation程序（可基于python, ruby, C, C++，LISP等），输入是一个虚拟地址和一个物理地址，能够自动计算出对应的页目录项的index值,页目录项内容的值，页表项的index值，页表项内容的值。即(pde_idx, pde_ctx, pte_idx, pte_cxt)

请用如下值来验证你写的程序的正确性：

```
va 0xc2265b1f, pa 0x0d8f1b1f
va 0xcc386bbc, pa 0x0414cbbc
va 0xc7ed4d57, pa 0x07311d57
va 0xca6cecc0, pa 0x0c9e9cc0
va 0xc18072e8, pa 0x007412e8
va 0xcd5f4b3a, pa 0x06ec9b3a
va 0xcc324c99, pa 0x0008ac99
va 0xc7204e52, pa 0x0b8b6e52
va 0xc3a90293, pa 0x0f1fd293
va 0xce6c3f32, pa 0x007d4f32
```

参考的输出格式为：

```
va 0xcd82c07c, pa 0x0c20907c, pde_idx 0x00000336, pde_ctx  0x00037003, pte_idx 0x0000002c, pte_ctx  0x0000c20b
```

- [x]

开放思考题

（1）请简要分析Intel的x64 64bit体系结构下的分页机制是如何实现的

```
+ 采分点：说明Intel x64架构的分页机制的大致特点和页表执行过程
- 答案没有涉及如下3点；（0分）
- 正确描述了x64支持的物理内存大小限制（1分）
- 正确描述了x64下的多级页表的级数和多级页表的结构（2分）
- 除上述两点外，进一步描述了在多级页表下的虚拟地址-->物理地址的映射过程（3分）
```

- [x]

（2）Intel8086不支持页机制，但有hacker设计过包含未做任何改动的8086CPU的分页系统。猜想一下，hacker是如何做到这一点的？提示：想想MMU的逻辑位置

- [x]

Virtual Memory Management

单选题

(2012联考)下列关于虚拟存储器的叙述中，正确的是（ ）

- [ ] 虚拟存储只能基于连续分配技术
- [x] 虚拟存储只能基于非连续分配技术
- [ ] 虚拟存储容量只受外存容量的限制
- [ ] 虚拟存储容量只受内容容量的限制

> 采用连续分配方式的时候，会使得相当一部分内存空间都处于空闲状态，造成内存资源的严重浪费，无法从逻辑上扩大内存容量。

（2011年联考）在缺页处理过程中，操作系统执行的操作可能是（ ） 1)修改页表 2)磁盘I/O 3)分配页框

- [ ] 仅1、2
- [ ] 仅2、3
- [ ] 仅1、3
- [x] 1、2、3

> 如果还有可分配给程序的内存，那么会分配新的页框，修改页表，从磁盘读取内容放入到分配的页框中。

（2013计算机联考）若系统发生抖动(Thrashing)时，可用采取的有效措施是（ ） 1）撤销部分进程 2）增加磁盘交换区的容量 3）提高用户进程的优先级

- [x] 仅1
- [ ] 仅2
- [ ] 仅3
- [ ] 仅1、2

> 撤销部分进程可以增大可用内存，减少抖动。而磁盘交换区容量和进程优先级则跟抖动无关。

(南昌大学)一个虚拟存储器系统中，主存容量16MB，辅存容量1GB，地址寄存器位数32位。那么虚存最大容量为（ ）

- [ ] 1GB
- [ ] 16MB
- [ ] 1GB + 16MB
- [x] 4GB

> 虚拟存储器的最大容量跟虚拟地址空间有关，是2^32。

（上海交通大学）分页式虚拟存储管理系统中，分页是（ ）实现的

- [ ] 程序员
- [ ] 编译器
- [ ] 系统调用
- [x] 系统

> 分页是系统内部实现的，对用户透明。

为了使得内存需求较大的程序能够正常运行，常需要通过外存和内存的交换技术，这被叫做__技术

- [ ] 虚拟机
- [ ] 内存分配
- [ ] 进程调度
- [x] 虚拟存储

> 解释：虚拟机用于模拟真实物理机器，单独的内存分配技术可以不考虑使用外存，进程调度则用于管理进程的执行时间和次序等。虚拟存储是指当真实内存不能满足需求的时候，可以将程序需要的代码和数据放到内存中，暂时不需要的放到外存上；通过内存和外存的不断交换，来满足程序的运行需求。

虚拟内存是为了应对__的问题（）

- [ ] 内存访问速度过慢
- [ ] 内存管理困难
- [x] 内存容量不满足程序需求
- [ ] 磁盘访问过慢

> 解释：虚拟内存是应对内存容量不能满足程序需求的情况，并不能解决内存内存和外存访问速度的问题。

一般来讲，虚拟内存使得程序的运行速度__

- [ ] 加快
- [ ] 不变
- [x] 变慢
- [ ] 变得极不稳定

> 解释：由于虚拟内存有可能造成外存和内存的不断交换，虽然能够满足大程序的运行需求，但是程序的运行速度相比没有虚拟内存的情况下会变慢。

虚拟内存常用的页面淘汰技术，主要利用了程序的__特征

- [ ] 健壮性
- [ ] 完整性
- [x] 局部性
- [ ] 正确性

> 解释：程序的局部性是指程序呈现在某段时间内只访问程序的某一部分代码和数据的特性，而页面置换算法可以利用这一特性使常被访问的页面不被淘汰也就减少了缺页率。

在一个系统中，页面大小设定为4k，分配给每个进程的物理页面个数为1，在某应用程序中需要访问一个int[1024][1024]的数组（逐行访问），那么按行存储和按列存储的不同情况下，__

- [x] 按行存储时，执行效率高
- [ ] 按列存储时，执行效率高
- [ ] 执行效率相同
- [ ] 执行效率不确定

> 解释：按行存储的时候，每访问一行才会出现一次页面置换；而按列存储，则每访问一次就发生一次缺页。由于缺页的时候需要调用页面置换算法进行内外存交换，所以缺页率高的时候效率就低。

虚拟内存技术__

- [ ] 只能应用于分段系统
- [ ] 只能应用于分页系统

- [x] 可应用于分段系统、分页系统
- [ ] 只能应用于段页式系统

> 解释：虚拟内存技术是一种内外存交换的思想，可应用与分段系统、分页系统等。而目标系统是分段系统还是分页系统，影响的只是虚拟内存技术思想的具体实现。

在虚拟页式内存管理系统中，页表项中的‘访问位’给__提供参考价值。

- [ ] 分配页面
- [x] 页面置换算法
- [ ] 换出页面
- [ ] 程序访问

> 解释：页面置换算法可能需要根据不同页面是否被访问，访问时间和访问频率等进行淘汰页面的选择。

在虚拟页式内存管理系统中，页表项中的‘修改位’供__使用

- [ ] 分配页面
- [ ] 页面置换算法
- [x] 换出页面
- [ ] 程序访问

> 解释：页面换出的时候，需要判断外存上的相应页面是否需要重写。如果内存中该页面在使用期间发生了修改，则相应的修改位被设置，用于换出的时候通知操作系统进行外存相应页面的修改。

在虚拟页式内存管理系统中，页表项中的__供程序访问时使用

- [ ] 访问位
- [ ] 修改位
- [x] 状态位
- [ ] 保护位

> 解释：页表项的状态位用于指示该页是否已经调入内存，供程序访问时使用，如果发现该页未调入内存，则产生缺页中断，由操作系统进行相应处理。

在虚拟页式内存管理系统中，发生缺页的概率一般取决于__

- [ ] 内存分配算法
- [ ] 内存读取速度
- [ ] 内存写入速度
- [x] 页面置换算法

> 解释：缺页率的高低跟实际能分配的物理内存的大小，以及系统中的页面置换算法相关。差的页面置换算法可能造成需要访问的页面经常没有在内存中，而需要进行缺页中断处理。

页面置换算法的优劣，表现在__

- [ ] 程序在运行时能够分配到的页面数
- [ ] 单位时间内，程序在运行时得到的CPU执行时间
- [x] 程序在运行时产生的页面换入换出次数
- [ ] 程序本身的访存指令个数

> 解释：页面置换算法在满足程序运行需求的同时，应尽量降低页面的置换次数，从而降低运行开销。

选择在将来最久的时间内不会被访问的页面作为换出页面的算法叫做__

- [x] 最优页面置换算法
- [ ] LRU
- [ ] FIFO
- [ ] CLOCK

> 解释：LRU是换出在过去的时间里最久未被访问的页面；FIFO是换出最先被换入的页面；CLOCK类似于LRU，也是对FIFO的改进。但是以上三种算法都是根据过去一段时间内的页面访问规律进行换出页面的选择。而最优页面置换算法是指换出将来在最久的时间内不会被访问的页面，是一种理想情况也是不可能实现的。

Belady异常是指__

- [ ] 频繁的出页入页现象
- [x] 分配的物理页数变多，缺页中断的次数却增加
- [ ] 进程的内存需求过高，不能正常运行
- [ ] 进程访问内存的时间多于读取磁盘的时间

> 解释：一般情况下，分配的物理页数越多，缺页率会越低。但是某些页面置换算法如FIFO就可能造成相反的情况，也即分配的物理页数增多，缺页率却增高的情况。这种情况称为Belady异常。

在各种常见的页面置换算法中，__会出现Belady异常现象

- [x] FIFO
- [ ] LRU
- [ ] LFU
- [ ] CLOCK

> 解释：FIFO可能出现Belady异常，如访问顺序1,2,3,4,1,2,5,1,2,3,4,5，在最多分配3个物理块的情况下缺页9次，而在最多分配4个物理块的情况下缺页10次。

当进程访问的页面不存在，且系统不能继续给进程分配物理页面的时候，系统处理过程为__

- [ ] 确定换出页面->页面换出->页面换入->缺页中断
- [ ] 缺页中断->页面换入->确定换出页面->页面换出
- [ ] 缺页中断->确定换出页面->页面换入->页面换出
- [x] 缺页中断->确定换出页面->页面换出->页面换入

> 解释：首先在程序访问的时候发现页面不在内存中，从而发出缺页中断，进入页面置换的流程。需要确定换出页面才能执行页面交换，而页面换入之前要保证页面已经正确的换出，因为页面换出可能需要重写外存中相应的页面。

某进程的页面访问顺序为1、3、2、4、2、3、1、2，系统最多分配3个物理页面，那么采用LRU算法时，进程运行过程中会发生__缺页。

- [ ] 三次
- [ ] 四次
- [x] 五次
- [ ] 六次

> 解释：1（缺页）-3（缺页）-2（缺页）-4（缺页，换出1）-2-3-1（缺页，换出4）-2

在现代提供虚拟内存的系统中，用户的逻辑地址空间__

- [ ] 不受限制
- [ ] 受物理内存空间限制
- [ ] 受页面大小限制
- [x] 受指令地址结构

> 解释：逻辑地址空间受到逻辑地址的结构限制，也即为指令地址的结构限制。

多选题

以下哪些页面置换算法是可以实现的__

- [ ] 最优页面置换算法
- [x] LRU
- [x] FIFO
- [x] CLOCK

> 解释：最优页面置换算法是根据将来的页面访问次序来选择应该换出的页面，因为在程序执行之前不可能已知将来的页面访问次序，所以不可能实现。而其它的页面置换算法则是根据已经发生的页面访问次序来决定换出的页面，都是可以实现的。

影响缺页率的因素有__

- [x] 页面置换算法
- [x] 分配给进程的物理页面数
- [x] 页面本身的大小
- [x] 程序本身的编写方法

> 解释：总体来讲，缺页率的主要影响因素的页面置换算法和分配给进程的物理页面数。但是页面本身的大小和程序本身的编写方法则涉及到页面访问次序的变化，对缺页率也会造成影响。

判断题

发生缺页的时候，一定会使用页面置换算法__

- [ ] 对
- [x] 错

> 解释：发生缺页的时候，如果分配给程序的物理页面数还有空闲，则直接换入新的页面，不需要使用页面置换算法来挑选需要换出的页面。

lec8 虚拟内存spoc练习

NOTICE

- 有"w4l2"标记的题是助教要提交到学堂在线上的。
- 有"w4l2"和"spoc"标记的题是要求拿清华学分的同学要在实体课上完成，并按时提交到学生对应的git repo上。
- 有"hard"标记的题有一定难度，鼓励实现。
- 有"easy"标记的题很容易实现，鼓励实现。
- 有"midd"标记的题是一般水平，鼓励实现。

个人思考题

内存访问局部性的应用程序例子

(1)(w4l2)下面是一个体现内存访问局部性好的简单应用程序例子，请参考，在linux中写一个简单应用程序，体现内存局部性差，并给出其执行时间。

```
#include <stdio.h>
#define NUM 1024
#define COUNT 10
int A[NUM][NUM];
void main (void) {
  int i,j,k;
  for (k = 0; k<COUNT; k++)
  for (i = 0; i < NUM; i++)
  for (j = 0; j     < NUM; j++)
      A[i][j] = i+j;
  printf("%d count computing over!\n",i*j*k);
}
```

可以用下的命令来编译和运行此程序：

```
gcc -O0 -o goodlocality goodlocality.c
time ./goodlocality
```

可以看到其执行时间。

小组思考题目

缺页异常嵌套

（1）缺页异常可用于虚拟内存管理中。如果在中断服务例程中进行缺页异常的处理时，再次出现缺页异常，这时计算机系统（软件或硬件）会如何处理？请给出你的合理设计和解释。

缺页中断次数计算

（2）如果80386机器的一条机器指令(指字长4个字节)，其功能是把一个32位字的数据装入寄存器，指令本身包含了要装入的字所在的32位地址。这个过程最多会引起几次缺页中断？

> 提示：内存中的指令和数据的地址需要考虑地址对齐和不对齐两种情况。需要考虑页目录表项invalid、页表项invalid、TLB缺失等是否会产生中断？

虚拟页式存储的地址转换

（3）(spoc) 有一台假想的计算机，页大小（page size）为32 Bytes，支持8KB的虚拟地址空间（virtual address space）,有4KB的物理内存空间（physical memory），采用二级页表，一个页目录项（page directory entry，PDE）大小为1 Byte,一个页表项（page-table entries PTEs）大小为1 Byte，1个页目录表大小为32 Bytes，1个页表大小为32 Bytes。页目录基址寄存器（page directory base register，PDBR）保存了页目录表的物理地址（按页对齐）。

PTE格式（8 bit）：

```
VALID | PFN6 ... PFN0
```

PDE格式（8 bit）：

```
VALID | PT6 ... PT0
```

其

```
VALID==1表示，表示映射存在；VALID==0表示，表示内存映射不存在（有两种情况：a.对应的物理页帧swap out在硬盘上；b.既没有在内存中，页没有在硬
PFN6..0:页帧号或外存中的后备页号
PT6..0:页表的物理基址>>5
```

已经建立好了1个页目录表和8个页表，且页目录表的index为0~7的页目录项分别对应了这8个页表。

在物理内存模拟数据文件中，给出了4KB物理内存空间和4KBdisk空间的值，PDBR的值。

请回答下列虚地址是否有合法对应的物理内存，请给出对应的pde index, pde contents, pte index, pte contents，the value of addr in phy page OR disk sector。

```
Virtual Address 6653:
Virtual Address 1c13:
Virtual Address 6890:
Virtual Address 0af6:
Virtual Address 1e6f:
```

提示：

```
页大小（page size）为32 Bytes(2^5)
页表项1B

8KB的虚拟地址空间(2^13)
一级页表：2^5
PDBR content: 0xd80（1101_100 0_0000, page 0x6c）

page 6c: e1(1110 0001) b5(1011 0101) a1(1010 0001) c1(1100 0001)
         b3(1011 0011) e4(1110 0100) a6(1010 0110) bd(1011 1101)
二级页表：2^5
页内偏移：2^5

4KB的物理内存空间（physical memory）(2^12)
物理帧号：2^7

Virtual Address 0330(0 00000 11001 1_0000):
```

```
   --> pde index:0x0(00000)  pde contents:(0xe1, 11100001, valid 1, pfn 0x61(page 0x61))
  page 6c: e1 b5 a1 c1 b3 e4 a6 bd 7f 7f 7f 7f 7f 7f 7f 7f
           7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f
  page 61: 7c 7f 7f 4e 4a 7f 3b 5a 2a be 7f 6d 7f 66 7f a7
           69 96 7f c8 3a 7f a5 83 07 e3 7f 37 62 30 7f 3f
    --> pte index:0x19(11001)  pte contents:(0xe3, 1 110_0011, valid 1, pfn 0x63)
  page 63: 16 00 0d 15 00 1c 1d 16 02 02 0b 00 0a 00 1e 19
           02 1b 06 06 14 1d 03 00 0b 00 12 1a 05 03 0a 1d
      --> To Physical Address 0xc70(110001110000, 0xc70) --> Value: 02

Virtual Address 1e6f(0 001_11 10_011 0_1111):
  --> pde index:0x7(00111)  pde contents:(0xbd, 10111101, valid 1, pfn 0x3d)
  page 6c: e1 b5 a1 c1 b3 e4 a6 bd 7f 7f 7f 7f 7f 7f 7f 7f
           7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f 7f
  page 3d: f6 7f 5d 4d 7f 04 29 7f 1e 7f ef 51 0c 1c 7f 7f
           7f 76 d1 16 7f 17 ab 55 9a 65 ba 7f 7f 0b 7f 7f
    --> pte index:0x13  pte contents:(0x16, valid 0, pfn 0x16)
  disk 16: 00 0a 15 1a 03 00 09 13 1c 0a 18 03 13 07 17 1c
           0d 15 0a 1a 0c 12 1e 11 0e 02 1d 10 15 14 07 13
      --> To Disk Sector Address 0x2cf(0001011001111) --> Value: 1c
```

扩展思考题

(1)请分析原理课的缺页异常的处理流程与lab3中的缺页异常的处理流程（分析粒度到函数级别）的异同之处。

(2)在X86-32虚拟页式存储系统中，假定第一级页表的起始地址是0xE8A3 B000，进程地址空间只有第一级页表的4KB在内存。请问这4KB的虚拟地址是多少？它对应的第一级页表项和第二级页表项的物理地址是多少？页表项的内容是什么？

lec9 在线练习

选择题

物理页帧数量为3，虚拟页访问序列为 0,1,2,0,1,3,0,3,1,0,3，请问采用最优置换算法的缺页次数为（） s2

- [ ] 1
- [ ] 2
- [ ] 3
- [x] 4

> 4

物理页帧数量为3，虚拟页访问序列为 0,1,2,0,1,3,0,3,1,0,3，请问采用LRU置换算法的缺页次数为（） s2

- [ ] 1
- [ ] 2
- [ ] 3
- [x] 4

> 4

物理页帧数量为3，虚拟页访问序列为 0,1,2,0,1,3,0,3,1,0,3，请问采用FIFO置换算法的缺页次数为（） s2

- [ ] 1
- [ ] 2
- [ ] 4
- [x] 6

> 6

物理页帧数量为4，虚拟页访问序列为 0,3,2,0,1,3,4,3,1,0,3,2,1,3,4 ，请问采用CLOCK置换算法（用1个bit表示存在时间）的缺页次数为（） s3

- [ ] 8
- [x] 9
- [ ] 10
- [ ] 11

> 9

物理页帧数量为4，虚拟页访问序列为 0,3,2,0,1,3,4,3,1,0,3,2,1,3,4 ，请问采用CLOCK置换算法（用2个bit表示存在时间）的缺页次数为（） s3

- [ ] 8
- [ ] 9
- [x] 10
- [ ] 11

> 10

虚拟页访问序列为 1,2,3,4,1,2,5,1,2,3,4,5，物理页帧数量为3和4，采用FIFO置换算法，请问是否会出现bealdy现象() s4

- [x] 会
- [ ] 不会

> 3页时9次缺页，4页时10次缺页。

下面哪些页面淘汰算法不会产生Belady异常现象 s4

- [ ] 先进先出页面置换算法（FIFO)
- [ ] 时钟页面置换算法（CLOCK)
- [x] 最佳页面置换算法（OPT）
- [x] 最近最少使用页面置换算法（LRU）

> LRU和OPT属于一种栈算法

物理页帧数量为5，虚拟页访问序列为 4,3,0,2,2,3,1,2,4,2,4,0,3，请问采用工作集置换算法（工作集窗口T=4）的缺页次数为（） s5

- [ ] 2
- [ ] 3
- [ ] 4
- [x] 5

> 5

物理页帧数量为5，虚拟页访问序列为 4,3,0,2,2,3,1,2,4,2,4,0,3，请问采用缺页率置换算法（窗口T=2）的缺页次数为（） s6

- [ ] 2
- [ ] 3
- [ ] 4
- [x] 5

> 5

lec9 虚存置换算法spoc练习

个人思考题

1. 置换算法的功能？
2. 全局和局部置换算法的不同？
3. 最优算法、先进先出算法和LRU算法的思路？
4. 时钟置换算法的思路？
5. LFU算法的思路？
6. 什么是Belady现象？
7. 几种局部置换算法的相关性：什么地方是相似的？什么地方是不同的？为什么有这种相似或不同？
8. 什么是工作集？
9. 什么是常驻集？
10. 工作集算法的思路？
11. 缺页率算法的思路？
12. 什么是虚拟内存管理的抖动现象？
13. 操作系统负载控制的最佳状态是什么状态？

小组思考题目

(1)（spoc）请证明为何LRU算法不会出现belady现象

(2)（spoc）根据你的`学号 mod 4`的结果值，确定选择四种替换算法（0：LRU置换算法，1:改进的clock 页置换算法，2：工作集页置换算法，3：缺页率置换算法）中的一种来设计一个应用程序（可基于python, ruby, C, C++，LISP等）模拟实现，并给出测试。请参考如python代码或独自实现。

- 页置换算法实现的参考实例

扩展思考题

（1）了解LIRS页置换算法的设计思路，尝试用高级语言实现其基本思路。此算法是江松博士（导师：张晓东博士）设计完成的，非常不错！

参考信息：

- LIRS conf paper
- LIRS journal paper
- LIRS-replacement ppt1-LIRS-replacement.pdf)

- LIRS-replacement ppt2

lab3 在线练习

选择题

lab3中虚存管理需要直接借助的机制包括() s1

- [x] 页映射机制
- [ ] 段映射机制
- [x] 中断异常处理机制
- [x] IDE硬盘读写机制

> 段映射机制不直接需要

lab3中实现虚存管理的过程包括() s2

- [x] 实现对硬盘swap分区的读写
- [x] 建立处理页访问错误的异常/中断服务例程
- [x] 实现页替换算法
- [x] 定义不在物理内存中的“合法”虚拟页

> 都包括

lab3中用于描述“合法”虚拟页的数据结构是（）s3

- [x] vma_struct
- [ ] trapframe
- [ ] gatedesc
- [ ] segdesc

> vma_struct

lab3中访问“合法”虚拟页产生缺页异常的原因是（）s4

- [x] 页表项的P bit为0
- [ ] 页目录项的I/D bit为0
- [ ] 页表项的U/S bit为0
- [ ] 页目录项的W/R bit位0

> 页表项的P bit为0，表示此页不存在

lab3中把扇区索引信息放在（）s5

- [x] 页表项中
- [ ] 页目录项中
- [ ] 内存中的Page结构中
- [ ] 内存中的vma_struct结构中

> 页表项中的高24位

lab3 SPOC思考题

=======

lab3 SPOC思考题

NOTICE

- 有"w5l2"标记的题是助教要提交到学堂在线上的。
- 有"w5l2"和"spoc"标记的题是要求拿清华学分的同学要在实体课上完成，并按时提交到学生对应的git repo上。
- 有"hard"标记的题有一定难度，鼓励实现。
- 有"easy"标记的题很容易实现，鼓励实现。
- 有"midd"标记的题是一般水平，鼓励实现。

个人思考题

10.1 实验目标：虚存管理

(1)缺页异常的处理流程？

(2)从外存的页面的存储和访问代码？

(3)缺页和页访问非法的返回地址有什么不同？

> 硬件设置、软件可修改； 中断号是

(4)虚拟内存管理中是否用到了段机制

(5)ucore如何知道页访问异常的地址？

10.2 回顾历史和了解当下

(6)中断处理例程的段表在GDT还是LDT？

(7)物理内存管理的数据结构在哪？

(8)页表项的结构？

(9)页表项的修改代码？

(10)如何设置一个虚拟地址到物理地址的映射关系？

(11)为了建立虚拟内存管理，需要在哪个数据结构中表示“合法”虚拟内存

10.3 处理流程、关键数据结构和功能

(12)swap_init()做了些什么？

(13)vmm_init()做了些什么？

(14)vma_struct数据结构的功能？

(15)mmap_list是什么列表？

(16)外存中的页面后备如何找到？

(17)vma_struct和mm_struct的关系是什么？

> 合法的连续虚拟地址区域、整个进程的地址空间

(18)画数据结构图，描述进程的虚拟地址空间、页表项、物理页面和后备页面的关系；

10.4 页访问异常

(19)页面不在内存和页面访问非法的处理中有什么区别？对应的代码区别在哪？

(20)find_vma()做了些什么？

(21)swapfs_read()做了些什么？

(22)缺页时的页面创建代码在哪？

(23)struct rb_tree数据结构的原理是什么？在虚拟管理中如何用它的？

(24)页目录项和页表项的dirty bit是何时，由谁置1的？

(25)页目录项和页表项的access bit是何时，由谁置1的？

10.5 页换入换出机制

(26)虚拟页与磁盘后备页面的对应有关系？

(27)如果在开始加载可执行文件时，如何改？

(28)check_swap()做了些什么检查？

(29)swap_entry_t数据结构做什么用的？放在什么地方？

(30)空闲物理页面的组织数据结构是什么？

(21)置换算法的接口数据结构？

> swap_manager

==========

小组思考题

(1)(spoc) 请参考lab3_result的代码，思考如何在lab3_results中实现clock算法，并给出你的概要设计方案，可4人一个小组，说明你的方案中clock算法与LRU算法上相比，潜在的性能差异性。并进一说明LRU算法在lab3实现的可能性评价（给出理

由）。

(2)(spoc) 理解内存访问的异常。在x86中内存访问会受到段机制和页机制的两层保护，请基于lab3_results的代码（包括lab1的challenge练习实现），请实践并分析出段机制和页机制各种内存非法访问的后果。，可4人一个小组，，找出尽可能多的各种内存访问异常，并在代码中给出实现和测试用例，在执行了测试用例后，ucore能够显示出是出现了哪种异常和尽量详细的错误信息。请在说明文档中指出：某种内存访问异常的原因，硬件的处理过程，以及OS如何处理，是否可以利用做其他有用的事情（比如提供比物理空间更大的虚拟空间）？哪些段异常是否可取消，并用页异常取代？

进程、线程管理

单选题

(2010年计算机联考真题)下列选项中，导致创建新进程的操作是（） 1)用户登陆成功 2)设备分配 3)启动程序执行

- [ ] 仅1和2
- [x] 仅2和3
- [ ] 仅1和3
- [ ] 1、2、3

> 解释：设备分配是通过系统中设置相应的数据结构实现的，不需要创建进程

（2012年计算机联考真题）下列关于进程和线程的叙述中，正确的是（）

- [x] 不管系统是否支持线程，进程都是资源分配的基本单位
- [ ] 线程是资源分配的基本单元，进程是调度的基本单位
- [ ] 系统级线程和用户级线程的切换都需要内核的支持
- [ ] 同一进程中的各个线程拥有各自不同的地址空间

> 解释：引入线程的操作系统中，通常都是把进程作为资源分配的基本单位，而把线程作为独立运行的基本单位。同一进程中的各个线程都可以共享进程所拥有的系统资源，这表现在所有线程都有相同的地址空间。对于用户级线程的切换，通常是发生在一个应用进程的诸多线程之间，这时，也同样无须内核的支持

(2010年计算机联考真题)下列选项中，降低进程优先级的合理时机是（）

- [x] 进程时间片用完
- [ ] 进程刚完成I/O操作，进入就绪队列
- [ ] 进程长期处于就绪队列
- [ ] 进程从就绪状态转为运行状态

> 解释：进程时间片用完，从执行态进入就绪态应降低优先级以让别的进城那个调度进入执行状态，B中进程刚完成I/O，进入就绪队列后应该等待被处理器调度，故应提高优先级，C中类似，D中不应该降低，应该在时间片用完后再降低

（上海交通大学）OS对（）分配内存资源

- [ ] 线程
- [ ] 高速缓冲存储器
- [x] 进程
- [ ] 快表

> 解释：进程是系统资源分配的基本单位，线程是调度的基本单位，高速缓冲存储器和快表都是硬件

(四川大学)一进程基本状态可以从其他两种基本状态转变过去，这个基本状态一定是（）

- [ ] 执行状态
- [ ] 阻塞状态
- [x] 就绪状态
- [ ] 完成状态

> 解释：处于就绪状态的进程，已具备了运行条件，但由于未能获得CPU，故仍不能运行，就绪状态可以从运行状态和阻塞状态转换得到

（上海交通大学）下列说法（）不是创建进程必须的

- [ ] 建立一个进程的进程表项
- [ ] 为进程分配内存
- [x] 为进程分配CPU
- [ ] 将进程表项放入就绪队列

> 解释：进程刚被创建时，实际上是处于就绪状态的，所以不需为进程分配CPU

（2011年全国统考）在支持多线程的系统中，进程P创建的若干个线程不能共享的是（）

- [ ] 进程P的代码段
- [ ] 进程P打开的文件
- [ ] 进程P的全局变量
- [x] 进程P中某线程的栈指针

> 解释：多线程系统中，一个进程的多个线程共享进程的代码段、文件和全局变量，进程中某线程的栈指针是归该线程所独有，对其他线程透明，但不恩能够与其他线程共享。

（2011年全国统考）下列选项中，在用户态执行的是（）

- [x] 命令解释程序
- [ ] 缺页处理程序
- [ ] 进程调度层序
- [ ] 时钟中断处理程序

> 解释：缺页处理程序和时钟中断都属于中断，进程调度属于系统调用，均在核心态执行，命令解释程序属于命令借口，它在用户态执行

（南京理工大学）进程和程序之间有密切联系，但又有不同的概念，两者的一个本质区别是（）

- [x] 程序是静态概念，进程是动态概念
- [ ] 程序是动态概念，进程是静态概念
- [ ] 程序保存在文件中，进程存放在内存中
- [ ] 程序顺序执行，进程并发执行

> 解释：进程和程序的本质区别是程序是静态的，进程是动态的

（电子科技大学）若一进程拥有100个线程，这些线程属于用户级线程，则在系统调度执行时间上占用（）个时间片

- [x] 1
- [ ] 100
- [ ] 1/100
- [ ] 0

> 解释：在引入线程的系统中，资源仍然是按进程分配的，由于分配给该进程1个时间片，所以在执行时间上总共占1个时间片

(上海交通大学)一个进程被唤醒，意味着（）

- [x] 该进程可以重新占用CPU
- [ ] 优先级变为最大
- [ ] PCB移到就绪队列之首

- [ ] 进程变为运行态

> 解释：在一个进程被唤醒时，它将从阻塞状态变成就绪状态，从而可以重新获得CPU并投入运行

对进程的描述中，下列说法错误的是（）

- [x] 一个程序只对应一个进程
- [ ] 一个进程可以包含若干个程序
- [ ] 进程是有生命周期的
- [ ] 一个程序可以对应多个进程

> 解释：进程是执行中的程序，它是有生命周期的，程序本身不是进程，程序只是被动实体，一个程序可能会有多个进程相关

下列的进程状态变化中，()变化是不可能发生的

- [ ] 运行一等待
- [x] 等待一运行
- [ ] 等待一就绪
- [ ] 运行一就绪

> 解释： 进程状态是由当前活动所定义，运行状态表示指令正在被执行，等待状态表示进程等待某个事件的发生，就绪态表示进程等待分配处理器，由进程状态图我们可以看到等待状态无法直接转变成运行状态，需要从等待态先变成就绪态

一个运行的进程用完了分配给它的时间片后，它的状态变为（）

- [ ] 运行
- [ ] 等待
- [x] 就绪
- [ ] 终止

> 解释： 当一个进程用完了分配给它的时间片后，状态会变为就绪态，之后会继续等待分配处理器

将进程的（）连接在一起形成进程队列

- [ ] 堆栈段
- [ ] 数据段
- [ ] 堆
- [x] PCB

> 解释：进程调度选择一个可用的进程到CPU上执行，而进程进入洗头膏时，会被加到作业队列中，改队列包括系统中的所有进程，驻留在内存中就绪、等待运行的进程保存在就绪队列中，改队列通常用链表来实现，其头节点指向链表的第一个和最后一个PCB块的指针。

下列关于进程控制块的描述中，说法错误的是（）

- [ ] 进程控制块记录进程的状态及名称等
- [ ] 进程控制块位于主存储区内
- [x] 进程控制块对每个进程不止有一个
- [ ] 进程控制块的内容、格式及大小可能不同

> 解释：每个进程在操作系统内用一个进程控制块来表示，每个进程控制块都记录进程的状态及名称等，并且每个进程对应一个进程控制块，进程控制块的内容、格式及大小可能不同，并且进程控制快位于主存储区内

PCB是进程存在的唯一标志，下列（）不属于PCB

- [ ] 堆栈指针
- [ ] 全局变量
- [ ] 进程ID
- [x] CPU状态

> 解释：进程描述块包含许多与一个特定进程相关的信息，主要有：进程状态、程序计数器、CPU调度信息、内存管理信息、记账信息以及I/O状态信息。从题目中我们可以看出CPU状态信息并不包含在内。

对于标准的线程，下列叙述中，错误的是（）

- [ ] 进程中可以包含多个线程
- [ ] 线程并不拥有资源，只是使用他们
- [ ] 线程可以创建其他线程
- [x] 线程没有生命期

> 解释：线程依然有生命周期

（）系统调用是用来被父进程等待子进程结束的

- [x] wait()
- [ ] fork()
- [ ] exit()
- [ ] exec()

> 解释：当进程完成执行最后的语句并使用系统调用的exit()请求操作系统删除自身时，进程终止。这时，进程可以返回状态值到父进程，而这个父进程等待子进程结束的方法是通过父进程系统调用wait()

多个进程的实体能存在于同一内存中，在一段时间内都得到运行。这种性质称为进程的（）

- [ ] 动态性
- [ ] 调度性
- [x] 并发性
- [ ] 独立性

> 解释：概念题,进程有四个特性，动态性：进程的实质是程序在多道程序系统中的一次执行过程，进程是动态产生，动态消亡的；并发性：任何进程都可以同其他进程一起并发执行；独立性：进程是一个能独立运行的基本单位，同时也是系统分配资源和调度的独立单位；异步性：由于进程间的相互制约，使进程具有执行的间断性，即进程按各自独立的、不可预知的速度向前推进

现在操作系统中，（）是资源分配的基本单位,（）是CPU调度的基本单位。

- [ ] 作业，程序
- [ ] 内存，进程
- [x] 进程，线程
- [ ] 代码，数据

> 解释：概念题，在现代操作系统中，进程使资源分配的基本单位，线程是CPU调度的基本单位。其中线程与属于同一进程的其他线程共享代码段、数据段和其他操作系统资源，如果进程有多个控制线程，那么它能同时做多个任务

下列各项工作步骤中，()不是创建进程所必需的步骤

- [ ] 为进程分配内存等资源
- [ ] 将PCB链入进程就绪队列
- [x] 作业调度程序为进程分配CPU
- [ ] 建立一个PCB

> 解释：创建进程时不需要用作业调度程序为进程分配CPU

在多线程操作系统中，对线程具有属性阐述正确的是（）

- [x] 具有进程控制块，共享所属进程资源，处理机的独立调度单位，具有动态性
- [ ] 具有线程控制块，共享所属进程资源，处理机的独立调度单位，具有动态性
- [ ] 具有进程控制块，独享所属进程资源，处理机的独立调度单位，具有动态性
- [ ] 具有进程控制块，共享所属进程资源，处理机的独立调度单位，具有静态性

> 解释：概念题，线程具有进程控制块，共享所属进程资源，处理机的独立调度单位，具有动态

多选题

（西安电子科技大学）能正确描述进程和线程的概念是（）

- [x] 线程可以是进程中独立执行的实体，一个进程可以包含一个或多个线程
- [ ] 线程又称为轻型进程，因为线程都比进程小
- [x] 多线程计数具有明显的优越性，如速度快、通信简便、设备并行性高
- [ ] 由于线程不作为资源分配单位，线程之间可以无约束地并行执行
- [x] 一个线程可以属于一个或多个进程

> 解释：虽然线程被称为轻量级线程，这并不意味着线程比进程小，进程和线程之间无法进行大小比较

（电子科技大学）引起挂起状态的原因有（）

- [x] 终端用户的请求
- [x] 父进程请求
- [x] 负荷调节的需要
- [ ] 操作系统的需要
- [ ] 平衡各队列中的进程控制块

> 解释：考察引起挂起的原因

下列各项中属于进程特性的是（）

- [x] 动态性
- [x] 异步性
- [x] 独立性
- [x] 并发性

> 解释：概念题,进程有四个特性,动态性：进程的实质是程序在多道程序系统中的一次执行过程，进程是动态产生，动态消亡的;并发性：任何进程都可以同其他进程一起并发执行;独立性：进程是一个能独立运行的基本单位，同时也是系统分配资源和调度的独立单位；异步性：由于进程间的相互制约，使进程具有执行的间断性，即进程按各自独立的、不可预知的速度向前推进

采用多线程技术的操作系统具有()

- [x] 一个进程中可以有一个或多个线程
- [x] 把进程作为资源分配单位,把线程作为调度和执行单位
- [ ] 不同的线程一定执行不同的程序
- [x] 允许多个线程并发执行

> 解释：不同的线程可能执行相同的程序，一个线程中可以有一个或多个线程，把进程作为资源分配单位,把线程作为调度和执行单位，允许多个线程并发执行

判断题：

（北京工业大学）子进程可以继承它的父进程所拥有的所有资源（）

- [ ] 对
- [x] 错
  > 解释：子进程继承了父进程的代码段和数据段资源，堆栈段则是自己的

（首都师范大学）属于同一进程的用户级线程阻塞了，那么同一个进程的其他用户级线程还可以占有CPU运行，直到时间片用完（）

- [x] 对
- [ ] 错

> 解释：在同一进程中，线程的切换不会引起进程的切换，在由一个进程中的线程切换到另一个进程中的线程时，将会引起进程的切换

在操作系统中，进程是一个静态的概念（）

- [ ] 对
- [x] 错

> 解释：动态概念

一般来说用户进程的PCB存放在用户区，系统进程的PCB存放在操作系统区（）

- [ ] 对
- [x] 错

> 解释： PCB通常是系统内存占用区中的一个连续存区，它存放着操作系统用于描述进程情况及控制进程运行所需的全部信息。

在linux环境里使用fork（）来创建新进程（）

- [x] 对
- [ ] 错

> 解释： 概念题，了解进程的创建是如何进行的

在多对一模型的线程中，如果一个线程执行了阻塞系统调用，并不影响整个进程（）

- [ ] 对
- [x] 错

> 解释： 多对一模型的线程中，如果一个线程执行了阻塞系统调用，会影响整个进程，整个进程会阻塞

启动一个线程使用的是start()方法（）

- [x] 对
- [ ] 错

> 解释： 概念题

在父进程还存活的情况下, 不会产生僵死状态（）

- [ ] 对
- [x] 错

> 解释：一个已经终止但是其父进程尚未对其进行善后处理（获取终止子进程的有关信息，释放它仍占用的资源）的进程称为僵尸进程(zombie)。这时进程在调用exit命令结束自己的生命的时候，其实它并没有真正的被销毁，而是留下一个称为僵尸进程（Zombie）的数据结构

---

lec11 进程与线程　在线练习

选择题

进程与程序的关系描述正确的是（） s1

- [x] 进程是指一个具有一定独立功能的程序在一个数据集合上的一次动态执行过程
- [ ] 进程是一个具有一定独立功能的程序
- [ ] 程序是一个动态执行的进程
- [x] 进程包含了正在运行的一个程序的所有状态信息

> 1,4

关于进程控制块的描述正确的是（） s2

- [x] 操作系统用进程控制块来描述进程的基本情况以及运行变化的过程
- [x] 进程控制块是进程存在的唯一标志
- [x] 每个进程都在操作系统中有一个对应的进程控制块
- [x] 操作系统管理控制进程运行所用的信息集合是进程控制块

> 都对

关于进程的生命周期的描述正确的是（） s3

- [x] 内核选择一个就绪态的进程，让它占用处理机并执行，此时进程处于运行态
- [x] 进程请求并等待系统服务，无法马上完成，此时进程处于等待态
- [x] 进程执行的当前时间片用完了，此时进程处于就绪态
- [x] 进程退出了，但还没被父进程回收，此时进程处于zombie态

> 都对

操作系统来维护一组队列，表示系统中所有进程的当前状态，有关管理进程的描述正确的是（） s5

- [x] 就绪态进程维护在进程就绪队列中
- [x] 等待态进程维护在进程等待队列中
- [ ] 运行态进程维护在进程运行队列中
- [ ] zombie态进程不在任何队列中

> 1,2

有关线程或进程的描述正确的是（） s6

- [x] 进程是资源分配单位，线程是CPU调度单位
- [x] 进程拥有一个完整的资源平台，而线程只独享指令流执行的必要资源，如寄存器和栈
- [x] 线程能减少并发执行的时间和空间开销
- [x] 同一进程的各线程间共享内存和文件资源，可不通过内核进行直接通信

> 都对

常见的线程种类有() s7

- [x] 用户线程
- [x] 内核线程
- [x] 轻量级进程

> 都对

内核线程的描述正确的是() s8

- [x] 由内核维护内核线程的线程控制块
- [ ] 由用户线程库维护内核线程的线程控制块
- [ ] 内核无法调度内核线程
- [ ] 内核线程间无法共享所属进程的资源

> 1

lec10 进程／线程概念spoc练习

NOTICE

- 有"w5l1"标记的题是助教要提交到学堂在线上的。
- 有"w5l1"和"spoc"标记的题是要求拿清华学分的同学要在实体课上完成，并按时提交到学生对应的git repo上。
- 有"hard"标记的题有一定难度，鼓励实现。
- 有"easy"标记的题很容易实现，鼓励实现。
- 有"midd"标记的题是一般水平，鼓励实现。

个人思考题

11.1 进程的概念

- 什么是进程？什么是程序？
- 程序和进程联系和区别是什么？

11.2 进程控制块

- 进程控制块的功能是什么？
- 进程控制块中包括什么信息？

11.3 进程状态

- 进程生命周期中的相关事件有些什么？它们对应的进程状态变化是什么？
- 进程切换过程中的几个关键代码分析
- 时钟中断触发调度函数的启动
- 当前进程的现场保存代码
- 进程切换代码 ＞ 下一个运行进程的现场恢复

11.4 三状态进程模型

- 运行、就绪和等待三种状态的含义？
- 分析的4个相关状态转换代码和状态修改代码

11.5 挂起进程模型

- 引入挂起状态的目的是什么？内存中的什么内容放到外存中，就算是挂起状态？

11.6 线程的概念

- 引入线程的目的是什么？什么是线程？
- 进程与线程的联系和区别是什么？

11.8 内核线程

- 用户线程与内核线程的区别是什么？

SPOC小组思考题

(1) (spoc)设计一个简化的进程管理子系统，可以管理并调度如下简化进程.给出了参考代码，请理解代码，并完成 " YOUR CODE"部分的内容.　可 2 个人一组

进程的状态

- RUNNING - 进程正在使用CPU
- READY - 进程可使用CPU
- DONE - 进程结束

进程的行为

- 使用CPU,
- 发出YIELD请求,放弃使用CPU

进程调度

- 使用FIFO/FCFS：先来先服务,
  - 先查找位于proc_info队列的curr_proc元素(当前进程)之后的进程(curr_proc+1..end)是否处于READY态，
  - 再查找位于proc_info队列的curr_proc元素(当前进程)之前的进程(begin..curr_proc-1)是否处于READY态
  - 如都没有，继续执行curr_proc直到结束

关键模拟变量

- 进程控制块

```
PROC_CODE = 'code_'
PROC_PC = 'pc_'
PROC_ID = 'pid_'
PROC_STATE = 'proc_state_'
```

- 当前进程 curr_proc
- 进程列表：proc_info是就绪进程的队列（list），
- 在命令行（如下所示）需要说明每进程的行为特征：（1）使用CPU ;(2)等待I/O

```
-l PROCESS_LIST, --processlist= X1:Y1,X2:Y2,...
X 是进程的执行指令数;
Y是执行CPU的比例(0..100)，如果是100，表示不会发出yield操作
```

- 进程切换行为：系统决定何时(when)切换进程:进程结束或进程发出yield请求

进程执行

```
instruction_to_execute = self.proc_info[self.curr_proc][PROC_CODE].pop(0)
```

关键函数

- 系统执行过程：run
- 执行状态切换函数:　move_to_ready/running/done
- 调度函数：next_proc

执行实例

例 1

```
$./process-simulation.py -l 5:50
Process 0
  yld
  yld
  cpu
  cpu
  yld

Important behaviors:
  System will switch when the current process is FINISHED or ISSUES AN YIELD
Time     PID: 0
  1     RUN:yld
  2     RUN:yld
  3     RUN:cpu
  4     RUN:cpu
  5     RUN:yld
```

例 2

```
$./process-simulation.py  -l 5:50,5:50
Produce a trace of what would happen when you run these processes:
Process 0
  yld
  yld
  cpu
  cpu
  yld

Process 1
  cpu
  yld
  cpu
  cpu
  yld

Important behaviors:
  System will switch when the current process is FINISHED or ISSUES AN YIELD
Time     PID: 0     PID: 1
  1     RUN:yld      READY
  2       READY    RUN:cpu
  3       READY    RUN:yld
  4     RUN:yld      READY
  5       READY    RUN:cpu
  6       READY    RUN:cpu
  7       READY    RUN:yld
  8     RUN:cpu      READY
  9     RUN:cpu      READY
 10     RUN:yld      READY
 11     RUNNING       DONE
```

lec12 进程控制　在线练习

选择题

关于进程切换描述正确的是（） s1

- [x] 进程切换会暂停当前运行进程，使其从运行状态变成就绪等其他状态
- [x] 进程切换要保存当前进程的上下文
- [x] 进程切换要恢复下一个进程的上下文
- [ ] 进程切换的进程上下文不包括CPU的寄存器等硬件信息

> 1,2,3

关于创建新进程的描述正确的是（） s2

- [x] fork() 创建子进程中，会复制父进程的所有变量和内存
- [x] 子进程的fork()返回0
- [x] 父进程的fork()在创建子进程成功后，返回子进程标识符
- [x] fork() 创建子进程中，会复制父进程的页表

> 都对

关于进程加载执行的描述正确的是（） s3

- [x] 系统调用exec( )加载新程序取代当前运行进程
- [x] 系统调用exec( )允许进程“加载”一个完全不同的程序，并从main开始执行
- [x] exec调用成功时，它是相同的进程，但是运行了不同的程序
- [x] exec调用成功时，代码段、堆栈和堆(heap)等完全重写了

> 都对

有关管理进程等待的描述正确的是（） s4

- [x] wait()系统调用用于父进程等待子进程的结束
- [x] 子进程结束时通过exit()向父进程返回一个值
- [x] 当某子进程调用exit()时,唤醒父进程，将exit()返回值作为父进程中wait的返回值
- [x] 进程结束执行时调用exit()，完成进程的部分占用资源的回收

> 都对

lec10 进程／线程控制spoc练习

SPOC个人练习

进程切换

(1)ucore的进程控制块数据结构是如何组织的？主要字段分别表示什么？有哪些函数对它进行了修改？有哪些函数用到它？

```
arch_proc_struct
mm_struct
need_resched
wait_state
run_link、list_link、hash_link
```

进程创建

(1)fork()的返回值是唯一的吗？父进程和子进程的返回值是不同的。请找到相应的赋值代码。

(2)新进程创建时的进程标识是如何设置的？请指明相关代码。

(3)fork()的例子中进程标识的赋值顺序说明进程的执行顺序。

(4)请在ucore启动时显示空闲进程（idleproc）和初始进程（initproc）的进程标识。

进程加载

(1)加载进程后，新进程进入就绪状态，它开始执行时的第一条指令的位置，在elf中保存在什么地方？在加载后，保存在什么地方？

> 进程等待与退出

(2)试分析wait()和exit()的结果放在什么地方？exit()是在什么时候放进去的？wait()在什么地方取到出的？

(3)试分析sleep()系统调用的实现。在什么地方设置的定时器？它对应的等待队列是哪个？它的唤醒操作在什么地方？

SPOC小组思考题

(1) (spoc)设计一个简化的进程管理子系统，可以管理并调度如下简化进程.给出了参考代码，请理解代码，并完成 " YOUR CODE"部分的内容.　可２个人一组

进程的状态

```
- RUNNING - 进程正在使用CPU
- READY   - 进程可使用CPU
- WAIT    - 进程等待I/O完成
- DONE    - 进程结束
```

进程的行为

```
- 使用CPU,
- 发出YIELD请求,放弃使用CPU
- 发出I/O操作请求,放弃使用CPU
```

进程调度

- 使用FIFO/FCFS：先来先服务, 只有进程done, yield, io时才会执行切换
  - 先查找位于proc_info队列的curr_proc元素(当前进程)之后的进程(curr_proc+1..end)是否处于READY态，
  - 再查找位于proc_info队列的curr_proc元素(当前进程)之前的进程(begin..curr_proc-1)是否处于READY态
  - 如都没有，继续执行curr_proc直到结束

关键模拟变量

- io_length : IO操作的执行时间
- 进程控制块

```
PROC_CODE = 'code_'
PROC_PC = 'pc_'
PROC_ID = 'pid_'
PROC_STATE = 'proc_state_'
```

- 当前进程 curr_proc
- 进程列表：proc_info是就绪进程的队列（list），
- 在命令行（如下所示）需要说明每进程的行为特征：（1）使用CPU ;(2)等待I/O

```
-l PROCESS_LIST, --processlist= X1:Y1,X2:Y2,...
X 是进程的执行指令数;
Y是执行yield指令（进程放弃CPU,进入READY状态）的比例(0..100)
Z是执行I/O请求指令（进程放弃CPU,进入WAIT状态）的比例(0..100)
```

- 进程切换行为：系统决定何时(when)切换进程:进程结束或进程发出yield请求

进程执行

```
instruction_to_execute = self.proc_info[self.curr_proc][PROC_CODE].pop(0)
```

关键函数

- 系统执行过程：run
- 执行状态切换函数: move_to_ready/running/done
- 调度函数：next_proc

执行实例

例1

```
$./process-simulation.py  -l 5:30:30,5:40:30 -c
Produce a trace of what would happen when you run these processes:
Process 0
  io
  io
  yld
  cpu
  yld
```

```
Process 1
  yld
  io
  yld
  yld
  yld

Important behaviors:
  System will switch when the current process is FINISHED or ISSUES AN YIELD or IO
Time     PID: 0     PID: 1        CPU        IOs
  1      RUN:io      READY          1
  2     WAITING    RUN:yld          1          1
  3     WAITING     RUN:io          1          1
  4     WAITING    WAITING                     2
  5     WAITING    WAITING                     2
  6*     RUN:io    WAITING          1          1
  7     WAITING    WAITING                     2
  8*    WAITING    RUN:yld          1          1
  9     WAITING    RUN:yld          1          1
 10     WAITING    RUN:yld          1          1
 11*    RUN:yld       DONE          1
 12     RUN:cpu       DONE          1
 13     RUN:yld       DONE          1
```

lab4 在线练习

选择题

关于进程切换描述正确的是（） s1

- [x] 进程切换会暂停当前运行进程，使其从运行状态变成就绪等其他状态
- [x] 进程切换要保存当前进程的上下文
- [x] 进程切换要恢复下一个进程的上下文
- [ ] 进程切换的进程上下文不包括CPU的寄存器等硬件信息

> 1,2,3

关于创建新进程的描述正确的是（） s2

- [x] fork() 创建子进程中，会复制父进程的所有变量和内存
- [x] 子进程的fork()返回0
- [x] 父进程的fork()在创建子进程成功后，返回子进程标识符
- [x] fork() 创建子进程中，会复制父进程的页表

> 都对

关于进程加载执行的描述正确的是（） s3

- [x] 系统调用exec( )加载新程序取代当前运行进程
- [x] 系统调用exec( )允许进程“加载”一个完全不同的程序，并从main开始执行
- [x] exec调用成功时，它是相同的进程，但是运行了不同的程序
- [x] exec调用成功时，代码段、堆栈和堆(heap)等完全重写了

> 都对

有关管理进程等待的描述正确的是（） s4

- [x] wait()系统调用用于父进程等待子进程的结束
- [x] 子进程结束时通过exit()向父进程返回一个值
- [x] 当某子进程调用exit()时,唤醒父进程，将exit()返回值作为父进程中wait的返回值
- [x] 进程结束执行时调用exit()，完成进程的部分占用资源的回收

> 都对

lab4 spoc 思考题

- 有"spoc"标记的题是要求拿清华学分的同学要在实体课上完成，并按时提交到学生对应的ucore_code和os_exercises的git repo上。

个人思考题

总体介绍

(1) ucore的线程控制块数据结构是什么？

关键数据结构

(2) 如何知道ucore的两个线程同在一个进程？

(3) context和trapframe分别在什么时候用到？

(4) 用户态或内核态下的中断处理有什么区别？在trapframe中有什么体现？

执行流程

(5) do_fork中的内核线程执行的第一条指令是什么？它是如何过渡到内核线程对应的函数的？

```
tf.tf_eip = (uint32_t) kernel_thread_entry;
/kern-ucore/arch/i386/init/entry.S
/kern/process/entry.S
```

(6)内核线程的堆栈初始化在哪？

```
tf和context中的esp
```

(7)fork()父子进程的返回值是不同的。这在源代码中的体现中哪？

(8)内核线程initproc的第一次执行流程是什么样的？能跟踪出来吗？

小组练习与思考题

(1)(spoc) 理解内核线程的生命周期。

> 需写练习报告和简单编码，完成后放到git server 对应的git repo中

掌握知识点

1. 内核线程的启动、运行、就绪、等待、退出
2. 内核线程的管理与简单调度
3. 内核线程的切换过程

练习用的**lab4 spoc exercise project source code**

请完成如下练习，完成代码填写，并形成spoc练习报告

1. 分析并描述创建分配进程的过程

> 注意 state、pid、cr3，context，trapframe的含义

练习2：分析并描述新创建的内核线程是如何分配资源的

> 注意 理解对kstack, trapframe, context等的初始化

当前进程中唯一，操作系统的整个生命周期不唯一，在get_pid中会循环使用pid，耗尽会等待

练习3：阅读代码，在现有基础上再增加一个内核线程，并通过增加cprintf函数到ucore代码中

能够把内核线程的生命周期和调度动态执行过程完整地展现出来

练习4（非必须，有空就做）：增加可以睡眠的内核线程，睡眠的条件和唤醒的条件可自行设计，并给出测试用例，并在spoc练习报告中给出设计实现说明

扩展练习1: 进一步裁剪本练习中的代码，比如去掉页表的管理，只保留段机制，中断，内核线程切换，print功能。看看代码规模会小到什么程度。

lab5 用户进程 在线练习

选择题

下列叙述中正确的是() s2

- [ ] lab5建立了用户进程，且0~3GB都是用户可访问空间，用户进程可进行正常读写
- [ ] lab5建立了用户进程，且3GB～4GB都是内核可访问空间，内核可进行正常读写
- [x] lab5中的第一个用户进程是内核创建的。
- [x] lab5中的用户进程可通过fork创建新的用户进程。

> 3,4

lab5通过do_execve函数执行新的程序，为此需要完成（） s3

- [x] 更新用户进程的context
- [x] 更新用户进程的代码内容
- [x] 更新用户进程的数据内容
- [ ] 更新用户进程的页表基址

> 1,2,3,4

lab5通过do_icode函数执行新的程序，为此需要完成（）s4

- [x] 设置用户堆栈
- [x] 修改页表
- [x] 根据ELF执行文件的格式描述分配内存并填写内容
- [x] 设置用户态的EFLAG寄存器不可屏蔽中断

> 都包括

关于进程管理的COW(Copy On Write)机制叙述正确的是（）s6

- [ ] 父进程创建子进程需要复制父进程的内存空间
- [ ] 父进程创建子进程需要给子进程分配内核堆栈
- [ ] 父进程创建子进程需要给子进程分配用户堆栈
- [x] 父进程创建子进程需要创建子进程的页表,但不复制父进程内存空间

> 4

lab5 spoc 思考题

- 有"spoc"标记的题是要求拿清华学分的同学要在实体课上完成，并按时提交到学生对应的ucore_code和os_exercises的git repo上。

个人思考题

总体介绍

- 第一个用户进程创建有什么特殊的？
- 系统调用的参数传递过程？
- getpid的返回值放在什么地方了？

进程的内存布局

- 尝试在进程运行过程中获取内核堆栈和用户堆栈的调用栈？
- 尝试在进程运行过程中获取内核空间中各进程相同的页表项（代码段）和不同的页表项（内核堆栈）？

执行ELF格式的二进制代码-do_execve的实现

- 在do_execve中进程清空父进程时，当前进程是哪一个？在什么时候开始使用新加载进程的地址空间？
- 新加载进程的第一级页表的建立代码在哪？

执行ELF格式的二进制代码-load_icode的实现

- 第一个内核线程和第一个用户进程的创建有什么不同？
- 尝试跟踪分析新创建的用户进程的开始执行过程？

进程复制

- 为什么新进程的内核堆栈可以先于进程地址空间复制进行创建？
- 进程复制的代码在哪？复制了哪些内容？
- 进程复制过程中有哪些修改？为什么要修改？

内存管理的copy-on-write机制

- 什么是写时复制？
- 写时复制的页表在什么时候进行复制？共享地址空间和写时复制有什么不同？

小组练习与思考题

(1)(spoc) 在真实机器的u盘上启动并运行ucore lab,

请准备一个空闲u盘，然后请参考如下网址完成练习

https://github.com/chyyuu/ucore_lab/blob/master/related_info/lab1/lab1-boot-with-grub2-in-udisk.md

> 注意，grub_kernel的源码在ucore_lab的lab1_X的git branch上，位于 `ucore_lab/labcodes_answer/lab1_result`

(报告可课后完成)请理解grub multiboot spec的含义，并分析ucore_lab是如何实现符合grub multiboot spec的，并形成spoc

练习报告。

(2)(spoc) 理解用户进程的生命周期。

> 需写练习报告和简单编码，完成后放到git server 对应的git repo中

练习用的lab5 spoc exercise project source code

掌握知识点

1. 用户进程的启动、运行、就绪、等待、退出
2. 用户进程的管理与简单调度
3. 用户进程的上下文切换过程
4. 用户进程的特权级切换过程
5. 用户进程的创建过程并完成资源占用
6. 用户进程的退出过程并完成资源回收

> 注意，请关注：内核如何创建用户进程的？用户进程是如何在用户态开始执行的？用户态的堆栈是保存在哪里的？

阅读代码，在现有基础上再增加一个用户进程A，并通过增加cprintf函数到ucore代码中， 能够把个人思考题和上述知识点中的内容展示出来：即在ucore运行过程中通过 `cprintf` 函数来完整地展现出来进程A相关的动态执行和内部数据/状态变化的细节。(约全面细致约好)

请完成如下练习，完成代码填写，并形成spoc练习报告

CPU调度

单选题

若当前进程因时间片用完而让出处理机时，该进程应转变为（）状态。

- [x] 就绪
- [ ] 等待
- [ ] 运行
- [ ] 完成

> 解释：只有处于就绪队列中的进程才能得到时间片，因此因为时间片用完而让出CPU的进程应该再次返回到就绪队列中。时间片是轮循调度算法中的概念，所有的进程都会按照顺序被分配一个时间片，当时间片用完时如果进程执没有结束，那么应该让出CPU进入就绪队列等待下一个属于自己的时间片。

最高响应比优先算法的特点是（）

- [ ] 有利于短作业但不利于长作业
- [x] 有利于短作业又兼顾到长作业
- [ ] 不利于短作业也不利于长作业
- [ ] 不利于短作业但有利于长作业

> 解释：最高响应比优先算法的响应值公式为R＝（w+s)/s，其中w为等待时间，s为服务时间，因此在等待时间相同的情况下优先选择服务时间短的进程，而当服务时间长的进程等待到一定时间后，其响应值会增加到能够被首先选择，避免了一直被服务时间短的进程超过，所以该算法有利于短作业又兼顾到长作业。

在单处理器的多进程系统中，进程什么时候占用处理器和能占用多长时间，取决于（）

- [ ] 进程相应的程序段的长度
- [ ] 进程总共需要运行时间多少
- [x] 进程自身和进程调度策略
- [ ] 进程完成什么功能

> 解释：在单处理器的多进程系统中，系统是依靠所使用的调度策略来对进程进行调度的，而其所采用的调度策略可能不止一种，所以什么时候选择什么进程占用处理器和能占用多长时间并不仅仅取决于进程的某一项特性。

时间片轮转调度算法是为了（）

- [x] 多个终端都能得到系统的及时响应
- [ ] 先来先服务
- [ ] 优先级高的进程先使用CPU
- [ ] 紧急事件优先处理

> 解释：时间片轮转调度算法在选择进程时是按照到达时间进行选择的，所以不存在优先级高的进程，而每个进程每次只能占用同等的CPU时间，所以优先执行的进程并不一定比后执行的进程先完成，对于新加入的进程，只要是队列中等待的进程不是很多，都可以很及时地得到时间片来使用CPU，所以该算法能够使多个终端得到系统的及时响应。

在基于优先级的可抢占的调度机制中，当系统强制使高优先级任务等待低优先级任务时，会发生（）

- [x] 优先级反转
- [ ] 优先级重置

- [ ] 系统错误
- [ ] 死循环

> 解释：优先级反转的定义：（1）可以发生在任何基于优先级的可抢占的调度机制中；（2）当系统内的环境强制使高优先级等待低优先级任务时发生。

下面关于硬时限（hard deadlines）和软时限（soft deadlines）的描述错误的是（）。

- [ ] 如果错过了硬时限，将会发生严重的后果
- [x] 硬时限是通过硬件实现的，软时限是通过软件实现的
- [ ] 如果软时限没有被满足，系统也可以继续运行
- [ ] 硬时限可以保证系统的确定性

> 解释：硬时限是指必须满足的时间限制，如果没有满足可能会导致非常严重的后果；软时限是指在理想情况下应该被满足的时间限制，如果没有被满足，系统可以降低对该时限的要求，以保证不会产生太严重的后果。

下面的调度算法中那个是公平的（）

- [ ] FCFS 先来先服务
- [ ] SPN 短进程优先
- [x] RR 轮循
- [ ] SRT

> 解释：FCFS算法可能导致某些进程长时间占用CPU，所以并不公平；SPN算法可能会使长进程在很长时间内得不到响应，所以也不公平；RR算法由于每个进程都能及时得到响应，并且不会长时间占用CPU，所以是公平的；SRT也就是SPN。

FCFS调度算法的特点不包括（）

- [ ] 简单
- [ ] 平均等待时间变化大
- [x] 不会导致I/O和CPU之间的重叠处理
- [ ] 花费时间少的任务可能排在花费时间长的任务后面

> 解释：FCFS算法的优点是简单，缺点有（1）平均等待时间变化较大；（2）花费时间较少的任务可能排在花费时间较长的任务后面；（3）可能导致i/o和CPU之间的重叠处理。

CPU调度策略的目标不包括（）

- [ ] 减少响应时间
- [x] 提高系统处理单任务的速度
- [ ] 减少等待时间
- [ ] 增加吞吐量

> 解释：系统处理单任务的速度不能通过CPU调度策略来改善，只能通过改善硬件性能和改良系统架构来提高。

有5个批处理作业(A,B,C,D,E)几乎同时到达一个计算中心,估计运行时间分别为2,4,6,8,10分钟,在使用时间片轮转作法（时间片为2分钟）,作业的平均周转时间为（）

- [x] 18分钟
- [ ] 6分钟
- [ ] 14分钟
- [ ] 22分钟

> 解释：进程A在第一次时间片轮转后就完成了，所以等待时间为0；进程B在第二次时间片轮转后完成，等待时间为

> 2+23=8；进程C在第三次时间片轮转后完成，等待时间为2+2+22+2+2*2=14；进程D在第四次时间片轮转后完成，等待时间为2+2+2+2+2+2+2+2=18；进程E在第五次时间片轮转后完成，等待时间为2+2+2+2+2+2+2+2+2+2=20；因此总的周转时间为2+0+4+8+6+14+8+18+10+20=90，所以平均周转时间为90/5=18。

多选题

对上下文切换的描述正确的是（）

- [x] 切换CPU的当前任务到另一个任务
- [ ] 不需要保存当前进程在PCB/TCP中的执行上下文
- [x] 需要读取下一个进程的上下文
- [ ] 只能读取没有被执行过的进程

> 解释：上下文切换的相关概念：（1）切换CPU的当前任务，从一个进程到另一个进程；（2）保存当前进程在PCB/TCP的执行上下文；（3）读取下一个进程的上下文。被切换的进程可以是新来的，也可以是之前没有执行完的。

可以作为进程调度算法的有（）。

- [x] 先来先服务调度算法
- [x] 时间片轮转调度算法
- [ ] 最高优先级调度算法
- [x] 最高响应比优先调度算法
- [ ] 均衡调度算法

> 解释：不存在最高优先级调度算法和均衡调度算法。

下面可以作为比较调度算法的指标有（）

- [x] CPU使用率
- [x] 吞吐量
- [x] 周转时间
- [x] 等待时间
- [x] 响应时间

> 解释：衡量调度算法的5个方面：CPU使用率，吞吐量，周转时间，等待时间和响应时间。

CPU调度策略可以通过哪几种方式增加系统的吞吐量（）

- [x] 减少操作系统开销
- [x] 减少上下文切换次数
- [ ] 加快系统处理单个任务的速度
- [x] 高效利用系统资源

> 解释：增加吞吐量可以从两个方面入手：（1）减少开销（操作系统开销，上下文切换）；（2）系统资源的高效利用（CPU，I/O设备）。

下面对FFS公平共享调度控制用户对系统资源的访问的描述中，正确的是（）

- [ ] 所有的用户组都是平等的
- [x] 能够保证不重要的用户组无法垄断资源
- [x] 未使用的资源按照每个组所分配的资源的比例来分配

- [x] 没有达到资源使用率目标的组可以获得更高的优先级

> 解释：公平共享调度控制用户对系统资源的访问：（1）一些用户组比其他用户组重要；（2）保证不重要的组无法垄断资源；（3）未使用的资源按照每个组所分配的资源的比例来分配；（3）没有达到资源使用率目标的组获得更高的优先级。

判断题

作业调度选择一个作业装入主存后，该作业能否占用处理器必须由作业控制来决定。

- [ ] 对
- [x] 错

> 解释：作业能够占用处理器是由进程调度来决定的。

在进行作业调度时，要想兼顾作业等待时间和计算时间，可选取响应比高者优先算法。

- [x] 对
- [ ] 错

> 解释：最高响应比优先算法的公式为 R =（w+s)/s，其中w为等待时间，s为计算时间，所以兼顾作业的等待时间和计算时间。

在作业调度时，采用最高响应比优先的作业调度算法可以得到最短的作业平均周转时间。

- [ ] 对
- [x] 错

> 解释：短进程优先算法的平均等待时间最小。

轮循算法的时间量子越大越好。（错）

- [ ] 对
- [x] 错

> 解释：轮循算法的时间量子太大的话会导致进程等待的时间过长，极限情况下会退化成FCFS。

可抢占式的调度算法比不可抢占式的调度算法开销要小。（错）

- [ ] 对
- [x] 错

> 解释：可抢占式的调度算法比不可抢占式的调度算法开销要大，因为其上下文切换比不可抢占式的要多。

lec15 处理器调度　在线练习

选择题

若当前进程因时间片用完而让出处理机时，该进程应转变为（）状态。 s1

- [x] 就绪
- [ ] 等待
- [ ] 运行
- [ ] 完成

> 1

最高响应比优先算法的特点是（） s3

- [ ] 有利于短作业但不利于长作业
- [x] 有利于短作业又兼顾到长作业
- [ ] 不利于短作业也不利于长作业
- [ ] 不利于短作业但有利于长作业

> 2

在单处理器的多进程系统中，进程什么时候占用处理器和能占用多长时间，取决于（） s4

- [ ] 进程相应的程序段的长度
- [ ] 进程总共需要运行时间多少
- [x] 进程自身和进程调度策略
- [ ] 进程完成什么功能

> 3

时间片轮转调度算法是为了（） s4

- [x] 多个终端都能得到系统的及时响应
- [ ] 先来先服务
- [ ] 优先级高的进程先使用CPU
- [ ] 紧急事件优先处理

> 1

下面关于硬时限（hard deadlines）和软时限（soft deadlines）的描述错误的是（）。s5

- [ ] 如果错过了硬时限，将会发生严重的后果
- [x] 硬时限是通过硬件实现的，软时限是通过软件实现的
- [ ] 如果软时限没有被满足，系统也可以继续运行
- [ ] 硬时限可以保证系统的确定性

> 2

在基于优先级的可抢占的调度机制中，当系统强制使高优先级任务等待低优先级任务时，会发生（）s6

- [x] 优先级反转
- [ ] 优先级重置
- [ ] 系统错误
- [ ] 死循环

```
1
```

调度算法概念(lec 15) spoc 思考题

- 有"spoc"标记的题是要求拿清华学分的同学要在实体课上完成，并按时提交到学生对应的ucore_code和os_exercises的git repo上。

个人思考题

处理机调度概念

- 如何判断操作系统是否是可抢先的？
- 当操作系统的处理机调度导致线程切换时，暂停进程的当前指令指针可能在什么位置？用户态代码或内核代码？给出理由和实例。

调度准则

- 尝试在ucore上写一个外排序程序，然后分析它的执行时间分布统计（每次切换后开始执行时间和放弃CPU的时间、当前用户和内核栈信息）。
- 在Linux上有一个应用程序time，可以统计应用程序的执行时间信息。请分析它是如何统计进程执行时间信息的。如可能，请在ucore上实现相同功能的应用程序。下面是可能的参考。
    - Linux用户态程序计时方式详解
    - Get Source Code for any Linux Command
    - How does time command work
    - http://cvs.opensolaris.org/source/xref/onnv/onnv-gate/usr/src/cmd/time/time.c
    - https://github.com/illumos/illumos-gate/blob/master/usr/src/cmd/time/time.c
- 尝试获取一个操作系统的调度算法的性能统计数据（CPU使用率、进程执行）。

先来先服务、短进程优先和最高响应比优先调度算法

- FIFO、SPN、HRRN算法的思路？
- 如何调试你的调度算法？

时间片轮转、多级反馈队列、公平共享调度算法和ucore调度框架

- RR、MLFQ和FSS算法的思路？
- RR算法选择时间片长度的依据有哪些？
- 定义调度算法接口需要考虑哪些因素？
- 尝试跟踪就绪进程选择、进程切换和调度算法的选择依据。

实时调度和多处理器调度

- 有兴趣的同学，请阅读下面论文，然后说明实时调度面临的主要困难是什么？
    - Buttazzo, “Rate monotonic vs. EDF: Judgement Day”, EMSOFT 2003.
    - 单调速率及其扩展算法的可调度性判定
- 多处理机调度中每个处理机一个就绪队列与整个系统一个就绪队列有什么不同？

优先级反置

- 什么是优先级继承(Priority Inheritance)和优先级天花板协议(priority ceiling protocol)？它们的区别是什么？

小组练习与思考题

(1)(spoc) 理解并完善调度算法

实现３种调度算法（SJF，FIFO，RR），可基于python, ruby, C, C++，LISP等）模拟实现，并给出测试。请参考scheduler-homework.py代码或独自实现。 最后统计采用不同调度算法的每个任务的相关时间和总体的平均时间： - turnaround time 周转时间 - response time 响应时间 - wait time 等待时间

对模拟环境的抽象

- 任务/进程，及其执行时间 Job 0 (length = 1) Job 1 (length = 4) Job 2 (length = 7)
  - 何时切换？
  - 如何统计？

执行结果

采用FIFO调度算法

```
 ./scheduler-homework.py  -p FIFO
ARG policy FIFO
ARG jobs 3
ARG maxlen 10
ARG seed 0

Here is the job list, with the run time of each job:
  Job 0 ( length = 9 )
  Job 1 ( length = 8 )
  Job 2 ( length = 5 )


** Solutions **

Execution trace:
  [ time   0 ] Run job 0 for 9.00 secs ( DONE at 9.00 )
  [ time   9 ] Run job 1 for 8.00 secs ( DONE at 17.00 )
  [ time  17 ] Run job 2 for 5.00 secs ( DONE at 22.00 )

Final statistics:
  Job   0 -- Response: 0.00  Turnaround 9.00  Wait 0.00
  Job   1 -- Response: 9.00  Turnaround 17.00  Wait 9.00
  Job   2 -- Response: 17.00  Turnaround 22.00  Wait 17.00

  Average -- Response: 8.67  Turnaround 16.00  Wait 8.67
```

采用SJF调度算法

```
 ./scheduler-homework.py  -p SJF
ARG policy SJF
ARG jobs 3
ARG maxlen 10
ARG seed 0

Here is the job list, with the run time of each job:
  Job 0 ( length = 9 )
  Job 1 ( length = 8 )
  Job 2 ( length = 5 )


** Solutions **

Execution trace:
  [ time   0 ] Run job 2 for 5.00 secs ( DONE at 5.00 )
  [ time   5 ] Run job 1 for 8.00 secs ( DONE at 13.00 )
```

```
  [ time  13 ] Run job 0 for 9.00 secs ( DONE at 22.00 )

Final statistics:
  Job   2 -- Response: 0.00  Turnaround 5.00  Wait 0.00
  Job   1 -- Response: 5.00  Turnaround 13.00  Wait 5.00
  Job   0 -- Response: 13.00  Turnaround 22.00  Wait 13.00

  Average -- Response: 6.00  Turnaround 13.33  Wait 6.00
```

采用RR调度算法

```
 ./scheduler-homework.py  -p RR
ARG policy RR
ARG jobs 3
ARG maxlen 10
ARG seed 0

Here is the job list, with the run time of each job:
  Job 0 ( length = 9 )
  Job 1 ( length = 8 )
  Job 2 ( length = 5 )


** Solutions **

Execution trace:
  [ time   0 ] Run job   0 for 1.00 secs
  [ time   1 ] Run job   1 for 1.00 secs
  [ time   2 ] Run job   2 for 1.00 secs
  [ time   3 ] Run job   0 for 1.00 secs
  [ time   4 ] Run job   1 for 1.00 secs
  [ time   5 ] Run job   2 for 1.00 secs
  [ time   6 ] Run job   0 for 1.00 secs
  [ time   7 ] Run job   1 for 1.00 secs
  [ time   8 ] Run job   2 for 1.00 secs
  [ time   9 ] Run job   0 for 1.00 secs
  [ time  10 ] Run job   1 for 1.00 secs
  [ time  11 ] Run job   2 for 1.00 secs
  [ time  12 ] Run job   0 for 1.00 secs
  [ time  13 ] Run job   1 for 1.00 secs
  [ time  14 ] Run job   2 for 1.00 secs ( DONE at 15.00 )
  [ time  15 ] Run job   0 for 1.00 secs
  [ time  16 ] Run job   1 for 1.00 secs
  [ time  17 ] Run job   0 for 1.00 secs
  [ time  18 ] Run job   1 for 1.00 secs
  [ time  19 ] Run job   0 for 1.00 secs
  [ time  20 ] Run job   1 for 1.00 secs ( DONE at 21.00 )
  [ time  21 ] Run job   0 for 1.00 secs ( DONE at 22.00 )

Final statistics:
  Job   0 -- Response: 0.00  Turnaround 22.00  Wait 13.00
  Job   1 -- Response: 1.00  Turnaround 21.00  Wait 13.00
  Job   2 -- Response: 2.00  Turnaround 15.00  Wait 10.00

  Average -- Response: 1.00  Turnaround 19.33  Wait 12.00
```

(２)扩展练习1:理解并实现MLFQ调度算法　可基于python, ruby, C, C++，LISP等）模拟实现，并给出测试，在试验报告写出设计思路和测试结果分析。

(3)扩展练习2:理解并实现stride调度算法　可基于python, ruby, C, C++，LISP等）模拟实现，并给出测试，在试验报告写出设计思路和测试结果分析。

(4)扩展练习３：理解并实现EDF, RM实时调度算法和优先级反置方法　可基于python, ruby, C, C++，LISP等）模拟实现，并给出测试，在试验报告写出设计思路和测试结果分析。

lab6 调度算法　在线练习

选择题

lab6的调度过程包括() s2

- [x] 触发：trigger scheduling
- [x] 入队：‘enqueue’
- [x] 选取：pick up
- [x] 出队：‘dequeue’
- [x] 切换：process switch

> 全部

lab6中涉及到的调度点包括（） s3

- [x] proc.c:do_exit　户线程执行结束，主动放弃CPU
- [x] proc.c:do_wait　用户线程等待子进程结束，主动放弃CPU
- [x] proc.c::cpu_idle　idleproc内核线程选取一个就绪进程并切换
- [x] ｔ rap.c::trap　　若时间片用完，则设置need_resched为1，让当前进程放弃CPU

> 全部

lab6调度算法支撑框架包括的函数指针有（）s4

- [x] (*enqueue)(struct run_queue* rq, …);
- [x] (*dequeue)(struct run_queue* rq, …);
- [x] (*pick_next)(struct run_queue* rq);
- [x] (*proc_tick)(struct run_queue* rq, …);

> 都包括

lab6调度算法支撑框架中与时钟中断相关的函数指针有（）s4

- [ ] (*enqueue)(struct run_queue* rq, …);
- [ ] (*dequeue)(struct run_queue* rq, …);
- [ ] (*pick_next)(struct run_queue* rq);
- [x] (*proc_tick)(struct run_queue* rq, …);

> 4

lab6中的RR调度算法在( )时对当前进程的完成时间片的递减 s5

- [ ] 等待进程结束
- [ ] 进程退出
- [ ] 进程睡眠
- [x] 进程被时钟中断打断

> 4

lab6 spoc 思考题

- 有"spoc"标记的题是要求拿清华学分的同学要在实体课上完成，并按时提交到学生对应的ucore_code和os_exercises的git repo上。

个人思考题

总体介绍

- ucore中的调度点在哪里，完成了啥事？
- 进程控制块中与调度相关的字段有哪些？
- ucore的就绪队列数据结构在哪定义？在哪进行修改？
- ucore的等待队列数据结构在哪定义？在哪进行修改？

调度算法支撑框架

- 调度算法支撑框架中的各个函数指针的功能是啥？会被谁在何种情况下调用？
- 调度函数schedule()的调用函数分析，可以了解进程调度的原因。请分析ucore中所有可能的调度位置，并说明可能的调用原因。

时间片轮转调度算法

- 时间片轮转调度算法是如何基于调度算法支撑框架实现的？
- 时钟中断如何调用RR_proc_tick()的？

stride调度算法

- stride调度算法的思路？
- stride算法的特征是什么？
- stride调度算法是如何避免stride溢出问题的？
- 无符号数的有符号比较会产生什么效果？
- 什么是斜堆(skew heap)?

小组练习与思考题

(1)(spoc) 理解调度算法支撑框架的执行过程

即在ucore运行过程中通过`cprintf`函数来完整地展现出来多个进程在调度算法和框架的支撑下，在相关调度点如何动态调度和执行的细节。(越全面细致越好)

请完成如下练习，完成代码填写，并形成spoc练习报告

> 需写练习报告和简单编码，完成后放到git server 对应的git repo中

练习用的**lab6 spoc exercise project source code**

同步

单选题

操作系统中，两个或多个并发进程各自占有某种资源而又都等待别的进程释放它们所占有的资源的现象叫做什么（）

- [ ] 饥饿
- [x] 死锁
- [ ] 死机
- [ ] 死循环

> 解释：饥饿状态的进程不会进入等待状态，死锁是指两个或多个进程各自占有某种资源而又等待别的进程释放其所占有的资源。

临界资源是什么类型的共享资源（）

- [ ] 临界资源不是共享资源
- [ ] 用户共享资源
- [x] 互斥共享资源
- [ ] 同时共享资源

> 解释：临界资源是指能够被多个进程共享，但是同一时间只能由一个进程访问的资源，因此是互斥的。

要想进程互斥地进入各自的同类资源的临界区，需要（）

- [ ] 在进程间互斥使用共享资源
- [ ] 在进程间非互斥使用临界资源
- [x] 在进程间互斥地使用临界资源
- [ ] 在进程间不使用临界资源

> 解释：临界资源位于临界区，共享资源不一定位于临界区，因此无法保证进程进入临界区；非互斥使用临界资源和不使用临界资源均无法保证进程互斥地进入临界区，因为不存在临界资源的互斥使用的话个进程之间不存在互斥关系。

一个进程由阻塞队列进入就绪队列，可能发生了哪种情况（）

- [x] 一个进程释放一种资源
- [ ] 系统新创建了一个进程
- [ ] 一个进程从就绪队列进入阻塞队列
- [ ] 一个在阻塞队列中的进程被系统取消了

> 解释：一个进程释放了一种资源后，可能该资源正是位于阻塞队列中的一个进程所必需的资源，因此该进程便可以从阻塞队列进入就绪队列；其余三种情况均不会使某个进程从阻塞队列进入就绪队列。

设两个进程共用一个临界区的互斥信号量mutex，当一个进程进入了临界区，另一个进程等待时，mutex应该等于多少（）

- [x] -1
- [ ] 0
- [ ] 1
- [ ] 2

> 解释：两个进程共用一个临界区的互斥信号量mutex，那么mutex的取值范围应该是1到-1，1表示没有进程进入临界区

> 并且也没有进程等待，0表示有一个进程进入临界区，-1表示有一个进程进入临界区并且另一个进程等待。

共享变量是指（）访问的变量

- [ ] 只能被系统进程
- [ ] 只能被多个进程互斥
- [ ] 只能被用户进程
- [x] 可被多个进程

> 解释：共享变量可以被多个进程访问，并且不需要互斥访问，可以访问的进程既可以是系统进程，也可以是用户进程。

临界区是指并发进程中访问共享变量的（）段

- [ ] 管理信息
- [ ] 信息存储
- [ ] 数据
- [x] 程序

> 解释：临界区是指进程中的一段需要访问共享资源并且当另一个进程处于相应代码区域时便不会被执行的代码区域。

假定在一个处理机上执行以下五个作业，其中采用HRN（最高响应比优先）算法时第三个被选择的作业号是（）

作业号	到达时间	运行时间
A	0	4
B	1	3
C	2	5
D	3	2
E	4	4

- [ ] A
- [ ] B
- [ ] C
- [x] D
- [ ] E

> 解释：A到达时没有其他进程到达，所以先处理A，当A处理完时其余所有进程都到达，此时计算各自的响应比：B=(3+3)/3=2, C=(2+5)/5=1.4, D=(1+2)/2=1.5, E=4/4=1。所以第二个被选择的是B，B完成后再次计算响应比：C=(5+5)/5=2, D=(4+2)/2=3, E=(3+4)/4=1.75，所以第三个被选择的是D。

下面关于Bakery算法的描述错误的是（）

- [ ] 进入临界区前，每个进程都会得到一个数字
- [ ] 得到数字最小的进程可以进入临界区
- [x] 如果P2和P4两个进程得到的数字相同，那么P4先进入临界区
- [ ] 数字是按照从小到大生成的\

> 解释：Bakery算法描述：（1）进入临界区之前，进程接收一个数字；（2）得到的数字最小的进入临界区；（3）如果进程Pi和Pj收到相同的数字，那么如果i<j，Pi先进入临界区，否则Pj先进入临界区；（4）编号方案总是按照枚举的增加顺序生成数字

Peterson算法是解决Pi和Pj之间互斥的经典的（）的解决方法

- [ ] 基于中断禁用
- [x] 基于软件
- [ ] 基于硬件

- [ ] 基于原子操作

> 解释：Peterson算法是满足进程Pi和Pj之间互斥的经典的基于软件的解决方法（1981年）。

如果有5个进程共享同一程序段，每次允许3个进程进入该程序段，若用PV操作作为同步机制则信号量S为-1时表示什么（）

- [ ] 有四个进程进入了该程序段
- [ ] 有一个进程在等待
- [x] 有三个进程进入了程序段，有一个进程在等待
- [ ] 有一个进程进入了该程序段，其余四个进程在等待

> 解释：S初始为3，当有一个进程进入程序段或等待时，S减一. S为-1，意味着有四次减1的操作，也即3个进程获准进入，1个在等待。

多选题

产生死锁的必要条件（1345）

- [x] 互斥
- [ ] 可抢占
- [x] 不可抢占
- [x] 占有且申请
- [x] 循环等待

> 解释：产生死锁的四个必要条件：（1）互斥--一个资源每次只能给一个进程使用（2）不可抢占--资源申请者不能强行的从资源占有者手中夺取资源，资源只能由占有者自愿释放（3）占有且申请--一个进程在申请新的资源的同时保持对原有资源的占有（只有这样才是动态申请，动态分配）（4）循环等待--存在一个进程等待队列 {P1 , P2 , … , Pn}, 其中P1等待P2占有的资源，P2等待P3占有的资源，…，Pn等待P1占有的资源，形成一个进程等待环路。

锁的实现方法有哪几种（124）

- [x] 禁用中断
- [x] 软件方法
- [ ] 添加硬件设备
- [x] 原子操作指令

> 解释：实现锁机制的三种方法：禁用中断（仅限于单处理器），软件方法（复杂）和原子操作指令（单处理器或多处理器均可）。锁机制是高等级的编程抽象，因此无法使用硬件设备实现。

判断题

产生死锁的根本原因是供使用的资源数少于需求资源的进程数。

- [x] 对
- [ ] 错

> 解释：死锁是指两个或多个进程各自占有某种资源而又等待别的进程释放其所占有的资源，因此根本原因就是提供的资源少于需求的资源。

一旦出现死锁, 所有进程都不能运行。

- [ ] 对
- [x] 错

> 解释：出现死锁后，处于死锁状态的进程无法继续运行，但是其他无关的进程可以继续运行。

所有进程都挂起时, 系统陷入死锁。

- [ ] 对
- [x] 错

> 解释：死锁是指两个或多个进程各自占有某种资源而又等待别的进程释放其所占有的资源。所有进程都挂起并不代表其无法被完成。

参与死锁的所有进程都占有资源。

- [ ] 对
- [x] 错

> 解释：应该是参与死锁的所有进程都等待资源。不占有资源的进程也可能进入死锁。

有m个进程的操作系统出现死锁时, 死锁进程的个数为1<k≤m。

- [x] 对
- [ ] 错

> 解释：死锁并不会将所有的进程都牵扯进去，但出现死锁时一定会有进程参与。

进程间的互斥是一种特殊的同步关系。

- [x] 对
- [ ] 错

> 解释：基本概念，互斥是实现同步的一种方式，因此也代表一种同步关系。

所有进程都进入等待状态时，系统陷入死锁。

- [ ] 对
- [x] 错

> 解释：产生死锁的四个必要条件：（1）互斥--一个资源每次只能给一个进程使用（2）不可抢占--资源申请者不能强行的从资源占有者手中夺取资源，资源只能由占有者自愿释放（3）占有且申请--一个进程在申请新的资源的同时保持对原有资源的占有（只有这样才是动态申请，动态分配）（4）循环等待--存在一个进程等待队列 {P1 , P2 , … , Pn}, 其中P1等待P2占有的资源，P2等待P3占有的资源，…，Pn等待P1占有的资源，形成一个进程等待环路。

lec17 同步互斥　在线练习

选择题

临界资源是什么类型的共享资源（） s2

- [ ] 临界资源不是共享资源
- [ ] 用户共享资源
- [x] 互斥共享资源
- [ ] 同时共享资源

> 3

操作系统中，两个或多个并发进程各自占有某种资源而又都等待别的进程释放它们所占有的资源的现象叫做什么（） s2

- [ ] 饥饿
- [x] 死锁
- [ ] 死机
- [ ] 死循环

> 2

共享变量是指（）访问的变量　s2

- [ ] 只能被系统进程
- [ ] 只能被多个进程互斥
- [ ] 只能被用户进程
- [x] 可被多个进程

> 4

要想进程互斥地进入各自的同类资源的临界区，需要（） s3

- [ ] 在进程间互斥使用共享资源
- [ ] 在进程间非互斥使用临界资源
- [x] 在进程间互斥地使用临界资源
- [ ] 在进程间不使用临界资源

> 3

锁的实现方法有哪几种（） s4

- [x] 禁用中断
- [x] 软件方法
- [ ] 添加硬件设备
- [x] 原子操作指令

> 124

一个进程由阻塞队列进入就绪队列，可能发生了哪种情况（） s5

- [x] 一个进程释放一种资源
- [ ] 系统新创建了一个进程
- [ ] 一个进程从就绪队列进入阻塞队列
- [ ] 一个在阻塞队列中的进程被系统取消了

```
1
```

同步互斥(lec 17) spoc 思考题

- 有"spoc"标记的题是要求拿清华学分的同学要在实体课上完成，并按时提交到学生对应的ucore_code和os_exercises的git repo上。

个人思考题

背景

- 请给出程序正确性的定义或解释。
- 在一个新运行环境中程序行为与原来的预期不一致，是错误吗？
- 程序并发执行有什么好处和障碍？
- 什么是原子操作？

现实生活中的同步问题

- 家庭采购中的同步问题与操作系统中进程同步有什么区别？
- 如何通过枚举和分类方法检查同步算法的正确性？
- 尝试描述方案四的正确性。
- 互斥、死锁和饥饿的定义是什么？

临界区和禁用硬件中断同步方法

- 什么是临界区？
- 临界区的访问规则是什么？
- 禁用中断是如何实现对临界区的访问控制的？有什么优缺点？

基于软件的同步方法

- 尝试通过枚举和分类方法检查Peterson算法的正确性。
- 尝试准确描述Eisenberg同步算法，并通过枚举和分类方法检查其正确性。

高级抽象的同步方法

- 如何证明TS指令和交换指令的等价性？
- 为什么硬件原子操作指令能简化同步算法的实现？

小组思考题

1. （spoc）阅读简化x86计算机模拟器的使用说明，理解基于简化x86计算机的汇编代码。

2. （spoc)了解race condition. 进入race-condition代码目录。

   - 执行 `./x86.py -p loop.s -t 1 -i 100 -R dx`， 请问 `dx` 的值是什么？
   - 执行 `./x86.py -p loop.s -t 2 -i 100 -a dx=3,dx=3 -R dx` ， 请问 `dx` 的值是什么？
   - 执行 `./x86.py -p loop.s -t 2 -i 3 -r -a dx=3,dx=3 -R dx`， 请问 `dx` 的值是什么？
   - 变量x的内存地址为2000, `./x86.py -p looping-race-nolock.s -t 1 -M 2000`, 请问变量x的值是什么？
   - 变量x的内存地址为2000, `./x86.py -p looping-race-nolock.s -t 2 -a bx=3 -M 2000`, 请问变量x的值是什么？为何每个线程要循环3次？
   - 变量x的内存地址为2000, `./x86.py -p looping-race-nolock.s -t 2 -M 2000 -i 4 -r -s 0`， 请问变量x的值是什么？

- 变量x的内存地址为2000, `./x86.py -p looping-race-nolock.s -t 2 -M 2000 -i 4 -r -s 1`, 请问变量x的值是什么？
- 变量x的内存地址为2000, `./x86.py -p looping-race-nolock.s -t 2 -M 2000 -i 4 -r -s 2`, 请问变量x的值是什么？
- 变量x的内存地址为2000, `./x86.py -p looping-race-nolock.s -a bx=1 -t 2 -M 2000 -i 1`, 请问变量x的值是什么？

3. （spoc） 了解software-based lock, hardware-based lock, software-hardware-lock代码目录

   - 理解flag.s,peterson.s,test-and-set.s,ticket.s,test-and-test-and-set.s 请通过x86.py分析这些代码是否实现了锁机制？请给出你的实验过程和结论说明。能否设计新的硬件原子操作指令Compare-And-Swap,Fetch-And-Add？ ```
     Compare-And-Swap

int CompareAndSwap(int *ptr, int expected, int new) { int actual* = ptr; if (actual == expected) *ptr = new; return actual; }

```

Fetch-And-Add

int FetchAndAdd(int *ptr) { int old* = ptr; *ptr = old + 1; return old; } ```
```

lec18 信号量与管程　在线练习

选择题

如果有5个进程共享同一程序段，每次允许3个进程进入该程序段，若用PV操作作为同步机制则信号量S为-1时表示什么（） s1

- [ ] 有四个进程进入了该程序段
- [ ] 有一个进程在等待
- [x] 有三个进程进入了程序段，有一个进程在等待
- [ ] 有一个进程进入了该程序段，其余四个进程在等待

> 3

2元信号量可以初始化为（） s2

- [x] 0或1
- [ ] 0或-1
- [ ] 只能为1
- [ ] 任意值

> 1

多个进程对信号量S进行了6次P操作，2次V操作后，现在信号量的值是-3，与信号量S相关的处于阻塞状态的进程有几个（） s2

- [ ] 1个
- [ ] 2个
- [x] 3个
- [ ] 4个

> 3

(2011年全国统考)有两个并发执行的进程P1和P2，共享初值为1的变量x。P1对x加1，P2对x减一。加1和减1操作的指令序列分别如下所示,两个操作完成后，x的值（）。 s2

```
加一操作            减一操作
Load R1,x          load R2,x
inc R1             dec R2
store x,R1         store x,R2
```

- [ ] 可能为-1或3
- [ ] 只能为1
- [x] 可能为0、1或2
- [ ] 可能为-1、0、1、1或2

> 3

管程的主要特点有（） s3

- [x] 局部数据变量只能被管程的过程访问

- [x] 一个进程通过调用管程的一个过程进入管程
- [ ] 不会出现死锁
- [x] 在任何时候，只能有一个进程在管程中执行

> 124

关于管程的叙述正确的是（） s3

- [ ] 管程中的局部数据变量可以被外部直接访问
- [ ] 当一个进程在管程中执行时，调用管程的其他进程都不会被阻塞
- [ ] 在管程中的signal()与信号量中的signal()操作实现及意义完全相同
- [x] 管程通过使用条件变量提供对同步的支持，这些条件变量包含在管程中，并且只有管程才能访问

> 4

同步互斥(lec 18) spoc 思考题

- 有"spoc"标记的题是要求拿清华学分的同学要在实体课上完成，并按时提交到学生对应的ucore_code和os_exercises的git repo上。

个人思考题

基本理解

- 什么是信号量？它与软件同步方法的区别在什么地方？
- 什么是自旋锁？它为什么无法按先来先服务方式使用资源？
- 下面是一种P操作的实现伪码。它能按FIFO顺序进行信号量申请吗？

```
while (s.count == 0) {  //没有可用资源时，进入挂起状态；
     调用进程进入等待队列s.queue;
     阻塞调用进程;
}
s.count--;              //有可用资源，占用该资源；
```

> 参考回答： 它的问题是，不能按FIFO进行信号量申请。 它的一种出错的情况
>
> ```
> 一个线程A调用P原语时，由于线程B正在使用该信号量而进入阻塞状态；注意，这时value的值为0。
> 线程B放弃信号量的使用，线程A被唤醒而进入就绪状态，但没有立即进入运行状态；注意，这里value为1。
> 在线程A处于就绪状态时，处理机正在执行线程C的代码；线程C这时也正好调用P原语访问同一个信号量，并得到使用权。注意，这时value又变回0。
> 线程A进入运行状态后，重新检查value的值，条件不成立，又一次进入阻塞状态。
> 至此，线程C比线程A后调用P原语，但线程C比线程A先得到信号量。
> ```

信号量使用

- 什么是条件同步？如何使用信号量来实现条件同步？
- 什么是生产者-消费者问题？
- 为什么在生产者-消费者问题中先申请互斥信息量会导致死锁？

管程

- 管程的组成包括哪几部分？入口队列和条件变量等待队列的作用是什么？
- 为什么用管程实现的生产者-消费者问题中，可以在进入管程后才判断缓冲区的状态？
- 请描述管程条件变量的两种释放处理方式的区别是什么？条件判断中while和if是如何影响释放处理中的顺序的？

哲学家就餐问题

- 哲学家就餐问题的方案2和方案3的性能有什么区别？可以进一步提高效率吗？

读者-写者问题

- 在读者-写者问题的读者优先和写者优先在行为上有什么不同？
- 在读者-写者问题的读者优先实现中优先于读者到达的写者在什么地方等待？

小组思考题

1. （spoc） 每人用python threading机制用信号量和条件变量两种手段分别实现47个同步问题中的一题。向勇老师的班级从前往后，陈渝老师的班级从后往前。请先理解]python threading 机制的介绍和实例

lec19 lab7 同步互斥　在线练习

选择题

ucore为支持内核中的信号量机制，需用到的支撑机制包括（） s2 底层支撑

- [x] 处理器调度
- [x] 屏蔽中断
- [x] 等待队列
- [ ] 动态内存分配

> 需用到前三个，动态内存分配不是必须的

ucore实现的信号量机制被用于（） s3 信号量设计与实现

- [x] 条件变量实现
- [x] mm内存管理实现
- [x] 哲学家问题实现
- [ ] 中断机制实现

> 中断机制是支持信号量的，所以不选

关于ucore实现的管程和条件变量的阐述正确的是（） s4 管程和条件变量设计实现

- [x] 管程中采用信号量用于互斥操作
- [x] 管程中采用信号量用于同步操作
- [x] 管程中采用条件变量用于同步操作
- [x] 属于管程的共享变量访问的函数需要用互斥机制进行保护

> 都对

同步互斥(lec 19) spoc 思考题

- 有"spoc"标记的题是要求拿清华学分的同学要在实体课上完成，并按时提交到学生对应的ucore_code和os_exercises的git repo上。

个人思考题

总体介绍

- 信号量与条件变量有什么不同？
- 同步机制、处理机调度、等待队列和定时器有些什么相互影响？

底层支撑

- 操作系统内核如何利用定时器实现sleep系统？在定时队列中保存的时间格式是什么？
- 中断屏蔽控制位在哪个寄存器中？如何修改中断屏蔽控制位？
- 在ucore中有多少种等待队列？
- 等待队列的两个基本操作down()和up()对线程状态和等待队列有什么影响？

信号量设计实现

- ucore中的信号量实现能实现是按FIFO获取信号量资源吗？给出你的理由。
- 为什么down()要加一个_down()函数来进行实质的处理？类似情况还出现在up()和_up()。

  > 参考回答：有多个有小差异的类似函数要使用相同的核心功能实现。类似情况还出现在do_fork()函数。

管程和条件变量设计实现

- 管程与信号量是等价的。如何理解？
- 基于信号量的管程中在什么地方用到信号量？
- 管程实现中的monitor.next的作用是什么？
- 分析管程实现中PV操作的配对关系，并解释PV操作的目的。
- 基于视频中对管程的17个状态或操作的分析，尝试分析管程在入口队列和条件变量等待队列间的等待和唤醒关系。注意分析各队列在什么情况下会有线程进入等待，在什么时候会被唤醒，以及这个等待和唤醒的依赖关系。

哲学家就餐问题

- 哲学家就餐问题的管程实现中的外部操作成员函数有哪几个？
- 哲学家就餐问题的管程实现中用了几个条件变量？每个条件变量的作用是什么？

小组思考题

1. (扩展练习) 每人用ucore中的信号量和条件变量两种手段分别实现47个同步问题中的一题。向勇老师的班级从前往后，陈渝老师的班级从后往前。请先理解与采用python threading 机制实现的异同点。

死锁和进程间通信

单选题

若P,V操作的信号量S初值为4,当前值为-1,则表示有（）进程处于等待状态。

- [ ] 0
- [x] 1
- [ ] 2
- [ ] 3

> p操作会使s减1，如果s<0，则p操作进程进入等待；v操作会使s加1，如果s<=0,则会唤醒一个等待的程序。处于等待状态的进程的数目只和信号量当前值有关，而和信号量的初始值无关。

任何两个并发进程之间（）。

- [ ] 一定存在互斥关系
- [ ] 一定存在同步关系
- [ ] 一定彼此独立无关
- [x] 可能存在同步或互斥关系

> 如果两个并发程序为互斥关系，则必定存在临界区，但是实际上不是多有的并发程序之间都存在临界区；我们把异步环境下的一组并发进程因直接制约而互相发送消息、进行互相合作、互相等待，使得各进程按一定的速度执行的过程称为进程间的同步，实际上不是所有的程序间都存在直接制约关系。所有两个并发的程序之间只是有可能存在同步或互斥关系。

银行家算法是一种（）算法。

- [ ] 死锁解除
- [x] 死锁避免
- [ ] 死锁预防
- [ ] 死锁检测

> 银行家算法是一种最有代表性的避免死锁的算法。在避免死锁方法中允许进程动态地申请资源，但系统在进行资源分配之前，应先计算此次分配资源的安全性，若分配不会导致系统进入不安全状态，则分配，否则等待。

在为多道程序所提供的可共享的系统资源不足时，可能出现死锁。但是，不适当的（）也可能产生死锁。

- [ ] 进程优先权
- [ ] 资源的线性分配
- [x] 进程推进顺序
- [ ] 分配队列优先权

> 不合理的进程推进顺序可能产生死锁的原因是相互等待资源。如进程p1申请资源的顺序是资源1和资源2，进程p2申请资源的顺序是资源2和资源1；这时当两个进行都申请成功了第一个资源后，在申请第二个资源的时候就会出现死锁。

产生死锁的四个必要条件是：互斥、( )、循环等待和不剥夺。

- [ ] 请求与阻塞
- [x] 请求与保持
- [ ] 请求与释放

- [ ] 释放与阻塞

> 互斥是一次只有一个进程可以使用一个资源，其他进程不能访问已分配给其他进程的资源；请求与保持是当一个进程等待其他进程时，继续占有已经分配的资源；不剥夺是不能强行抢占进程已占有的资源；循环等待指存在一个封闭的进程链，使得每个进程至少占有此链中下一个进程所需要的一个资源。这四个是产生死锁的必要条件。

在下列解决死锁的方法中，属于死锁预防策略的是 ( )。

- [ ] 银行家算法（死锁避免）
- [x] 资源有序分配法
- [ ] 死锁检测法
- [ ] 资源分配图化简法

> 资源有序分配法将资源按某种规则系统中的所有资源统一编号，申请的时候必须按照编号的顺序申请。对进行必须使用的同类资源，必须一次申请；不同类的资源必须按照资源编号顺序申请，这样就破坏了死锁环路。

采用资源剥夺法可以解除死锁，还可以采用 ( ) 方法解除死锁。

- [ ] 执行并行操作
- [x] 撤销进程
- [ ] 拒绝分配新资源
- [ ] 修改信号量

> 当检测到系统中已发生死锁时，须将进程从死锁状态中解脱出来。常用的实施方法是撤销或挂起一些进程，以便回收一些资源，再将这些资源分配给已处于阻塞状态的进程，使之转为就绪状态，以继续运行。

进程从运行态进入阻塞态可能是由于（）。

- [ ] 现运行进程运行结束
- [x] 现运行进程执行了P操作
- [ ] 现运行进程执行了V操作
- [ ] 现运行进程时间片用完

> p操作使信号量减1，表明程序申请了资源，当信号量小于0时，表明没有可供使用的资源，程序将从运行态进入阻塞态。

在 ( )情况下，系统出现死锁。

- [ ] 计算机系统发生了重大故障
- [ ] 有多个封锁的进程同时存在
- [x] 若干进程因竞争而无休止地相互等待他方释放已占有的资源
- [ ] 资源数远远小于进程数或进程同时申请的资源数量远远超过资源总数

> 死锁产生的必要条件是互斥、请求与保持、不可抢占、循环等待。所以当若干进程因竞争而无休止地相互等待他方释放已占有的资源时，系统会产生死锁。

若信号量S的初值为2，且有三个进程共享此信号量，则S的取值范围是( )。

- [ ] [-3,2]
- [ ] [-2,2]
- [x] [-1,2]
- [ ] [0,2]

> 因为s的初始值为2，而有3个进行共享信号量，所以s的最小值为2-3=-1，最大值为2-0=2。

对于记录型信号量，在执行一次P操作(wait操作)时，信号量的值应当为减1；当其值为( )时，进程应阻塞。

- [ ] 大于0
- [x] 小于0
- [ ] 大于等于0
- [ ] 小于等于0

> 在执行p操作时，如果信号量的值小于0，则表明没有资源可以分配了，进行应进入等待状态。

预防死锁的论述中，（）条是正确的论述。

- [ ] 由于产生死锁的基本原因是系统资源不足，因而预防死锁的有效方法，是根据系统规模，配置足够的系统资源。
- [ ] 由于产生死锁的另一种基本原因是进程推进顺序不当，因而预防死锁的有效方法，是使进程的推进顺序合法。
- [ ] 因为只要系统不进入不安全状态，便不会产生死锁，故预防死锁的有效方法，是防止系统进入不安全状态。
- [x] 可以通过破坏产生死锁的四个必要条件之一或其中几个的方法，来预防发生死锁。

> 死锁是因为若干进程因竞争而无休止地相互等待他方释放已占有的资源，通过破坏四个必要条件，可以预防死锁产生。通过增加系统资源的方法相当于增加了信号量的初始值，可以一定程度上减少死锁的出现，但是当众多进程申请同一资源时，还是会出现死锁的情况；合理的进程推进顺序可以降低死锁出现的可能，但是当四个必要条件存在时，还是有出现死锁的可能；让系统不进入安全状态是可以预防死锁，但是因为破坏了互斥关系，导致程序执行错误。

操作系统中，进程与程序的重要区别之一是( )。

- [ ] 程序有状态而进程没有
- [x] 进程有状态而程序没有
- [ ] 程序可占有资源而进程不可
- [ ] 进程能占有资源而程序不能

> 进程是一个“执行中的程序”，所以进程是有状态的，而程序只是一个静态的概念，并不存在状态。

进程从阻塞状态进入就绪状态可能是由于( )。

- [ ] 现运行进程运行结束
- [ ] 现运行进程执行了P操作
- [x] 现运行进程执行了V操作
- [ ] 现运行进程时间片用完

> v操作会使信号量加1，表明有程序释放了资源，而需要该资源的一个进程会被唤醒，成阻塞状态转换成就绪状态，准备获取资源并执行。

发生死锁的必要条件有四个，要防止死锁的发生，可以破坏这四个必要条件，但破坏( )条件是不太实际的。

- [x] 互斥
- [ ] 不可抢占
- [ ] 占有且等待
- [ ] 循环等待

> 互斥是不可能被禁止的，因为如果需要对资源进行互斥访问，那么操作系统必须支持互斥；预防占有且等待可以要求进行一次性请求所有需要的资源，并且阻塞这个进程直到所有请求都同时满足；对于不可抢占可以要求如果占有某些资源的一个进程进一步申请资源时被拒绝，则该进行必须释放它最初占有的资源；循环等待可以通过定义资源类型的线性顺序来预防。

(北京理工大学)资源的有序分配策略可以破坏死锁的（）条件。

- [ ] 互斥
- [ ] 请求和保持
- [ ] 不剥夺

- [x] 循环等待

> 资源的有序分配策略属于死锁预防的一种，死锁预防是通过破坏4个必要条件中的1个或者多个以确保系统不会发生死锁。采用资源有序分配法是破坏了“环路”条件，即破坏了循环等待。

(南京理工大学)一进程在获得资源后，只能在使用完资源后由自己释放，这属于死锁必要条件的（）。

- [ ] 互斥条件
- [ ] 请求和释放条件
- [x] 不剥夺条件
- [ ] 环路等待条件

> 死锁的必要条件包括互斥、请求和保持、不可剥夺、循环等待。如果一个程序占有的资源只能由其使用完后自己释放，则满足其中的不剥夺条件。

(四川大学)死锁产生的原因之一是：（）。

- [ ] 系统中没有采用Spooling技术
- [ ] 使用PV操作过多
- [ ] 有共享资源存在
- [x] 资源分配不当

> 任何系统的资源都是有限的，所以不恰当的资源分配可能会导致死锁的产生。

(南京理工大学)计算机系统产生死锁的根本原因是（）。

- [ ] 资源有限
- [ ] 进程推进顺序不当
- [ ] 系统中进程太多
- [x] A和B

> 产生死锁的原因有两个，一是系统提供的资源不能满足每个进程的使用需求；二是在多道程序运行时，进程推进顺序不合法。

(上海交通大学)某系统中有11台打印机，N个进程共享打印机资源，每个进程要求3台，当N不超过（）时，系统不会死锁。

- [ ] 4
- [x] 5
- [ ] 6
- [ ] 7

> 考虑下面的极端情况，每个进程都刚好分到了2台打印机，则只需要再分到一台打印机，某个进程就可以获得该打印机，完成自己的工作，并释放所有的打印机。其他的进程就可以完成，这样，N*2+1=11,所以N=5

(电子科技大学)死锁定理是用于处理死锁的哪一种方法（）。

- [ ] 预防死锁
- [ ] 避免死锁
- [x] 检测死锁
- [ ] 解除死锁

> 死锁定理是操作系统中用于检测死锁的充分必要条件的方法，所以死锁定理属于检测死锁。

(青岛大学)通常，（）是预防系统死锁的主要策略。

- [ ] 动态分配与静态分配相结合

- [ ] 静态分配与银行家算法相结合
- [ ] 死锁检测与死锁解除相结合
- [x] 静态分配、剥夺式分配和按序分配

> 死锁预防是通过破坏4个必要条件中的1个或者多个以确保系统不会发生死锁。采用资源的静态预分配策略，破坏了保持和请求；运行进程剥夺使用其他进程占有的资源，从而破坏不剥夺性条件，采用资源有序分配法，破坏了循环等待条件。

(兰州大学)死锁检测检查的是（）。

- [x] 资源分配图
- [ ] 前趋图
- [ ] 搜索树
- [ ] 安全图

> 如果资源分配图中不存在环路，则系统不存在死锁；反之如果资源分配图中存在环路，则系统可能存在死锁，也可能不存在死锁。

(兰州大学)采用资源剥夺法可以解除死锁，还可以采用（）方法解除死锁。

- [ ] 执行并行操作
- [x] 撤销进程
- [ ] 拒绝分配资源
- [ ] 修改信号量

> 解除死锁通常的做法有三种：一是撤销处于死锁状态的进程并收回它们的资源；二是资源剥夺法；三是进程回退。所以这里选择撤销进程。

(四川大学)当进程A使用磁带机时，进程B又申请该磁带机，这种情况（）。

- [ ] 是不可能出现的
- [ ] 是没法解决的
- [ ] 就是死锁
- [x] 以上均不正确

> 首先，这种情况在多道程序系统中是可能出现的，甚至是会经常出现的。同时，死锁是指多个进程因竞争资源而形成的一种僵局，若无外力作用，这些进程都将永远不能再向前推进。通常情况下，进程都在等待彼此已经占据的资源。本题中的情况没有构成死锁。

(电子科技大学)下面关于检测死锁的正确描述是（）。

- [ ] 银行家算法是典型的检测死锁算法
- [ ] 检测死锁中系统需要反复检测各个进程资源申请和分配情况
- [ ] 检测死锁是预防卷入了死锁
- [x] 检测死锁方法对系统吱呀u年的分配不加限制，只要有则可以进行分配

> 银行家算法是死锁避免算法，死锁检测方法是对资源分配不加限制，即允许死锁发生。但是系统定时地运行一个死锁检测程序，判断系统是否已发生死锁，若检测到死锁发生，则设法加以解除。

判断题

死锁与程序的死循环一样。

- [ ] 对
- [x] 错

> 死锁是因为若干进程因竞争而无休止地相互等待他方释放已占有的资源，造成程序不能顺利执行；而程序的死循环的程序在执行逻辑上存在缺陷，导致程序不能够结束循环。

信号量机制中，P、V操作必须成对出现。

- [x] 对
- [ ] 错

> p操作为申请资源操作，p操作成功执行后，信号量会减1；v操作为释放资源操作，v操作执行成功后，信号量会加1；所有资源的申请和释放必须成对出现。

当系统同时具备了死锁的四个必要条件时就肯定会产生死锁。

- [ ] 对
- [x] 错

> 在系统存在死锁的四个必要条件只是表明该系统可能会出现死锁，而不是肯定会产生死锁。在存在这四个必要条件的时候可以通过明智的选择，确保永远都不会到达死锁点。

死锁是指两个或多个进程都处于互等状态而无法继续工作。

- [x] 对
- [ ] 错

> 死锁是因为若干进程因竞争而无休止地相互等待他方释放已占有的资源。

死锁避免比死锁预防对系统条件限制更严格，所以使得系统资源利用率不高。

- [ ] 对
- [x] 错

> 死锁预防是通过破坏死锁的四个必要条件来预防死锁，但这样会导致低效的资源使用和低效的进程执行；死锁避免则相反，它允许死锁的互斥、占有且等待、不可抢占三个条件，但通过明智的选择，确保永远不会到达死锁点，因此死锁避免比死锁预防允许更多的并发，所以其资源利用率要高于死锁预防。

---

lec20 死锁和进程间通信　在线练习

选择题

死锁产生的必要条件包括（） s1

- [x] 互斥
- [x] 持有并等待
- [x] 非抢占
- [x] 循环等待

> 都是

死锁处理方法主要包括（） s2

- [x] 死锁预防(Deadlock Prevention):确保系统永远不会进入死锁状态
- [x] 死锁避免(Deadlock Avoidance):在使用前进行判断，只允许不会出现死锁的进程请求资源
- [x] 死锁检测和恢复(Deadlock Detection & Recovery)：在检测到运行系统进入死锁状态后，进行恢复
- [x] 由应用进程处理死锁：通常操作系统忽略死锁

> 都是

可以使用银行家算法_ 死锁。s3

- [ ] 预防
- [ ] 检测
- [ ] 解除
- [x] 避免

> 是死锁避免

对于进程个数为n，资源类型为m的死锁检测算法的时间复杂度为（） s4

- [x] O(m*n^2)
- [ ] O(m^2*n)
- [ ] O(m^2*n^2)
- [ ] O(m*n)

> 是O(m*n^2)

关于进程通信原理的阐述正确的是（） s5

- [x] 进程通信是进程进行通信和同步的机制
- [x] 进程通信可划分为阻塞（同步）或非阻塞（异步）
- [x] 进程通信可实现为直接通信和间接通信
- [x] 进程通信的缓冲区是有限的

> 都对

关于信号和管道的进程通信机制的阐述正确的是（） s6

- [x] 信号（signal）是一种进程间的软件中断通知和处理机制
- [x] 信号的接收处理方式包括：捕获(catch)，忽略(Ignore)，屏蔽（Mask）
- [x] 管道（pipe）是一种进程间基于内存文件（或内存缓冲区 ）的通信机制
- [ ] 管道（pipe）的实现需要在磁盘文件系统上创建一个文件

> 管道（pipe）的实现只需基于内存即可。

关于消息队列和共享内存的进程通信机制的阐述正确的是（） s7

- [x] 消息队列是由操作系统维护的以字节序列为基本单位的间接通信机制
- [x] 共享内存是把同一个物理内存区域同时映射到多个进程的内存地址空间的通信机制
- [x] 消息队列机制可用于进程间的同步操作
- [x] 共享内存机制可用于进程间的数据共享

> 都对

死锁与IPC(lec 20) spoc 思考题

- 有"spoc"标记的题是要求拿清华学分的同学要在实体课上完成，并按时提交到学生对应的ucore_code和os_exercises的git repo上。

个人思考题

死锁概念

- 尝试举一个生活中死锁实例。
- 可重用资源和消耗资源有什么区别？

可重用和不可撤销；

- 资源分配图中的顶点和有向边代表什么含义？
- 出现死锁的4个必要条件是什么？

死锁处理方法

- 死锁处理的方法有哪几种？它们的区别在什么地方？
- 安全序列的定义是什么？

进程的最大资源需要量小于可用资源与前面进程占用资源的总合；

- 安全、不安全和死锁有什么区别和联系？

银行家算法

- 什么是银行家算法？
- 安全状态判断和安全序列是一回事吗？

死锁检测

- 死锁检测与安全状态判断有什么区别和联系？

> 死锁检测、安全状态判断和安全序列判断的本质就是资源分配图中的循环等待判断。

进程通信概念

- 直接通信和间接通信的区别是什么？ 本质上来说，间接通信可以理解为两个直接通信，间接通信中假定有一个永远有效的直接通信方。
- 同步和异步通信有什么区别？

信号和管道

- 尝试视频中的信号通信例子。
- 写一个检查本机网络服务工作状态并自动重启相关服务的程序。
- 什么是管道？

消息队列和共享内存

- 写测试用例，测试管道、消息队列和共享内存三种通信机制进行不同通信间隔和通信量情况下的通信带宽、通信延时、带宽抖动和延时抖动方面的性能差异。

小组思考题

- （spoc） 每人用python实现银行家算法。大致输出可参考参考输出。除了 `YOUR CODE` 部分需要填写代码外，在算法的具体实现上，效率也不高，可进一步改进执行效率。
- (spoc) 以小组为单位，请思考在lab1~lab5的基础上，是否能够实现IPC机制，请写出如何实现信号，管道或共享内存（三选一）的设计方案。
- (spoc) 扩展：用C语言实现某daemon程序，可检测某网络服务失效或崩溃，并用信号量机制通知重启网络服务。信号机制的例子
- (spoc) 扩展：用C语言写测试用例，测试管道、消息队列和共享内存三种通信机制进行不同通信间隔和通信量情况下的通信带宽、通信延时、带宽抖动和延时抖动方面的性能差异。管道的例子

文件系统

单选题

文件系统的主要目的是（）。

- [x] 实现对文件的按名存取
- [ ] 实现虚拟存贮器
- [ ] 提高外围设备的输入输出速度
- [ ] 用于存贮系统文档

> 在现在操作系统中，几乎都有一个文件管理系统，这个系统的目的主要实现对文件的按名存取。

按逻辑结构划分，文件主要有两类，UNIX中的文件系统采用（）。

- [ ] 网状文件
- [ ] 只读文件
- [ ] 读写文件
- [ ] 记录式文件
- [ ] 索引文件
- [x] 流式文件

> 按文件的逻辑结构可分为记录文件和流式文件，而UNIX、DOS、WINDOWS系统中的普通文件都是流式文件。

通常，文件的逻辑结构可以分为两大类：无结构的（）和有结构的记录式文件。

- [ ] 堆文件
- [x] 流式文件
- [ ] 索引文件
- [ ] 直接（Hash）文件

> 按文件的逻辑结构可分为记录文件和流式文件。

链接文件解决了顺序结构中存在的问题，它（）。

- [x] 提高了存储空间的利用率
- [ ] 适合于随机存取方式
- [ ] 不适用于顺序存取
- [ ] 指针存入主存，速度快

> 顺序结构：把逻辑文件的记录（内容）按其本身的顺序（逻辑记录的顺序）在磁盘上也按序存放在连续的块中。读取时也从第一个记录开始按顺序进行。在文件目录中指出文件名，存放的起始块号和占用块数。其最大优点是存取速度快。而问题主要是存储空间利用率不高、输出文件时难以估计需要多少磁盘块、影响文件扩展。链接结构：如果逻辑文件中的各个逻辑记录任意存放到一些磁盘块中，再用指针把各个块按逻辑记录的顺序链接起来，在文件目录中只记录第一块的地址和最后一块的地址，那么这种文件组织方式就是链接结构。链接结构解决了顺序结构中的所有问题，所有空闲块都可以被利用，在顺序读取时效率较高但需要随机存取时效率低下（因为要从第一个记录开始读取查。

文件管理实际上是对（2）的管理。

- [ ] 主存空间
- [x] 辅助存储空间

- [ ] 逻辑地址空间
- [ ] 物理地址空间

> 从系统角度看，文件系统是一个负责文件存储空间的管理机构，文件管理实际是对辅助存储空间的管理。

下面关于索引文件的论述中，第（）条是正确的论述。

- [ ] 索引文件中，索引表的每个表项中含有相应记录的关键字和存放该记录的物理地址。
- [x] 对顺序文件进行检索时，首先从FCB中读出文件的第一个盘块号；而对索引文件进行检索时，应先从FCB中读出文件索引表始址。
- [ ] 对于一个具有三级索引表的文件，存取一个记录通常要访问三次磁盘。
- [ ] 在文件较大时，无论是进行顺序存取还是随机存取，通常都是以索引文件方式为最快。

> 索引结构 ：索引结构是实现非连续存储的另一种方法，索引结构为每个文件建立一张“索引表”，把指示每个逻辑记录存放位置的指针集中在索引表中。（最直观的索引结构就比如我们的网站，首页就相当于一个索引表，每个链接记录了一个文件的位置，当我们点击时，就可以找到那个文件）。文件目录中指出文件名的索引表位置，而索引表中每个项指出一个逻辑记录的存放位置。存取文件时根据索引表中的登记项来查找磁盘上的逻辑记录。索引结构既适合顺序存取记录，也可以方便地随机存取记录，并且容易实现记录的增删和插入，所以索引结构被广泛应用。但是索引结构增加了索引表，要占用部分空间并增加读写索引表的时间。当索引项很多时，还要考虑采用多级索引结构。

下面关于顺序文件和链接文件的论述中错误的论述是（）。

- [x] 顺序文件适于建立在顺序存储设备上，而不适合建立在磁盘上。
- [ ] 在链接文件中是在每个盘块中设置一链接指针，用于将文件的所有盘块链接起来。
- [ ] 顺序文件必须采用连续分配方式，而链接文件和索引文件则都可采取离散分配方式。
- [ ] 在MS-DOS中采用的是链接文件结构。
- [ ] 链接文件解决了顺序结构中存在的问题，它提高了存储空间的利用率。

> 看关于顺序文件、链接文件和索引文件的介绍。

在文件系统中，（）要求逻辑记录顺序与磁盘块顺序一致。

- [x] 顺序文件
- [ ] 链接文件
- [ ] 索引文件
- [ ] 串联文件

> 看4关于顺序文件、链接文件和索引文件的介绍。

下列文件中，（）的物理结构不便于文件的扩充。

- [x] 顺序文件
- [ ] 链接文件
- [ ] 索引文件
- [ ] 多级索引文件

> 看关于顺序文件、链接文件和索引文件的介绍。

()的物理结构对文件随机存取时必须按指针进行，效率较低。

- [ ] 连续文件
- [x] 链接文件
- [ ] 索引文件
- [ ] 多级索引文件

> 看关于顺序文件、链接文件和索引文件的介绍。

一个采用二级索引文件系统，存取一块盘块信息通常要访问（）次磁盘。

- [ ] 1
- [ ] 2
- [x] 3
- [ ] 4

> 二级索引取需要访问2次，存需要一次，共需要三次。

设有一个包含1000个记录的索引文件，每个记录正好占用一个物理块。一个物理块可以存放10个索引表目。建立索引时，一个物理块应有一个索引表目，试问索引及其文件本身应占（）个物理块？

- [ ] 1000
- [ ] 1001
- [ ] 1011
- [x] 1111

> 共1000个记录，即有1000个索引表目，索引级数：lg1000=3；一个物理块可以放10个索引表目，三级索引需1000/10=100个物理块；二级索引：100/10=10；一级索引10/10=1。所以索引及文件共需1000+100+10+1=1111物理快。

打开文件操作的使用是（）。

- [ ] 把整个文件从磁盘拷贝到内存
- [x] 把文件目录项(FCB)从磁盘拷贝到内存
- [ ] 把整个文件和文件目录项(FCB)从磁盘拷贝到内存
- [ ] 把磁盘文件系统的控制管理信息从辅存读到内存

> 文件目录项是系统管理文件的必须信息结构，是文件存在的唯一标志，打开文件把文件目录项(FCB)从磁盘拷贝到内存。

如果文件系统中有两个文件重名，不应采用（1）。

- [x] 单级目录结构
- [ ] 树型目录结构
- [ ] 二级目录结构
- [ ] 单级和二级目录结构

> 一级目录结构，要求所有的文件名均不相同，一般只适用于微机的单用户系统。

文件系统采用二级文件目录可以（）。

- [ ] 缩短访问存储器的时间
- [ ] 实现文件共享
- [ ] 节省内存空间
- [x] 解决不同用户间的文件命名冲突

> 二级目录结构 则增加一级主文件目录，此目录是为用户建立的独立文件目录，用户访问文件时先要找到用户自己的目录再查找该目录下的指定文件。实际上，二级目录结构中，文件系统把用户名和文件名合起来作为文件标识。

【2010年计算机联考真题】设当前工作目录的主要目的是（）。

- [ ] 节省外存空间
- [ ] 节省内存空间

- [x] 加快文件的索引速度
- [ ] 加快文件的读/写速度

> 当文件系统含有多级目录时，每访问一个文件，都要使用从树根开始到树叶为止、包含中间节点名的全路径名。当前目录又称工作目录，进程对各个文件的访问都是相对于当前目录进行，而不需要一层一层的检索，加快了文件的检索速度。 选项12都是与相对目录无关；选项4加快文件的读/写速度取决于磁盘的性能。

【2009年计算机联考真题】文件系统中，文件访问控制信息存储的合理位置是（）。

- [x] 文件控制块
- [ ] 文件分配表
- [ ] 用户口令表
- [ ] 系统注册表

> 为了实现“按名存取”，文件系统为每个文件设置用于描述和控制文件的数据结构，称为文件控制块（FCB）。在文件控制块中，通常包含三类信息，即基本信息、存取控制信息级使用信息。

【2012年计算机联考真题】若一个用户进程通过read系统调用读取一个磁盘文件中的数据，则下列关于此进程的叙述中，正确的是（）。 I. 若文件的数据不在内存中，则该进程进入睡眠等待状态 II. 请求read系统调用会导致CPU从用户态切到核心态 III. read系统调用的参数应包含文件的名称

- [x] 仅I、II
- [ ] 仅I、III
- [ ] 仅II、III
- [ ] I、II和III

> 对于I，当所读文件的数据不在内存是，产生中断（缺页中断），原进程进入阻塞状态，知道所需数据从外存调入内存后，才将该进程唤醒。对于II，read系统调用通过陷入将CPU从用户态进入核心态，从而获取操作系统提供的服务。对于III，读一个文件首先要用open系统调用打开该文件。open参数包含文件的路径名与文件名，read只需要open返回的文件描述符，不用文件名作为参数。read要求三个输入参数：1文件描述符fd；2buf缓冲区首地址；3传送的字节数n。read的功能试图从fd所指示的文件中读入n个字节的数据，并将它们送到buf所指示的缓冲区中。

【2013年计算机联考真题】用户删除某文件的过程中，操作系统不可能执行的操作是（）。

- [x] 删除文件所在的目录
- [ ] 删除与此文件关联的目录项
- [ ] 删除与此文件对应的文件控制块
- [ ] 释放与此文件关联的内存缓冲区

> 此文件的目录下可能还存在其他的文件，因此删除文件不能删除文件所在的目录，而与此文件关联的目录项和文件控制块需要随着文件一同删除，同时释放文件关联的内存缓冲区。

【2009年计算机联考真题】 设文件F1的当前引用计数值为1，先建立文件F1的符号链接（软链接）文件F2，在建立文件F1的硬链接F3，然后删除文件F1.此时，文件F2和文件F3的引用技术支持分别是（）。

- [ ] 0,1
- [x] 1,1
- [ ] 1,2
- [ ] 2,1

> 建立符号链接时，引用计数直接复制；建立硬链接时，引用计数加1.删除文件时， 删除操作对于符号链接是不可见的，这并不影响文件系统，当以后通过符号链接访问文件时，发现文件不存咋，直接删除符号链接；但对于硬链接则不可以直接删除，引用计数值减1，若值不为0，则不能删除文件，因为还有其他的硬链接指向此文件。当建立F2时，F1和F2的引用计数值都为1.当建立F3时，F1和F3的引用计数值都为2了。删除F1，F3的引用计数值为2-1=1，F2的引用计数值不变。

【河北大学】文件系统采用两级索引分配方式，如果每个磁盘块的大小为1KB，每个盘块号占4个字节，则在该系统中，文件的最大长度是（）。

- [x] 64MB
- [ ] 128MB
- [ ] 32MB
- [ ] 以上都不对

> 每个磁盘块大小1KB，每个盘块号占4个字节，则一个盘块可以存放1KB/4B=256个盘块，则2级索引文件的最大长度是256*256*1KB=64MB。

【2013年统考真题】为支持CD-ROM中视频文件的快速随机播放，播放性能最好的文件数据块组织方式是（）。

- [x] 连续结构
- [ ] 链式结构
- [ ] 直接索引结构
- [ ] 多级索引结构

> 视频文件属于有结构文件中的定长记录文件，适合用连续分配来组织，连续分配的优点主要有顺序访问容易，顺序访问速度快。为了实现快速随机播放，要保证最短时间查询，不宜选取链式和索引结构。

【2013年统考真题】若某文件系统索引节点（inode）中有直接地址项和间接地址项，则下列选项中，与单个文件长度无关的因素是（）。

- [x] 索引节点的总数
- [ ] 间接地址索引级数
- [ ] 地址项的个数
- [ ] 文件块大小

> 一个文件对应一个索引节点，索引节点的总数只能说明有多少个文件，跟单个文件的长度没有关系。而间接地址索引的级数、地址项的个数和文件块大小都跟单个文件长度相关。

【燕山大学，2006年】在磁盘上容易导致存储碎片的物理文件结构式（）。

- [ ] 链接
- [x] 连续
- [ ] 索引
- [ ] 索引和链接

> 连续文件的优点是在顺序存取时速度较快。存在如下缺点：1要求建立文件时就确定它的长度，2不便于文件的动态扩充。3可能出现外部碎片，就是在存储介质上存在很多空闲块，但不连续，无法被连续文件使用。

【南昌大学，2006年】采用直接存取法来读写磁盘上的物理记录时，效率最高的是（）。

- [x] 连续结构的文件
- [ ] 索引结构的文件
- [ ] 链接结构文件
- [ ] 其它结构文件

> 在直接存取法下，连续文件只要知道文件在存储设备上的起始地址和文件长度，就能很快的进行存取。适合随机存取的程度总结为：连续>索引>链接。

多选题

按用途分类，文件主要能分为（）

- [x] 系统文件
- [ ] 档案文件
- [x] 用户文件
- [x] 库文件

> 按用途 ： 系统文件、库文件、用户文件 按保护级别 ： 可执行文件、只读文件、读写文件 按信息流向： 输入文件、输出文件、输入输出文件 按存放时限： 临时文件、永久文件、档案文件 按设备类型： 磁盘文件、磁带文件、卡片文件、打印文件 按文件组织结构：逻辑文件、物理文件（顺序文件、链接文件、索引文件）

允许多个用户同时使用同一个共享文件时，下列（）做法是正确的。

- [x] 允许多个用户同时打开共享文件执行读操作
- [ ] 允许读者和写者同时使用共享文件
- [x] 不允许读者和写者同时使用共享文件
- [x] 不允许多个写者同时对共享文件执行写操作

> 进行文件读写要保持文件的正确性，所以不能进行同时读写，多个同时写等操作。

文件系统的功能有（）

- [x] 文件系统实现对文件的按名存取
- [x] 负责实现数据的逻辑结构到物理结构的转换
- [ ] 提高磁盘的读写速度
- [x] 提供对文件的存取方法和对文件的操作

> 文件系统的功能主要有：1、管理文件的存储介质 2、实现文件名到物理地址的映射 3、提供用户对文件和目录的操作命令 4、 提供用户共享文件机制 5、提供文件存取机制，保证文件安全性。

文件的物理结构可分为（）

- [x] 顺序结构
- [x] 链表结构
- [x] 索引结构
- [ ] 目录结构

> 文件的物理结构 ：由文件系统在存储介质上的文件构造方式称为文件的物理结构。不论用户看来是什么文件，在存储介质上存储时，按何种构造方式记录呢，因为介质上的存储单位是物理块，那么这些物理快是顺序存放，还是链式结构，或者索引结构，都要由文件系统结构来实现。

从对文件信息的存取次序考虑，存取方法可分为（）。

- [x] 顺序存取
- [x] 随机存取
- [ ] 索引存取
- [ ] 连续存取

> 文件的存取方式有顺序存取和随机存取两种。磁带上的文件只能顺序存取，磁盘上的文件既可采用顺序方式也可用随机方式存取。

---

lec21 文件系统　在线练习

选择题

关于文件系统功能的阐述正确的是（） s1

- [x] 负责数据持久保存
- [x] 文件分配
- [x] 文件管理
- [x] 数据可靠和安全

> 都对

打开文件时，文件系统要维护哪些信息（） s2

- [x] 文件指针
- [x] 打开文件计数
- [x] 文件访问权限
- [x] 文件位置和数据缓存

> 都是

关于目录和别名的阐述正确的是（）s3

- [x] 目录是一类特殊的文件
- [x] 目录的内容是文件索引表<文件名, 指向文件的指针>
- [x] 可通过硬链接机制实现文件别名
- [x] 可通过软链接机制实现文件别名

> 都对

虚拟文件系统可支持的具体文件系统包括（） s4

- [x] 磁盘文件系统
- [x] 设备文件系统
- [x] 网络文件系统
- [x] 系统状态文件系统（proc...）

> 都对

关于文件缓存和打开文件的阐述正确的是（） s5

- [x] 打开文件后，可通过把文件数据块按需读入内存来减少IO操作次数
- [x] 文件数据块使用后被缓存在内存中，可用于再次读写，从而减少IO操作次数
- [x] 在虚拟地址空间中虚拟页面可映射到本地外存文件中，这样访问文件就像访问内存一样
- [ ] 多个进程打开同一文件进行读写访问不需要用锁机制进行互斥保护

> 文件是共享资源，对于写操作需要互斥保护

关于文件分配的阐述正确的是（） s6

- [x] 连续分配会产生外碎片
- [ ] 链式分配会产生外碎片
- [ ] 索引分配会产生外碎片
- [x] 多级索引分配可支持大文件

> 链式分配和索引分配不会产生外碎片

关于冗余磁盘阵列(RAID, Redundant Array of Inexpensive Disks)的阐述正确的是（） s7

- [x] 采用RAID机制可提高磁盘IO的吞吐量(通过并行)
- [x] 采用RAID机制可提高磁盘IO的可靠性和可用性 (通过冗余)
- [ ] 采用RAID-0可提高磁盘IO的可靠性和可用性
- [ ] 采用RAID-1可提高磁盘IO的吞吐量

> RAID-0提高并行性，RAID-1提高可靠性

文件系统(lec 21) spoc 思考题

个人思考题

文件系统和文件

1. 文件系统的功能是什么？

> 负责数据持久保存，功能是数据存储和访问
>
> 具体功能：文件分配、文件管理、数据可靠和安全

1. 什么是文件？

> 文件系统中具有符号名的基本数据单位。

文件描述符

1. 打开文件时，文件系统要维护哪些信息？

> 文件指针、打开文件计数、访问权限、文件位置和数据缓存

1. 文件系统的基本数据访问单位是什么？这对文件系统有什么影响？
2. 文件的索引访问有什么特点？如何优化索引访问的速度？

目录、文件别名和文件系统种类

1. 什么是目录？

> 由文件索引项组成的特殊文件。

1. 目录的组织结构是什么样的？

> 树结构、有向图

1. 目录操作有哪些种类？
2. 什么是文件别名？软链接和硬链接有什么区别？
3. 路径遍历的流程是什么样的？如何优化路径遍历？
4. 什么是文件挂载？
5. 为什么会存在大量的文件类型？

虚拟文件系统

1. 虚拟文件系统的功能是什么？

> 对上对下的接口、高效访问实现

1. 文件卷控制块、文件控制块和目录项的相互关系是什么？
2. 可以把文件控制块放到目录项中吗？这样做有什么优缺点？

文件缓存和打开文件

1. 文件缓存和页缓存有什么区别和联系？
2. 为什么要同时维护进程的打开文件表和操作系统的打开文件表？这两个打开文件表有什么区别和联系？为什么没有线程的打开文件表？

文件分配

1. 文件分配的三种方式是如何组织文件数据块的？各有什么特征？
2. UFS多级索引分配是如何组织文件数据块的位置信息的？

空闲空间管理和冗余磁盘阵列RAID

1. 硬盘空闲空间组织和文件分配有什么异同？
2. RAID-0、1、4和5分别是如何组织磁盘数据块的？各有什么特征？

小组思考题

1. (spoc)完成Simple File System的功能，支持应用程序的一般文件操作。具体帮助和要求信息请看sfs-homework

1. (spoc)FAT、UFS、YAFFS、NTFS这几种文件系统中选一种，分析它的文件卷结构、目录结构、文件分配方式，以及它的变种。 wikipedia上的文件系统列表参考

    - http://en.wikipedia.org/wiki/List_of_file_systems
    - http://en.wikipedia.org/wiki/File_system
    - http://en.wikipedia.org/wiki/List_of_file_systems

    请同学们依据自己的选择，在下面链接处回复分析结果。

    - [https://piazza.com/class/i5j09fnsl7k5x0?cid=416 FAT文件系统分析]
    - [https://piazza.com/class/i5j09fnsl7k5x0?cid=417 NTFS文件系统分析]
    - [https://piazza.com/class/i5j09fnsl7k5x0?cid=418 UFS文件系统分析]
    - [https://piazza.com/class/i5j09fnsl7k5x0?cid=419 ZFS文件系统分析]
    - [https://piazza.com/class/i5j09fnsl7k5x0?cid=420 YAFFS文件系统分析]

lec22 lab8 文件系统　在线练习

选择题

ucore实现的文件系统抽象包括（） s1 总体介绍

- [x] 文件
- [x] 目录项
- [x] 索引节点
- [ ] 安装点

> 都是

ucore实现的simple FS（简称SFS）采用的文件分配机制是（） s2 ucore 文件系统架构

- [ ] 连续分配
- [ ] 链式分配
- [x] 索引分配
- [ ] 位图分配

> 索引分配

关于ucore实现的SFS阐述正确的是（） s3 Simple File System分析

- [x] SFS的超级块保存在硬盘上，在加载simple FS时会读入内存中
- [x] SFS的free map结构保存在硬盘上，表示硬盘可用的数据块（扇区）
- [x] SFS的root-dir inode结构保存在硬盘上，表示SFS的根目录的元数据信息
- [ ] 硬盘上的SFS，除保存上述三种结构外，剩下的都用于保存文件的数据内容

> 除了前三种结构，剩下的用于保存文件的inode, dir/file的data

关于ucore实现的Virtual FS（简称VFS）阐述正确的是() s4 Virtual File System分析

- [x] 已支持磁盘文件系统
- [x] 已支持设备文件系统
- [ ] 已支持网络文件系统
- [ ] 已支持系统状态文件系统

> 后两种可实现，但现在还没实现

关于ucore文件系统支持的I/O 设备包括() s5 I/O 设备接口分析

- [x] 串口设备
- [x] 并口设备
- [x] CGA设备
- [x] 键盘设备

> 都支持

lab8 文件系统 (lec 22) spoc 思考题

- 有"spoc"标记的题是要求拿清华学分的同学要在实体课上完成，并按时提交到学生对应的ucore_code和os_exercises的git repo上。

个人思考题

总体介绍

- 请简要描述ucore文件系统支持的文件系统抽象

ucore 文件系统架构

- 请简要阐述ucore 文件系统架构的四个组成部分
- 请简要说明进程proc_struct、文件file、inode之间的关系

Simple File System分析

- SFS在硬盘上的四大部分主要是什么，有何作用？
- 硬盘上的SFS是如何加载到ucore中并初始化的？
- 硬盘上的inode和内存中的inode的关系和区别是什么?

Virtual File System分析

- file数据结构的主要内容是什么？与进程的关系是什么？
- inode数据结构的主要内容是什么？与file的数据结构的关系是什么？
- inode_ops包含哪些与文件相关的操作？

I/O 设备接口分析

- device数据结构的主要内容是什么？与fs的关系是什么？与inode的关系是什么？

小组思考题

1. (spoc) 理解文件访问的执行过程，即在ucore运行过程中通过 `cprintf` 函数来完整地展现出来读一个文件在ucore中的整个执行过程，(越全面细致越好) 完成代码填写，并形成spoc练习报告，需写练习报告和简单编码，完成后放到git server对应的git repo中

2. （spoc） 在下面的实验代码的基础上，实现基于文件系统的pipe IPC机制

练习用的**lab8 spoc exercise project source code**

I/O子系统

单项选择题

在操作系统中，用户在使用I/O设备时，通常采用（）。

- [ ] 物理设备名
- [x] 逻辑设备名
- [ ] 虚拟设备名
- [ ] 设备号

> 用户程序提出使用设备申请时，使用系统规定的设备类型号和自己规定的设备相对号（即逻辑设备名）由操作系统进行地址转换，变成系统的设备绝对号（物理设备号）。

操作系统中采用缓冲技术的目的是为了增强系统（）的能力。

- [ ] 串行操作
- [ ] 控制操作
- [ ] 重执操作
- [x] 并行操作

> 为了提高CPU和设备之间的并行程度。

操作系统采用缓冲技术，能够减少对CPU的（）次数，从而提高资源的利用率。

- [x] 中断
- [ ] 访问
- [ ] 控制
- [ ] 依赖

> I/O中断：是指中央处理器和通道协调工作的一种手段。通道借助I/O中断请求CPU进行干预，CPU根据产生的I/O中断事件了解输入输出操作的执行情况，I/O中断事件是由于通道程序的执行或其他外界原因引起的，采用缓冲技术可以减少CPU中断。

CPU输出数据的速度远远高于打印机的打印速度，为了解决这一矛盾，可采用（）。

- [ ] 并行技术
- [ ] 通道技术
- [x] 缓冲技术
- [ ] 虚存技术

> 在设备I/O中引入缓存技术是为了改善CPU与设备I/O直接速度不匹配的矛盾。

缓冲技术用于（）。

- [x] 提高主机和设备交换信息的速度
- [ ] 提供主、辅存接口
- [ ] 提高设备利用率
- [ ] 扩充相对地址空间

> 在设备I/O中引入缓存技术是为了改善CPU与设备I/O直接速度不匹配的矛盾。

通道是一种（）。

- [ ] I/O端口
- [ ] 数据通道
- [x] I/O专用处理机
- [ ] 软件工具

> 通道：计算机系统中能够独立完成输入输出操作的硬件装置，也称为“输入输出处理机”，能接收中央处理机的命令，独立执行通道程序，协助中央处理机控制与管理外部设备。一个独立于CPU的专门I/O控制的处理机，控制设备与内存直接进行数据交换。

设备管理的主要程序之一是设备分配程序，当进程请求在内存和外设之间传送信息时，设备分配程序分配设备的过程通常是（）。

- [x] 先分配设备，再分配控制器，最后分配通道
- [ ] 先分配控制器，再分配设备，最后分配通道
- [ ] 先分配通道，再分配设备，最后分配控制器
- [ ] 先分配通道，再分配控制器，最后分配设备

> 主机对外部设备的控制可分为四个层次：主机、通道、控制器、设备。所以，设备分配过程是先设备、在分控制器、最后是通道。

下列描述中，不是设备管理的功能的是（）。

- [ ] 实现外围设备的分配与回收
- [ ] 缓冲管理与地址转换
- [x] 实现按名存取
- [ ] 实现I/O操作

> 设备管理的功能有：1、缓冲管理。为达到缓解CPU和I/O设备速度不匹配的矛盾，达到提高CPU和I/O设备利用率，提高系统吞吐量的目的，许多操作系统通过设置缓冲区的办法来实现。2、设备分配。设备分配的基本任务是根据用户的I/O请求，为他们分配所需的设备。如果在I/O设备和CPU之间还存在设备控制器和通道，则还需为分配出去的设备分配相应的控制器和通道。3、设备处理。设备处理程序又称设备驱动程序。其基本任务是实现CPU和设备控制器之间的通信。4、设备独立性和虚拟设备。用户向系统申请和使用的设备与实际操作的设备无关。 按名存取是文件管理的功能。

用户编制的程序与实际使用的物理设备无关是由（）功能实现的。

- [ ] 设备分配
- [ ] 设备驱动
- [ ] 虚拟设备
- [x] 设备独立性

> 设备独立性是设备管理要达到的目的之一，就是，用户程序应与实际使用的物理设备无关，由操作系统来考虑因实际设备不同使用不同的驱动等问题。采用“设备类、相对号”方式使用设备时，用户编程就不必指定特定设备，在程序中由“设备类、相对号”定义逻辑设备。程序执行时由系统根据用户指定的逻辑设备转换成与其对应的具体物理设备。所以，用户编程时使用的设备与实际使用哪台设备无关，这就是“设备独立性”

SPOOLing技术利用于（）。

- [ ] 外设概念
- [x] 虚拟设备概念
- [ ] 磁带概念
- [ ] 存储概念

> SPOOLing技术有3个特点：(1)提高了I/O速度。从对低速I/O设备进行的I/O操作变为对输入井或输出井的操作,如同脱机操作一样,提高了I/O速度,缓和了CPU与低速I/O设备速度不匹配的矛盾。(2)设备并没有分配给任何进程。在输入井或输出井中,分配给进程的是一存储区和建立一张I/O请求表。(3)实现了虚拟设备功能。多个进程同时使用一个独享设备,而对每一进程而言,都认为自己独占这一设备,从而实现了设备的虚拟分配。不过,该设备是逻辑上的设备。

【2010年计算机联考真题】本地用户通过键盘登陆系统时，首先获得键盘输入信息的程序是（）。

- [ ] 命令解释程序
- [x] 中断处理程序
- [ ] 系统调用服务程序
- [ ] 用户登陆程序

> 键盘是典型的通过中断I/O方式工作的外设，当用户输入信息时，计算机响应中断并通过中断处理程序获得输入信息。

【2011年计算机联考真题】操作系统的I/O子系统通常有4个层次组成，每一层明确定义了与邻近层次的接口，其合理的层次组织排列顺序是（）。

- [x] 用户级I/O软件、设备无关软件、设备驱动程序、中断处理程序
- [ ] 用户级I/O软件、设备无关软件、中断处理程序、设备驱动程序
- [ ] 用户级I/O软件、设备驱动程序、设备无关软件、中断处理程序
- [ ] 用户级I/O软件、中断处理程序、设备无关软件、设备驱动程序

> 设备管理软件一般分为四个层次：用户层、与设备无关的软件层、设备驱动程序以及中断处理程序。

【2013年计算机联考真题】用户程序发出磁盘I/O请求后，系统的处理流程是：用户程序-系统调用处理程序-设备驱动程序-中断处理程序。其中，计算数据所在磁盘的柱面号、磁头号、扇区号的程序是（）。

- [ ] 用户程序
- [ ] 系统调用处理程序
- [x] 设备驱动程序
- [ ] 中断处理程序

> 计算数据所在磁盘的柱面号、磁头号、扇区号的工作是由设备驱动程序完成的。题中的功能因设备硬件的不用而不同，因此应有厂家提供的设备驱动程序完成。

【2011年计算机统考真题】某文件占用10个磁盘块，现在要把该文件磁盘块逐个读入主缓冲区，并送用户区进行分析。假设一个缓冲区与一个磁盘块大小相同，把一个磁盘块读入缓冲区的时间为100μs，将缓冲区的数据传送到用户区的时间是50μs，CPU对一块数据进行分析的时间为50μs。在单缓冲区和双缓冲区结构下，读入并分析完该文件的时间分别是（）。

- [ ] 1500μs，1000μs
- [x] 1550μs，1100μs
- [ ] 1550μs，1550μs
- [ ] 2000μs，2000μs

> 单缓冲区下，当上一个磁盘块从缓冲区读入用户区完成时下一磁盘块才能开始读入，所以当最后一块磁盘块读入用户区完毕时，所用时间为１５０×１０＝１５００，加上处理最后一个磁盘块的cpu处理时间５０为１５５０.双缓冲区下，读入第一个缓冲区之后可以立刻开始读入第二个缓冲区，读完第二个缓冲区之后，第一个缓冲区的数据已经传送到用户区，因此不存在等待磁盘块从缓冲区读入用户区的问题，也就是１００×１０＝１１００，再加上最后一个缓冲区的数据传输到用户区并有CPU处理的时间50+50=100，总的时间是1000+100=1100。

【2005年，北京理工大学】（）是操作系统中采用的以空间换取时间的技术。

- [x] Spooling技术
- [ ] 虚拟存储技术
- [ ] 覆盖与交换技术

- [ ] 通道技术

> SPOOLing技术是操作系统中采用的以空间换取时间的技术；虚拟存储技术和覆盖交换技术是为了扩充内存容量，是以时间换空间技术；通道技术是为了提高设备速度, 增加了硬件，不属于这两者的任何一个。

【2012年统考真题】下列选项中，不能改善磁盘设备I/O性能的是（）。

- [ ] 重排I/O请求次序
- [x] 在一个磁盘上设置多个分区
- [ ] 预读和滞后写
- [ ] 优化文件物理的分布

> 重排I/O请求次序就是将磁盘请求访问序列进行重新排序，就是有关磁盘访问调度策略的选择对I/O有性能的影响，不同的调度策略有不同的寻道时间;磁盘分区实质上就是对磁盘的一种格式化。但磁盘设备I/O性能是由调用顺序和磁盘本身性质决定的，和分区的多少无太大关系，如果设置过多分区，还会导致一个I/O需要启动多个分区，反而会减低效率;预读和滞后写是常见的提升磁盘设备I/O速度的方法，具有重复性及阵发性的I/O进程和改善磁盘I/O性能很有帮助;优化文件物理块的分布可以减少寻找时间与延迟时间，从而提高磁盘性能。

【2009年全国统考】假设磁头当前位于第105道，正在向磁道序号增加的方向移动。现有一个磁道访问请求序列为35，45，12，68，110，180，170，195，采用SCAN调度（电梯调度）算法得到的磁道访问序列是()

- [x] 110，170，180，195，68，45，35，12
- [ ] 110，68，45，35，12，170，180，195
- [ ] 110，170，180，195，12，35，45，68
- [ ] 12，35，45，68，110，170，180，195

> SCAN调度算法就是电梯调度算法，顾名思义就是如果开始时磁头往外就一直要到最外面，然后再返回向里（磁头编号一般是最外面为0号往里增加），就像电梯若往下则一直要下到最底层才会再上升一样。

【西安电子科技大学，2000年】在关于SPOOLing的叙述中，（）描述不正确。

- [ ] SPOOLing系统必须使用独占设备
- [ ] SPOOLing系统加快了作业执行的速度
- [ ] SPOOLing系统使独占设备变成共享设备
- [x] SPOOLing系统利用了处理器与通道并行工作的能力

> SPOOLing是操作系统中采用的一种将独占设备改造为共享设备的技术，它有效减少了进程等待读入读出信息的时间，加快了作业执行的速度。不过，无论有没有通道，SPOOL系统都可以运行，因此4选项不对。

【河北大学】系统设备是通过一些数据结构来进行的，下面的（）不属于设备管理数据结构。

- [x] FCB
- [ ] DCT
- [ ] SDT
- [ ] COCT

> FCB是文件控制块，与设备管理无关。DCT是设备控制表，SDT是系统设备表，COCT控制器控制表，这3者都死设备管理中的重要数据结构。

【电子科技大学】 不属于DMA控制器的是（）

- [ ] 命令/状态寄存器
- [ ] 内存寄存器
- [ ] 数据寄存器
- [x] 堆栈指针寄存器

> DMA控制器与CPU的接口有3类信号线：数据线、地址线和控制线。通常与：数据寄存器、命令/状态寄存器两类寄存器相连。为了将数据送到内存，DMA控制器还需要内存地址寄存器。

lec23 IO设备　在线练习

选择题

字符设备包括（） s1

- [x] 键盘
- [x] 鼠标
- [x] 并口
- [x] 串口

> 都是

块设备包括（） s1

- [x] 硬盘
- [x] 软盘
- [x] 光盘
- [x] U盘

> 都是

网络设备包括（） s1

- [x] 以太网卡
- [x] wifi网卡
- [x] 蓝牙设备
- [ ] 网盘设备

> 网盘在模拟实现上应该算块设备

关于CPU与设备的通信方式包括（） s2

- [x] 轮询
- [x] 设备中断
- [x] DMA
- [ ] PIPE

> PIPE是用于进程间通信

关于IO数据传输的阐述正确的是（） s3

- [x] 程序控制I/O(PIO, Programmed I/O)通过CPU的in/out或者load/store传输所有数据
- [x] DMA设备控制器可直接访问系统总线并直接与内存互相传输数据
- [ ] DMA机制适合字符设备
- [ ] PIO机制适合块设备

> DMA机制适合块设备，PIO机制适合简单，低速的字符设备等

常用移臂调度算法包括（） s4 磁盘调度

- [x] 先来先服务（FIFO）算法
- [x] 最短寻道时间优先（SSTF）算法
- [x] 电梯调度（SCAN）算法
- [x] 单向扫描（C-SCAN）算法

> 都对

在设备管理子系统中，引入缓冲区的目的主要有() s5 缓冲区

- [x] 缓和CPU与I/O设备间速度不匹配的矛盾
- [x] 减少对CPU的中断频率，放宽对CPU中断响应时间的限制
- [x] 解决基本数据单元大小（即数据粒度）不匹配的问题
- [x] 提高CPU和I/O设备之间的并行性

> 都对

IO设备(lec 23) spoc 思考题

个人思考题

IO特点

- 字符设备的特点是什么？
- 块设备的特点是什么？
- 网络设备的特点是什么？

IO结构

- IO访问的手段有哪些？
- 请描述IO请求到完成的整个执行过程
- CPU与device通信的手段有哪些？

> 显式的IO指令，如x86的in, out； 或者是memory读写方式，即把device的寄存器，内存等映射到物理内存中

IO数据传输

- IO数据传输有哪几种？
- 轮询方式的特点是什么？
- 中断方式的特点是什么？
- DMA方式的特点是什么？

磁盘调度

- 请简要阐述磁盘的工作过程
- 请用一表达式（包括寻道时间，旋转延迟，传输时间）描述磁盘I/O传输时间
- 请说明磁盘调度算法的评价指标
- FIFO磁盘调度算法的特点是什么?
- 最短服务时间优先(SSTF)磁盘调度算法的特点是什么?
- 扫描(SCAN)磁盘调度算法的特点是什么?
- 循环扫描(C-SCAN)磁盘调度算法的特点是什么?
- C-LOOK磁盘调度算法的特点是什么?
- N步扫描(N-step-SCAN)磁盘调度算法的特点是什么?
- 双队列扫描(FSCAN)磁盘调度算法的特点是什么?

磁盘缓存

- 磁盘缓存的作用是什么？
- 请描述单缓存(Single Buffer Cache)的工作原理
- 请描述双缓存(Double Buffer Cache)的工作原理
- 请描述访问频率置换算法(Frequency-based Replacement)的基本原理

小组思考题

- (spoc)完成磁盘访问与磁盘寻道算法的作业，具体帮助和要求信息请看disksim指导信息和disksim参考代码