1
下列四个操作系统中，是分时系统的为 。

- ( ) A.CP / M
- ( ) B.MS - DOS
- (x) C.UNIX
- ( ) D.WINDOWS NT

知识点:操作系统概述
出处:网络
难度:1

> C
> 1
> 用户要在程序一级获得系统帮助，必须通过 。

- ( ) A.进程调度
- ( ) B.作业调度
- ( ) C.键盘命令
- (x) D.系统调用

知识点:中断、异常与系统调用
出处:网络
难度:1

> D
> 1
> 批处理系统的主要缺点是 。

- ( ) A.CPU的利用率不高
- (x) B.失去了交互性
- ( ) C.不具备并行性
- ( ) D.以上都不是

知识点:操作系统概述
出处:网络
难度:1

> B
> 1
> DOS操作系统主要的功能是 。

- (x) A.文件管理程序
- ( ) B.中断处理程序
- ( ) C.作业管理程序
- ( ) D.打印管理程序

知识点:操作系统概述
出处:网络
难度:1

> A
> 1
> 在Pascal程序中调用的sin（x）是 。

- ( ) A.系统调用
- (x) B.标准子程序
- ( ) C.操作系统命令
- ( ) D.进程

知识点:中断、异常与系统调用
出处:网络
难度:1

> B
> 1
> 计算机操作系统的功能是 。

- ( ) A.把源程序代码转换为目标代码
- ( ) B.实现计算机用户之间的相互交流
- ( ) C.完成计算机硬件与软件之间的转换
- (x) D.控制、管理计算机系统的资源和程序的执行

知识点:操作系统概述
出处:网络
难度:1

> D
> 1
> 在分时系统中，时间片一定时，，响应时间越长。

- ( ) A.内存越多
- (x) B.用户数越多
- ( ) C.内存越少
- ( ) D.用户数越少

知识点:操作系统概述
出处:网络
难度:1

> B
> 4
> "1）试说明硬中断（hardware interrupt）、异常（exception）和系统调用（system call）的相同点和不同点。
> 2）下面代码完成在进入trap()函数前的准备工作。其中pushal完成包括esp在内的CPU寄存器压栈。试说明“pushl %esp”的作用是什么?
> ==========trapentry.S (kern\trap)========== #include # vectors.S sends all traps here. .text .globl **alltraps** alltraps: # push registers to build a trap frame # therefore make the stack look like a struct trapframe pushl %ds pushl %es pushl %fs pushl %gs pushal # load GD_KDATA into %ds and %es to set up data segments for kernel movl Missing open brace for subscript 0x8, %esp iret ==========Trap.c (kern\trap)========== ..... / trap - handles or dispatches an exception/interrupt. if and when trap() returns, the code in kern/trap/trapentry.S restores the old CPU state saved in the trapframe and then uses the iret instruction to return from the exception. / void trap(struct trapframe tf) { // dispatch based on what type of trap occurred trap_dispatch(tf); } ....."

- [x]

知识点:中断、异常与系统调用
出处:网络
难度:1

> Hardware interrupt Interruption based on an external hardware event external to the CPU An interrupt is generally initiated by an I/O device, and causes the CPU to stop what it's doing Exception an exceptional condition in the processor (Illed program) an interrupt that is caused by software (by executing an instruction) System call a programmer initiated in user mode and expected transfer of control to the kernel an interrupt that is caused by software (by executing an instruction) 共同：中断当前执行/保存现场（3分） 不同：产生原因(每个2分) 2)3分 给trap函数传参数，汇编调用C时如何传参。 # push %esp to pass a pointer to the trapframe as an argument to trap() pushl %esp # call trap(tf), where tf=%esp call trap
> 4
> "1）系统调用的参数传递有几种方式？各有什么特点?
> 2）sys_exec是一个加载和执行指定可执行文件的系统调用。请说明在下面的ucore实现中，它的三个参数分别是以什么方式传递的。
> ==========Proc.c (kern\process)========== ..... // do_execve - call exit_mmap(mm)&pug;_pgdir(mm) to reclaim memory space of current process //

- call load_icode to setup new memory space accroding binary prog. int do_execve(const char name, int argc, const char argv) { static_assert(EXEC_MAX_ARG_LEN >= FS_MAX_FPATH_LEN); struct mm_struct mm = current->mm; if (!(argc >= 1 && argc <= EXEC_MAX_ARG_NUM)) { return -E_INVAL; } char local_name[PROC_NAME_LEN + 1]; memset(local_name, 0, sizeof(local_name)); char kargv[EXEC_MAX_ARG_NUM]; const char path; int ret = -E_INVAL; lock_mm(mm); if (name == NULL) { snprintf(local_name, sizeof(local_name), " %d", current->pid); } else { if (!copy_string(mm, local_name, name, sizeof(local_name))) { unlock_mm(mm); return ret; } } if ((ret = copy_kargv(mm, argc, kargv, argv)) != 0) { unlock_mm(mm); return ret; } path = argv[0]; unlock_mm(mm); files_closeall(current->filesp); / sysfile_open will check the first argument path, thus we have to use a user-space pointer, and argv[0] may be incorrect / int fd; if ((ret = fd = sysfile_open(path, O_RDONLY)) < 0) { goto execve_exit; } if (mm != NULL) { lcr3(boot_cr3); if (mm_count_dec(mm) == 0) { exit_mmap(mm); put_pgdir(mm); mm_destroy(mm); } current->mm = NULL; } ret= -E_NO_MEM;; if ((ret = load_icode(fd, argc, kargv)) != 0) { goto execve_exit; } put_kargv(argc, kargv); set_proc_name(current, local_name); return 0; execve_exit: put_kargv(argc, kargv); do_exit(ret); panic("already exit: %e. ", ret); } ..... ==========Syscall.c (kern\syscall)========== ..... static int sys_exec(uint32_t arg[]) { const char name = (const char)arg[0]; int argc = (int)arg[1]; const char argv = (const char)arg[2]; return do_execve(name, argc, argv); } ..... static int (syscalls[])(uint32_t arg[]) = { [SYS_exit] sys_exit, [SYS_fork] sys_fork, [SYS_wait] sys_wait, [SYS_exec] sys_exec, [SYS_yield] sys_yield, [SYS_kill] sys_kill, [SYS_getpid] sys_getpid, [SYS_putc] sys_putc, [SYS_pgdir] sys_pgdir, }; #define NUM_SYSCALLS ((sizeof(syscalls)) / (sizeof(syscalls[0]))) void syscall(void) { struct trapframe tf = current->tf; uint32_t arg[5]; int num = tf->tf_regs.reg_eax; if (num >= 0 && num < NUM_SYSCALLS) { if (syscalls[num] != NULL) { arg[0] = tf->tf_regs.reg_edx; arg[1] = tf->tf_regs.reg_ecx; arg[2] = tf->tf_regs.reg_ebx; arg[3] = tf->tf_regs.reg_edi; arg[4] = tf->tf_regs.reg_esi; tf->tf_regs.reg_eax = syscallsnum; return ; } } print_trapframe(tf); panic("undefined syscall %d, pid = %d, name = %s. ", num, current->pid, current->name); } ..... ==========libs-user-ucore/syscall.c========== ..... int sys_exec(const char filename, const char argv, const char envp) { return syscall(SYS_exec, filename, argv, envp); } ..... ==========libs-user-ucore/arch/i386/syscall.c========== ..... uint32_t syscall(int num, ...) { va_list ap; va_start(ap, num); uint32_t a[MAX_ARGS]; int i; for (i = 0; i < MAX_ARGS; i++) { a[i] =

va_arg(ap, uint32_t); } va_end(ap); uint32_t ret; asm volatile ("int %1;":"=a"
(ret) :"i"(T_SYSCALL), "a"(num), "d"(a[0]), "c"(a[1]), "b"(a[2]), "D"(a[3]),
"S"(a[4]) :"cc", "memory"); return ret; }"

- [x]

知识点:中断、异常与系统调用
出处:网络
难度:1

> 1）Three general methods used to pass parameters to the OS Simplest: pass the parameters in registers. （2分） In some cases, may be more parameters than registers（2分） Parameters stored in a block, or table, in memory（2分）, and address of block passed as a parameter in a register（1分） Parameters placed, or pushed, onto the stack（2分） by the program and popped off the stack by the operating system (ucore methord) Block and stack methods do not limit the number or length（1分） of parameters being passed 2）三个参数都是通过堆栈来传递的。（3分）
> 从用户态到内核态时参数是在寄存器中的； name可理解为是在内存块中来传递的； 综合而言，言之有理即可。
> 4
> "1）描述伙伴系统（Buddy System）中对物理内存的分配和回收过程。2）假定一个操作系统内核中由伙伴系统管理的物理内存有1MB，试描述按下面顺序进行
> 物理内存分配和回收过程中，每次分配完成后的分配区域的首地址和大小，或每次回收完成后的空闲区域队列（要求说明，每个空闲块的首地址和大小）。建议给出分配和回收的
> 中间过程。 a) 进程A申请50KB； b) 进程B申请100KB； c) 进程C申请40KB； d) 进程D申请70KB； e) 进程B释放100KB；
> f) 进程E申请127KB； g) 进程D释放70KB； h) 进程A释放50KB； i) 进程E释放127KB； j) 进程C释放40KB；"

- [x]

知识点:连续内存分配
出处:网络
难度:1

> 整个空间被分成2U大小；（2分） 分配：找到2U大小的块， 满足2U-1 < s <= 2U（2分） 如果比它大，就划分成两个等大小的块（2分） 释放：相邻且大小相同2U-1的两块中第一块起始地址为2U倍数（2分）时，合并（2分）； a) 进程A申请50KB；
> Addr:0,Size:64KB b) 进程B申请100KB；Addr:128K,Size:128KB c) 进程C申请40KB；Addr:64K,Size:64KB d) 进程D申请70KB；
> Addr:256K,Size:128KB e) 进程B释放100KB；Addr:128K,Size:128KB f) 进程E申请127KB；Addr:128K,Size:128KB g) 进程D释放70KB；Addr:256K,Size:512KB h) 进程A释放50KB；Addr:0,Size:64KB i) 进程E释放127KB；Addr:128K,Size:128KB j) 进程C释放40KB；Addr:0,Size:1024KB
> 4
> "1）试用图示描述32位X86系统在采用4KB页面大小时的虚拟地址结构和地址转换过程。2）在采用4KB页面大小的32位X86的ucore虚拟存储系统中，进程
> 页面的起始地址由宏VPT确定。 #define VPT 0x0D000000
> 请计算：2a)试给出页目录中自映射页表项的虚拟地址；2b)虚拟地址0X87654321对应的页目录项和页表项的虚拟地址。"

- [x]

知识点:非连续内存分配
出处:网络
难度:1

> 1）（12分）地址划分：10 + 10 + 12（6分）
> 地址转换过程关键点：两级页面（2分）、缺页处理（2分）（分配物理页面、更新页表项、重新访问）（有一个就给2分） 2a）（4分） 自映射页表项地址4分
> 每个地址3分，每个地址中的三段，二进制每段1分；（结果对了，就给全分） 0D00 0000 0000 1101 0000 0000 0000 0000 0000 0000 0000 1101 0000 0011 0100 0000 1101 0000 0X0D0340D0 2b）
> 虚拟地址0X87654321对应的页目录项和页表项的虚拟地址（4分,每个2分，二进制对，就给全分） 87654321 1000 0111 0110 0101 0100 0011 0010 0001 PDE: 0000 1101 0000 0011 0100 1000 0111 01 00 0X0D034874
> PTE: 0000 1101 00 10 00 01 11 01 10 01 01 01 00 00 0X0D21 D950
> 4
> "试描述FIFO页面替换算法的基本原理，并swap_fifo.c中未完成FIFA页面替换算法实验函数map_swappable()和swap_out_vic
> tim() 。 ==========Defs.h (libs)========== / to_struct - get the
> struct from a ptr @ptr: a struct pointer of member @type: the type of the
> struct this is embedded in @member: the name of the member within the struct
> / #define to_struct(ptr, type, member) \ ((type)((char)(ptr) -
> offsetof(type, member))) ==========Memlayout.h (kern\mm)========== //
> convert list entry to page #define le2page(le, member) \ to_struct((le),
> struct Page, member) ==========List.h (libs)========== #ifndef
> **LIBS_LIST_H** #define **LIBS_LIST_H** #ifndef **ASSEMBLER** #include /
> Simple doubly linked list implementation. Some of the internal functions
> ("**xxx") are useful when manipulating whole lists rather than single
> entries, as sometimes we already know the next/prev entries and we can
> generate better code by using them directly rather than using the generic
> single-entry routines. / struct list_entry { struct list_entry prev,
> next; }; typedef struct list_entry list_entry_t; static inline void
> list_init(list_entry_t elm)** attribute**((always_inline)); static inline
> void list_add(list_entry_t listelm, list_entry_t elm)**
> attribute**((always_inline)); static inline void
> list_add_before(list_entry_t listelm, list_entry_t elm)**
> attribute**((always_inline)); static inline void list_add_after(list_entry_t
> listelm, list_entry_t elm)** attribute**((always_inline)); static inline
> void list_del(list_entry_t listelm)** attribute**((always_inline)); static
> inline void list_del_init(list_entry_t listelm)**
> attribute**((always_inline)); static inline bool list_empty(list_entry_t**

list) attribute**((always_inline)); static inline list_entry_t list_next(list_entry_t listelm)** attribute**((always_inline)); static inline list_entry_t list_prev(list_entry_t listelm)** attribute**((always_inline)); static inline void** list_add(list_entry_t elm, list_entry_t prev, list_entry_t next) **attribute**((always_inline)); static inline void **list_del(list_entry_t prev, list_entry_t next)** attribute__((always_inline)); / list_init - initialize a new entry @elm: new entry to be initialized / static inline void list_init(list_entry_t elm) { elm->prev = elm->next = elm; } / list_add

- add a new entry @listelm: list head to add after @elm: new entry to be added Insert the new element @elm after the element @listelm which is already in the list. / static inline void list_add(list_entry_t listelm, list_entry_t elm) { list_add_after(listelm, elm); } / list_add_before - add a new entry @listelm: list head to add before @elm: new entry to be added Insert the new element @elm before the element @listelm which is already in the list. / static inline void list_add_before(list_entry_t listelm, list_entry_t elm) { **list_add(elm, listelm->prev, listelm); } / list_add_after - add a new entry @listelm: list head to add after @elm: new entry to be added Insert the new element @elm after the element @listelm which is already in the list. / static inline void list_add_after(list_entry_t listelm, list_entry_t elm) {** list_add(elm, listelm, listelm->next); } / list_del - deletes entry from list @listelm: the element to delete from the list Note: list_empty() on @listelm does not return true after this, the entry is in an undefined state. / static inline void list_del(list_entry_t listelm) { **list_del(listelm->prev, listelm->next); } / list_del_init - deletes entry from list and reinitialize it. @listelm: the element to delete from the list. Note: list_empty() on @listelm returns true after this. / static inline void list_del_init(list_entry_t listelm) { list_del(listelm); list_init(listelm); } / list_empty - tests whether a list is empty @list: the list to test. / static inline bool list_empty(list_entry_t list) { return list->next == list; } / list_next - get the next entry @listelm: the list head / static inline list_entry_t list_next(list_entry_t listelm) { return listelm->next; } / list_prev - get the previous entry @listelm: the list head / static inline list_entry_t list_prev(list_entry_t listelm) { return listelm->prev; } / Insert a new entry between two known consecutive entries. This is only for internal list manipulation where we know the prev/next entries already! / static inline void** list_add(list_entry_t elm, list_entry_t prev, list_entry_t next) { prev->next = next->prev = elm; elm->next = next; elm->prev = prev; } / Delete a list entry by making the prev/next entries point to each other. This is only for internal list manipulation where we know the prev/next entries already! / static inline void **list_del(list_entry_t prev, list_entry_t next) { prev->next = next; next->prev = prev; } #endif /** !ASSEMBLER_ */ #endif / !***LIBS_LIST_H*** / ============= Swap_fifo.c (kern\mm)============= #include #include #include #include #include #include #include / [wikipedia]The simplest Page Replacement Algorithm(PRA) is a FIFO algorithm. (1) Prepare: In order to implement FIFO PRA, we should manage all swappable pages, so we can link these pages into pra_list_head according the time order. At first you should be familiar to the struct list in list.h. struct list is a simple doubly linked list implementation. You should know howto USE: list_init, list_add(list_add_after), list_add_before, list_del, list_next, list_prev. Another tricky method is to transform a general list struct to a special struct (such as struct page). You can find some MACRO: le2page (in memlayout.h), (in future labs: le2vma (in vmm.h), le2proc (in proc.h),etc. / list_entry_t pra_list_head; / (2) _fifo_init_mm: init pra_list_head and let mm->sm_priv point to the addr of pra_list_head. Now, From the memory control struct mm_struct, we can access FIFO PRA / static int _fifo_init_mm(struct mm_struct mm) { list_init(&pra;list_head); mm->sm_priv = &pra;list_head; //cprintf(" mm->sm_priv %x in fifo_init_mm ",mm->sm_priv); return 0; } / (3)**fifo_map_swappable: According FIFO PRA, we should link the most recent arrival page at the back of pra_list_head qeueue / static int *fifo_map_swappable(struct mm_struct mm, uintptr_t addr, struct Page page, int swap_in) { list_entry_t head=(list_entry_t) mm->sm_priv; list_entry_t entry=&(page->pra_page_link); assert(entry != NULL && head != NULL); //record the page access situlation /LAB3 EXERCISE 2: YOUR CODE/ //(1)link the most recent arrival page at the back of the pra_list_head qeueue. ===Your code 2=== return 0; } / (4)**fifo_swap_out_victim: According FIFO PRA, we should unlink the earliest arrival page in front of pra_list_head qeueue, then set the addr of addr of this page to ptr_page. / static int *fifo_swap_out_victim(struct mm_struct mm, struct Page ptr_page, int in_tick) { list_entry_t head=(list_entry_t) mm->sm_priv; assert(head != NULL); assert(in_tick==0); / Select the victim / /LAB3 EXERCISE 2: YOUR CODE/ //(1) unlink the earliest arrival page in front of pra_list_head qeueue //(2) set the addr of

> *addr of this page to ptr_page / Select the tail / ===Your code 3=== return 0; } static int _fifo_check_swap(void) { cprintf("write Virt Page c in fifo_check_swap ");* (unsigned char)0x3000 = 0x0c; assert(pgfault_num==4); *cprintf("write Virt Page a in fifo_check_swap ");* (unsigned char)0x1000 = 0x0a; assert(pgfault_num==4); *cprintf("write Virt Page d in fifo_check_swap ");* (unsigned char)0x4000 = 0x0d; assert(pgfault_num==4); *cprintf("write Virt Page b in fifo_check_swap ");* (unsigned char)0x2000 = 0x0b; assert(pgfault_num==4); *cprintf("write Virt Page e in fifo_check_swap ");* (unsigned char)0x5000 = 0x0e; assert(pgfault_num==5); *cprintf("write Virt Page b in fifo_check_swap ");* (unsigned char)0x2000 = 0x0b; assert(pgfault_num==5); *cprintf("write Virt Page a in fifo_check_swap ");* (unsigned char)0x1000 = 0x0a; assert(pgfault_num==6); *cprintf("write Virt Page b in fifo_check_swap ");* (unsigned char)0x2000 = 0x0b; assert(pgfault_num==7); *cprintf("write Virt Page c in fifo_check_swap ");* (unsigned char)0x3000 = 0x0c; assert(pgfault_num==8); *cprintf("write Virt Page d in fifo_check_swap ");* (unsigned char)0x4000 = 0x0d; assert(pgfault_num==9); return 0; } static int _fifo_init(void) { return 0; } static int _fifo_set_unswappable(struct mm_struct mm, uintptr_t addr) { return 0; } static int _fifo_tick_event(struct mm_struct mm) { return 0; } struct swap_manager swap_manager_fifo = { .name = "fifo swap manager", .init = &fifo_init, .init_mm = &*fifo_init_mm, .tick_event* = &fifo_tick_event, .map_swappable = &*fifo_map_swappable, .set_unswappable* = &fifo_set_unswappable, .swap_out_victim = &*fifo_swap_out_victim, .check_swap* = &fifo_check_swap, }; "

- [x]

知识点:置换算法
出处:网络
难度:1

> 算法：（4分）占用页面按置换时间先后排序；缺页时置换最先进入内存的页面； 实现： map_swappable() //record the page access situation /LAB3 EXERCISE 2: YOUR CODE/ //(1)link the most recent arrival page at the back of the pra_list_head qeueue. list_add(head, entry);// （3分） swap_out_victim() / Select the victim / /LAB3 EXERCISE 2: YOUR CODE/ //(1) unlink the earliest arrival page in front of pra_list_head qeueue //(2) set the addr of addr of this page to ptr_page / Select the tail / list_entry_t le = head->prev; // 找到链表尾（2分） assert(head!=le); struct Page p = le2page(le, pra_page_link); //找到物理页面数据结构，并保存 （2分） list_del(le); // 从链表中取出页面（2分） assert(p !=NULL); ptr_page = p; //返回被置换的物理页面数据结构指针 （2分） return 0;
> 4
> "描述int fork(void)系统调用的功能和接口，给出程序fork.c的输出结果，并用图示给出所有进程的父子关系。注：1）getpid()和 getpp id()是两个系统调用，分别返回本进程标识和父进程标识。2）你可以假定每次新进程创建时生成的进程标识是顺序加1得到的；在进程标识为1000的命令解释程序sh ell中启动该程序的执行。 #include #include / getpid() and fork() are system calls declared in unistd.h. They return / / values of type pid_t. This pid_t is a special type for process ids. / / It's equivalent to int. / int main(void) { pid_t childpid; int x = 5; int i; childpid = fork(); for (i = 0; i < 3; i++) { printf("This is process %d; childpid = %d; The parent of this process has id %d; i = %d; x = %d ", getpid(), childpid, getppid(), i, x); sleep(1); x++; } return 0; }"

- [x]

知识点:进程状态与控制
出处:网络
难度:1

> 功能：复制当前进程，生成一个子进程（2分），并从当前位置继续执行（2分）； 接口：没有输入，父进程返回子进程标识（2分）； 子进程返回零（2分）； 输出：
> 三次循环（3分）；i的值输出正确（2分）；x的值输出正确（2分）；父子进程标识正确（2分）； xyong@ubuntu:~/work$ ./a.out This is process 13724; childpid = 13725; The parent of this process has id 9917; i = 0; x = 5 This is process 13725; childpid = 0; The parent of this process has id 13724; i = 0; x = 5 This is process 13724; childpid = 13725; The parent of this process has id 9917; i = 1; x = 6 This is process 13725; childpid = 0; The parent of this process has id 13724; i = 1; x = 6 This is process 13724; childpid = 13725; The parent of this process has id 9917; i = 2; x = 7 This is process 13725; childpid = 0; The parent of this process has

> id 13724; i = 2; x = 7 父子关系图：1分
> 1
> 操作系统是（）。

- ( ) A.硬件
- (x) B.系统软件
- ( ) C.应用软件
- ( ) D.虚拟机

知识点:操作系统概述
出处:网络
难度:1

> B
> 1
> 下面关于SPOOL的叙述错误的是()

- ( ) A.SPOOL又称“斯普林”，是Simultaneous Peripheral Operation On Line的缩写
- (x) B.SPOOL处理方式只是方便操作员，不能直接提高系统效率
- ( ) C.SPOOL是把磁盘作为巨大缓冲器的技术
- ( ) D.SPOOL处理方式不仅方便操作员，而且还提高系统效率

知识点:操作系统概述
出处:网络
难度:1

> B
> 1
> 对于下列文件的物理结构，()只能采用顺序存取方式

- ( ) A.顺序文件
- (x) B.链接文件
- ( ) C.索引文件
- ( ) D.Hash文件

知识点:连续内存分配
出处:网络
难度:1

> B
> 1
> 设备分配问题中，算法实现时，同样要考虑安全性问题，防止在多个进程进行设备请求时，因相互等待对方释放所占设备所造成的()现象

- ( ) A.瓶颈
- ( ) B.碎片
- ( ) C.系统抖动
- (x) D.死锁

知识点:死锁
出处:网络
难度:1

> D
> 1
> 下面有关可变分区管理中采用的主存分配算法说法错误的是（）

- ( ) A.可变分区管理常采用的主存分配算法包括首次适应、最优适应和循环首次适应等算法
- ( ) B.首次适应算法实现简单，但碎片过多使主存空间利用率降低
- (x) C.最优适应算法是最好的算法，但后到的较大作业很难得到满足
- ( ) D.循环首次适应算法能使内存中的空闲分区分布得更均匀

知识点:非连续内存分配
出处:网络
难度:1

> C
> 1
> 如下表所示，虚拟段页式存储管理方案的特性为() 地址空间 空间浪费 存储共享 存储保护 动态扩充 动态连接

- ( ) A.一维 大 不易 易 不可 不可
- ( ) B.一维 小 易 不易 可以 不可
- ( ) C.二维 大 不易 易 可以 可以
- (x) D.二维 小 易 易 可以 可以

知识点:非连续内存分配
出处:网络
难度:1

> D
> 1
> 执行一次磁盘输入输出操作所花费的时间包括

- ( ) A.寻道时间、旋转延迟时间、传送时间和等待时间
- ( ) B.寻道时间、等待时间、传送时间
- ( ) C.等待时间、寻道时间、旋转延迟时间和读写时间
- (x) D.寻道时间、旋转延迟时间、传送时间

知识点:I/O子系统
出处:网络
难度:1

> D
> 1
> 在下列操作系统的各个功能组成部分中,哪一个不需要有硬件的支持

- (x) A.进程调度
- ( ) B.时钟管理
- ( ) C.地址映射
- ( ) D.中断系统

知识点:操作系统概述
出处:网络
难度:1

> A
> 1
> 一个正在访问临界资源的进程由于申请等待I/O操作而被中断时

- ( ) A.可以允许其他进程进入与该进程相关的临界区
- ( ) B.不允许其他进程进入任何临界区
- (x) C.可以允许其他就绪进程抢占处理器，继续运行
- ( ) D.不允许任何进程抢占处理器

知识点:同步互斥
出处:网络
难度:1

> C
> 1
> 批处理操作系统的特点不包括

- ( ) A.提高了系统资源的利用率
- (x) B.用户可以直接干预作业的运行，具有交互性
- ( ) C.提高了单位时间内的处理能力
- ( ) D.提高了系统的吞吐率

知识点:操作系统概述
出处:网络
难度:1

> B
> 1
> 下面不属于操作系统提供虚拟设备技术原因的是

- ( ) A.独占设备可以作为共享设备来使用
- ( ) B.独占设备使用的静态分配技术既不能充分利用设备，又不利于提高系统效率
- ( ) C.在一定硬件和软件条件的基础上共享设备可以部分或全部地模拟独占设备的工作，提高独占设备的利用率和系统效率
- (x) D.计算机系统具有多道处理功能，允许多道作业同时执行

知识点:I/O子系统
出处:网络
难度:1

> D
> 1
> 采用多道程序设计的实质之一是

- (x) A.以空间换取时间
- ( ) B.将独享设备改造为共享设备
- ( ) C.提高内存和I/O设备利用率
- ( ) D.虚拟设备

知识点:操作系统概述
出处:网络
难度:1

> A
> 1
> 访管指令的作用是

- ( ) A.嵌套调用
- ( ) B.用户使用的命令
- (x) C.用户态转换为核心态
- ( ) D.保证运行在不同状态

知识点:中断、异常与系统调用
出处:网络
难度:1

> C
> 1
> 不属于I/O控制方式的是

- ( ) A.程序查询方式
- (x) B.复盖方式

- ( ) C.DMA方式
- ( ) D.中断驱动方式

知识点:I/O子系统
出处:网络
难度:1

> B
> 1
> 软件共享的必要性是为了

- ( ) A.节约内存空间
- ( ) B.缩短运行时间
- ( ) C.减少内外存对换信息量
- (x) D.A和C

知识点:非连续内存分配
出处:网络
难度:1

> D
> 1
> 下面软件系统中完全属于系统软件的一组是

- (x) A.操作系统、编译系统、windowsNT
- ( ) B.接口软件、操作系统、软件开发工具
- ( ) C.专用程序、财务管理软件、编译系统、操作系统
- ( ) D.操作系统、接口软件、Office 2000

知识点:操作系统概述
出处:网络
难度:1

> A
> 1
> 主存储器是

- ( ) A.以“字”为单位进行编址的
- (x) B.是中央处理机能够直接访问的惟一的存储空间
- ( ) C.与辅助存储器相比速度快、容量大、价格低的一类存储器
- ( ) D.只能被CPU访问的存储器

知识点:连续内存分配
出处:网络
难度:1

> B
> 1
> 特权指令

- (x) A.是可能影响系统安全的一类指令
- ( ) B.既允许操作系统程序使用，又允许用户程序使用
- ( ) C.是管态和目态运行的基本单位
- ( ) D.是一种存储保护方法

知识点:操作系统概述
出处:网络
难度:1

> A
> 1
> 下面有关选择进程调度算法的准则错误的是

- ( ) A.尽量提高处理器利用率
- ( ) B.尽可能提高系统吞吐量
- (x) C.适当增长进程在就绪队列中的等待时间
- ( ) D.尽快响应交互式用户的请求

知识点:处理机调度
出处:网络
难度:1

> C
> 1
> 下面是关于重定位的有关描述，其中错误的是

- ( ) A.绝对地址是主存空间的地址编号
- ( ) B.用户程序中使用的从0地址开始的地址编号是逻辑地址
- ( ) C.动态重定位中装入主存的作业仍保持原来的逻辑地址
- (x) D.静态重定位中装人主存的作业仍保持原来的逻辑地址

知识点:非连续内存分配
出处:网络
难度:1

> D
> 3
> 操作系统的所有程序都必须常驻内存

- ( ) A.对
- (x) B.错

知识点:操作系统概述
出处:网络
难度:1

> B
> 3
> 虚拟存储系统可以在每一台计算机上实现

- ( ) A.对
- (x) B.错

知识点:缺页中断
出处:网络
难度:1

> B
> 3
> 执行系统调用时可以被中断

- (x) A.对
- ( ) B.错

知识点:中断、异常与系统调用
出处:网络
难度:1

> A
> 3
> 选择通道主要用于连接低速设备

- ( ) A.对
- (x) B.错

知识点:操作系统概述
出处:网络
难度:1

> B
> 3
> 在请求分页存储管理中，从主存中刚刚移走某一页面后，根据请求马上又调进该页，这种反复调进调出的现象，称为系统颠簸，也叫系统抖动

- (x) A.对
- ( ) B.错

知识点:置换算法
出处:网络
难度:1

> A
> 3
> 通道程序解决了I / O操作的独立性和各部件工作的并行性，采用通道技术后，能实现CPU与通道的并行操作

- (x) A.对
- ( ) B.错

知识点:I/O子系统
出处:网络
难度:1

> A
> 3
> 程序的顺序执行具有顺序性，封闭性和不可再现性

- ( ) A.对
- (x) B.错

知识点:操作系统概述
出处:网络
难度:1

> B
> 3
> 快表是高速缓存，是内存的一部分区域

- ( ) A.对
- (x) B.错

知识点:非连续内存分配
出处:网络
难度:1

> B
> 3
> 磁盘上物理结构为链接结构的文件只能顺序存取

- (x) A.对
- ( ) B.错

知识点:I/O子系统
出处:网络
难度:1

> A
> 3
> 一旦出现死锁, 所有进程都不能运行

- ( ) A.对
- (x) B.错

知识点:死锁
出处:网络
难度:1

> B
> 4
> "什么叫进程同步和互斥?举例说明"

- [x]

知识点:同步互斥
出处:网络
难度:1

> 进程同步是在几个进程合作完成一项任务时，体现各进程相互联系相互协调的关系。例如：A、B两个进程合作通过缓存区输出数据。
> 把两个以上进程不能同时访问临界区的工作
> 规则称为进程互斥。例如：两个进程同时使用打印机
> 4
> "什么是动态链接"

- [x]

知识点:非连续内存分配
出处:网络
难度:1

> 指用户程序中的各程序段，不是在程序开始运行前就链接好，而是在程序装入或运行过程中，当发现要调用的程序段未链接时，才进行链接。
> 4
> "在下面的条件下，若用一个位图来实现空闲表，那么存储空闲表需要多少位? （a） 共有500000个块，有200000个空闲块 （b） 共有500000个块，有0个空闲块"

- [x]

知识点:置换算法
出处:网络
难度:1

> 在任何一种情况下，每个地址所用的位数和空闲块数目无关。在500000个块中，需要500000位。
> 4
> "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。若缺页率是10%，为使有效访问时间达到0.5ms,求不在内存的页面的平均访问时间
> 。"

- [x]

知识点:置换算法
出处:网络
难度:1

> 4.99865ms
> 4
> "设P,Q,R共享一个缓冲区,P,Q构成一对生产者-消费者,R既为生产者又为消费?者。使用P,V 实现其同步。"

- [x]

知识点:信号量
出处:网络
难度:1

> Semaphore 方法 设置三个信号量：full(itemCounter)、empty(vacancyCounter)和mutex。
> full表示有数据的缓冲块数目，初值是0；empty表示空的缓冲块数初值是n；mutex用于访问缓冲区时的互斥，初值是1。
> 三种进程，consumer,producer,both，both表示既是producer又是consumer。 producer 伪码 while true
> p(empty); P(mutex); produce one; v(mutex); v(full); end while consumer 伪码
> while true p(full); P(mutex); consume one; v(mutex); v(empty); end while both
> 伪码 if empty>=1 then begin p(empty); p(mutex); product one; v(mutex); v(full);
> end if full>=1 then begin p(full); p(mutex); consume one; v(mutex); v(empty);
> end Monitor 方法
> 设置一个monitor，内有两个条件变量：notFull和notEmpty。其中，notFull表示缓存满，notEmpty表示缓存空 producer
> 伪码 lock.Acquire(); while (count == n) notFull.Wait(&lock;); produce one;
> count++; notEmpty.Signal(); end while lock.Release(); consumer 伪码
> lock.Acquire(); while (count == 0); notEmpty.Wait(&lock;); consume one;
> count--; notFull.Signal(); end while lock.Release(); both 伪码 lock.Acquire();
> notEmpty.Wait(&lock;); consume one; count--; notFull.Signal();

> notFull.Wait(&lock;); produce one; count++; notEmpty.Signal();
> lock.Release();
> 4
> "此问题是对读者-写者问题的一个扩展，既如果读者写者均是平等的即二者都不优先情况下。
> 此问题的一个更高的版本是说，每个资源可以同时读取的人的个数也是有限的（限制数RN）。"

- [x]

知识点:信号量
出处:网络
难度:1

> "为了达到公平的目的，即在读者进行读取的时候，如果有写者在排队，后面的读者不能够加入到读取的队列中来，应该等待写者执行完写操作之后再进行读取。
> 针对上面一种情况引入一个排队信号量q,每次有操作必须等待这个信号量释放再进行操作（如果有写操作在排队，q没有释放，下一个读操作没有办法进入并进行读操作）
> 算法流程 q,s, mutex <=1, ReadCount <= 0 Reader: while True: wait(q) wait(mutex) if
> ReadCount ==0 wait(s) ReadCount++ signal(mutex) signal(q) READING.....
> signal(mutex) ReadCount-- if ReadCount==0 signal(s) signal(mutex) end while
> Writer: While True: wait(q) wait(s) WRITING..... singal(s) singal(w)
> 问题二使用一个计数器计算当前还有几个剩下的读者名额，当写者掌控时，直接进行0/RN级别的替换。 代码无需修改。"
> 4
> "有一个许多进程共享的数据区，有一些只读这个数据区的进程(reader)和一些只往数据区中写数据的进程(writer)；此外还需满足如下条件：
> 1.任意多的读进程可以同时读这个文件。 2.一次只有一个写进程可以往文件中写。 3.如果一个写进程正在往文件中写时，则禁止任何读进程和其他写进程。
> 实现基于先来先服务策略的读者 - 写者的问题，具体要求描述如下： 1.存在m个读者和n个写者，共享同一个缓冲区。
> 2.当没有读者在读，写者在写时，读者写者均可进入读或写。 3.当有读者在读时： (1) 写者来了，则写者等待。 (2)
> 读者来了，则分两种情况处理：无写者等待，则读者可以直接进入读操作，如果有写者等待，则读者必须依次等待。 4.当有写者在写时，写者或读者来了，均需等待。
> 5.当写者写完后，如果等待队列中第一个是写者，则唤醒该写者；如果等待队列中第一个是读者，则唤醒该队列中从读者开始连续的所有读者。
> 6.当最后一个读者读后，如果有写者在等待，则唤醒第一个等待的写者。"

- [x]

知识点:信号量
出处:网络
难度:1

> 前面的实现方法中可能出现多个写和读同时等待同一个锁打开，一旦锁打开，会随机挑选一个操作执行，但我们知道在写操作之后加入的读操作是不能在写操作之前执行的，所以
> 上述的方法会 有错误产生。 可以考虑建立一个读写操作队列，给队列设置两个队列锁（read锁锁定read操作，write锁锁定write操作），每次挑选队列中
> 最早加入的操作执行，由于数组删除很复杂，所以采用循环数组。以信号量实现为例，管程的实现方法也是对前一位同学的代码做出相应类似的修改即可。贴出主要代码(读写队
> 列操作部分，monitor不再赘述，跟很多人是一样的)： 变量定义 #define OP_NUM 200; //操作队列上限 int op_num = 0;
> //队列当前等待数目 int op_list[OP_NUM]; //等待队列，奇数为读，偶数为写 int start=0;//队首位置 int
> end=-1;//队尾位置 semaphore_t op_sem;//队首和队尾位置,等待数目锁 semaphore_t
> list_read_sem;//队列读互斥锁 semaphore_t list_write_sem;//队列写互斥锁 读操作 int read_op(int
> id){ down(&list;*write_sem);//只锁写操作 cprintf("No.%d Reader is*
> *reading*
> *",i); do_sleep(50); cprintf("No.%d Reader finished reading*
> *",i);*
> *up(&list;*write_sem); cprintf("No.%d Reader Sem Proc Quit
> ",i); return 0;
> } 写操作 int write_op(int id){ down(&list;*write_sem);*
> *down(&list;*read_sem);//同时锁定读写操作 cprintf("No.%d Writer is writing
> ",i);
> do_sleep(50); cprintf("No.%d Writer finished writing
> ",i);
> up(&list;*write_sem); up(&list;*read_sem);//同时解锁 cprintf("No.%d Writer
> Sem Proc Quit
> ",i); return 0; } 加入操作 int add_op(int id){
> down(&op;*sem);//锁定队列信息 if(op_num>OP_NUM) return -1;//队列已满*
> *end=(end+1)%OP_NUM; op_list[end]=id; op_num_sem++; up(&op;*sem); return 0;
> } 队列执行操作 int run_op(){ if(op_num==0) return -1;//队列为空
> if(op_list[start]%2==1){//读操作 read_op(op_list[start]); } else{
> write_op(op_list[start]); } down(&op;*sem);//锁住队列信息*
> *start=(start+1)%OP_NUM; op_num--; up(&op;*sem); return 0; }
> 1
> 下列哪一条不是批处理系统的优点?

- ( ) A.吞吐量大
- ( ) B.资源利用率高
- ( ) C.系统开销小
- (x) D.响应及时

知识点:操作系统概述
出处:网络
难度:1

D
4
"在一间酒吧里有三个音乐爱好者队列，第一队的音乐爱好者只有随身听，第二队的只有音乐磁带，第三队只有电池。而要听音乐就必须随身听，音乐磁带和电池这三种物品俱全
。酒吧老板依次出售这三种物品中的任意两种。当一名音乐爱好者得到这三种物品并听完一首乐曲后，酒吧老板才能再一次出售这三种物品中的任意两种。于是第二名音乐爱好者
得到这三种物品，并开始听乐曲。全部买卖就这样进行下去。试用P，V操作正确解决这一买卖。"

- [x]

知识点:信号量
出处:网络
难度:1

```
#include  #include  #include  #include  #include  #define ROUND 10 const char
GOODS[3][20] = { "Walkman", "Tape", "Battery" }; const char WANT[3][20] = {
"Tape&Battery;", "Walkman&Battery;", "Walkman&Tape;" }; int
sema_flag; int condvar_flag; semaphore_t listener[3]; semaphore_t seller;
struct proc_struct listener_sema_proc[3]; struct proc_struct
seller_sema_proc; void listener_sema(void arg){ int i = (int) arg;
while(sema_flag){ down(&listener;[i]); if (sema_flag){ cprintf("No %d
listener has %s, and bought %s. sema
",i,GOODS[i],WANT[i]);
up(&seller;); } } cprintf("No %d listener quit! sema
",i); } void
seller_sema(void arg){ int i; int pos; for(i=0;i<ROUND;i ++){ pos = rand() %
3; cprintf("Iter %d : Seller is selling: %s. sema
",i,WANT[pos]);
up(&listener;[pos]); down(&seller;); } sema_flag = 0; for(i = 0;i <
3;i ++) up(&listener;[i]); cprintf("Seller quit! sema
"); } monitor_t
lmt, mtp2= &lmt; struct proc_struct listener_condvar_proc[3]; struct
proc_struct seller_condvar_proc; void seller_condvar(void arg){ int i; int
pos; for(i = 0;i < ROUND;i ++){ down(&mtp2-;>mutex); pos = rand() % 3;
cprintf("Iter %d : Seller is selling: %s. condvar
",i,WANT[pos]);
cond_signal(&mtp2-;>cv[pos + 1]); cond_wait(&mtp2-;>cv[0]); if
(mtp2->next_count > 0) up(&mtp2-;>next); else up(&mtp2-;>mutex); }
condvar_flag = 0; down(&mtp2-;>mutex); for(i = 0;i < 3;i ++)
cond_signal(&mtp2-;>cv[i + 1]); cprintf("Seller_condvar quit!
"); if
(mtp2->next_count > 0) up(&mtp2-;>next); else up(&mtp2-;>mutex); }
void listener_condvar(void arg){ int num = (int)arg; down(&mtp2-;>mutex);
cprintf("No %d listener is waiting
", num); cond_wait(&mtp2-;>cv[num+1]);
if (mtp2->next_count > 0) up(&mtp2-;>next); else up(&mtp2-;>mutex);
while(condvar_flag){ down(&mtp2-;>mutex); if(condvar_flag){ cprintf("No %d
listener has %s, and bought %s and is listening music now.condvar
",num,GOODS[num],WANT[num]); cond_signal(&mtp2-;>cv[0]);
cond_wait(&mtp2-;>cv[num + 1]); } if (mtp2->next_count > 0)
up(&mtp2-;>next); else up(&mtp2-;>mutex); } cprintf("No %d listener
quit! condvar
",num); } void check_sync(void) {/ 吸烟者问题拓展一（北大1999) / int i,
pid; //check semaphore sem_init(&seller;, 0); pid =
kernel_thread(seller_sema, NULL, 0); if (pid <= 0) { panic("create seller_sema
failed.
"); } seller_sema_proc = find_proc(pid);
set_proc_name(seller_sema_proc, "seller_sema_proc"); sema_flag = 1; for(i = 0;
i < 3; ++i){ sem_init(&listener;[i], 0); pid =
kernel_thread(listener_sema, (void )i, 0); if (pid <= 0) { panic("create
No.%d listener_sema failed.
", i); } listener_sema_proc[i] = find_proc(pid);
set_proc_name(listener_sema_proc[i], "listener_sema_proc"); } //check
condition variable monitor_init(&lmt;, 4); pid =
kernel_thread(seller_condvar, NULL, 0); if (pid <= 0) { panic("create
seller_condvar failed.
"); } seller_condvar_proc = find_proc(pid);
set_proc_name(seller_condvar_proc, "seller_condvar_proc"); condvar_flag = 1;
for(i = 0; i < 3; ++i){ pid = kernel_thread(listener_condvar, (void )i, 0);
if (pid <= 0) { panic("create No.%d listener_condvar failed.
"); }
listener_condvar_proc[i] = find_proc(pid);
set_proc_name(listener_condvar_proc[i], "listener_condvar_proc"); } }
```

4
"假设一个录像厅有0,1，2三种不同的录像片可由观众选择放映，录像厅的放映规则为:
任一时刻最多只能放映一种录像片，正在放映的录像片是自动循环放映的，最后一个观众主动离开时结束当前录像片的放映；
选择当前正在放映的录像片的观众可立即进入，允许同时有多位选择同一种录像片的观众同时观看，同时观看的观众数量不受限制；
等待观看其他录像片的观众按到达顺序排队，当一种新的录像片开始放映时，所有等待观看该录像片的观众可依次序进入录像厅同时

> 观看。用一个进程代表一个观众。
> 要求:用信号量方法PV实现，并给出信号量定义和初始值。（最好也能写出录像厅的进程）"

- [x]

知识点:信号量
出处:网络
难度:1

> #include #include #include #include #include int cinema=-1; int people=0;
> semaphore_t mov[num]; / 每个电影一个信号量 / int wait[3]; void semaphore_test(i) /
> i：影片编号 / { if(cinema==-1 || (cinema==i && people>0)) { cinema=i;
> up(&mov;[i]); } } void semaphore_movie_play(int i) { down(&mutex;);
> semaphore_test(i); int ifwait=0; if (i!=cinema) ifwait=1; wait[i]+=ifwait;
> //cprintf("testing %d %d %d
> ",cinema,i,mov[i].value); up(&mutex;);
> down(&mov;[i]); down(&mutex;); wait[i]-=ifwait; people++; cinema=i;
> cprintf("No.%d movie_sema is playing,remain people num:%d
> ",i,people);
> /电影放映/ //cprintf("testING %d %d %d %d
> ",cinema,i,mov[i].value,wait[i]); if
> (wait[i]!=0) up(&mov;[i]); up(&mutex;); //if (bf==people)
> down(&mov;[i]); } void semaphore_cinema_end(int i) / i：影片编号从0到N-1 / {
> down(&mutex;); / 进入临界区 / people--; cprintf("No.%d movie_sema quit,remain
> people num: %d
> ",i,people); if(people==0) cinema=-1; semaphore_test(left);
> semaphore_test(right); / 看一下其他影片可否播放 / up(&mutex;); / 离开临界区 / } int
> semaphore_movie(void arg) / i：电影编号，从0到N-1 / { int i, iter=0; i=(int)arg;
> cprintf("I am No.%d movie_sema
> ",i); cprintf("Iter %d, No.%d movie_sema is
> ready
> ",iter,i); do_sleep(SLEEP_TIME); semaphore_movie_play(i); / 开始电影放映 /
> do_sleep(SLEEP_TIME); semaphore_cinema_end(i); / 结束放映 / cprintf("No.%d
> movie_sema quit
> ",i); return 0; }
> 4
> "银行有n个柜员,每个顾客进入银行后先取一个号,并且等着叫号,当一 个柜员空闲后,就叫下一个号."

- [x]

知识点:信号量
出处:网络
难度:1

> 将顾客号码排成一个队列,顾客进入银行领取号码后,将号码由队尾插入;柜员空闲 时,从队首取得顾客号码,并且为这个顾客服务,由于队列为若干进程共享, 所以需要互
> 斥.柜员空闲时,若有顾客,就叫下一个顾客为之服务.因此,需要设置一个信号量来记录等 待服务的顾客数. begin var
> mutex=1,customer_count=0:semaphore; cobegin process customer begin repeat 取号码;
> p(mutex); 进入队列； v(mutex); v(customer_count); end process serversi(i=1,...,n)
> begin repeat p(customer_count); p(mutex); 从队列中取下一个号码； v(mutex); 为该号码持有者服务； end
> 4
> "假设缓冲区buf1和缓冲区buf2无限大，进程p1向buf1写数据，进程p2向buf2写数据，
> 要求buf1数据个数和buf2数据个数的差保持在(m,n)之间(m<n,m,n都是正数)."

- [x]

知识点:信号量
出处:网络
难度:1

> 题中没有给出两个进程执行顺序之间的制约关系，只给出了一个数量上的制约 关系，即m≤|buf1数据个数 - buf2数据个数≤n．不需要考虑缓冲区的大小，只需要考
> 虑两个进程的同步和互斥．p2向buf2写数据比p1向buf1写数据的次数最少不超过m次，
> 最多不能超过n次，反之也成立．所以是一个生产者和消费者问题。将等式展开得： (1)m≤(buf1数据个数 - buf2数据个数)≤n;
> (2)m≤(buf2数据个数 - buf1数据个数)≤n;由 于m,n都是正数，等式只有一个成立，不妨设(1)成立．在进程p1和p2都没有运行时，
> 两个缓冲区数据个数之差为0,因此，p1必须先运行，向buf1至少写m+1个数据后再唤
> 醒p2运行．信号量s1表示p1一次写入的最大量,初值为n，s2表示p2一次写入的最大量,初 值为-m. begin var
> mutex1=1,mutex2=1,s1=n,s2=-m:semaphore; cobegin process p1 begin repeat get
> data; p(s1); p(mutex1); 写数据到buf1; v(mutex1); v(s2); end process p2 begin
> repeat; get data; p(s2); p(mutex2); 写数据到buf2; v(mutex2); v(s1); end
> 4
> 有三个并发进程P、Q和R以及一对供存储数据的缓冲BufI和BufO，P进程把数据输入BufI，R进程输出BufO中的数据。Q地把BufI中的数据变换后送入B
> ufO，在上述假定之下,使三个进程实现最大并行性。试在下述类PASCAL程序中虚线位置分别填上信号量、信号量初值和P、V操作实现三个进程正确的并发执行。
> 

- [x]

知识点:信号量
出处:网络
难度:1

Program ito; var Bufl,BufO:buffer; (信号量)_____:SEMAPHORE:=
(信号量初值)_____;
begin
parbegin
procedure P
begin
repeat
input from IO; _____ Add to Bufl; _____
until false end; procedure Q; begin
repeat _____
Remove from Bufl;
_____
transform;
_____
Add to BufO;
_____ until false end;
procedure R; begin
repeat _____
Remove from BufO;
_____
Output ...;
until false end;parend
end
4
（10分）当一个进程释放一个包含某虚地址的物理内存页时，需要让对应此物理内存页的管理数据结构Page进行清除处理，使得此物理内存页成为空闲。同时，还需把表示
虚地址与物理地址映射关系的二级页表项清除，这个工作由page_remove_pte函数完成。
page_remove_pte函数的调用关系图如下所示。请补全在 kern/mm/pmm.c中的page_remove_pte函数。

图1 page_remove_pte函数的调用关系图
==========Pmm.h (kern\mm)==========
#define alloc_page() alloc_pages(1)
#define free_page(page) free_pages(page, 1)
.....
static inline struct Page
pte2page(pte_t pte) {
if (!(pte & PTE_P)) {
panic("pte2page called with invalid pte");
}
return pa2page(PTE_ADDR(pte));
}
.....
static inline int
page_ref_inc(struct Page page) {
page->ref += 1;
return page->ref;
}
static inline int
page_ref_dec(struct Page page) {
page->ref -= 1;
return page->ref;
}
.....
==========Pmm.c (kern\mm)==========
.....
//page_remove_pte - free an Page sturct which is related linear address la
// - and clean(invalidate) pte which is related linear address la
//note: PT is changed, so the TLB need to be invalidate
static inline void
page_remove_pte(pde_t pgdir, uintptr_t la, pte_t ptep) {
/ LAB2 EXERCISE 3: YOUR CODE

Please check if ptep is valid, and tlb must be manually updated if mapping is updated

Maybe you want help comment, BELOW comments can help you finish the code

Some Useful MACROs and DEFINEs, you can use them in below implementation.
MACROs or Functions:
struct Page page pte2page(ptep): get the according page from the value of a ptep
free_page : free a page
page_ref_dec(page) : decrease page->ref. NOTICE: ff page->ref == 0 , then this page should be free.
tlb_invalidate(pde_t pgdir, uintptr_t la) : Invalidate a TLB entry, but only if the page tables being
edited are the ones currently in use by the processor.
DEFINEs:
PTE_P 0x001 // page table/directory entry flags bit : Present
/
#if 0
if (0) { //(1) check if page directory is present
struct Page page = NULL; //(2) find corresponding page to pte

//(3) decrease page reference
//(4) and free this page when page reference reachs 0
//(5) clear second page table entry
//(6) flush tlb
}
#endif
===Your code 1===
}
.....
// invalidate a TLB entry, but only if the page tables being
// edited are the ones currently in use by the processor.
void
tlb_invalidate(pde_t pgdir, uintptr_t la) {
if (rcr3() == PADDR(pgdir)) {
invlpg((void)la);
}
}
static void
check_alloc_page(void) {
pmm_manager->check();
cprintf("check_alloc_page() succeeded!
");
}
=============Mmu.h (kern\mm)=============
/ page table/directory entry flags /
#define PTE_P 0x001 // Present
#define PTE_W 0x002 // Writeable
#define PTE_U 0x004 // User
#define PTE_PWT 0x008 // Write-Through
#define PTE_PCD 0x010 // Cache-Disable
#define PTE_A 0x020 // Accessed
#define PTE_D 0x040 // Dirty
#define PTE_PS 0x080 // Page Size
#define PTE_MBZ 0x180 // Bits must be zero
#define PTE_AVAIL 0xE00 // Available for software use
// The PTE_AVAIL bits aren't used by the kernel or interpreted by the
// hardware, so user processes are allowed to set them arbitrarily.
#define PTE_USER (PTE_U | PTE_W | PTE_P)

- [x]

知识点:虚拟内存管理实验
出处:网络
难度:1

> if (ptep & PTE_P) {//判断页面存在（2分）
> struct Page page = pte2page(ptep);//获取物理页面数据结构指针（1分）
> if (page_ref_dec(page) == 0) {//物理页面数据结构中引用计数减一（2分）
> free_page(page); //释放占用页面（2分）
> }
> ptep = 0; //页表项内容清除（2分）
> tlb_invalidate(pgdir, la); //更新TLB（1分）
> }
> 4
> 在南开大学至天津大学间有一条弯曲的路，每次只允许一辆自行车通过，但中间有小的安全岛M（同时允许两辆车），可供两辆车在已进入两端小车错车，设计算法并使用P，V
> 实现。
> 

- [x]

知识点:同步互斥
出处:网络
难度:1

> 由于安全岛M仅仅允许两辆车停留,本应该作为临界资源而要设置信号量, 但根据题意,任意时刻进入安全岛的车不会超过两辆(两个方向最多各有一辆), 因此，不需要为
> M设置信号量,在路口s和路口t都需要设置信号量,以控制来自两个方向的车对路口资源的争夺.这两个信号量的初值都是1.此外，由于从s到t的一段路只允许一辆车通过
> ,所以还需要设置另外的信号量用于控制,由于M的存在,可以为两端的小路分别设置一个互斥信号量.

```
  var T2N, N2T,L,M,K:semaphore;
  T2N:=1;
  N2T:=1;
  L:=1;
  K:=1;
  M:=2;
cobegin
  Procedure Bike T2N
  begin
    p(T2N);
    p(L);
```

```
      go T to L;
    p(M);
      go into M;
    V(L);
    P(k);
      go K to s;
    V(M);
    V(k);
    V(T2N);
  end
  Procedure Bike N2T
  begin
    P(N2T);
    p(k);
      go v to k;
    p(M);
      go into M;
    V(k);
    P(L);
      go L to T;
    V(M);
    V(L);
    V(N2T);
  end
coend
```

4
在一个盒子里，混装了数量相等的黑白围棋子·现在用自动分拣系统把黑子、白子分开，设分拣系统有二个进程P1 和P2 ，其中P1 拣白子；P2
拣黑子。规定每个进程每次拣一子；当一个进程在拣时，不允许另一个进程去拣；当一个进程拣了一子时，必须让另一个进程去拣.
试写出两进程P1 和P2
能并发正确执行的程序。

- [x]

知识点:同步互斥
出处:网络
难度:1

大家熟悉了生产-消费问题(PC)，这个问题很简单。题目较为新颖，但是本质非常简单即：生产-消费问题的简化或者说是两个进程的简单同步问题。答案如下：

```
设信号量s1 和s2 分别表示可拣白子和黑子;
不失一般性，若令先拣白子。
var S1 , S2 : semaphore;
S1 : = l; S2 : =0;
cobegin
  process P1            process P2
  begin                 begin
    repeat                repeat
      P(S1);                p(S2);
      pick The white;       pick the black;
      V(S2);                v(s1);
    until false;          until false;
  end                   end
coend
```

4
设公共汽车上，司机和售票员的活动分别如下：司机的活动：启动车辆：正常行车；到站停车。售票员的活动：关车门；售票；开车门。在汽车不断地到站、停车、行驶过程中，
这两个活动有什么同步关系？用信号量和P 、V 操作实现它们的同步。


- [x]

知识点:同步互斥
出处:网络
难度:1

在汽车行驶过程中，司机活动与售票员活动之间的同步关系为：售票员关车门后，向司机发开车信号，司机接到开车信号后启动车辆，在汽车正常行驶过程中售票员售票，到站时
司机停车，售票员在车停后开门让乘客上下车。因此，司机启动车辆的动作必须与售票员关车门的动作取得同步；售票员开车门的动作也必须与司机停车取得同步。应设置两个信
号量：S1 、S2 ;
S1表示是否允许司机启动汽车（其初值为0 ）
S2表示是否允许售票员开门（其初值为0 ）
用P 、v 原语描述如下：

```
var S1,S2 : semaphore ;
    S1=0；S2=0；
cobegin
Procedure driver    Procedure Conductor
  begin               begin
  while TRUE            while TRUE
  begin                 begin
```

```
    P(S1);                关车门;
    Start;                v(s1);
    Driving;              售票;
    Stop;                 p(s2);
    V(S2);                开车门;
  end                     上下乘客;
end                       end
                        end
coend
```

4
某寺庙，有小和尚、老和尚若干．庙内有一水缸，由小和尚提水入缸，供老和尚饮用。水缸可容纳10桶水，每次入水、取水仅为1桶，不可同时进行。水取自同一井中，水井径
窄，每次只能容纳一个水桶取水。设水桶个数为3个，试用信号灯和PV操作给出老和尚和小和尚的活动。

- [x]

知识点:同步互斥
出处:网络
难度:1

从井中取水并放入水缸是一个连续的动作可以视为一个进程，从缸中取水为另一个进程。
设水井和水缸为临界资源，引入mutex1,mutex2；三个水桶无论从井中取水还是放入水缸中都一次一个，应该给他们一个信号量count，抢不到水桶的进程只好
为等待，水缸满了时，不可以再放水了。设empty控制入水量，水缸空了时，不可取水设full。

```
var mutex1,mutex2,empty,full,count:semaphore;
mutex1:=mutex2:=1;
empty:=10;
full:=0;
count:=3;
cobegin
  Procedure Fetch_Water     Procedure Drink_Water
    begin                     begin
    while true                  while true
      p(empty);                   p(full);
      P(count);                   p(count);
      P(mutex1);                  p(mutex2);
        Get Water;                  Get water and
      v(mutex1);                    Drink water;
      P(mutex2);                  p(mutex2);
      pure water into the jar;    v(empty);
      v(mutex2);                  v(count);
      v(count);                 end
      v(full);
    end
coend
coend
```

1
I/O请求完成会导致哪种进程状态演变?

- ( ) A.就绪 → 执行
- (x) B.阻塞 → 就绪
- ( ) C.阻塞 → 执行
- ( ) D.执行 → 阻塞

知识点:操作系统概述
出处:网络
难度:1

B
4
一座小桥(最多只能承重两个人)横跨南北两岸，任意时刻同一方向只允许一人过桥，南侧桥段和北侧桥段较窄只能通过一人，桥中央一处宽敞，允许两个人通过或歇息。试用信
号灯和PV操作写出南、北两岸过桥的同步算法。

- [x]

知识点:同步互斥
出处:网络
难度:1

桥上可能没有人，也可能有一人，也可能有两人。
两人同时过桥
两人都到中间
南(北)来者到北(南)段
共需要三个信号量，load用来控制桥上人数，初值为2，表示桥上最多有2人；north用来控制北段桥的使用，初值为1，用于对北段桥互斥；south用来控制南段
桥的使用，初值为1，用于对南段桥互斥。

```
var load,north,south:semaphore;
load=2;
north=1;
south=1;
```

```
    GO_South()
      P(load);
      P(north);
        过北段桥;
        到桥中间;
      V(north);
      P(south);
        过南段桥;
        到达南岸;
      V(south);
      V(load);
    GO_North()
      P(load);
      P(south);
        过南段桥;
        到桥中间
      V(south);
      P(north);
        过北段桥;
        到达北岸
      V(north);
      V(load);
```

4
两人公用一个账号，每次限存或取10元；

- [x]

知识点:同步互斥
出处:网络
难度:1

```
begin
var mutex=1:semaphore;
amount =0:integer;
cobegin
  process save
    m1: integer;
    begin
    repeat
    p(mutex);
    m1= amount ;
    m1 = m1 +10;
    amout = m1;
    v(mutex);
    end
  process take
    m2: integer;
    begin
    repeat;
    p(mutex);
    m2= amount ;
    m2 = m2 -10;
    amout = m2;
    v(mutex);
    end
coend
```

4
某高校计算机系开设网络课并安排上机实习，假设机房共有2m台机器，有2n名学生选课（m，n均大于等于1），规定：
每两个学生组成一组，各占一台及其协同完成上机实习；
只有一组两个学生到齐，并且此时机房有空闲机器时，该组学生才能进入机房；
上机实习由一名教师检查，检查完毕，一组学生同时离开机房
试用P、V实现其过程。
注意：
本题目隐含一个进程(Guard)。

- [x]

知识点:同步互斥
出处:网络
难度:1

```
var stu,computer,enter,finish,test:semaphore;
ste:=2N;
computer:=2M;
enter:=0;
finish:=0;
test:=0;
cobegin
Procedure Student   Procedure Teacher   Procedure Guard
  begin               begin               begin
  p(computer);        p(finish);            p(stu);
  p(stu);              Test the work;       p(stu);
    Start computer;   v(test);              Enter;
  v(finish);          v(test);              v(enter);
```

```
    v(test);            end                  v(enter);
    v(computer);                           end
    end
  coend
```

4
(18分)调度器是操作系统内核中依据调度算法进行进程切换选择的模块。
1）试描述步进调度算法(Stride Scheduling)的基本原理。
2）请补全下面 ucore代码中调度器和步进调度算法实现中所缺代码，以实现调度器和调度算法的功能。提示：每处需要补全的代码最少只需要一行，一共有9个空要填。
当然，你可以在需要补全代码的地方写多行来表达需要实现的功能，也允许修改已给出的代码。
3）试描述斜堆(skew heap)在这个步进调度算法中的作用。
```
kern/process/proc.h
================== kern/process/proc.h ======================
#ifndef KERN_PROCESS_PROC_H
#define KERN_PROCESS_PROC_H
#include
#include
#include
#include
#include
// process's state in his life cycle
enum proc_state {
    PROC_UNINIT = 0,  // uninitialized
    PROC_SLEEPING,    // sleeping
    PROC_RUNNABLE,    // runnable(maybe running)
    PROC_ZOMBIE,      // almost dead, and wait parent proc to reclaim his resource
};
// Saved registers for kernel context switches.
// Don't need to save all the %fs etc. segment registers,
// because they are constant across kernel contexts.
// Save all the regular registers so we don't need to care
// which are caller save, but not the return register %eax.
// (Not saving %eax just simplifies the switching code.)
// The layout of context must match code in switch.S.
struct context {
    uint32_t eip;
    uint32_t esp;
    uint32_t ebx;
    uint32_t ecx;
    uint32_t edx;
    uint32_t esi;
    uint32_t edi;
    uint32_t ebp;
};
#define PROC_NAME_LEN               15
#define MAX_PROCESS                 4096
#define MAX_PID                     (MAX_PROCESS  2)
extern list_entry_t proc_list;
struct proc_struct {
    enum proc_state state;                      // Process state
    int pid;                                    // Process ID
    int runs;                                   // the running times of Proces
    uintptr_t kstack;                           // Process kernel stack
    volatile bool need_resched;                 // bool value: need to be rescheduled to release CPU?
    struct proc_struct parent;                  // the parent process
    struct mm_struct mm;                        // Process's memory management field
    struct context context;                     // Switch here to run process
    struct trapframe tf;                        // Trap frame for current interrupt
    uintptr_t cr3;                              // CR3 register: the base addr of Page Directroy Table(PDT)
    uint32_t flags;                             // Process flag
    char name[PROC_NAME_LEN + 1];               // Process name
    list_entry_t list_link;                     // Process link list
    list_entry_t hash_link;                     // Process hash list
    int exit_code;                              // exit code (be sent to parent proc)
    uint32_t wait_state;                        // waiting state
    struct proc_struct cptr, yptr, optr;     // relations between processes
    struct run_queue rq;                        // running queue contains Process
    list_entry_t run_link;                      // the entry linked in run queue
    int time_slice;                             // time slice for occupying the CPU
    skew_heap_entry_t lab6_run_pool;            // FOR LAB6 ONLY: the entry in the run pool
    uint32_t lab6_stride;                       // FOR LAB6 ONLY: the current stride of the process
    uint32_t lab6_priority;                     // FOR LAB6 ONLY: the priority of process, set by lab6_set_priority(uint32_t)
};
#define PF_EXITING                  0x00000001      // getting shutdown
#define WT_CHILD                    (0x00000001 | WT_INTERRUPTED)
#define WT_INTERRUPTED              0x80000000                    // the wait state could be interrupted
```

```
#define le2proc(le, member)         \
    to_struct((le), struct proc_struct, member)
extern struct proc_struct idleproc, initproc, current;
void proc_init(void);
void proc_run(struct proc_struct proc);
int kernel_thread(int (fn)(void ), void arg, uint32_t clone_flags);
char set_proc_name(struct proc_struct proc, const char name);
char get_proc_name(struct proc_struct proc);
void cpu_idle(void) attribute((noreturn));
struct proc_struct find_proc(int pid);
int do_fork(uint32_t clone_flags, uintptr_t stack, struct trapframe tf);
int do_exit(int error_code);
int do_yield(void);
int do_execve(const char name, size_t len, unsigned char binary, size_t size);
int do_wait(int pid, int code_store);
int do_kill(int pid);
void lab6_set_priority(uint32_t priority);
#endif / !KERN_PROCESS_PROC_H /
==============================================================
kern/schedule/default_sched.c
=============kern/schedule/default_sched.c=======================
#include
#include
#include
#include
#include
#define USE_SKEW_HEAP 1
/ You should define the BigStride constant here/
/ LAB6: YOUR CODE /
#define BIG_STRIDE    0x7FFFFFFF / ??? /
/ The compare function for two skew_heap_node_t's and the
  corresponding procs/
static int
proc_stride_comp_f(void a, void b)
{
     struct proc_struct p = le2proc(a, lab6_run_pool);
     struct proc_struct q = le2proc(b, lab6_run_pool);
     int32_t c = p->lab6_stride - q->lab6_stride;
     if (c > 0) return 1;
     else if (c == 0) return 0;
     else return -1;
}
/
  stride_init initializes the run-queue rq with correct assignment for
  member variables, including:
```

```
        - run_list: should be a empty list after initialization.
        - lab6_run_pool: NULL
        - proc_num: 0
        - max_time_slice: no need here, the variable would be assigned by the caller.

      hint: see proj13.1/libs/list.h for routines of the list structures.
     /
    static void
    stride_init(struct run_queue rq) {
         / LAB6: YOUR CODE /
         list_init(&(rq->run_list));
         rq->lab6_run_pool = NULL;
         rq->proc_num = 0;
    }
    /
      stride_enqueue inserts the process ``proc'' into the run-queue
      ``rq''. The procedure should verify/initialize the relevant members
      of ``proc'', and then put the ``lab6_run_pool'' node into the
      queue(since we use priority queue here). The procedure should also
      update the meta date in ``rq'' structure.

      proc->time_slice denotes the time slices allocation for the
      process, which should set to rq->max_time_slice.

      hint: see proj13.1/libs/skew_heap.h for routines of the priority
      queue structures.
     /
    static void
    stride_enqueue(struct run_queue rq, struct proc_struct proc) {
         / LAB6: YOUR CODE /
    #if USE_SKEW_HEAP
         rq->lab6_run_pool = ……(1)……;
    #else
         assert(list_empty(&(proc->run_link)));
         list_add_before(&(rq->run_list), &(proc->run_link));
```

```
#endif
     if (proc->time_slice == 0 || proc->time_slice > rq->max_time_slice) {
          proc->time_slice = rq->max_time_slice;
     }
     proc->rq = rq;
     rq->proc_num ++;
}
/
  stride_dequeue removes the process ``proc'' from the run-queue
  ``rq'', the operation would be finished by the skew_heap_remove
  operations. Remember to update the ``rq'' structure.

  hint: see proj13.1/libs/skew_heap.h for routines of the priority
  queue structures.
 /
static void
stride_dequeue(struct run_queue rq, struct proc_struct proc) {
     / LAB6: YOUR CODE /
#if USE_SKEW_HEAP
     rq->lab6_run_pool =  ……(2)……;
#else
     assert(!list_empty(&(proc->run_link)) && proc->rq == rq);
     list_del_init(&(proc->run_link));
#endif
     rq->proc_num --;
}
/
  stride_pick_next pick the element from the ``run-queue'', with the
  minimum value of stride, and returns the corresponding process
  pointer. The process pointer would be calculated by macro le2proc,
  see proj13.1/kern/process/proc.h for definition. Return NULL if
  there is no process in the queue.

  When one proc structure is selected, remember to update the stride
  property of the proc. (stride += BIG_STRIDE / priority)

  hint: see proj13.1/libs/skew_heap.h for routines of the priority
  queue structures.
 /
static struct proc_struct
stride_pick_next(struct run_queue rq) {
     / LAB6: YOUR CODE /
#if USE_SKEW_HEAP
     if (rq->lab6_run_pool == NULL) return NULL;
     struct proc_struct p = le2proc(rq->lab6_run_pool, lab6_run_pool);
#else
     list_entry_t le = list_next(&(rq->run_list));
     if (le == &rq-;>run_list)
          return NULL;
     struct proc_struct p = le2proc(le, run_link);
     le = list_next(le);
     while (le != &rq-;>run_list)
     {
          struct proc_struct q = le2proc(le, run_link);
          if ((int32_t)(p->lab6_stride - q->lab6_stride) > 0)
               p = q;
          le = list_next(le);
     }
#endif
     if (p->lab6_priority == 0)
          p->lab6_stride += BIG_STRIDE;
     else p->lab6_stride = ……(3)……;
     return p;
}
/
  stride_proc_tick works with the tick event of current process. You
  should check whether the time slices for current process is
  exhausted and update the proc struct ``proc''. proc->time_slice
  denotes the time slices left for current
  process. proc->need_resched is the flag variable for process
  switching.
 /
static void
stride_proc_tick(struct run_queue rq, struct proc_struct proc) {
     / LAB6: YOUR CODE /
     if (proc->time_slice > 0) {
          ……(4)……;
     }
     if (proc->time_slice == 0) {
          ……(5)……;
     }
}
struct sched_class default_sched_class = {
     .name = "stride_scheduler",
     .init = stride_init,
     .enqueue = ……(6)……,
     .dequeue = ……(7)……,
```

```
        .pick_next = ……(8)……,
        .proc_tick = ……(9)……,
    };
    ============================================================
    libs/skew_heap.h
    ==================libs/skew_heap.h============================
    #ifndef __LIBS_SKEW_HEAP_H__
    #define __LIBS_SKEW_HEAP_H__
    struct skew_heap_entry {
        struct skew_heap_entry parent, left, right;
    };
    typedef struct skew_heap_entry skew_heap_entry_t;
    typedef int(compare_f)(void a, void b);
    static inline void skew_heap_init(skew_heap_entry_t a) __attribute__((always_inline));
    static inline skew_heap_entry_t skew_heap_merge(
        skew_heap_entry_t a, skew_heap_entry_t b,
        compare_f comp);
    static inline skew_heap_entry_t skew_heap_insert(
        skew_heap_entry_t a, skew_heap_entry_t b,
        compare_f comp) __attribute__((always_inline));
    static inline skew_heap_entry_t skew_heap_remove(
        skew_heap_entry_t a, skew_heap_entry_t b,
        compare_f comp) __attribute__((always_inline));
    static inline void
    skew_heap_init(skew_heap_entry_t a)
    {
        a->left = a->right = a->parent = NULL;
    }
    static inline skew_heap_entry_t
    skew_heap_merge(skew_heap_entry_t a, skew_heap_entry_t b,
                    compare_f comp)
    {
        if (a == NULL) return b;
        else if (b == NULL) return a;
        skew_heap_entry_t l, r;
        if (comp(a, b) == -1)
        {
            r = a->left;
            l = skew_heap_merge(a->right, b, comp);
            a->left = l;
            a->right = r;
            if (l) l->parent = a;
            return a;
        }
        else
        {
            r = b->left;
            l = skew_heap_merge(a, b->right, comp);
            b->left = l;
            b->right = r;
            if (l) l->parent = b;
            return b;
        }
    }
    static inline skew_heap_entry_t
    skew_heap_insert(skew_heap_entry_t a, skew_heap_entry_t b,
                     compare_f comp)
    {
        skew_heap_init(b);
        return skew_heap_merge(a, b, comp);
    }
    static inline skew_heap_entry_t
    skew_heap_remove(skew_heap_entry_t a, skew_heap_entry_t b,
                     compare_f comp)
    {
        skew_heap_entry_t p   = b->parent;
        skew_heap_entry_t rep = skew_heap_merge(b->left, b->right, comp);
        if (rep) rep->parent = p;
        if (p)
        {
            if (p->left == b)
                p->left = rep;
            else p->right = rep;
            return a;
        }
        else return rep;
    }
    #endif  / !__LIBS_SKEW_HEAP_H__ /
    ============================================================
    ```
```

- [x]

知识点:处理机调度
出处:网络
难度:1
```

> 基本原理7分；填空9分；斜堆2分；
> *基本原理*
> 1）
> Tickets: Abstract, relative, and uniform resource rights（2分）
> Strides (stride = stride1 / tickets): Intervals between selections（2分）
> Passes (pass += stride)（2分）
> ? Virtual time index for next selection
> ? Clients with smallest pass gets selected（1分）
> 2）
> rq->lab6_run_pool = ……(1)……;
> rq->lab6_run_pool =
> skew_heap_insert(rq->lab6_run_pool, &(proc->lab6_run_pool),
> proc_stride_comp_f);
> rq->lab6_run_pool = ……(2)……;
> rq->lab6_run_pool =
> skew_heap_remove(rq->lab6_run_pool, &(proc->lab6_run_pool),
> proc_stride_comp_f);
> else p->lab6_stride = ……(3)……;
> else p->lab6_stride += BIG_STRIDE / p->lab6_priority;
> if (proc->time_slice > 0) {
> ……(4)……;
> }
> if (proc->time_slice == 0) {
> ……(5)……;
> }
> if (proc->time_slice > 0) {
> proc->time_slice --;
> }
> if (proc->time_slice == 0) {
> proc->need_resched = 1;
> }
> struct sched_class default_sched_class = {
> .name = "stride_scheduler",
> .init = stride_init,
> .enqueue = ……(6)……,
> .dequeue = ……(7)……,
> .pick_next = ……(8)……,
> .proc_tick = ……(9)……,
> };
> .enqueue = stride_enqueue,
> .dequeue = stride_dequeue,
> .pick_next = stride_pick_next,
> .proc_tick = stride_proc_tick,
> 3）就绪线程形成一个树状结构，根优先级（pass）最小（1分）；按斜堆的规则进行就绪线程的插入和删除（1分）；
> 4
> (15分)公平的读者-写者（Reader-Writer
> Problem）问题是指，多个读者进程（Reader）与多个写者进程（Writer）共享一个数据区；读者进程和写者进程对共享数据区的访问满足下列条件。
> 1）多个读者进程可以同时对共享数据区进行访问；
> 2）多个写者进程只能对共享数据区进行互斥访问；
> 3）读者进程与写者进程只能对共享数据区进行互斥访问；
> 4）当有写者进程等待时，其后到达的读者进程不能先于该写者进程对共享数据区进行访问；
> 5）当有读者进程等待时，其后到达的写者进程不能先于该读者进程对共享数据区进行访问；
> 试用信号量机制实现读者进程Reader（）和写者进程
> Writer（）。要求：用信号量方法（不允许使用信号量集），并给出信号量定义和初始值；在代码中要有适当的注释，以说明信号量定义的作用和代码的含义；用类 C
> 语言描述共享变量和函数。

- [x]

知识点:同步互斥
出处:网络
难度:1

> 三个信号量，共13分（初值1分，共3分；两对mutex，每对2分；两对
> rcount_mutex，每对1分；两对waiter_mutex，每对2分）；计数变量2分（条件1分，加一和减一1分）；
> 只会配对使用PV原语，给4分；

```
var
waiter_mutex, mutex, rcount_mutex:semaphore;
reader_counter:integer;
waiter_mutex,mutex,rcount_mutex:=1;
reader_counter:=0;
cobegin
Procedure Reader
begin
while TRUE
{
p(waiter_mutex);
p(rcount_mutex); (没有这一句会导致reader_counter读和写冲突)
```

```
if reader_counter ==0 then
p(mutex);
reader_counter:=reader_counter+1;
v(rcount_mutex);
v(waiter_mutex);
Reading;
p(rcount_mutex);
reader_counter:=reader_counter-1;
if reader_counter==0 then
v(mutex);
v(rcount_mutex);
};
end
Procedure Writer
begin
while TRUE
{
p(waiter_mutex);
p(mutex);
Writing;
v(mutex);
v(waiter_mutex);（这一句放在Writing的前面好像也行。我不确信。）
}
coend
```

4
(8分)某计算机系统中有18个同类型共享资源，有K个进程竞争使用，每个进程最多需要3个共享资源。该系统不会发生死锁的K的最大值是多少？要求给出计算过程，并说
明理由。

- [x]

知识点:同步互斥
出处:网络
难度:1

结果3分，计算过程3分，理由2分；
不死锁需要2K+1<18（3分）；
理由是，不会出现所有进程都只占用2个资源的死锁情况（2分）；
所以 K=8（3分）
4
(8分)给出下面程序fork-example.cpp的输出结果；

```
====================fork-example.cpp==========================
#include
#include
#include
#include
#include
using namespace std;
int globalVariable = 2;
main()
{
    string sIdentifier;
    int    iStackVariable = 20;
    pid_t pID = fork();
    if (pID == 0)
    {
      sIdentifier = "Child Process: ";
      globalVariable++;
      iStackVariable++;
     }
    else if (pID < 0)
    {
       cerr << "Failed to fork" << endl;
       exit(1);
    }
    else
    {
      sIdentifier = "Parent Process:";
    }
    cout << sIdentifier;
    cout << " Global variable: " << globalVariable;
    cout << " Stack variable: "  << iStackVariable << endl;
}
=============================================================
```

- [x]

知识点:进程状态与控制
出处:网络
难度:1

8分，六个点（4个数每个1.5分），父和子每个1分；
Parent Process: Global variable: 2 Stack variable: 20
Child Process: Global variable: 3 Stack variable: 21

```
fork-example.cpp
====================fork-example.cpp=========================
#include
#include
// Required by for routine
#include
#include
#include    // Declaration for exit()
using namespace std;
int globalVariable = 2;
main()
{
    string sIdentifier;
    int    iStackVariable = 20;
    pid_t pID = fork();
    if (pID == 0)                // child
    {
        // Code only executed by child process
        sIdentifier = "Child Process: ";
        globalVariable++;
        iStackVariable++;
    }
    else if (pID < 0)            // failed to fork
    {
        cerr << "Failed to fork" << endl;
        exit(1);
        // Throw exception
    }
    else                                   // parent
    {
        // Code only executed by parent process
        sIdentifier = "Parent Process:";
    }
    // Code executed by both parent and child.
    cout << sIdentifier;
    cout << " Global variable: " << globalVariable;
    cout << " Stack variable: "  << iStackVariable << endl;
}
============================================================
```

4
(16分)下面是ucore内核中与yield()系统调用实现相关源代码，可实现用户线程主动放弃CPU使用权的功能。
1）试描述ucore中用户进程利用yield()进行主动让出CPU的工作过程；
2）请补全其中所缺的代码，以正确完成从用户态函数yield()的功能。提示：每处需要补全的代码最少只需要一行，一共有11个空要填。当然，你可以在需要补全代码
的地方写多行来表达需要实现的功能，也允许修改已给出的代码。

```
libs-user-ucore/syscall.h
================ libs-user-ucore/syscall.h ======================
#ifndef __USER_LIBS_SYSCALL_H__
#define __USER_LIBS_SYSCALL_H__
#include
......
int sys_yield(void);
......
#endif / !__USER_LIBS_SYSCALL_H__ /
============================================================
libs-user-ucore/arch/i386/syscall.c
=============libs-user-ucore/arch/i386/syscall.c====================
#include
#include
#include
#include
#include
#include
#include
#define MAX_ARGS            5
uint32_t
syscall(int num, ...) {
    va_list ap;
    va_start(ap, num);
    uint32_t a[MAX_ARGS];
    int i;
    for (i = 0; i < MAX_ARGS; i ++) {
        a[i] = va_arg(ap, uint32_t);
    }
    va_end(ap);
```

```
        uint32_t ret;
        asm volatile (
            "int %1;"
            : "=a" (ret)
            : "i" (T_SYSCALL),
              "a" (num),
              "d" (a[0]),
              "c" (a[1]),
              "b" (a[2]),
              "D" (a[3]),
              "S" (a[4])
            : "cc", "memory");
        return ret;
    }
    ==============================================================
    libs-user-ucore/syscall.c
    ==================libs-user-ucore/syscall.c=======================
    #include
    #include
    #include
    #include
    #include
    #include
    #include
    extern uintptr_t syscall (int num, ...);
    ......
    int
    sys_yield(void) {
        return ……(1)……;
    }
    ......
    ==============================================================
    kern-ucore/glue-ucore/libs/unistd.h
    =============kern-ucore/glue-ucore/libs/unistd.h===================
    #ifndef __LIBS_UNISTD_H__
    #define __LIBS_UNISTD_H__
    #define T_SYSCALL           0x80
    / syscall number /
    ......
    #define SYS_yield           10
    ......
    #endif / !__LIBS_UNISTD_H__ /
    ==============================================================
    kern-ucore/arch/i386/glue-ucore/trap.c
    ============= kern-ucore/arch/i386/glue-ucore/trap.c ===============
    ......
    static void
    trap_dispatch(struct trapframe tf) {
        char c;
        int ret;
        switch (tf->tf_trapno) {
        case T_DEBUG:
        case T_BRKPT:
            debug_monitor(tf);
            break;
        case T_PGFLT:
            if ((ret = pgfault_handler(tf)) != 0) {
                print_trapframe(tf);
                if (pls_read(current) == NULL) {
                    panic("handle pgfault failed. %e
", ret);
                }
                else {
                    if (trap_in_kernel(tf)) {
                        panic("handle pgfault failed in kernel mode. %e
", ret);
                    }
                    kprintf("killed by kernel.
");
                    do_exit(-E_KILLED);
                }
            }
            break;
        case ……(2)……:
```

```
            syscall();
            break;
        case IRQ_OFFSET + IRQ_TIMER:
            ticks ++;
            assert(pls_read(current) != NULL);
            run_timer_list();
            break;
        case IRQ_OFFSET + IRQ_COM1:
        case IRQ_OFFSET + IRQ_KBD:
            if ((c = cons_getc()) == 13) {
                debug_monitor(tf);
            }
            else {
                extern void dev_stdin_write(char c);
                dev_stdin_write(c);
            }
            break;
        case IRQ_OFFSET + IRQ_IDE1:
        case IRQ_OFFSET + IRQ_IDE2:
            / do nothing /
            break;
        default:
            print_trapframe(tf);
            if (pls_read(current) != NULL) {
                kprintf("unhandled trap.
");
                do_exit(-E_KILLED);
            }
            panic("unexpected trap in kernel.
");
        }
    }
    void
    trap(struct trapframe tf) {
        // used for previous projects
        if (pls_read(current) == NULL) {
            trap_dispatch(tf);
        }
        else {
            // keep a trapframe chain in stack
            struct trapframe otf = pls_read(current)->tf;
            pls_read(current)->tf = tf;
            bool in_kernel = trap_in_kernel(tf);
            trap_dispatch(tf);
            pls_read(current)->tf = otf;
            if (!in_kernel) {
                may_killed();
                if (pls_read(current)->need_resched) {
                    ……(3)……;
                }
            }
        }
    }
    ============================================================
    kern-ucore/schedule/sched.c
    =============kern-ucore/schedule/sched.c=========================
    #include
    #include
    #include
    #include
    #include
    #include
    #include
    #include
    #include
    #define current (pls_read(current))
    #define idleproc (pls_read(idleproc))
    ......
    #include
    #define MT_SUPPORT
    void
    schedule(void) {
        bool intr_flag;
        struct proc_struct next;
```

```
    #ifndef MT_SUPPORT
        list_entry_t head;
        int lapic_id = pls_read(lapic_id);
    #endif
        local_intr_save(intr_flag);
        int lcpu_count = pls_read(lcpu_count);
        {
            current->need_resched = ……(4)……;
    #ifndef MT_SUPPORT
            if (current->mm)
            {
                assert(current->mm->lapic == lapic_id);
                current->mm->lapic = -1;
            }
    #endif
            if (current->state == PROC_RUNNABLE && current->pid >= lcpu_count) {
                sched_class_enqueue(current);
            }
    #ifndef MT_SUPPORT
            list_init(&head;);
            while (1)
            {
                next = ……(5)……;
                if (next != NULL) sched_class_dequeue(next);
                if (next && next->mm && next->mm->lapic != -1)
                {
                    list_add(&head;, &(next->run_link));
                }
                else
                {
                    list_entry_t cur;
                    while ((cur = list_next(&head;)) != &head;)
                    {
                        list_del_init(cur);
                        sched_class_enqueue(le2proc(cur, run_link));
                    }
                    break;
                }
            }
    #else
            next = ……(6)……;
            if (next != NULL)
                sched_class_dequeue(next);
    #endif  / !MT_SUPPORT /
            if (next == NULL) {
                next = ……(7)……;
            }
            next->runs ++;
            / Collect information here/
            if (sched_collect_info) {
                int lcpu_count = pls_read(lcpu_count);
                int lcpu_idx = pls_read(lcpu_idx);
                int loc = sched_info_head[lcpu_idx];
                int prev = sched_info_pid[loclcpu_count + lcpu_idx];
                if (next->pid == prev)
                    sched_info_times[loclcpu_count + lcpu_idx] ++;
                else {
                    sched_info_head[lcpu_idx] ++;
                    if (sched_info_head[lcpu_idx] >= PGSIZE / sizeof(uint16_t) / lcpu_count)
                        sched_info_head[lcpu_idx] = 0;
                    loc = sched_info_head[lcpu_idx];
                    uint16_t prev_pid = sched_info_pid[loclcpu_count + lcpu_idx];
                    uint16_t prev_times = sched_info_times[loclcpu_count + lcpu_idx];
                    if (prev_times > 0 && prev_pid >= lcpu_count + 2)
                        sched_slices[lcpu_idx][prev_pid % SLICEPOOL_SIZE] += prev_times;
                    sched_info_pid[loclcpu_count + lcpu_idx] = next->pid;
                    sched_info_times[loclcpu_count + lcpu_idx] = 1;
                }
            }
    #ifndef MT_SUPPORT
            assert(!next->mm || next->mm->lapic == -1);
            if (next->mm)
                next->mm->lapic = lapic_id;
    #endif
```

```
            if (next != current) {
                ……(8)……;
            }
        }
        local_intr_restore(intr_flag);
    }
    void
    add_timer(timer_t timer) {
        bool intr_flag;
        local_intr_save(intr_flag);
        {
            assert(timer->expires > 0 && timer->proc != NULL);
            assert(list_empty(&(timer->timer_link)));
            list_entry_t le = list_next(&timer;_list);
            while (le != &timer;_list) {
                timer_t next = le2timer(le, timer_link);
                if (timer->expires < next->expires) {
                    next->expires -= timer->expires;
                    break;
                }
                timer->expires -= next->expires;
                le = list_next(le);
            }
            list_add_before(le, &(timer->timer_link));
        }
        local_intr_restore(intr_flag);
    }
    ......
    ============================================================
    kern-ucore/process/proc.c
    ================kern-ucore/process/proc.c=======================
    ......
    // proc_run - make process "proc" running on cpu
    // NOTE: before call switch_to, should load  base addr of "proc"'s new PDT
    void
    proc_run(struct proc_struct proc) {
        if (proc != current) {
            bool intr_flag;
            struct proc_struct prev = current, next = proc;
            // kprintf("(%d) => %d
", lapic_id, next->pid);
            local_intr_save(intr_flag);
            {
                pls_write(current, proc);
                load_rsp0(next->kstack + KSTACKSIZE);
                mp_set_mm_pagetable(next->mm);
                ……(9)……;
            }
            local_intr_restore(intr_flag);
        }
    }
    ......
    // do_yield - ask the scheduler to reschedule
    int
    do_yield(void) {
        current->need_resched = ……(10)……;
        return 0;
    }
    ......
    ============================================================
    kern-ucore/arch/i386/syscall/syscall.c
    =============kern-ucore/arch/i386/syscall/syscall.c=================
    ......
    static uint32_t
    sys_yield(uint32_t arg[]) {
        return ……(11)……;
    }
    ......
    static uint32_t (syscalls[])(uint32_t arg[]) = {
    ......
        [SYS_yield]             sys_yield,
    ......
    };
    #define NUM_SYSCALLS        ((sizeof(syscalls)) / (sizeof(syscalls[0])))
```

```
    void
    syscall(void) {
        struct trapframe tf = pls_read(current)->tf;
        uint32_t arg[5];
        int num = tf->tf_regs.reg_eax;
        if (num >= 0 && num < NUM_SYSCALLS) {
            if (syscalls[num] != NULL) {
                arg[0] = tf->tf_regs.reg_edx;
                arg[1] = tf->tf_regs.reg_ecx;
                arg[2] = tf->tf_regs.reg_ebx;
                arg[3] = tf->tf_regs.reg_edi;
                arg[4] = tf->tf_regs.reg_esi;
                tf->tf_regs.reg_eax = syscalls[num](arg);
                return ;
            }
        }
        print_trapframe(tf);
        panic("undefined syscall %d, pid = %d, name = %s.
",
                num, pls_read(current)->pid, pls_read(current)->name);
    }
    ==========================================================
```

- [x]

知识点:进程状态与控制
出处:网络
难度:1

第一问5分；第二问11分；
1）yield()的工作过程：（1）设置调度标志need_sched（2分）；（2）在系统调用返回时检查调度标志，并进行线程切换（2分）；（3）再次调度yi
eld()所在线程继续执行时返回用户态（1分）；
2)

```
return ……(1)……;
return syscall(SYS_yield);
case ……(2)……:
    case T_SYSCALL:
if (pls_read(current)->need_resched) {
                ……(3)……;
            }
        if (pls_read(current)->need_resched) {
                schedule();
            }
current->need_resched = ……(4)……;
current->need_resched = 0;
next = ……(5)……;
next = sched_class_pick_next();
next = ……(6)……;
next = sched_class_pick_next();
if (next == NULL) {
            next = ……(7)……;
        }
if (next == NULL) {
            next = idleproc;
        }
if (next != current) {
            ……(8)……;
        }
if (next != current) {
            proc_run(next);
        }
mp_set_mm_pagetable(next->mm);
            ……(9)……;
            mp_set_mm_pagetable(next->mm);
            switch_to(&(prev->context), &(next->context));
current->need_resched = ……(10)……;
current->need_resched = 1;
return ……(11)……;
return do_yield();
```

4
(18分)文件系统是操作系统内核中用于持久保存数据的功能模块。
1）试描述SFS文件系统中的文件存储组织，即文件内部数据块存储位置和顺序的组织方法；
2）试描述ucore文件系统在一个SFS文件的最后附加一个新数据块实现方法；
3）试解释下面 ucore代码中文件系统实现中与append_block()函数相关的指定代码行的作用。注意：需要解释的代码共有12处。
`kern/fs/sfs/sfs.h`

```
    ======================kern/fs/sfs/sfs.h=======================
    #ifndef __KERN_FS_SFS_SFS_H__
    #define __KERN_FS_SFS_SFS_H__
```

```
#include
#include
#include
#include
#include
#define SFS_MAGIC           0x2f8dbe2a              / magic number for sfs /
#define SFS_BLKSIZE         PGSIZE                  / size of block /
#define SFS_NDIRECT         12                      / # of direct blocks in inode /
#define SFS_MAX_INFO_LEN    31                      / max length of infomation /
#define SFS_MAX_FNAME_LEN   FS_MAX_FNAME_LEN        / max length of filename /
#define SFS_MAX_FILE_SIZE   (1024UL  1024  128)   / max file size (128M) /
#define SFS_BLKN_SUPER      0                   / block the superblock lives in /
#define SFS_BLKN_ROOT       1                   / location of the root dir inode /
#define SFS_BLKN_FREEMAP    2                   / 1st block of the freemap /
/ # of bits in a block /
#define SFS_BLKBITS                                 (SFS_BLKSIZE  CHAR_BIT)
/ # of entries in a block /
#define SFS_BLK_NENTRY                              (SFS_BLKSIZE / sizeof(uint32_t))
/ file types /
#define SFS_TYPE_INVAL                              0       / Should not appear on disk /
#define SFS_TYPE_FILE                               1
#define SFS_TYPE_DIR                                2
#define SFS_TYPE_LINK                               3
/
  On-disk superblock
 /
struct sfs_super {
    uint32_t magic;                                 / magic number, should be SFS_MAGIC /
    uint32_t blocks;                                / # of blocks in fs /
    uint32_t unused_blocks;                         / # of unused blocks in fs /
    char info[SFS_MAX_INFO_LEN + 1];                / infomation for sfs  /
};
/ inode (on disk) /
struct sfs_disk_inode {
    uint32_t size;                                  / size of the file (in bytes) /
    uint16_t type;                                  / one of SYS_TYPE_ above /
    uint16_t nlinks;                                / # of hard links to this file /
    uint32_t blocks;                                / ……(1)…… /
    uint32_t direct[SFS_NDIRECT];                   / ……(2)…… /
    uint32_t indirect;                              / ……(3)…… /
//    uint32_t db_indirect;                           / double indirect blocks /
//   unused
};
/ file entry (on disk) /
struct sfs_disk_entry {
    uint32_t ino;                                   / inode number /
    char name[SFS_MAX_FNAME_LEN + 1];               / file name /
};
#define sfs_dentry_size                             \
    sizeof(((struct sfs_disk_entry )0)->name)
/ inode for sfs /
struct sfs_inode {
    struct sfs_disk_inode din;                    / on-disk inode /
    uint32_t ino;                                   / inode number /
    bool dirty;                                     / true if inode modified /
    int reclaim_count;                              / kill inode if it hits zero /
    semaphore_t sem;                                / semaphore for din /
    list_entry_t inode_link;         / entry for linked-list in sfs_fs /
    list_entry_t hash_link;          / entry for hash linked-list in sfs_fs /
};
#define le2sin(le, member)                          \
    to_struct((le), struct sfs_inode, member)
/ filesystem for sfs /
struct sfs_fs {
    struct sfs_super super;                         / on-disk superblock /
    struct device dev;                             / device mounted on /
    struct bitmap freemap;                         / blocks in use are mared 0 /
    bool super_dirty;                               / true if super/freemap modified /
    void sfs_buffer;                  / buffer for non-block aligned io /
    semaphore_t fs_sem;                             / semaphore for fs /
    semaphore_t io_sem;                             / semaphore for io /
    semaphore_t mutex_sem;              / semaphore for link/unlink and rename /
    list_entry_t inode_list;                        / inode linked-list /
    list_entry_t hash_list;                        / inode hash linked-list /
```

```
};
/ hash for sfs /
#define SFS_HLIST_SHIFT                 10
#define SFS_HLIST_SIZE                  (1 << SFS_HLIST_SHIFT)
#define sin_hashfn(x)                   (hash32(x, SFS_HLIST_SHIFT))
/ size of freemap (in bits) /
#define sfs_freemap_bits(super)         ROUNDUP((super)->blocks, SFS_BLKBITS)
/ size of freemap (in blocks) /
#define sfs_freemap_blocks(super)       ROUNDUP_DIV((super)->blocks, SFS_BLKBITS)
struct fs;
struct inode;
void sfs_init(void);
int sfs_mount(const char devname);
void lock_sfs_fs(struct sfs_fs sfs);
void lock_sfs_io(struct sfs_fs sfs);
void lock_sfs_mutex(struct sfs_fs sfs);
void unlock_sfs_fs(struct sfs_fs sfs);
void unlock_sfs_io(struct sfs_fs sfs);
void unlock_sfs_mutex(struct sfs_fs sfs);
int sfs_rblock(struct sfs_fs sfs, void buf, uint32_t blkno, uint32_t nblks);
int sfs_wblock(struct sfs_fs sfs, void buf, uint32_t blkno, uint32_t nblks);
int sfs_rbuf(struct sfs_fs sfs, void buf, size_t len, uint32_t blkno, off_t offset);
int sfs_wbuf(struct sfs_fs sfs, void buf, size_t len, uint32_t blkno, off_t offset);
int sfs_sync_super(struct sfs_fs sfs);
int sfs_sync_freemap(struct sfs_fs sfs);
int sfs_clear_block(struct sfs_fs sfs, uint32_t blkno, uint32_t nblks);
int sfs_load_inode(struct sfs_fs sfs, struct inode node_store, uint32_t ino);
#endif / !__KERN_FS_SFS_SFS_H__ /
=============================================================
tools/mksfs.c
====================== tools/mksfs.c==========================
......
#define SFS_MAGIC                       0x2f8dbe2a
#define SFS_NDIRECT                     12
#define SFS_BLKSIZE                     4096                    // 4K
#define SFS_MAX_NBLKS                   (1024UL  512)           // 4K  512K
#define SFS_MAX_INFO_LEN                31
#define SFS_MAX_FNAME_LEN               255
#define SFS_MAX_FILE_SIZE               (1024UL  1024  128)     // 128M
#define SFS_BLKBITS                     (SFS_BLKSIZE  CHAR_BIT)
#define SFS_TYPE_FILE                   1
#define SFS_TYPE_DIR                    2
#define SFS_TYPE_LINK                   3
#define SFS_BLKN_SUPER                  0
#define SFS_BLKN_ROOT                   1
#define SFS_BLKN_FREEMAP                2
struct cache_block {
    uint32_t ino;
    struct cache_block hash_next;
    void cache;
};
struct cache_inode {
    struct inode {
        uint32_t size;
        uint16_t type;
        uint16_t nlinks;
        uint32_t blocks;
        uint32_t direct[SFS_NDIRECT];
        uint32_t indirect;
        uint32_t db_indirect;
    } inode;
    ino_t real;
    uint32_t ino;
    uint32_t nblks;
    struct cache_block l1, l2;
    struct cache_inode hash_next;
};
struct sfs_fs {
    struct {
        uint32_t magic;
        uint32_t blocks;
        uint32_t unused_blocks;
        char info[SFS_MAX_INFO_LEN + 1];
    } super;
```

```
        struct subpath {
            struct subpath next, prev;
            char subname;
        } __sp_nil, sp_root, sp_end;
        int imgfd;
        uint32_t ninos, next_ino;
        struct cache_inode root;
        struct cache_inode inodes[HASH_LIST_SIZE];
        struct cache_block blocks[HASH_LIST_SIZE];
    };
    struct sfs_entry {
        uint32_t ino;
        char name[SFS_MAX_FNAME_LEN + 1];
    };
    static uint32_t
    sfs_alloc_ino(struct sfs_fs sfs) {
        if (sfs->next_ino < sfs->ninos) {
            sfs->super.unused_blocks --;
            return sfs->next_ino ++;
        }
        bug("out of disk space.
");
    }
    ......
    #define show_fullpath(sfs, name) subpath_show(stderr, sfs, name)
    void open_dir(struct sfs_fs sfs, struct cache_inode current, struct cache_inode parent);
    void open_file(struct sfs_fs sfs, struct cache_inode file, const char filename, int fd);
    void open_link(struct sfs_fs sfs, struct cache_inode file, const char filename);
    #define SFS_BLK_NENTRY                          (SFS_BLKSIZE / sizeof(uint32_t))
    #define SFS_L0_NBLKS                            SFS_NDIRECT
    #define SFS_L1_NBLKS                            (SFS_BLK_NENTRY + SFS_L0_NBLKS)
    #define SFS_L2_NBLKS                            (SFS_BLK_NENTRY  SFS_BLK_NENTRY + SFS_L1_NBLKS)
    #define SFS_LN_NBLKS                            (SFS_MAX_FILE_SIZE / SFS_BLKSIZE)
    static void
    update_cache(struct sfs_fs sfs, struct cache_block cbp, uint32_t inop) {
        uint32_t ino = inop;
        struct cache_block cb = cbp;
        if (ino == 0) {
            cb = alloc_cache_block(sfs, 0);
            ino = cb->ino;
        }
        else if (cb == NULL || cb->ino != ino) {
            cb = search_cache_block(sfs, ino);
            assert(cb != NULL && cb->ino == ino);
        }
        cbp = cb, inop = ino;
    }
    static void
    append_block(struct sfs_fs sfs, struct cache_inode file, size_t size, uint32_t ino, const char filename)
{
        static_assert(SFS_LN_NBLKS <= SFS_L2_NBLKS);
        assert(size <= SFS_BLKSIZE);
        uint32_t nblks = file->nblks;
        struct inode inode = &(file->inode);
        if (nblks >= SFS_LN_NBLKS) {
            open_bug(sfs, filename, "file is too big.
");
        }
        if (nblks < SFS_L0_NBLKS) { / ……(4)…… /
            inode->direct[nblks] = ino;  / ……(5)…… /
        }
        else if (nblks < SFS_L1_NBLKS) {     / ……(6)…… /
            nblks -= SFS_L0_NBLKS;        / ……(7)…… /
            update_cache(sfs, &(file->l1), &(inode->indirect));
            uint32_t data = file->l1->cache;
            data[nblks] = ino;            / ……(8)…… /
        }
        else if (nblks < SFS_L2_NBLKS) {     / ……(9)…… /
            nblks -= SFS_L1_NBLKS;        / ……(10)…… /
            update_cache(sfs, &(file->l2), &(inode->db_indirect));
            uint32_t data2 = file->l2->cache;
            update_cache(sfs, &(file->l1), &data2;[nblks / SFS_BLK_NENTRY]);
            uint32_t data1 = file->l1->cache;
            data1[nblks % SFS_BLK_NENTRY] = ino; / ……(11)…… /
```

```
        }
        file->nblks ++;
        inode->size += size;
        inode->blocks ++;     / ……(12)…… /
    }
    ......
    ============================================================
```

- [x]

知识点:文件系统
出处:网络
难度:1

第1问6分；第2问6分；第3问6分；
1）文件内部数据块存储位置和顺序的组织方法
（3分）前12块的数据块编号组成一个直接索引数组，存于"uint32_t direct[SFS_NDIRECT]; / direct blocks /"
（3分）后面的数据块编号组成一个一级索引数组，指向该索引的指针为"uint32_t indirect; / indirect blocks /"
2）在一个SFS文件的最后附加一个新数据块实现方法：
通过判断数据块编号是在直接索引、一级索引还是二级索引（3分）；在相应数组元素处填入数据块编号（3分）；
3）每两个空1分；

```
/ inode (on disk) /
struct sfs_disk_inode {
    uint32_t size;                                  / size of the file (in bytes) /
    uint16_t type;                                  / one of SYS_TYPE_ above /
    uint16_t nlinks;                                / # of hard links to this file /
    uint32_t blocks;                                / ……(1)……文件占用数据块数 /
    uint32_t direct[SFS_NDIRECT];                   / ……(2)……直接索引数组 /
    uint32_t indirect;                              / ……(3)……1级索引指针 /
//    uint32_t db_indirect;                           / double indirect blocks /
//   unused
};
    if (nblks < SFS_L0_NBLKS) {    / ……(4)……最后一个数据块序号位于直接索引块 /
        inode->direct[nblks] = ino;    / ……(5)……将最后一个数据块的序号存入对应直接索引数组元素中 /
    }
    else if (nblks < SFS_L1_NBLKS) {   / ……(6)…… 最后一个数据块序号位于1级索引块  /
        nblks -= SFS_L0_NBLKS;     / ……(7)……计算1级索引数组下标 /
        update_cache(sfs, &(file->l1), &(inode->indirect));
        uint32_t data = file->l1->cache;
        data[nblks] = ino;         / ……(8)……将最后一个数据块的序号存入对应1级索引数组元素中/
    }
    else if (nblks < SFS_L2_NBLKS) {   / ……(9)…… 最后一个数据块序号位于2级索引块/
        nblks -= SFS_L1_NBLKS;     / ……(10)……计算2级索引数组下标/
        update_cache(sfs, &(file->l2), &(inode->db_indirect));
        uint32_t data2 = file->l2->cache;
        update_cache(sfs, &(file->l1), &data2;[nblks / SFS_BLK_NENTRY]);
        uint32_t data1 = file->l1->cache;
        data1[nblks % SFS_BLK_NENTRY] = ino;   / ……(11)…… 将最后一个数据块的序号存入对应2级索引数组中第2级子数组元素中/
    }
    file->nblks ++;
    inode->size += size;
    inode->blocks ++;  / ……(12)……文件数据占用的数据块总数加1 /
}
```

4
(6分)设文件F1的当前引用计数值为1，先建立F1的符号链接（软链接）文件F2，再建立F1的硬链接文件F3，然后删除F1。此时，F2和F3的引用计数值分别是
多少？要求说明理由。

- [x]

知识点:文件系统
出处:网络
难度:1

每个3分；建立符号链接不影响引用计数（1分），于是F2引用计数值是1（2分）；F3与F1指向同一文件，删除F1导致引用计数值减1（1分），
F3的引用计数值是1（2分）；
4
(11分)I/O子系统是操作系统中负责计算机系统与外界进行信息交互功能。键盘和显示器是计算机系统中最基本的I/O设备。
1）试描述ucore内核中是如何实现命令行状态的键盘输入时屏幕回显的；
2）试解释下面与I/O子系统中指定代码行的作用。注意：需要解释的代码共有10处。

```
    kern-ucore/arch/i386/driver/console.c
    ============ kern-ucore/arch/i386/driver/console.c=================
    #include
    #include
    #include
    #include
    #include
    #include
```

```
    #include
    #include
    #include
    #include
    / stupid I/O delay routine necessitated by historical PC design flaws /
    static void
    delay(void) {
        inb(0x84);
        inb(0x84);
        inb(0x84);
        inb(0x84);
    }
    ......
    static uint16_t crt_buf;
    static uint16_t crt_pos;
    static uint16_t addr_6845;
    / TEXT-mode CGA/VGA display output /
    static void
    cga_init(void) {
        volatile uint16_t cp = (uint16_t )(CGA_BUF + KERNBASE);
        uint16_t was = cp;
        cp = (uint16_t) 0xA55A;
        if (cp != 0xA55A) {
            cp = (uint16_t)(MONO_BUF + KERNBASE);
            addr_6845 = MONO_BASE;
        } else {
            cp = was;
            addr_6845 = CGA_BASE;
        }
        // Extract cursor location
        uint32_t pos;
        outb(addr_6845, 14);
        pos = inb(addr_6845 + 1) << 8; / ……(1)…… /
        outb(addr_6845, 15);
        pos |= inb(addr_6845 + 1); / ……(2)…… /
        crt_buf = (uint16_t) cp; / ……(3)…… /
        crt_pos = pos;
    }
    static bool serial_exists = 0;
    static void
    serial_init(void) {
    ......
    }
    ......
    / cga_putc - print character to console /
    static void
    cga_putc(int c) {
        // set black on white
        if (!(c & ~0xFF)) {
            c |= 0x0700;
        }
        switch (c & 0xff) {
        case '□':
            if (crt_pos > 0) {
                crt_pos --;
                crt_buf[crt_pos] = (c & ~0xff) | ' ';
            }
            break;
        case '
':
            crt_pos += CRT_COLS;
        case '
':
            crt_pos -= (crt_pos % CRT_COLS);
            break;
        default:
            crt_buf[crt_pos ++] = c;     // write the character
            break;
        }
        // What is the purpose of this?
        if (crt_pos >= CRT_SIZE) {
            int i;
            memmove(crt_buf, crt_buf + CRT_COLS, (CRT_SIZE - CRT_COLS)  sizeof(uint16_t));
            for (i = CRT_SIZE - CRT_COLS; i < CRT_SIZE; i ++) {
```

```
                crt_buf[i] = 0x0700 | ' ';
            }
            crt_pos -= CRT_COLS;
        }
        // move that little blinky thing
        outb(addr_6845, 14);
        outb(addr_6845 + 1, crt_pos >> 8);
        outb(addr_6845, 15);
        outb(addr_6845 + 1, crt_pos);
    }
    ......
    /
      Here we manage the console input buffer, where we stash characters
      received from the keyboard or serial port whenever the corresponding
      interrupt occurs.
      /
    #define CONSBUFSIZE 512
    static struct {
        uint8_t buf[CONSBUFSIZE];
        uint32_t rpos;
        uint32_t wpos;
    } cons;
    /
      cons_intr - called by device interrupt routines to feed input
      characters into the circular console input buffer.
      /
    static void
    cons_intr(int (proc)(void)) {
        int c;
        while ((c = (proc)()) != -1) {
            if (c != 0) {
                cons.buf[cons.wpos ++] = c;  / ……(4)…… /
                if (cons.wpos == CONSBUFSIZE) {
                    cons.wpos = 0;   / ……(5)…… /
                }
            }
        }
    }
    / serial_proc_data - get data from serial port /
    static int
    serial_proc_data(void) {
        if (!(inb(COM1 + COM_LSR) & COM_LSR_DATA)) {
            return -1;
        }
        int c = inb(COM1 + COM_RX);
        if (c == 127) {
            c = '□';
        }
        return c;
    }
    / serial_intr - try to feed input characters from serial port /
    void
    serial_intr(void) {
        if (serial_exists) {
            cons_intr(serial_proc_data);
        }
    }
    / Keyboard input code /
    #define NO              0
    #define SHIFT           (1<<0)
    #define CTL             (1<<1)
    #define ALT             (1<<2)
    #define CAPSLOCK        (1<<3)
    #define NUMLOCK         (1<<4)
    #define SCROLLLOCK      (1<<5)
    #define E0ESC           (1<<6)
    static uint8_t shiftcode[256] = {
        [0x1D] CTL,
        [0x2A] SHIFT,
        [0x36] SHIFT,
        [0x38] ALT,
        [0x9D] CTL,
        [0xB8] ALT
    };
```

```
    static uint8_t togglecode[256] = {
        [0x3A] CAPSLOCK,
        [0x45] NUMLOCK,
        [0x46] SCROLLLOCK
    };
    static uint8_t normalmap[256] = {
        NO,   0x1B, '1',  '2',  '3',  '4',  '5',  '6',  // 0x00
        '7',  '8',  '9',  '0',  '-',  '=',  '□', '   ',
        'q',  'w',  'e',  'r',  't',  'y',  'u',  'i',  // 0x10
        'o',  'p',  '[',  ']',  '
', NO,   'a',  's',
        'd',  'f',  'g',  'h',  'j',  'k',  'l',  ';',  // 0x20
        '\'', '`',  NO,   '', 'z', 'x', 'c', 'v',
    'b', 'n', 'm', ',', '.', '/', NO,  '',  // 0x30
    NO,  ' ', NO,  NO,  NO,  NO,  NO,  NO,
    NO,  NO,  NO,  NO,  NO,  NO,  NO,  '7',  // 0x40
    '8', '9', '-', '4', '5', '6', '+', '1',
    '2', '3', '0', '.', NO,  NO,  NO,  NO,  // 0x50
    [0xC7] KEY_HOME,   [0x9C] '
' /KP_Enter/,
    [0xB5] '/' /KP_Div/,  [0xC8] KEY_UP,
    [0xC9] KEY_PGUP,   [0xCB] KEY_LF,
    [0xCD] KEY_RT,     [0xCF] KEY_END,
    [0xD0] KEY_DN,     [0xD1] KEY_PGDN,
    [0xD2] KEY_INS,    [0xD3] KEY_DEL
  };
  static uint8_t shiftmap[256] = {
    NO,  033, '!', '@', '#', '$', '%', '^',  // 0x00
    '&', '', '(', ')', '_', '+', '□', '   ',
    'Q', 'W', 'E', 'R', 'T', 'Y', 'U', 'I',  // 0x10
    'O', 'P', '{', '}', '
', NO,  'A', 'S',
    'D', 'F', 'G', 'H', 'J', 'K', 'L', ':',  // 0x20
    '"', '~', NO,  '|', 'Z', 'X', 'C', 'V',
    'B', 'N', 'M', '<', '>', '?', NO,  '',  // 0x30
    NO,  ' ', NO,  NO,  NO,  NO,  NO,  NO,
    NO,  NO,  NO,  NO,  NO,  NO,  NO,  '7',  // 0x40
    '8', '9', '-', '4', '5', '6', '+', '1',
    '2', '3', '0', '.', NO,  NO,  NO,  NO,  // 0x50
    [0xC7] KEY_HOME,   [0x9C] '
' /KP_Enter/,
    [0xB5] '/' /KP_Div/,  [0xC8] KEY_UP,
    [0xC9] KEY_PGUP,   [0xCB] KEY_LF,
    [0xCD] KEY_RT,     [0xCF] KEY_END,
    [0xD0] KEY_DN,     [0xD1] KEY_PGDN,
    [0xD2] KEY_INS,    [0xD3] KEY_DEL
  };
  #define C(x) (x - '@')
  static uint8_t ctlmap[256] = {
    NO,    NO,    NO,    NO,    NO,    NO,    NO,    NO,
    NO,    NO,    NO,    NO,    NO,    NO,    NO,    NO,
    C('Q'), C('W'), C('E'), C('R'), C('T'), C('Y'), C('U'), C('I'),
    C('O'), C('P'), NO,    NO,    '
',   NO,    C('A'), C('S'),
    C('D'), C('F'), C('G'), C('H'), C('J'), C('K'), C('L'), NO,
    NO,    NO,    NO,    C(''), C('Z'), C('X'), C('C'), C('V'),
    C('B'), C('N'), C('M'), NO,    NO,    C('/'), NO,    NO,
    [0x97] KEY_HOME,
    [0xB5] C('/'),     [0xC8] KEY_UP,
    [0xC9] KEY_PGUP,   [0xCB] KEY_LF,
    [0xCD] KEY_RT,     [0xCF] KEY_END,
    [0xD0] KEY_DN,     [0xD1] KEY_PGDN,
    [0xD2] KEY_INS,    [0xD3] KEY_DEL
  };
  static uint8_t charcode[4] = {
    normalmap,
    shiftmap,
    ctlmap,
    ctlmap
  };
  /
   kbd_proc_data - get data from keyboard
```

```
1       The kbd_proc_data() function gets data from the keyboard.
2       If we finish a character, return it, else 0. And return -1 if no data.
3       /
4      static int
5      kbd_proc_data(void) {
```

```
        int c;
        uint8_t data;
        static uint32_t shift;
        if ((inb(KBSTATP) & KBS_DIB) == 0) {
            return -1;
        }
        data = inb(KBDATAP);
        if (data == 0xE0) {
            // E0 escape character
            shift |= E0ESC;
            return 0;
        } else if (data & 0x80) {
            // Key released
            data = (shift & E0ESC ? data : data & 0x7F);
            shift &= ~(shiftcode[data] | E0ESC);
            return 0;
        } else if (shift & E0ESC) {
            // Last character was an E0 escape; or with 0x80
            data |= 0x80;
            shift &= ~E0ESC;
        }
        shift |= shiftcode[data]; / ……(6)…… /
        shift ^= togglecode[data];
        c = charcode[shift & (CTL | SHIFT)][data];
        if (shift & CAPSLOCK) {
            if ('a' <= c && c <= 'z')
                c += 'A' - 'a'; / ……(7)…… /
            else if ('A' <= c && c <= 'Z')
                c += 'a' - 'A';
        }
        // Process special keys
        // Ctrl-Alt-Del: reboot
        if (!(~shift & (CTL | ALT)) && c == KEY_DEL) {
            kprintf("Rebooting!
");
            outb(0x92, 0x3); // courtesy of Chris Frost
        }
        return c;
    }
    / kbd_intr - try to feed input characters from keyboard /
    void
    kbd_intr(void) {
        cons_intr(kbd_proc_data);   / ……(8)…… /
    }
    static void
    kbd_init(void) {
        // drain the kbd buffer
        kbd_intr();
        pic_enable(IRQ_KBD);
    }
    / cons_init - initializes the console devices /
    void
    cons_init(void) {
        cga_init();
        serial_init();
        kbd_init();
        if (!serial_exists) {
            kprintf("serial port does not exist!!
");
        }
    }
    / cons_putc - print a single character @c to console devices /
    void
    cons_putc(int c) {
        bool intr_flag;
        local_intr_save(intr_flag);
        {
            lpt_putc(c);
            cga_putc(c);
            serial_putc(c);
        }
        local_intr_restore(intr_flag);
    }
    /
      cons_getc - return the next input character from console,
      or 0 if none waiting.
      /
    int
    cons_getc(void) {
        int c = 0;
        bool intr_flag;
        local_intr_save(intr_flag);
        {
            // poll for any pending input characters,
            // so that this function works even when interrupts are disabled
            // (e.g., when called from the kernel monitor).
            serial_intr();
```

```
            kbd_intr();
            // grab the next character from the input buffer.
            if (cons.rpos != cons.wpos) {
                c = cons.buf[cons.rpos ++]; / ……(9)…… /
                if (cons.rpos == CONSBUFSIZE) {
                    cons.rpos = 0;  / ……(10)…… /
                }
            }
        }
        local_intr_restore(intr_flag);
        return c;
    }
    ============================================================
    ```
```

- [x]

知识点:管程
出处:网络
难度:1

> 第1问6分，第2问5分；
> 1）命令行状态的键盘输入时的屏幕回显：
> 键盘输入产生中断；
> 中断处理例程扫描键盘，将数据写入缓冲区；
> 并在屏幕回显；
> 2）每两个1分；

```
    // Extract cursor location
    uint32_t pos;
    outb(addr_6845, 14);
    pos = inb(addr_6845 + 1) << 8; / ……(1)……读取光标位置的高8位，并左移8位 /
    outb(addr_6845, 15);
    pos |= inb(addr_6845 + 1); / ……(2)……读取光标位置的低8位，并与高8位合并在一个16位 /
    crt_buf = (uint16_t) cp;   / ……(3)……读取显示缓存区起始地址 /
    crt_pos = pos;
        if (c != 0) {
            cons.buf[cons.wpos ++] = c;    / ……(4)……键盘输入保存到缓冲区对应的位置，并指针加1 /
            if (cons.wpos == CONSBUFSIZE) {
                cons.wpos = 0;     / ……(5)……缓冲区写指针位置移回缓冲区开始地址 /
    shift |= shiftcode[data]; / ……(6)……依据控制键码表得到控制键编码 /
    shift ^= togglecode[data];
    c = charcode[shift & (CTL | SHIFT)][data];
    if (shift & CAPSLOCK) {
        if ('a' <= c && c <= 'z')
            c += 'A' - 'a';    / ……(7)……小写字符编码转在大写字符 /
        else if ('A' <= c && c <= 'Z')
            c += 'a' - 'A';
    }
void
kbd_intr(void) {
    cons_intr(kbd_proc_data);  / ……(8)……从键盘输入并保存到输入缓冲区 /
}
        if (cons.rpos != cons.wpos) {
            c = cons.buf[cons.rpos ++];    / ……(9)……从输入缓冲区读取输入字符编码，并后移读出指针 /
            if (cons.rpos == CONSBUFSIZE) {
                cons.rpos = 0; / ……(10)……将缓冲区读出指针移到缓冲区开头 /
            }
        }
```

4
在一个只允许单向行驶的十字路口，分别有若干由东向西，由南向北的车辆在等待通过十字路口。为了安全，每次只允许一辆车通过。当有车辆通过时其它车辆必须等候，当无车
辆在路口行驶时则允许一辆车通过。请用PV操作实现保证十字路口安全行驶的自动管理系统。

- [x]

知识点:同步互斥
出处:网络
难度:1

> S：表示临界资源十字路口，S＝1

```
    int  S=1;
    main()
    { pew();  psn();}
    pew()                                          psn()
    {                                              {
      p(s);     wait(s)                              p(s);
      由东向西通过十字路口;                        由南向北通过十字路口;
      v(s);    signal(s)                             v(s);
    }                                              }
```
```

> 4
> 有4位哲学家围着一个圆桌在思考和进餐，每人思考时手中什么都不拿，当需要进餐时，每人需要用刀和叉各一把，餐桌上的布置如图2-12所示，共有2把刀和2把叉，每把
> 刀或叉供相邻的两个人使用。请用信号量及PV操作说明4位哲学家的同步过程。

- [x]

知识点:同步互斥
出处:网络
难度:1

```
Int fork1=1,fork2=1,knife1=1,knife2=1;
Pa()
{ while(1)
  { p(knife1);
    p(fork1);
    进餐;
    v(knife1);
    v(fork1);
  }
}
```

> 4
> 桌上有一个空盘子，只允许放一个水果。爸爸可以向盘中放苹果，也可以向盘中放桔子，儿子专等吃盘中的桔子，女儿专等吃盘中的苹果。规定当盘空时，一次只能放一只水果，
> 请用PV操作实现爸爸、儿子、女儿3个并发进程的同步。

- [x]

知识点:同步互斥
出处:网络
难度:1

```
Int sp=1; sa=0;so=0;
Main()
{ father(); son();daughter();}
Father()                                          son()
{while(1)                                         {while(1)
  {p(sp);                                            {p(so);
    将水果放入盘中;                                 从盘中取出桔子;
   if (放入的是桔子)                                v(sp);
      v(so);                                              吃桔子;
   else  v(sa);                                       }
  }                                               }
}
```

> 4
> 在一个页式存储管理系统中，页面大小为1KB，主存中用户区的起始地址为1000，假定页表如下。现有一逻辑地址，页号为2，页内地址为20，试设计相应的物理地址，
> 并画图说明地址转换过程。

- [x]

知识点:非连续内存分配
出处:网络
难度:1

> 物理地址 = 块号块长+块内地址+用户区基址=91024+20+1000=10236
> 4
> 设一页式存储管理系统，向用户提供逻辑地址空间最大为16页，每页2048字节，主存总共有8个存储块，试问逻辑地址应为多少位？主存空间有多大？

- [x]

知识点:非连续内存分配
出处:网络
难度:1

> 逻辑地址：页号+页内地址 24 = 16, 211 = 2048 所以15位；
> 主存空间：82K = 16K
> 4
> 在一个页式存储管理系统中，某作业的页表如下表所示。已知页面大小为1024字节，用户区的基址为1000，试将逻辑地址1011、2148、3000、4000、5
> 012转换为相应的物理地址。 页号| 块号
> ---|---
> 0| 2
> 1| 3
> 2| 1
> 3| 6

- [x]

知识点:非连续内存分配
出处:网络
难度:1

```
页号＝[逻辑地址/页长]
页内地址＝逻辑地址 mod 页长
物理地址＝块号块长+块内地址+用户区基址
1011：  21024+1011+1000＝4059
2148：  页号：2     块号：3
            31024+100+1000＝
```

4
在一个请求分页存储管理的系统中，一个程序的页面走向为6,0,1,2,0,3,0,4,2,3,分别采用最佳置换算法、先进先出置换算法、最近最久未使用算法，完成
下列要求。设分配给该程序的存储块数M=3,每调进一个新页就发生一次缺页中断。
完成下表，求缺页中断次数和缺页率

时刻	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
访问顺序	6	0	1	2	0	3	0	4	2	3
M=3										
f										

- [x]

知识点:非连续内存分配
出处:网络
难度:1

OPT

时刻	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
访问顺序	6	0	1	2	0	3	0	4	2	3
M=3	6	6	6	2	2	2	2	2	2	2
	0	0	0	0	0	0	4	4	4	
		1	1	1	3	3	3	3	3	
f	1	2	3	4		5		6		

FIFO

时刻	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
访问顺序	6	0	1	2	0	3	0	4	2	3
M=3	6	6	6	0	0	1	2	3	0	4
	0	0	1	1	2	3	0	4	2	
		1	2	2	3	0	4	2	3	
f	1	2	3	4		5	6	7	8	9

LRU

时刻	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
访问顺序	6	0	1	2	0	3	0	4	2	3
M=3			1	2	0	3	0	4	2	3
	0	0	1	2	0	3	0	4	2	
6	6	6	0	1	2	2	3	0	4	
f	1	2	3	4		5		6	7	8

4
假定磁带记录密度为每英寸800字符，每一逻辑记录为160字符，块间隙为0.6英寸。今有1500个逻辑记录需要存储，试计算磁带的利用率？若要使磁带空间利用率不
少于50%，至少应以多少个逻辑记录为一组？这说明了什么问题？

- [x]

知识点:I/O子系统
出处:网络
难度:1

> 一个记录占据的长度：160/800 = 0.2
> 1500个记录占据的长度：(0.2+0.6)1500=1200
> 磁带的利用率：0.2/(0.2+0.6)=25%
> 一组记录数：0.6/0.2 = 3
> 4
> 某软盘有40个磁道，磁头从一个磁道移到另一个磁道需要6ms。文件在磁盘上非连续存放，逻辑上相邻数据块的平均距离为13磁道，每块的旋转延迟时间及传输时间分别为
> 100ms、25ms，问读取一个100块的文件需要多少时间？如果系统对磁盘进行了整理，让同一个磁盘块尽可能靠拢，从而使逻辑上相邻的数据块的平均距离降为2磁道
> ，这时读取一个100块的文件需要多少时间?

- [x]

知识点:I/O子系统
出处:网络
难度:1

> 磁盘访问时间 = 寻道时间 + 延迟时间 + 传输时间
> 整理前：读取一个数据块的时间为：136+100+25 = 203ms
> 读取一个100块的文件需要：100203 = 20300ms
> 整理后：100(26+100+25)=13700ms
> 4
> 若磁头的当前位置为100磁道，磁头正向磁道号增加的方向移动。现有一磁盘读写请求队列：23、376、205、132、19、61、190、398、29、4、18
> 、40。若采用先来先服务、最短寻道时间优先和扫描（电梯调度）算法，试计算平均寻道长度各为多少?

- [x]

知识点:I/O子系统
出处:网络
难度:1

> FCFS:133; SSTF:58.3; SCAN:57.7
> 4
> 磁盘请求以10、22、20、2、40、6、38柱面的次序到达磁盘驱动器。寻道时每个柱面移动需要6ms，计算以下寻道次序和寻道时间。
> （1）先来先服务
> （2）电梯调度算法（起始向磁道号大的方向移动）
> 在所有情况下磁头臂起始都位于柱面20号上。

- [x]

知识点:I/O子系统
出处:网络
难度:1

```
                              10、22、20、2、40、6、38
FCFS:                  （10+12+2+ 18+ 38+34+32）6=876
SCAN:                 （2+16+2+ 20+ 10+4+4）6=348
                              22、38、40、20、10、6、2
```

> 4
> 有一磁盘组共有10个盘面，每个盘面上有100个磁道，每个磁道有16个扇区。假定分配以扇区为单位，若使用位示图管理磁盘空间，问位示图需要占用多少空间？若空闲文
> 件目录的每条记录占用5个字节，问什么时候空闲文件目录大于位示图?

- [x]

知识点:文件系统
出处:网络
难度:1

```
解：扇区总数：1010016=16000
        则位示图的位数：16000/8=2000字节
        位示图中空闲块数：216>16000   所以要用2字节存储
       则位示图的大小为：2000+0=2002字节
空闲文件目录的每条记录占用5个字节
2002字节可以存放的表目数为：2002/5=400
当空闲文件目录数为400时，空闲文件目录大于位示图
```

> 4
> 假定磁盘块的大小为1KB，对于540MB的硬盘，其文件分配表FAT需要占用多少存储空间？当硬盘容量为1.2GB时，FAT需要占用多少空间?

- [x]

知识点:文件系统
出处:网络
难度:1

> 硬盘总块数为：540M/1K = 540K个；因为：220>540K，即文件分配表的每个表目为20/8 = 2.5字节，则FAT占用：2.5540 = 1350K
> 4
> 在一个单道批处理系统中，一组作业的提交时间和运行时间作业

	提交时间	运行时间
J1	8：00	1.0

```
J2              8：50          0.50
J3              9：00          0.20
J4              9：10          0.10
```
试计算以下三种作业调度算法的平均周转时间和平均带权周转时间（1）先来先服务 （2）短作业优先 （3）响应比高者优先

- [x]

知识点:处理机调度
出处:网络
难度:1

```
（1）先来先服务 平均周转时间＝(1.0+0.67+0.7+0.63)/4=0.75
平均带权周转时间＝(1.0+1.34+3.5+6.3)/4=3.035
(2)短作业优先
作业执行顺序：J1  J3  J4  J2
平均周转时间＝（1.0+0.94+0.2+0.13）/4=0.5675
平均带权周转时间＝（1.0+1.94+1.0+1.3）=1.31
(3)响应比高者优先
同（1）
```

4
1）操作系统的微内核结构特征是什么？2）它有什么优点和缺点？3）在微内核结构中，内存管理、进程通信、文件系统、I/O管理这几种操作系统功能中，哪些是放在内核
中的？哪些是放在用户态的?

- [x]

知识点:操作系统概述
出处:网络
难度:1

1）只把必要的功能放在内核中；（1分） 2）优点：扩展和移植（1分，写对一个就行）、可靠和安全（1分，写对一个就行）；
缺点：内核与用户态的切换（1分）性能和进程间通信（1分）性能效率低（1分）； 3）内核功能：内存管理（放在哪都对）（1分）、进程通信（1分）
应用功能：文件系统（1分）、I/O管理（1分）
4
1）试描述进程执行中利用堆栈实现函数调用和返回的过程。2）请补全下面print_stackframe()函数所缺的代码，以利用函数调用时保存在堆栈中的信息输
出嵌套调用的函数入口地址和参数信息。

```
=============kern-ucore/arch/i386/debug/kdebug.c=============
/
  print_debuginfo - read and print the stat information for the address @eip,
  and info.eip_fn_addr should be the first address of the related function.
  /
void
print_debuginfo(uintptr_t eip) {
......
}
static uint32_t read_eip(void) attribute((noinline));
static uint32_t
read_eip(void) {
......
}
/
  print_stackframe - print a list of the saved eip values from the nested 'call'
  instructions that led to the current point of execution
```

```
      The x86 stack pointer, namely esp, points to the lowest location on the stack
      that is currently in use. Everything below that location in stack is free. Pushing
      a value onto the stack will invole decreasing the stack pointer and then writing
      the value to the place that stack pointer pointes to. And popping a value do the
      opposite.

      The ebp (base pointer) register, in contrast, is associated with the stack
      primarily by software convention. On entry to a C function, the function's
      prologue code normally saves the previous function's base pointer by pushing
      it onto the stack, and then copies the current esp value into ebp for the duration
      of the function. If all the functions in a program obey this convention,
      then at any given point during the program's execution, it is possible to trace
      back through the stack by following the chain of saved ebp pointers and determining
      exactly what nested sequence of function calls caused this particular point in the
      program to be reached. This capability can be particularly useful, for example,
      when a particular function causes an assert failure or panic because bad arguments
      were passed to it, but you aren't sure who passed the bad arguments. A stack
      backtrace lets you find the offending function.

      The inline function read_ebp() can tell us the value of current ebp. And the
      non-inline function read_eip() is useful, it can read the value of current eip,
      since while calling this function, read_eip() can read the caller's eip from
      stack easily.

      In print_debuginfo(), the function debuginfo_eip() can get enough information about
```

```
      calling-chain. Finally print_stackframe() will trace and print them for debugging.

      Note that, the length of ebp-chain is limited. In boot/bootasm.S, before jumping
      to the kernel entry, the value of ebp has been set to zero, that's the boundary.
      /
    void
    print_stackframe(void) {
        uint32_t ebp = read_ebp(), eip = read_eip();
        int i, j;
        for (i = 0; ebp != 0 && i < 10; i ++) {
            kprintf("ebp:0x%08x eip:0x%08x args:", ebp, eip);
            uint32_t args = (uint32_t ) _--YOUR CODE 1--_;
            for (j = 0; j < 4; j ++) {
                kprintf("0x%08x ", args[j]);
            }
            kprintf("
");
            print_debuginfo(eip - 1);
            eip = ((uint32_t ) _--YOUR CODE 2--_;
            ebp = ((uint32_t ) _--YOUR CODE 3--_;
        }
    }
    ```
```

- [x]

知识点:进程状态与控制
出处:网络
难度:1

1）函数调用和返回的过程：
1.参数压栈；2.函数调用跳转（指令指针（2分）和栈顶指针（2分）压栈（2分））；3.函数执行；4. 函数返回（指令指针和栈顶指针退栈（2分））
没有指针压退栈，但有参数压退栈的，给1分；
2）（6分，每空2分）

```
         uint32_t args = (uint32_t ) _--YOUR CODE 1--_;
         uint32_t args = (uint32_t )ebp + 2;
        eip = ((uint32_t ) _--YOUR CODE 2--_;
        eip = ((uint32_t )ebp)[1];
         ebp = ((uint32_t ) _--YOUR CODE 3--_;
         ebp = ((uint32_t )ebp)[0];
```

4
1）系统调用接口是操作系统内核向用户进程提供操作系统服务的接口。试描述用户进程通过系统调用使用操作系统服务的过程。2）gettime_msec是一个获取当前
系统时间的系统调用。请补全该系统调用的实现代码。

```
=============libs-user-ucore/ulib.c=============
unsigned int
gettime_msec(void) {
    return (unsigned int)sys_gettime();
}
=============libs-user-ucore/syscall.c=============
size_t
sys_gettime(void) {
    return (size_t) _--YOUR CODE 4--_;
}
=============libs-user-ucore/arch/i386/syscall.c=============
#define MAX_ARGS            5
uint32_t
syscall(int num, ...) {
    va_list ap;
    va_start(ap, num);
    uint32_t a[MAX_ARGS];
    int i;
    for (i = 0; i < MAX_ARGS; i ++) {
        a[i] = va_arg(ap, uint32_t);
    }
    va_end(ap);
    uint32_t ret;
    asm volatile (
        "int %1;"
        : "=a" (ret)
        : "i" (T_SYSCALL),
          "a" (num),
          "d" (a[0]),
          "c" (a[1]),
          "b" (a[2]),
          "D" (a[3]),
          "S" (a[4])
```
```

```
            : "cc", "memory");
        return ret;
    }
    =============libs-user-ucore/common/unistd.h=============
    / syscall number /
    #define SYS_exit            1
    #define SYS_fork            2
    #define SYS_wait            3
    #define SYS_exec            4
    #define SYS_clone           5
    #define SYS_exit_thread     9
    #define SYS_yield           10
    #define SYS_sleep           11
    #define SYS_kill            12
    #define SYS_gettime         17
    #define SYS_getpid          18
    #define SYS_brk             19
    ……
    =============kern-ucore/arch/i386/glue-ucore/trap.c=============
    static void
    trap_dispatch(struct trapframe tf) {
        char c;
        int ret;
        switch (tf->tf_trapno) {
        case T_DEBUG:
        case T_BRKPT:
            debug_monitor(tf);
            break;
        case T_PGFLT:
            if ((ret = pgfault_handler(tf)) != 0) {
                print_trapframe(tf);
                if (pls_read(current) == NULL) {
                    panic("handle pgfault failed. %e
", ret);
                }
                else {
                    if (trap_in_kernel(tf)) {
                        panic("handle pgfault failed in kernel mode. %e
", ret);
                    }
                    kprintf("killed by kernel.
");
                    do_exit(-E_KILLED);
                }
            }
            break;
        case T_SYSCALL:
            _--YOUR CODE 5--_;
            break;
        case IRQ_OFFSET + IRQ_TIMER:
            ticks ++;
            assert(pls_read(current) != NULL);
            run_timer_list();
            break;
        case IRQ_OFFSET + IRQ_COM1:
        case IRQ_OFFSET + IRQ_KBD:
            if ((c = cons_getc()) == 13) {
                debug_monitor(tf);
            }
            else {
                extern void dev_stdin_write(char c);
                dev_stdin_write(c);
            }
            break;
        case IRQ_OFFSET + IRQ_IDE1:
        case IRQ_OFFSET + IRQ_IDE2:
            / do nothing /
            break;
        default:
            print_trapframe(tf);
            if (pls_read(current) != NULL) {
                kprintf("unhandled trap.
");
                do_exit(-E_KILLED);
```

```
            }
            panic("unexpected trap in kernel.
");
        }
    }
    void
    trap(struct trapframe tf) {
        // used for previous projects
        if (pls_read(current) == NULL) {
            trap_dispatch(tf);
        }
        else {
            // keep a trapframe chain in stack
            struct trapframe otf = pls_read(current)->tf;
            pls_read(current)->tf = tf;
            bool in_kernel = trap_in_kernel(tf);
            _--YOUR CODE 6--_;
            pls_read(current)->tf = otf;
            if (!in_kernel) {
                may_killed();
                if (pls_read(current)->need_resched) {
                    schedule();
                }
            }
        }
    }
    =============kern-ucore/arch/i386/syscall/syscall.c=============
    ……
    static uint32_t
    sys_gettime(uint32_t arg[]) {
        return (int)ticks;
    }
    ……
    static uint32_t (syscalls[])(uint32_t arg[]) = {
        [SYS_exit]              sys_exit,
        [SYS_fork]              sys_fork,
        [SYS_wait]              sys_wait,
        [SYS_exec]              sys_exec,
        [SYS_clone]             sys_clone,
        [SYS_exit_thread]       sys_exit_thread,
        [SYS_yield]             sys_yield,
        [SYS_kill]              sys_kill,
        [SYS_sleep]             sys_sleep,
        [SYS_gettime]           _--YOUR CODE 7--_,
        [SYS_getpid]            sys_getpid,
    ……
    };
    #define NUM_SYSCALLS        ((sizeof(syscalls)) / (sizeof(syscalls[0])))
    void
    syscall(void) {
        struct trapframe tf = pls_read(current)->tf;
        uint32_t arg[5];
        int num = tf->tf_regs.reg_eax;
        if (num >= 0 && num < NUM_SYSCALLS) {
            if (syscalls[num] != NULL) {
                arg[0] = tf->tf_regs.reg_edx;
                arg[1] = tf->tf_regs.reg_ecx;
                arg[2] = tf->tf_regs.reg_ebx;
                arg[3] = tf->tf_regs.reg_edi;
                arg[4] = tf->tf_regs.reg_esi;
                tf->tf_regs.reg_eax = _--YOUR CODE 8--_;
                return ;
            }
        }
        print_trapframe(tf);
        panic("undefined syscall %d, pid = %d, name = %s.
",
                num, pls_read(current)->pid, pls_read(current)->name);
    }
```

- [x]

知识点:中断、异常与系统调用
出处:网络
难度:1

1）系统调用的过程：（8分，每个关键词1分）
1.准备参数和系统调用号；2. 执行系统中断；3. CPU响应中断，并依据中断号找到系统调用处理例程；4.
系统调用处理例程依据系统调用号找到对应系统调用实现代码；5.获取系统调用参数；6.
执行系统调用功能7.准备返回结果；8.执行中断返回到用户态；9.获取返回结果；
2）（10分，每个空2分）

```
    return (size_t) _--YOUR CODE 4--_;
    return (size_t)syscall(SYS_gettime);//必须有参数
        _--YOUR CODE 5--_;
syscall();//不能有参数
        _--YOUR CODE 6--_;
trap_dispatch(tf);
    [SYS_gettime]           _--YOUR CODE 7--_,
[SYS_gettime]           sys_gettime,
            tf->tf_regs.reg_eax = _--YOUR CODE 8--_;
tf->tf_regs.reg_eax = syscalls[num](arg);
```

4
1）试利用图示描述伙伴系统（Buddy System）中对物理内存的分配和回收过程。2）请补全下面伙伴系统实现中所缺的代码。

```
==========kern-ucore/arch/i386/mm/buddy_pmm.c==========
    // {1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024}
    // from 2^0 ~ 2^10
    #define MAX_ORDER 10
    static free_area_t free_area[MAX_ORDER + 1];
    //x from 0 ~ MAX_ORDER
    #define free_list(x) (free_area[x].free_list)
    #define nr_free(x) (free_area[x].nr_free)
    #define MAX_ZONE_NUM 10
    struct Zone {
        struct Page mem_base;
    } zones[MAX_ZONE_NUM] = {{NULL}};
    //buddy_init - init the free_list(0 ~ MAX_ORDER) & reset nr_free(0 ~ MAX_ORDER)
    static void
    buddy_init(void) {
        int i;
        for (i = 0; i <= MAX_ORDER; i ++) {
            list_init(&free_list(i));
            nr_free(i) = 0;
        }
    }
    //buddy_init_memmap - build free_list for Page base follow  n continuing pages.
    static void
    buddy_init_memmap(struct Page base, size_t n) {
    ……
    }
    //getorder - return order, the minmal 2^order >= n
    static inline size_t
    getorder(size_t n) {
        size_t order, order_size;
        for (order = 0, order_size = 1; order <= MAX_ORDER; order ++, order_size <<= 1) {
            if (n <= order_size) {
                return order;
            }
        }
        panic("getorder failed. %d
", n);
    }
    //buddy_alloc_pages_sub - the actual allocation implimentation, return a page whose size >=n,
    //                        - the remaining free parts insert to other free list
    static inline struct Page
    buddy_alloc_pages_sub(size_t order) {
        assert(order <= MAX_ORDER);
        size_t cur_order;
        for (cur_order = order; cur_order <= MAX_ORDER; cur_order ++) {
            if (!list_empty(&free_list(cur_order))) {
                list_entry_t le = list_next(&free_list(cur_order));
                struct Page page = le2page(le, page_link);
                nr_free(cur_order) --;
                _--YOUR CODE 9--_(le);
                size_t size = 1 << cur_order;
                while (cur_order > order) {
                    cur_order --;
                    size >>= 1;
                    struct Page buddy = page + size;
                    buddy->property = cur_order;
                    SetPageProperty(buddy);
```

```
                    nr_free(cur_order) ++;
                    _--YOUR CODE 10--_(&free;_list(cur_order), &(buddy->page_link));
                }
                ClearPageProperty(page);
                return page;
            }
        }
        return NULL;
    }
    //buddy_alloc_pages - call buddy_alloc_pages_sub to alloc 2^order>=n pages
    static struct Page 
    buddy_alloc_pages(size_t n) {
        assert(n > 0);
        size_t order = getorder(n), order_size = (1 << order);
        struct Page page = buddy_alloc_pages_sub(order);
        if (page != NULL && n != order_size) {
            free_pages(page + n, order_size - n);
        }
        return page;
    }
    //page_is_buddy - Does this page belong to the No. zone_num Zone & this page
    //              -  be in the continuing page block whose size is 2^order pages?
    static inline bool
    page_is_buddy(struct Page page, size_t order, int zone_num) {
        if (page2ppn(page) < npage) {
            if (page->zone_num == zone_num) {
                return !PageReserved(page) && PageProperty(page) && page->property == order;
            }
        }
        return 0;
    }
    //page2idx - get the related index number idx of continuing page block which this page belongs to
    static inline ppn_t
    page2idx(struct Page page) {
        return page - zones[page->zone_num].mem_base;
    }
    //idx2page - get the related page according to the index number idx of continuing page block
    static inline struct Page 
    idx2page(int zone_num, ppn_t idx) {
        return zones[zone_num].mem_base + idx;
    }
    //buddy_free_pages_sub - the actual free implimentation, should consider how to
    //                     - merge the adjacent buddy block
    static void
    buddy_free_pages_sub(struct Page base, size_t order) {
        ppn_t buddy_idx, page_idx = page2idx(base);
        assert((page_idx & ((1 << order) - 1)) == 0);
        struct Page p = base;
        for (; p != base + (1 << order); p ++) {
            assert(!PageReserved(p) && !PageProperty(p));
            p->flags = 0;
            set_page_ref(p, 0);
        }
        int zone_num = base->zone_num;
        while (order < MAX_ORDER) {
            buddy_idx = page_idx ^ (1 << order);
            struct Page buddy = idx2page(zone_num, buddy_idx);
            if (!page_is_buddy(buddy, order, zone_num)) {
                break;
            }
            nr_free(order) --;
            _--YOUR CODE 11--_(&(buddy->page_link));
            ClearPageProperty(buddy);
            page_idx &= buddy_idx;
            order ++;
        }
        struct Page page = idx2page(zone_num, page_idx);
        page->property = order;
        SetPageProperty(page);
        nr_free(order) ++;
        _--YOUR CODE 12--_(&free;_list(order), &(page->page_link));
    }
    //buddy_free_pages - call buddy_free_pages_sub to free n continuing page block
    static void
```

```
    buddy_free_pages(struct Page base, size_t n) {
        assert(n > 0);
        if (n == 1) {
            buddy_free_pages_sub(base, 0);
        }
        else {
            size_t order = 0, order_size = 1;
            while (n >= order_size) {
                assert(order <= MAX_ORDER);
                if ((page2idx(base) & order_size) != 0) {
                    buddy_free_pages_sub(base, order);
                    base += order_size;
                    n -= order_size;
                }
                order ++;
                order_size <<= 1;
            }
            while (n != 0) {
                while (n < order_size) {
                    order --;
                    order_size >>= 1;
                }
                buddy_free_pages_sub(base, order);
                base += order_size;
                n -= order_size;
            }
        }
    }
    //buddy_nr_free_pages - get the nr: the number of free pages
    static size_t
    buddy_nr_free_pages(void) {
        size_t ret = 0, order = 0;
        for (; order <= MAX_ORDER; order ++) {
            ret += nr_free(order)  (1 << order);
        }
        return ret;
    }
    //buddy_check - check the correctness of buddy system
    static void
    buddy_check(void) {
    ……
    }
    //the buddy system pmm
    const struct pmm_manager buddy_pmm_manager = {
        .name = "buddy_pmm_manager",
        .init = buddy_init,
        .init_memmap = buddy_init_memmap,
        .alloc_pages = buddy_alloc_pages,
        .free_pages = buddy_free_pages,
        .nr_free_pages = buddy_nr_free_pages,
        .check = buddy_check,
    };
```

- [x]

知识点:连续内存分配
出处:网络
难度:1

1)

分配：1）把用户要求的大小转成*2的幂*（1分）；2）从空闲数组对应大小开始向大的方向找，直到*有比需要大小不小的空闲块*（1分）；3）如果比需要的大，*切*
半（2分）后留下一个，另一个*放入空闲数组*（1分）；直到得到一个需要大小的块；
回收：1）回收块*按地址顺序*（1分）放入对应大小的空闲链；2）与相邻空闲块进行可能的合并（1分），合并条件：*相邻*（1分）且小地址的那一块的起始地址是_
当前块大小的2倍的整数倍_（2分）；
只画图，没有描述的，给4分；
2)8分，每个空2分；

```
            _--YOUR CODE 9--_(le);
list_del(le);
                _--YOUR CODE 10--_(&free;_list(cur_order), &(buddy->page_link));
list_add(&free;_list(cur_order), &(buddy->page_link));
        _--YOUR CODE 11--_(&(buddy->page_link));
list_del(&(buddy->page_link));
    _--YOUR CODE 12--_(&free;_list(order), &(page->page_link));
list_add(&free;_list(order), &(page->page_link));
```

4
1）试用图示描述32位X86系统在采用4KB页面大小时的页表结构。2）在采用4KB页面大小的32位X86的ucore虚拟存储系统中，进程页面的起始地址由宏V
PT确定。
`#define VPT 0xFAC00000`
请计算：2a)页目录（PDE）在虚拟地址空间中的起始地址；2b)虚拟地址0X87654321对应的页目录项和页表项的虚拟地址。

- [x]

知识点:虚拟内存管理实验
出处:网络
难度:1

1）（7分）地址划分：10＋10＋12（3分）
每个页表项占4字节（2分），每页1024项（2分）
2a）（9分）每个地址3分，每个地址中的三段，每段1分；（二进制对了，就给全分）

```
FAC0 0000
_1111 1010 11_00 0000 0000 0000 0000 0000
_1111 1010 11_ 11 11 10 10 11 _11 11 10 10 11_ 00
FAFE B_FAC_
2b）
87654321
_1000 0111 01_10 0101 0100 0011 0010 0001
PDE:
_1111 1010 11_ 11 11 10 10 11 _1000 0111 01_ 00
FAFE B874
PTE:
_1111 1010 11_ _10 00 01 11 01_ 10 0101 0100 00
FAE1 D950
```

4
1）试描述ucore的进程创建系统调用fork（）的基本过程。2）请补全fork系统调用的实现代码。
`==========kern-ucore/process/proc.c==========`

```
// get_pid - alloc a unique pid for process
static int
get_pid(void) {
……
}
……
int
do_fork(uint32_t clone_flags, uintptr_t stack, struct trapframe tf) {
    int ret = -E_NO_FREE_PROC;
    struct proc_struct proc;
    if (nr_process >= MAX_PROCESS) {
        goto fork_out;
    }
    ret = -E_NO_MEM;
    if ((proc = alloc_proc()) == NULL) {
        goto fork_out;
    }
    proc->parent = _--YOUR CODE 13--_;
    list_init(&(proc->thread_group));
    assert(current->wait_state == 0);
    assert(current->time_slice >= 0);
    proc->time_slice = current->time_slice / 2;
    current->time_slice -= proc->time_slice;
    if (setup_kstack(proc) != 0) {
        goto bad_fork_cleanup_proc;
    }
    if (copy_sem(clone_flags, proc) != 0) {
        goto bad_fork_cleanup_kstack;
    }
    if (copy_fs(clone_flags, proc) != 0) {
        goto bad_fork_cleanup_sem;
    }
    if (copy_mm(clone_flags, proc) != 0) {
        goto bad_fork_cleanup_fs;
    }
    if (copy_thread(clone_flags, proc, stack, tf) != 0) {
```

```
            goto bad_fork_cleanup_sem;
        }
        bool intr_flag;
        local_intr_save(intr_flag);
        {
            proc->pid = _--YOUR CODE 14--_;
            hash_proc(proc);
            set_links(proc);
            if (clone_flags & CLONE_THREAD) {
                list_add_before(&(current->thread_group), &(proc->thread_group));
            }
        }
        local_intr_restore(intr_flag);
        wakeup_proc(proc);
        ret = _--YOUR CODE 15--_;
    fork_out:
        return ret;
    bad_fork_cleanup_fs:
        put_fs(proc);
    bad_fork_cleanup_sem:
        put_sem_queue(proc);
    bad_fork_cleanup_kstack:
        put_kstack(proc);
    bad_fork_cleanup_proc:
        kfree(proc);
        goto fork_out;
    }
    =============kern-ucore/arch/i386/process/proc.c=============
    // forkret -- the first kernel entry point of a new thread/process
    // NOTE: the addr of forkret is setted in copy_thread function
    //       after switch_to, the current proc will execute here.
    static void
    forkret(void) {
        forkrets(pls_read(current)->tf);
    }
    ……
    // copy_thread - setup the trapframe on the  process's kernel stack top and
    //             - setup the kernel entry point and stack of process
    int
    copy_thread(uint32_t clone_flags, struct proc_struct proc,
                uintptr_t esp, struct trapframe tf) {
        proc->tf = (struct trapframe )(proc->kstack + KSTACKSIZE) - 1;
        (proc->tf) = tf;
        proc->tf->tf_regs.reg_eax = 0;
        proc->tf->tf_esp = esp;
        proc->tf->tf_eflags |= FL_IF;
        proc->context.eip = (uintptr_t) _--YOUR CODE 16--_;
        proc->context.esp = (uintptr_t)(proc->tf);
        return 0;
    }
```

- [x]

知识点:进程状态与控制
出处:网络
难度:1

1）fork()系统调用的执行过程： 进程通过调用fork（）进入内核，在内核完成子进程的复制（2分）后，放入就绪队列；父进程返回子进程的标识（2分）；子进
程在调度执行时返回用户态，返回值为0（2分），从fork（）后的指令开始子进程的执行（1分）； 2）

```
1     proc->parent = _--YOUR CODE 13--_;
2   proc->parent = current; （1分）
3         proc->pid = _--YOUR CODE 14--_;
4  proc->pid = get_pid();//分配新标识（2分）
5     ret = _--YOUR CODE 15--_;
6    ret = proc->pid;//返回子进程标识；（2分）
7     proc->context.eip = (uintptr_t) _--YOUR CODE 16--_;
8  proc->context.rip = (uintptr_t)forkret;//IP指向forkret （2分）
```

4
试用图示描述五状态进程模型，要求给出状态描述和各状态间的变迁。

- [x]

知识点:处理机调度
出处:网络
难度:1

每个状态一分，错一个扣一分，对一个加一分；每错一个变迁扣1分（最多扣5分）；

4

在管道通信机制中,用信号量描述读进程和写进程访问管道文件的过程,假设管 道文件大小为10KB.

- [x]

知识点:实验环境准备实验
出处:网络
难度:1

UNIX系统中,利用一个打开的共享文件来连接两个相互通信的进程,这个共享文 件叫管道.作为管道输入的发送进程,以字符流的形式将信息送入管道,而作为管道
输出的接收进程,从管道中获取信息.管道通信机制要提供三方面的协调能力:(1)互 斥.当一个进程对管道进行读/写操作时,另一个进程必须等待.(2)
同步.当写进程 把数据写入管道后便去睡眠等待,直到输出进程取走数据后唤醒它.若一次写入的数据
超过缓冲区剩余空间的大小,当缓冲区满时,写进程必须阻塞,并唤醒读进程。(3)对 方是否存在.只有确定对方存在时,才能够进行通信.本题只需要考虑互斥,同步问
题。由于只有一对进程访问管道,因此不需要设置互斥信号量,只要设置两个同步信号 量empty,full.分别表示管道可写和可读.

```
begin
pipe:array[09] of kilobytes;
ts=10,length,in=0,out=0:integer;
empty,full:semaphore=1,0;
cobegin
process PipeWriter
  begin
  repeat
  产生数据;
  p(empty);
  length = data length;
  while(length>0 and ts>0)
  begin
   pipe[in] = data of 1KB;
   in = (in+1) mod n;
   ts = ts-1;
   length = length - 1;
  end
  v(full);
  end
process Consumer
  begin
  repeat;
  p(full);
  从缓冲区取出一件物品;
  out = (out+1) mod n;
  ts = ts +1;
  v(empty);
  end
coend
end
```

4

桌上有一空盘,允许存放一只水果。爸爸可向盘中放苹果,也可向盘中放桔子,儿 子专等吃盘中的桔子,女儿专等吃盘中的苹果。规定当盘空时一次只能放一只水果供
吃者取用,请用P、V原语实现爸爸、儿子、女儿三个并发进程的同步

- [x]

知识点:实验环境准备实验
出处:网络
难度:1

在本题中,爸爸、儿子、女儿共用一个盘子,盘中一次只能放一个水果。当盘子为 空时,爸爸可将一个水果放入果盘中。若放入果盘中的是桔子,则允许儿子吃,女儿必
须等待;若放入果盘中的是苹果,则允许女儿吃,儿子必须等待。本题实际上是生产 者-消费者问题的一种变形。这里,生产者放入缓冲区的产品有两类,消费者也有两类,
每类消费者只消费其中固定的一类产品。
在本题中,应设置三个信号量S、So、Sa,信号量S表示盘子是否为空,其初值为l;
信号量So表示盘中是否有桔子,其初值为0;信号量Sa表示盘中是否有苹果,其初值为0。 同步描述如下:

```
S=1; Sa=0; So=0;
cobegin
 Procedure father;
 Procedure son;
 Procedure daughter;
coend
Procedure father:
  begin
  while(TRUE)
  begin
  P(S);
  将水果放入盘中;
  if(放入的是桔子)
  V(So);
```

```
  else
  V(Sa);
  end
  end
Procedure son:
  begin
  while(TRUE)
  begin
  P(So);
  从盘中取出桔子;
  V(S);
  吃桔子;
  end
  end
Procedure daughter:
  begin
  while(TRUE)
  begin
  P(Sa);
  从盘中取出苹果;
  V(S);
  吃苹果;
  end
end
```

4
在南开大学至天津大学间有一条弯曲的路,每次只允许一辆自行车通过,但中间
有小的安全岛M(同时允许两辆车),可供两辆车在已进入两端小车错车,设计算法并使用P,V实现。

- [x]

知识点:实验环境准备实验
出处:网络
难度:1

由于安全岛M仅仅允许两辆车停留,本应该作为临界资源而要设置信号量, 但根据 题意,任意时刻进入安全岛的车不会超过两辆(两个方向最多各有一辆), 因此,不需要
为M设置信号量,在路口s和路口t都需要设置信号量,以控制来自两个方向的车对路口资
源的争夺.这两个信号量的初值都是1.此外,由于从s到t的一段路只允许一辆车通过,所
以还需要设置另外的信号量用于控制,由于M的存在,可以为两端的小路分别设置一个互 斥信号量.

```
var T2N, N2T,L,M,K:semaphore;
T2N:=1;
N2T:=1;
L:=1;
K:=1;
M:=2;
cobegin
  Procedure Bike T2N
  begin
    p(T2N);
    p(L);
      go T to L;
    p(M);
      go into  M;
    V(L);
    P(k);
     go K to s;
    V(M);
    V(k);
    V(T2N);
  end
  Procedure Bike N2T
  begin
    P(N2T);
    p(k);
     go v to k;
    p(M);
      go into M;
    V(k);
    P(L);
      go L to T;
    V(M);
    V(L);
    V(N2T);
  end
coend
```

4
fork例子
第二题： 一、 （10分）给出程序fork.c的输出结果。
注：1）getpid()和getppid()是两个系统调用，分别返回本进程标识和父进程标识。
2）你可以假定每次新进程创建时生成的进程标识是顺序加1得到的；在进程标识为1000的命令解释程序shell中启动该程序的执行
```

  nclude
```

```
#include
/ getpid() and fork() are system calls declared in unistd.h.  They return /
/ values of type pid_t.  This pid_t is a special type for process ids. /
/ It's equivalent to int. /
int main(void)
{
  pid_t childpid;
  int x = 5;
      int i;
  childpid = fork();
  for ( i = 0;  i < 2;  i++) {
    printf(This is process %d; childpid = %d; The parent of this process has id %d; i = %d; x = %d
"
getpid()
```

- [x]

知识点:
出处:网络
难度:1

> getppid()
> 4
> 在一个盒子里,混装了数量相等的黑白围棋子·现在用自动分拣系统把黑子、白 子分开,设分拣系统有二个进程P1 和P2 ,其中P1 拣白子;P2
> 拣黑子。规定每个进程 每次拣一子;当一个进程在拣时,不允许另一个进程去拣;当一个进程拣了一子时,必 须让另一个进程去拣.试写出两进程P1 和P2
> 能并发正确执行的程序。

- [x]

知识点:实验环境准备实验
出处:网络
难度:1

> 大家熟悉了生产-消费问题(PC),这个问题很简单。题目较为新颖,但是本质非常 简单即:生产-消费问题的简化或者说是两个进程的简单同步问题。答案如下:
> 设信号量s1 和s2 分别表示可拣白子和黑子; 不失一般性,若令先拣白子。

```
var S1 , S2 : semaphore;
S1 : = l; S2 :=0;
cobegin
  process P1           Process P2
   begin                begin
   repeat               repeat
   P(S1);               P(S2);
   pick The white;      pick the black;
   V(S2);               V(S1);
  until false ;         until false;
  end                   end
coend
```

> 4
> 设公共汽车上,司机和售票员的活动分别如下:司机的活动:启动车辆:正常行
> 车;到站停车。售票员的活动:关车门;售票;开车门。在汽车不断地到站、停车、行驶过程中,这两个活动有什么同步关系?用信号量和P 、V 操作实现它们的同步

- [x]

知识点:实验环境准备实验
出处:网络
难度:1

> 在汽车行驶过程中,司机活动与售票员活动之间的同步关系为:售票员关车门后, 向司机发开车信号,司机接到开车信号后启动车辆,在汽车正常行驶过程中售票员售
> 票,到站时司机停车,售票员在车停后开门让乘客上下车。因此,司机启动车辆的动作 必须与售票员关车门的动作取得同步;售票员开车门的动作也必须与司机停车取得同
> 步。应设置两个信号量:S1 、S2 ;
> S1表示是否允许司机启动汽车(其初值为0)
> S2表示是否允许售票员开门(其初值为0)

```
var S1,S2 : semaphore ;
  S1=0;S2=0;
cobegin
Procedure driver             Procedure Conductor
begin                        begin
 while  TRUE                  while TRUE
 begin                        begin
   P(S1);                       关车门;
   Start;                       V(s1);
   Driving;                     售票;
   Stop;                        P(s2);
   V(S2);                       开车门;
 end                            上下乘客;
```

```
end                          end
coend
```

4
某寺庙,有小和尚、老和尚若干.庙内有一水缸,由小和尚提水入缸,供老和尚 饮用。水缸可容纳10桶水,每次入水、取水仅为1桶,不可同时进行。水取自同一井中,
水井径窄,每次只能容纳一个水桶取水。设水桶个数为3个,试用信号灯和PV操作给出 老和尚和小和尚的活动。

- [x]

知识点:实验环境准备实验
出处:网络
难度:1

从井中取水并放入水缸是一个连续的动作可以视为一个进程,从缸中取水为另一 个进程。
设水井和水缸为临界资源,引入mutex1,mutex2;三个水桶无论从井中取水还是放
入水缸中都一次一个,应该给他们一个信号量count,抢不到水桶的进程只好为等待,
水缸满了时,不可以再放水了。设empty控制入水量,水缸空了时,不可取水设full。

```
var mutex1,mutex2,empty,full,count:semaphore;
mutex1:=mutex2:=1;
empty:=10;
full:=0;
count:=3;
cobegin
  Procedure Fetch_Water
    begin
     while true
      p(empty);
      P(count);
      P(mutex1);
       Get Water;
      v(mutex1);
      P(mutex2);
      pure water into the jar;
      v(mutex2);
      v(count);
      v(full);
    end
coend
Procedure Drink_Water
 begin
   while true
    p(full);
    p(count);
    p(mutex2);
      Get water and
      Drink water;
    p(mutex2);
    v(empty);
    v(count);
end
```

4
一座小桥(最多只能承重两个人)横跨南北两岸,任意时刻同一方向只允许一人过 桥,南侧桥段和北侧桥段较窄只能通过一人,桥中央一处宽敞,允许两个人通过或歇
息。试用信号灯和PV操作写出南、北两岸过桥的同步算法。

- [x]

知识点:实验环境准备实验
出处:网络
难度:1

桥上可能没有人,也可能有一人,也可能有两人。
两人同时过桥
两人都到中间
南(北)来者到北(南)段
np>共需要三个信号量,load用来控制桥上人数,初值为2,表示桥上最多有2人;north用
来控制北段桥的使用,初值为1,用于对北段桥互斥;south用来控制南段桥的使用,初 值为1,用于对南段桥互斥。

```
var load,north,south:semaphore;
load=2;
north=1;
south=1;
GO_South()
P(load);
P(north);
过北段桥;
到桥中间;
V(north);
P(south);
过南段桥;
到达南岸;
V(south);
```

```
V(load);
GO_North()
P(load);
P(south);
过南段桥;
到桥中间
V(south);
P(north);
过北段桥;
到达北岸
V(north);
V(load);
```

4
在一个盒子里,混装了数量相等的黑白围棋子·现在用自动分拣系统把黑子、白 子分开,设分拣系统有二个进程P1 和P2 ,其中P1 拣白子;P2
拣黑子。规定每个进程 每次拣一子;当一个进程在拣时,不允许另一个进程去拣;当一个进程拣了一子时,必 须让另一个进程去拣.试写出两进程P1 和P2
能并发正确执行的程序。

- [x]

知识点:实验环境准备实验
出处:网络
难度:1

大家熟悉了生产-消费问题(PC),这个问题很简单。题目较为新颖,但是本质非常 简单即:生产-消费问题的简化或者说是两个进程的简单同步问题。答案如下:
设信号量s1 和s2 分别表示可拣白子和黑子; 不失一般性,若令先拣白子。

```
var S1 , S2 : semaphore;
S1 : = l; S2 :=0;
cobegin
  process P1           Process P2
   begin                begin
   repeat               repeat
   P(S1);               P(S2);
   pick The white;      pick the black;
   V(S2);               V(S1);
  until false ;         until false;
  end                   end
coend
```

4
设公共汽车上,司机和售票员的活动分别如下:司机的活动:启动车辆:正常行
车;到站停车。售票员的活动:关车门;售票;开车门。在汽车不断地到站、停车、行驶过程中,这两个活动有什么同步关系?用信号量和P 、V 操作实现它们的同步

- [x]

知识点:实验环境准备实验
出处:网络
难度:1

在汽车行驶过程中,司机活动与售票员活动之间的同步关系为:售票员关车门后, 向司机发开车信号,司机接到开车信号后启动车辆,在汽车正常行驶过程中售票员售
票,到站时司机停车,售票员在车停后开门让乘客上下车。因此,司机启动车辆的动作 必须与售票员关车门的动作取得同步;售票员开车门的动作也必须与司机停车取得同
步。应设置两个信号量:S1 、S2 ;
S1表示是否允许司机启动汽车(其初值为0)
S2表示是否允许售票员开门(其初值为0)

```
var S1,S2 : semaphore ;
  S1=0;S2=0;
cobegin
Procedure driver             Procedure Conductor
begin                        begin
 while  TRUE                  while TRUE
 begin                        begin
   P(S1);                       关车门;
   Start;                       V(s1);
   Driving;                     售票;
   Stop;                        P(s2);
   V(S2);                       开车门;
 end                            上下乘客;
end                            end
coend
```

4
某寺庙,有小和尚、老和尚若干.庙内有一水缸,由小和尚提水入缸,供老和尚 饮用。水缸可容纳10桶水,每次入水、取水仅为1桶,不可同时进行。水取自同一井中,
水井径窄,每次只能容纳一个水桶取水。设水桶个数为3个,试用信号灯和PV操作给出 老和尚和小和尚的活动。

- [x]

知识点:实验环境准备实验
出处:网络
难度:1

> 从井中取水并放入水缸是一个连续的动作可以视为一个进程,从缸中取水为另一 个进程。
> 设水井和水缸为临界资源,引入mutex1,mutex2;三个水桶无论从井中取水还是放
> 入水缸中都一次一个,应该给他们一个信号量count,抢不到水桶的进程只好为等待,
> 水缸满了时,不可以再放水了。设empty控制入水量,水缸空了时,不可取水设full。

```
var mutex1,mutex2,empty,full,count:semaphore;
mutex1:=mutex2:=1;
empty:=10;
full:=0;
count:=3;
cobegin
  Procedure Fetch_Water
    begin
     while true
      p(empty);
      P(count);
      P(mutex1);
       Get Water;
      v(mutex1);
      P(mutex2);
      pure water into the jar;
      v(mutex2);
      v(count);
      v(full);
    end
coend
Procedure Drink_Water
 begin
   while true
    p(full);
    p(count);
    p(mutex2);
      Get water and
      Drink water;
    p(mutex2);
    v(empty);
    v(count);
end
```

> 4
> 一座小桥(最多只能承重两个人)横跨南北两岸,任意时刻同一方向只允许一人过 桥,南侧桥段和北侧桥段较窄只能通过一人,桥中央一处宽敞,允许两个人通过或歇
> 息。试用信号灯和PV操作写出南、北两岸过桥的同步算法。

- [x]

知识点:实验环境准备实验
出处:网络
难度:1

> 桥上可能没有人,也可能有一人,也可能有两人。
> 两人同时过桥
> 两人都到中间
> 南(北)来者到北(南)段
> np>共需要三个信号量,load用来控制桥上人数,初值为2,表示桥上最多有2人;north用
> 来控制北段桥的使用,初值为1,用于对北段桥互斥;south用来控制南段桥的使用,初 值为1,用于对南段桥互斥。

```
var load,north,south:semaphore;
load=2;
north=1;
south=1;
GO_South()
P(load);
P(north);
过北段桥;
到桥中间;
V(north);
P(south);
过南段桥;
到达南岸;
V(south);
V(load);
GO_North()
P(load);
P(south);
过南段桥;
到桥中间
V(south);
P(north);
过北段桥;
到达北岸
```

```
25  V(north);
26  V(load);
```

4
下面是ucore中用于按需分页处理过程的内核代码。请补全其中所缺的代码，以正确完成按需分页过程.
kern/trap/trap.h

```
    ------------------------------
    ...
    struct trapframe {
        struct pushregs tf_regs;
        uint16_t tf_es;
        uint16_t tf_padding1;
        uint16_t tf_ds;
        uint16_t tf_padding2;
        uint32_t tf_trapno;
        / below here defined by x86 hardware /
        uint32_t tf_err;
        uintptr_t tf_eip;
        uint16_t tf_cs;
        uint16_t tf_padding3;
        uint32_t tf_eflags;
        // below here only when crossing rings, such as from user to kernel
        uintptr_t tf_esp;
        uint16_t tf_ss;
        uint16_t tf_padding4;
    } __attribute__((packed));
    ...
    ------------------------------
    kern/trap/trap.c
    ------------------------------
    ...
    static int
    pgfault_handler(struct trapframe tf) {
        extern struct mm_struct check_mm_struct;
    }
    print_pgfault(tf);
        if (check_mm_struct != NULL) {
            return do_pgfault(check_mm_struct, tf->tf_err, rcr2());
        }
        panic("unhandled page fault.
");
    }
    static void
    trap_dispatch(struct trapframe tf) {
    char c;
    int ret;
    switch ( --YOUR CODE 1-- ) {
          .YOUR..
    case T_PGFLT:
    if ( --YOUR CODE 2-- ) != 0) {
    print_trapframe(trapf);
    if (current == NULL) {
    panic("handle pgfault failed. %e
",              ret);
    }
    else { ... }
    }
    break;
       ...
    }
    void
    trap(struct trapframe tf) {
        // dispatch based on what type of trap occurred
        trap_dispatch(tf);
    }
    ...
    // do_pgfault - interrupt handler to process the page fault execption
    int
    do_pgfault(struct mm_struct mm, uint32_t error_code, uintptr_t addr) {
        int ret = -E_INVAL;
        struct vma_struct vma = find_vma(mm, addr);
        if (vma == NULL || vma->vm_start > addr) {
            goto failed;
        }
        switch (error_code & 3) {
```

```
        default:
                / default is 3: write, present /
        case 2: / write, not present /
            if (!(vma->vm_flags & VM_WRITE)) {
                goto failed;
            }
            break;
        case 1: / read, present /
            goto failed;
        case 0: / read, not present /
            if (!(vma->vm_flags & (VM_READ | VM_EXEC))) {
                goto failed;
            }
        }
        uint32_t perm = PTE_U;
        if (vma->vm_flags & VM_WRITE) {
            perm |= PTE_W;
        }
        addr = ROUNDDOWN(addr, PGSIZE);
        ret = -E_NO_MEM;
        if (pgdir_alloc_page(mm->pgdir, addr, perm) == 0) {
            goto failed;
        }
        ret = 0;
    failed:
        return ret;
    }
    ...
    ------------------------------
    Pmm.h
    ------------------------------
    ...
    //ppn is physical page number
    static inline ppn_t
    page2ppn(struct Page page) {
        return --YOUR CODE 3--;
    }
    //pa is physical address
    static inline uintptr_t
    page2pa(struct Page page) {
        return --YOUR CODE 4--;
    }
    ...
    ------------------------------
    pmm.c
    ------------------------------
    ...
    // virtual address of physicall page array
    struct Page pages;
    // amount of physical memory (in pages)
    size_t npage = 0;
    // virtual address of boot-time page directory
    pde_t boot_pgdir = NULL;
    ……
    // pgdir_alloc_page - call alloc_page & page_insert functions to
    //                  - allocate a page size memory & setup an addr map
    //                  - pa<->la with linear address la and the PDT pgdir
    struct Page 
    pgdir_alloc_page(pde_t pgdir, uintptr_t la, uint32_t perm) {
        struct Page page = alloc_page();
        if (page != NULL) {
            if (page_insert(pgdir, page, la, perm) != 0) {
                free_page(page);
                return NULL;
            }
        }
        return page;
    }
    ...
    //page_insert - build the map of phy addr of an Page with the linear addr la
    // paramemters:
    //  pgdir: the kernel virtual base address of PDT
    //  page:  the Page which need to map
    //  la:    the linear address need to map
```

```
    //  perm:  the permission of this Page which is setted in related pte
    // return value: always 0
    //note: PT is changed, so the TLB need to be invalidate
    int
    page_insert(pde_t pgdir, struct Page page, uintptr_t la, uint32_t perm) {
        pte_t ptep = get_pte(pgdir, la, 1);
        if (ptep == NULL) {
            return -E_NO_MEM;
        }
        page_ref_inc(page);
        if (ptep & PTE_P) {
            struct Page p = pte2page(ptep);
            if (p == page) {
                page_ref_dec(page);
            }
            else {
                page_remove_pte(pgdir, la, ptep);
            }
        }
        ptep = --YOUR CODE 5--
        tlb_invalidate(pgdir, la);
        return 0;
    }
    ------------------------------
```

- [x]

知识点:
出处:网络
难度:1

> Code1： tf->tf_trapno Code 2: ret = pgfault_handler(tf) Code 3: page - pages;
> Code 4: page2ppn(page) << PGSHIFT Code 5: page2pa(page) | PTE_P | perm;
> ---------- 评分标准 5个空，每个3分； 第4个空中，对了前半部分，给2分；移位正确给1分；
> 第5个空中，每一个部分1分
> 4
> 在管道通信机制中,用信号量描述读进程和写进程访问管道文件的过程,假设管 道文件大小为10KB.

- [x]

知识点:实验环境准备实验
出处:网络
难度:1

> UNIX系统中,利用一个打开的共享文件来连接两个相互通信的进程,这个共享文 件叫管道.作为管道输入的发送进程,以字符流的形式将信息送入管道,而作为管道
> 输出的接收进程,从管道中获取信息.管道通信机制要提供三方面的协调能力:(1)互 斥.当一个进程对管道进行读/写操作时,另一个进程必须等待.(2)
> 同步.当写进程 把数据写入管道后便去睡眠等待,直到输出进程取走数据后唤醒它.若一次写入的数据
> 超过缓冲区剩余空间的大小,当缓冲区满时,写进程必须阻塞,并唤醒读进程。(3)对 方是否存在.只有确定对方存在时,才能够进行通信.本题只需要考虑互斥,同步问
> 题。由于只有一对进程访问管道,因此不需要设置互斥信号量,只要设置两个同步信号 量empty,full.分别表示管道可写和可读.

```
begin
pipe:array[09] of kilobytes;
ts=10,length,in=0,out=0:integer;
empty,full:semaphore=1,0;
cobegin
process PipeWriter
  begin
  repeat
  产生数据;
  p(empty);
  length = data length;
  while(length>0 and ts>0)
  begin
   pipe[in] = data of 1KB;
   in = (in+1) mod n;
   ts = ts-1;
   length = length - 1;
  end
  v(full);
  end
process Consumer
  begin
  repeat;
  p(full);
  从缓冲区取出一件物品;
  out = (out+1) mod n;
  ts = ts +1;
  v(empty);
  end
```

```
coend
end
```

4
桌上有一空盘,允许存放一只水果。爸爸可向盘中放苹果,也可向盘中放桔子,儿 子专等吃盘中的桔子,女儿专等吃盘中的苹果。规定当盘空时一次只能放一只水果供
吃者取用,请用P、V原语实现爸爸、儿子、女儿三个并发进程的同步

- [x]

知识点:实验环境准备实验
出处:网络
难度:1

在本题中,爸爸、儿子、女儿共用一个盘子,盘中一次只能放一个水果。当盘子为 空时,爸爸可将一个水果放入果盘中。若放入果盘中的是桔子,则允许儿子吃,女儿必
须等待;若放入果盘中的是苹果,则允许女儿吃,儿子必须等待。本题实际上是生产 者-消费者问题的一种变形。这里,生产者放入缓冲区的产品有两类,消费者也有两类,
每类消费者只消费其中固定的一类产品。
在本题中,应设置三个信号量S、So、Sa,信号量S表示盘子是否为空,其初值为l;
信号量So表示盘中是否有桔子,其初值为0;信号量Sa表示盘中是否有苹果,其初值为0。 同步描述如下:

```
S=1; Sa=0; So=0;
cobegin
 Procedure father;
 Procedure son;
 Procedure daughter;
coend
Procedure father:
  begin
  while(TRUE)
  begin
  P(S);
  将水果放入盘中;
  if(放入的是桔子)
  V(So);
  else
  V(Sa);
  end
  end
Procedure son:
  begin
  while(TRUE)
  begin
  P(So);
  从盘中取出桔子;
  V(S);
  吃桔子;
  end
  end
Procedure daughter:
  begin
  while(TRUE)
  begin
  P(Sa);
  从盘中取出苹果;
  V(S);
  吃苹果;
  end
end
```

4
在南开大学至天津大学间有一条弯曲的路,每次只允许一辆自行车通过,但中间
有小的安全岛M(同时允许两辆车),可供两辆车在已进入两端小车错车,设计算法并使用P,V实现。

- [x]

知识点:实验环境准备实验
出处:网络
难度:1

由于安全岛M仅仅允许两辆车停留,本应该作为临界资源而要设置信号量, 但根据 题意,任意时刻进入安全岛的车不会超过两辆(两个方向最多各有一辆), 因此,不需要
为M设置信号量,在路口s和路口t都需要设置信号量,以控制来自两个方向的车对路口资
源的争夺.这两个信号量的初值都是1.此外,由于从s到t的一段路只允许一辆车通过,所
以还需要设置另外的信号量用于控制,由于M的存在,可以为两端的小路分别设置一个互 斥信号量.

```
var T2N, N2T,L,M,K:semaphore;
T2N:=1;
N2T:=1;
L:=1;
K:=1;
M:=2;
cobegin
  Procedure Bike T2N
```

```
  begin
    p(T2N);
    p(L);
      go T to L;
    p(M);
      go into  M;
    V(L);
    P(k);
     go K to s;
    V(M);
    V(k);
    V(T2N);
  end
  Procedure Bike N2T
  begin
    P(N2T);
    p(k);
     go v to k;
    p(M);
      go into M;
    V(k);
    P(L);
      go L to T;
    V(M);
    V(L);
    V(N2T);
  end
coend
```

4
在一个盒子里,混装了数量相等的黑白围棋子·现在用自动分拣系统把黑子、白 子分开,设分拣系统有二个进程P1 和P2 ,其中P1 拣白子;P2
拣黑子。规定每个进程 每次拣一子;当一个进程在拣时,不允许另一个进程去拣;当一个进程拣了一子时,必 须让另一个进程去拣.试写出两进程P1 和P2
能并发正确执行的程序。

- [x]

知识点:实验环境准备实验
出处:网络
难度:1

大家熟悉了生产-消费问题(PC),这个问题很简单。题目较为新颖,但是本质非常 简单即:生产-消费问题的简化或者说是两个进程的简单同步问题。答案如下:
设信号量s1 和s2 分别表示可拣白子和黑子; 不失一般性,若令先拣白子。

```
var S1 , S2 : semaphore;
S1 : = l; S2 :=0;
cobegin
  process P1           Process P2
   begin                begin
   repeat               repeat
   P(S1);               P(S2);
   pick The white;      pick the black;
   V(S2);               V(S1);
  until false ;         until false;
  end                   end
coend
```

4
设公共汽车上,司机和售票员的活动分别如下:司机的活动:启动车辆:正常行
车;到站停车。售票员的活动:关车门;售票;开车门。在汽车不断地到站、停车、行驶过程中,这两个活动有什么同步关系?用信号量和P 、V 操作实现它们的同步

- [x]

知识点:实验环境准备实验
出处:网络
难度:1

在汽车行驶过程中,司机活动与售票员活动之间的同步关系为:售票员关车门后, 向司机发开车信号,司机接到开车信号后启动车辆,在汽车正常行驶过程中售票员售
票,到站时司机停车,售票员在车停后开门让乘客上下车。因此,司机启动车辆的动作 必须与售票员关车门的动作取得同步;售票员开车门的动作也必须与司机停车取得同
步。应设置两个信号量:S1 、S2 ;
S1表示是否允许司机启动汽车(其初值为0)
S2表示是否允许售票员开门(其初值为0)

```
var S1,S2 : semaphore ;
  S1=0;S2=0;
cobegin
Procedure driver             Procedure Conductor
begin                        begin
 while  TRUE                  while TRUE
 begin                        begin
```

```
    P(S1);                          关车门;
    Start;                          V(s1);
    Driving;                        售票;
    Stop;                           P(s2);
    V(S2);                          开车门;
 end                                上下乘客;
end                                end
coend
```

4
某寺庙,有小和尚、老和尚若干.庙内有一水缸,由小和尚提水入缸,供老和尚 饮用。水缸可容纳10桶水,每次入水、取水仅为1桶,不可同时进行。水取自同一井中,
水井径窄,每次只能容纳一个水桶取水。设水桶个数为3个,试用信号灯和PV操作给出 老和尚和小和尚的活动。

- [x]

知识点:实验环境准备实验
出处:网络
难度:1

从井中取水并放入水缸是一个连续的动作可以视为一个进程,从缸中取水为另一 个进程。
设水井和水缸为临界资源,引入mutex1,mutex2;三个水桶无论从井中取水还是放
入水缸中都一次一个,应该给他们一个信号量count,抢不到水桶的进程只好为等待,
水缸满了时,不可以再放水了。设empty控制入水量,水缸空了时,不可取水设full。

```
var mutex1,mutex2,empty,full,count:semaphore;
mutex1:=mutex2:=1;
empty:=10;
full:=0;
count:=3;
cobegin
  Procedure Fetch_Water
    begin
     while true
      p(empty);
      P(count);
      P(mutex1);
       Get Water;
      v(mutex1);
      P(mutex2);
      pure water into the jar;
      v(mutex2);
      v(count);
      v(full);
    end
coend
Procedure Drink_Water
 begin
   while true
    p(full);
    p(count);
    p(mutex2);
      Get water and
      Drink water;
    p(mutex2);
    v(empty);
    v(count);
end
```

4
一座小桥(最多只能承重两个人)横跨南北两岸,任意时刻同一方向只允许一人过 桥,南侧桥段和北侧桥段较窄只能通过一人,桥中央一处宽敞,允许两个人通过或歇
息。试用信号灯和PV操作写出南、北两岸过桥的同步算法。

- [x]

知识点:实验环境准备实验
出处:网络
难度:1

桥上可能没有人,也可能有一人,也可能有两人。
两人同时过桥
两人都到中间
南(北)来者到北(南)段
np>共需要三个信号量,load用来控制桥上人数,初值为2,表示桥上最多有2人;north用
来控制北段桥的使用,初值为1,用于对北段桥互斥;south用来控制南段桥的使用,初 值为1,用于对南段桥互斥。

```
var load,north,south:semaphore;
load=2;
north=1;
south=1;
GO_South()
P(load);
P(north);
过北段桥;
```

```
到桥中间;
V(north);
P(south);
过南段桥;
到达南岸;
V(south);
V(load);
GO_North()
P(load);
P(south);
过南段桥;
到桥中间
V(south);
P(north);
过北段桥;
到达北岸
V(north);
V(load);
```

4

下面是ucore中用于按需分页处理过程的内核代码。请补全其中所缺的代码，以正确完成按需分页过程.

kern/trap/trap.h

```
    ------------------------------
    ...
    struct trapframe {
        struct pushregs tf_regs;
        uint16_t tf_es;
        uint16_t tf_padding1;
        uint16_t tf_ds;
        uint16_t tf_padding2;
        uint32_t tf_trapno;
        / below here defined by x86 hardware /
        uint32_t tf_err;
        uintptr_t tf_eip;
        uint16_t tf_cs;
        uint16_t tf_padding3;
        uint32_t tf_eflags;
        // below here only when crossing rings, such as from user to kernel
        uintptr_t tf_esp;
        uint16_t tf_ss;
        uint16_t tf_padding4;
    } __attribute__((packed));
    ...
    ------------------------------
    kern/trap/trap.c
    ------------------------------
    ...
    static int
    pgfault_handler(struct trapframe tf) {
        extern struct mm_struct check_mm_struct;
    }
    print_pgfault(tf);
        if (check_mm_struct != NULL) {
            return do_pgfault(check_mm_struct, tf->tf_err, rcr2());
        }
        panic("unhandled page fault.
");
    }
    static void
    trap_dispatch(struct trapframe tf) {
    char c;
    int ret;
    switch ( --YOUR CODE 1-- ) {
          .YOUR..
    case T_PGFLT:
    if ( --YOUR CODE 2-- ) != 0) {
    print_trapframe(trapf);
    if (current == NULL) {
    panic("handle pgfault failed. %e
",              ret);
    }
    else { ... }
    }
    break;
      ...
    }
    void
```

```
    trap(struct trapframe tf) {
        // dispatch based on what type of trap occurred
        trap_dispatch(tf);
    }
    ...
    // do_pgfault - interrupt handler to process the page fault execption
    int
    do_pgfault(struct mm_struct mm, uint32_t error_code, uintptr_t addr) {
        int ret = -E_INVAL;
        struct vma_struct vma = find_vma(mm, addr);
        if (vma == NULL || vma->vm_start > addr) {
            goto failed;
        }
        switch (error_code & 3) {
        default:
                / default is 3: write, present /
        case 2: / write, not present /
            if (!(vma->vm_flags & VM_WRITE)) {
                goto failed;
            }
            break;
        case 1: / read, present /
            goto failed;
        case 0: / read, not present /
            if (!(vma->vm_flags & (VM_READ | VM_EXEC))) {
                goto failed;
            }
        }
        uint32_t perm = PTE_U;
        if (vma->vm_flags & VM_WRITE) {
            perm |= PTE_W;
        }
        addr = ROUNDDOWN(addr, PGSIZE);
        ret = -E_NO_MEM;
        if (pgdir_alloc_page(mm->pgdir, addr, perm) == 0) {
            goto failed;
        }
        ret = 0;
    failed:
        return ret;
    }
    ...
    ------------------------------
    Pmm.h
    ------------------------------
    ...
    //ppn is physical page number
    static inline ppn_t
    page2ppn(struct Page page) {
        return --YOUR CODE 3--;
    }
    //pa is physical address
    static inline uintptr_t
    page2pa(struct Page page) {
        return --YOUR CODE 4--;
    }
    ...
    ------------------------------
    pmm.c
    ------------------------------
    ...
    // virtual address of physicall page array
    struct Page pages;
    // amount of physical memory (in pages)
    size_t npage = 0;
    // virtual address of boot-time page directory
    pde_t boot_pgdir = NULL;
    ……
    // pgdir_alloc_page - call alloc_page & page_insert functions to
    //                  - allocate a page size memory & setup an addr map
    //                  - pa<->la with linear address la and the PDT pgdir
    struct Page 
    pgdir_alloc_page(pde_t pgdir, uintptr_t la, uint32_t perm) {
        struct Page page = alloc_page();
```

```
        if (page != NULL) {
            if (page_insert(pgdir, page, la, perm) != 0) {
                free_page(page);
                return NULL;
            }
        }
        return page;
    }
    ...
    //page_insert - build the map of phy addr of an Page with the linear addr la
    // paramemters:
    //  pgdir: the kernel virtual base address of PDT
    //  page:  the Page which need to map
    //  la:    the linear address need to map
    //  perm:  the permission of this Page which is setted in related pte
    // return value: always 0
    //note: PT is changed, so the TLB need to be invalidate
    int
    page_insert(pde_t pgdir, struct Page page, uintptr_t la, uint32_t perm) {
        pte_t ptep = get_pte(pgdir, la, 1);
        if (ptep == NULL) {
            return -E_NO_MEM;
        }
        page_ref_inc(page);
        if (ptep & PTE_P) {
            struct Page p = pte2page(ptep);
            if (p == page) {
                page_ref_dec(page);
            }
            else {
                page_remove_pte(pgdir, la, ptep);
            }
        }
        ptep = --YOUR CODE 5--
        tlb_invalidate(pgdir, la);
        return 0;
    }
    --------------------------------
```

- [x]

知识点:
出处:网络
难度:1

Code1： tf->tf_trapno Code 2: ret = pgfault_handler(tf) Code 3: page - pages;
Code 4: page2ppn(page) << PGSHIFT Code 5: page2pa(page) | PTE_P | perm;
---------- 评分标准 5个空，每个3分； 第4个空中，对了前半部分，给2分；移位正确给1分；
第5个空中，每一个部分1分
4
fork例子
第二题： 一、 （10分）给出程序fork.c的输出结果。
注：1）getpid()和getppid()是两个系统调用，分别返回本进程标识和父进程标识。
2）你可以假定每次新进程创建时生成的进程标识是顺序加1得到的；在进程标识为1000的命令解释程序shell中启动该程序的执行
```
  nclude
  #include
  / getpid() and fork() are system calls declared in unistd.h.  They return /
  / values of type pid_t.  This pid_t is a special type for process ids. /
  / It's equivalent to int. /
  int main(void)
  {
    pid_t childpid;
    int x = 5;
        int i;
    childpid = fork();
    for ( i = 0;  i < 2;  i++) {
      printf(This is process %d; childpid = %d; The parent of this process has id %d; i = %d; x = %d
"
getpid()
```

- [x]

知识点:
出处:网络
难度:1

> getppid()
> 1
> 哪种设备属于块设备?

- ( ) A.键盘
- (x) B.磁盘
- ( ) C.显示器
- ( ) D.打印机

知识点:操作系统概述
出处:网络
难度:1

> B
> 1
> 年龄

- ( ) A.19岁以下
- (x) B.20~29岁
- ( ) C.30~39岁
- ( ) D.40岁以上

知识点:调查问卷
出处:网络
难度:1

> 解释：统计学生的年龄组成情况。
> 1
> 性别

- (x) A.男
- ( ) B.女

知识点:调查问卷
出处:网络
难度:1

> 解释：统计学生的性别组成情况
> 1
> 所在地区

- (x) A.北京
- ( ) B.其他地区

知识点:调查问卷
出处:网络
难度:1

> 解释：统计学生的地理位置分布情况。
> 2
> 为什么要学这门课?

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

知识点:调查问卷
出处:网络
难度:1

> 解释
> 1
> （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) A.系统调用
- ( ) B.图形用户
- ( ) C.原语
- ( ) D.键盘命令

知识点:操作系统概述
出处:网络
难度:1

> A 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
> 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。
> 1
> (2009计算机统考)单处理器系统中，可并行执行或工作的对象是（ ）

```
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
```

- ( ) A.1 2 3

- ( ) B.1 2 4
- ( ) C.1 3 4
- (x) D.2 3 4

知识点:操作系统概述
出处:网络
难度:1

> D 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。
> 1
> (2010统考）下列选项中，操作系统提供给应用程序的接口是（ ）

- (x) A.系统调用
- ( ) B.中断
- ( ) C.库函数
- ( ) D.原语

知识点:操作系统概述
出处:网络
难度:1

> A
> 1
> (2011统考)下列选项中，在用户态执行的是（ ）

- (x) A.命令解释程序
- ( ) B.缺页处理程序
- ( ) C.进程调度程序
- ( ) D.时钟中断处理程序

知识点:操作系统概述
出处:网络
难度:1

> A 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。
> 1
> (2013联考)计算机开机后，操作系统最终被加载到（ ）

- ( ) A.BIOS
- ( ) B.ROM
- ( ) C.EPROM
- (x) D.RAM

知识点:操作系统概述
出处:网络
难度:1

> D 操作系统被加载到内存（RAM）中。
> 1
> 操作系统属于

- ( ) A.硬件
- (x) B.系统软件
- ( ) C.通用库
- ( ) D.应用软件

知识点:操作系统概述
出处:网络
难度:1

> 解释：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。
> 1
> 以下不属于操作系统的功能是（）

- ( ) A.进程调度
- ( ) B.内存管理
- (x) C.视频编辑
- ( ) D.设备驱动

知识点:操作系统概述
出处:网络
难度:1

> C 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。
> 1
> 操作系统中的多道程序设计方式用于提高

- ( ) A.稳定性
- (x) B.效率

- ( ) C.兼容性
- ( ) D.可靠性

知识点:操作系统概述
出处:网络
难度:1

> B 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。
> 1
> 下面对于分时操作系统的说法，正确的是（）

- ( ) A.应用程序执行的先后顺序是完全随机的
- ( ) B.应用程序按照启动的时间依次执行
- (x) C.应用程序可以交替执行
- ( ) D.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

知识点:操作系统概述
出处:网络
难度:1

> C 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。
> 1
> Unix操作系统属于()

- (x) A.分时操作系统
- ( ) B.批处理操作系统
- ( ) C.实时操作系统
- ( ) D.分布式操作系统

知识点:操作系统概述
出处:网络
难度:1

> A 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。
> 1
> 批处理的主要缺点是()

- ( ) A.效率低
- ( ) B.失去了交互性
- (x) C.失去了并行性
- ( ) D.以上都不是

知识点:操作系统概述
出处:网络
难度:1

> C 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。
> 2
> 关于操作系统，说法正确的是（）

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

知识点:操作系统概述
出处:网络
难度:1

> ABC 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是不对的。
> 2
> 设备管理的功能包括（）

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

知识点:操作系统概述
出处:网络
难度:1

> ACD 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
> 2
> 多道批处理系统主要考虑的是()

- [ ] A.交互性
- [ ] B.及时性
- [x] C.系统效率
- [x] D.吞吐量

知识点:操作系统概述
出处:网络
难度:1

> CD 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。
> 4
> 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]

知识点:实验环境准备
出处:网络
难度:1

> http://www.imada.sdu.dk/Courses/DM18/Litteratur/IntelnATT.htm inb一般应用程序用不到的指令等。
> 4
> 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]

知识点:实验环境准备
出处:网络
难度:1

> 中断寄存器和非通用寄存器等。
> 4
> 哪些困难（请分优先级）会阻碍你自主完成lab实验？

- [x]

知识点:实验环境准备
出处:网络
难度:1

> 4
> 你希望从lab中学到什么知识？

- [x]

知识点:实验环境准备
出处:网络
难度:1

> 4
> 搭建好实验环境，请描述碰到的困难和解决的过程。

- [x]

知识点:实验环境准备
出处:网络
难度:1

> 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
> 4
> 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]

知识点:实验环境准备
出处:网络
难度:1

> clone 仓库 gitclone http://www.github.com/chyyuu/ucore_lab
> 4
> 尝试用qemu+gdb（or ECLIPSE-CDT）调试lab1

- [x]

知识点:实验环境准备
出处:网络
难度:1

> 清除文件夹：make clean 编译lab1：make 调出debug命令行：make debug
> 1
> 如何从用户方式（用户态）转入特权方式（核心态）？

- ( ) A.使用特权指令
- ( ) B.发生子程序调用
- ( ) C.使用共享代码
- (x) D.进行系统调用

知识点:操作系统概述
出处:网络
难度:1

> D
> 4
> 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]

知识点:实验环境准备
出处:网络
难度:1

> 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)。
> 5
> 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[ ]?

- [x]

知识点:实验环境准备
出处:网络
难度:1

> 0x10002 https://github.com/chyyuu/ucore_lab/blob/master/related_info/lab0/lab0_ex3.c
> 4
> 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]

知识点:实验环境准备
出处:网络
难度:1

> 4
> 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]

知识点:实验环境准备
出处:网络
难度:1

> 4
> 了解函数调用栈对lab实验有何帮助?

- [x]

知识点:实验环境准备
出处:网络
难度:1

> 4
> 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]

知识点:实验环境准备
出处:网络
难度:1

> 4
> 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]

知识点:实验环境准备
出处:网络
难度:1

> 4
> 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]

知识点:实验环境准备
出处:网络
难度:1

> 4
> 请给出你觉得的更准确的操作系统的定义?

- [x]

知识点:实验环境准备
出处:网络
难度:1

> 4
> 你希望从操作系统课学到什么知识?

- [x]

知识点:实验环境准备
出处:网络
难度:1

> 4
> 目前的台式PC机标准配置和价格?

- [x]

知识点:实验环境准备
出处:网络
难度:1

> 4
> 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]

知识点:实验环境准备
出处:网络
难度:1

> 4
> 为什么现在的操作系统基本上用C语言来实现?

- [x]

知识点:实验环境准备
出处:网络
难度:1

> 4
> 为什么没有人用python，java来实现操作系统?

- [x]

知识点:实验环境准备
出处:网络
难度:1

> 4
> 请评价用C++来实现操作系统的利弊?

- [x]

知识点:实验环境准备
出处:网络
难度:1

> 4
> 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]

知识点:实验环境准备
出处:网络
难度:1

> 4
> 请评价用LISP,OCcaml, GO, D，RUST等实现操作系统的利弊?

- [x]

知识点:实验环境准备
出处:网络
难度:1

> 4
> 进程切换的可能实现思路?

- [x]

知识点:实验环境准备
出处:网络
难度:1

> 4
> 计算机与终端间通过串口通信的可能实现思路?

- [x]

知识点:实验环境准备
出处:网络
难度:1

> 4
> 为什么微软的Windows没有在手机终端领域取得领先地位?

- [x]

知识点:实验环境准备
出处:网络
难度:1

> 4
> 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]

知识点:实验环境准备
出处:网络
难度:1

> 1
> 清华大学目前的操作系统实验中采用的OS对象是()

- ( ) A.Linux
- (x) B.ucore
- ( ) C.xv6
- ( ) D.Nachos

知识点:实验环境准备
出处:网络
难度:1

> B 是参考了xv6, OS161, Linux的教学操作系统ucore OS
> 1
> 在ucore lab的实验环境搭建中，使用的非开源软件是()

- ( ) A.eclipse CDT
- (x) B.Scitools Understand
- ( ) C.gcc

- ( ) D.qemu

知识点:实验环境准备
出处:网络
难度:1

> B 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。
> 1
> 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()

- ( ) A.apt
- ( ) B.git
- ( ) C.meld
- (x) D.qemu

知识点:实验环境准备
出处:网络
难度:1

> D qemu是一个支持模拟多种CPU的模拟软件
> 1
> ucore lab实验中8个实验是否可以不按顺序完成

- ( ) A.是
- (x) B.否

知识点:实验环境准备
出处:ucore
难度:1

> B 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
> 1
> 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) A.是
- ( ) B.否

知识点:实验环境准备
出处:网络
难度:1

> 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
> 2
> 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] A.实模式
- [x] B.保护模式
- [ ] C.SMM模式
- [ ] D.虚拟8086模式

知识点:实验环境准备
出处:网络
难度:1

> AB ucore需要碰到和处理16位的实模式和32位的保护模式
> 4
> 请完成piazza讨论区访问的设置，并熟悉piazza的使用。然后在操作系统课的piazza讨论区中找到“开课通知”的访问链接。

- [x]

知识点:实验环境准备
出处:网络
难度:1

> 4
> 在操作系统课的piazza讨论区中找到标签为“学习方法”的问答中关于“如何看内核源代码？”的访问链接。

- [x]

知识点:实验环境准备
出处:网络
难度:1

> 4
> 请在操作系统课程的“代码编辑器”中采用直接输入路径的方式打开“lab5/libs”目录下的defs.h文件。

- [x]

知识点:实验环境准备
出处:网络
难度:1

> 1
> 以下那个不是程序顺序执行的特性

- ( ) A.封闭性
- ( ) B.顺序性
- ( ) C.无关性

- (x) D.不可再现性

知识点:操作系统概述
出处:网络
难度:1

> D
> 4
> 请在（1）中打开的文件defs.h中添加“#TEST”注释，并将修改后的文件保存到git上。提交的commit messege为“code-edit test”.

- [x]

知识点:实验环境准备
出处:网络
难度:1

> 1
> (2012统考)下列选项中，不可能在用户态发生的是（ ）

- ( ) A.系统调用
- ( ) B.外部中断
- (x) C.进程切换
- ( ) D.缺页

知识点:中断、异常与系统调用
出处:网络
难度:1

> C 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。
> 1
> （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（ ）

- ( ) A.程序计数器
- (x) B.程序状态字寄存器
- ( ) C.通用数据寄存器
- ( ) D.通用地址寄存器

知识点:中断、异常与系统调用
出处:网络
难度:1

> B 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。
> 1
> （华中科技大学）中断向量地址是（ ）

- ( ) A.子程序入口地址
- (x) B.中断服务例程入口地址
- ( ) C.中断服务例程入口地址的地址
- ( ) D.例行程序入口地址

知识点:中断、异常与系统调用
出处:网络
难度:1

> B
> 1
> 下列选项中， 可以执行特权指令?

- (x) A.中断处理例程
- ( ) B.普通用户的程序
- ( ) C.通用库函数
- ( ) D.管理员用户的程序

知识点:中断、异常与系统调用
出处:网络
难度:1

> A 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?
> 1
> 一般来讲，中断来源于（）

- (x) A.外部设备
- ( ) B.应用程序主动行为
- ( ) C.操作系统主动行为
- ( ) D.软件故障

知识点:中断、异常与系统调用
出处:网络
难度:1

> A 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。
> 1
> 用户程序通过向操作系统提出访问外部设备的请求

- ( ) A.I/O指令
- (x) B.系统调用
- ( ) C.中断
- ( ) D.创建新的进程

知识点:中断、异常与系统调用
出处:网络
难度:1

> B 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。
> 1
> 应用程序引发异常的时候，操作系统可能的反应是（）

- ( ) A.删除磁盘上的应用程序
- ( ) B.重启应用程序
- (x) C.杀死应用程序
- ( ) D.修复应用程序中的错误

知识点:中断、异常与系统调用
出处:网络
难度:1

> C 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?
> 1
> 操作系统与用户的接口包括

- (x) A.系统调用
- ( ) B.进程调度
- ( ) C.中断处理
- ( ) D.程序编译

知识点:中断、异常与系统调用
出处:网络
难度:1

> A 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。
> 2
> 操作系统处理中断的流程包括（）

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

知识点:中断、异常与系统调用
出处:网络
难度:1

> ABCD 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。
> 2
> 下列程序工作在内核态的有()

- [x] A.系统调用的处理程序

- [x] B.中断处理程序
- [x] C.进程调度
- [x] D.内存管理

知识点:中断、异常与系统调用
出处:网络
难度:1

> ABCD 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
> 4
> Linux的系统调用有哪些？大致的功能分类有哪些？

- [x]

知识点:中断、异常与系统调用
出处:网络
难度:1

> 4
> 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？

- [x]

知识点:中断、异常与系统调用
出处:网络
难度:1

> 1
> (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（ ）

- ( ) A.自由态
- ( ) B.目态
- (x) C.管态
- ( ) D.就绪态

知识点:中断、异常与系统调用
出处:网络
难度:1

> C 内核态又称为管态
> 1
> （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
```

- ( ) A.1、2
- (x) B.1、3
- ( ) C.2、3
- ( ) D.1、2、3

知识点:中断、异常与系统调用
出处:网络
难度:1

> B 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。
> 1
> 系统调用的主要作用是（）

- ( ) A.处理硬件问题
- ( ) B.应对软件异常
- (x) C.给应用程序提供服务接口
- ( ) D.管理应用程序

知识点:中断、异常与系统调用
出处:网络
难度:1

> C 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）。
> 1
> 下列关于系统调用的说法错误的是（）

- ( ) A.系统调用一般有对应的库函数

- (x) B.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
- ( ) C.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
- ( ) D.系统调用可能执行失败

知识点:中断、异常与系统调用
出处:网络
难度:1

> B 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。
> 1
> 以下关于系统调用和常规调用的说法中，错误的是（）

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

知识点:中断、异常与系统调用
出处:网络
难度:1

> D 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。
> 4
> 通过分析lab1_ex0了解Linux应用的系统调用编写和含义。

- [x]

知识点:中断、异常与系统调用
出处:网络
难度:1

> 4
> 通过调试lab1_ex1了解Linux应用的系统调用执行过程

- [x]

知识点:中断、异常与系统调用
出处:网络
难度:1

> 4
> 举例说明Linux中有哪些中断，哪些异常?

- [x]

知识点:中断、异常与系统调用
出处:网络
难度:1

> 4
> Linux的系统调用有哪些？大致的功能分类有哪些？(w2l1)

- [x]

知识点:中断、异常与系统调用
出处:网络
难度:1

> - 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）
>
> 4
> 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)

- [x]

知识点:中断、异常与系统调用
出处:网络
难度:1

> - 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）
>
> 4
> 比较UEFI和BIOS的区别。

- [x]

知识点:中断、异常与系统调用
出处:网络
难度:1

> 4
> 描述PXE的大致启动流程。

- [x]

知识点:中断、异常与系统调用
出处:网络
难度:1

> 4
> 了解NTLDR的启动流程。

- [x]

知识点:中断、异常与系统调用
出处:网络
难度:1

> 4
> 了解GRUB的启动流程。

- [x]

知识点:中断、异常与系统调用
出处:网络
难度:1

> 4
> 比较NTLDR和GRUB的功能有差异。

- [x]

知识点:中断、异常与系统调用
出处:网络
难度:1

> 4
> 了解u-boot的功能。

- [x]

知识点:中断、异常与系统调用
出处:网络
难度:1

> 4
> ucore的系统调用中参数传递代码分析。

- [x]

知识点:中断、异常与系统调用
出处:网络
难度:1

> 4
> ucore的系统调用中返回结果的传递代码分析。

- [x]

知识点:中断、异常与系统调用
出处:网络
难度:1

> 4
> 以ucore lab8的answer为例，分析ucore 应用的系统调用编写和含义。

- [x]

知识点:中断、异常与系统调用
出处:网络
难度:1

> 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。

- [x]

知识点:中断、异常与系统调用
出处:网络
难度:1

> 4
> 请分析函数调用和系统调用的区别,请从代码编写和执行过程来说明。说明int、iret、call和ret的指令准确功能.

- [x]

知识点:中断、异常与系统调用
出处:网络
难度:1

> 1
> 80386机器加电启动后，CPU立刻跳转到()执行

- ( ) A.ucore第一条指令
- ( ) B.bootloader第一条指令
- (x) C.BIOS的第一条指令
- ( ) D.GRUB的第一条指令

知识点:中断、异常与系统调用
出处:网络
难度:1

> C 是调到BIOS去执行
> 1
> 应用程序中的C函数调用中不需要用到()指令

- ( ) A.push
- ( ) B.ret
- (x) C.iret
- ( ) D.call

知识点:中断、异常与系统调用
出处:网络
难度:1

> C iret用于中断返回
> 1
> GCC内联汇编 asm("movl %ecx, %eax"); 的含义是()

- (x) A.把 ecx 内容移动到 eax
- ( ) B.把 eax 内容移动到 ecx

知识点:中断、异常与系统调用
出处:网络
难度:1

> A 把 ecx 内容移动到 eax
> 2
> 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] A.全局描述符表
- [x] B.中断描述符表
- [x] C.中断服务例程
- [x] D.内核堆栈

知识点:中断、异常与系统调用
出处:网络
难度:1

> ABCD 在ucore处理中，上述几个都是要设置好的。
> 4
> 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)

- [x]

知识点:中断、异常与系统调用
出处:网络
难度:1

> - 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）
>
> 4
> lab1中完成了对哪些外设的访问？(w2l2)

- [x]

知识点:中断、异常与系统调用
出处:网络
难度:1

> - 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）
>
> 4
> 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)

- [x]

知识点:中断、异常与系统调用
出处:网络
难度:1

> - 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）
>
> 4
> 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]

知识点:中断、异常与系统调用
出处:网络
难度:1

> 4
> 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]

知识点:中断、异常与系统调用
出处:网络
难度:1

> 4
> 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]

知识点:中断、异常与系统调用
出处:网络
难度:1

> 4
> 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]

知识点:中断、异常与系统调用
出处:网络
难度:1

> 4
> 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]

知识点:中断、异常与系统调用
出处:网络
难度:1

> 4
> 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]

知识点:中断、异常与系统调用
出处:网络
难度:1

> 4
> 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]

知识点:中断、异常与系统调用
出处:网络
难度:1

> 4
> 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]

知识点:中断、异常与系统调用
出处:网络
难度:1

> 4
> 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]

知识点:中断、异常与系统调用
出处:网络
难度:1

> 4
> 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]

知识点:中断、异常与系统调用
出处:网络
难度:1

> 4
> 读入ucore内核的代码？

- [x]

知识点:中断、异常与系统调用
出处:网络
难度:1

> 4
> 跳转到ucore内核的代码？

- [x]

知识点:中断、异常与系统调用
出处:网络
难度:1

> 4
> 全局描述符表的初始化代码？

- [x]

知识点:中断、异常与系统调用
出处:网络
难度:1

> 4
> 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]

知识点:中断、异常与系统调用
出处:网络
难度:1

> 4
> 可执行文件格式elf的各个段的数据结构？

- [x]

知识点:中断、异常与系统调用
出处:网络
难度:1

> 4
> 如果ucore内核的elf是否要求连续存放？为什么？

- [x]

知识点:中断、异常与系统调用
出处:网络
难度:1

> 4
> 函数调用的stackframe结构？函数调用的参数传递方法有哪几种？

- [x]

知识点:中断、异常与系统调用
出处:网络
难度:1

> 4
> 系统调用的stackframe结构？系统调用的参数传递方法有哪几种？

- [x]

知识点:中断、异常与系统调用
出处:网络
难度:1

> 4
> 使用内联汇编的原因?

- [x]

知识点:中断、异常与系统调用
出处:网络
难度:1

> 特权指令、性能优化
> 4
> 对ucore中的一段内联汇编进行完整的解释?

- [x]

知识点:中断、异常与系统调用
出处:网络
难度:1

> 4
> 中断描述符表IDT的结构?

- [x]

知识点:中断、异常与系统调用
出处:网络
难度:1

> 4
> 中断描述表到中断服务例程的地址计算过程?

- [x]

知识点:中断、异常与系统调用
出处:网络
难度:1

> 4
> 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]

知识点:中断、异常与系统调用
出处:网络
难度:1

> 4
> 中断处理中硬件保存了哪些寄存器?

- [x]

知识点:中断、异常与系统调用
出处:网络
难度:1

> 4
> 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]

知识点:中断、异常与系统调用
出处:网络
难度:1

> 4
> 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]

知识点:中断、异常与系统调用
出处:网络
难度:1

> 4
> 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]

知识点:中断、异常与系统调用
出处:网络
难度:1

> 4
> 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]

知识点:中断、异常与系统调用
出处:网络
难度:1

> 4
> qemu的命令行参数含义解释?

- [x]

知识点:中断、异常与系统调用
出处:网络
难度:1

> 4
> gdb命令格式? 反汇编、运行、断点设置

- [x]

知识点:中断、异常与系统调用
出处:网络
难度:1

> 4
> A20的使能代码分析?

- [x]

知识点:中断、异常与系统调用
出处:网络
难度:1

> 4
> qemu的命令行参数含义解释?

- [x]

知识点:中断、异常与系统调用
出处:网络
难度:1

> 4
> 生成主引导扇区的过程分析?

- [x]

知识点:中断、异常与系统调用
出处:网络
难度:1

> 4
> 保护模式的切换代码?

- [x]

知识点:中断、异常与系统调用
出处:网络
难度:1

> 4
> 如何识别elf格式? 对应代码分析? ?

- [x]

知识点:中断、异常与系统调用
出处:网络
难度:1

> 4
> 跳转到elf的代码?

- [x]

知识点:中断、异常与系统调用
出处:网络
难度:1

> 4
> 函数调用栈获取?

- [x]

知识点:中断、异常与系统调用
出处:网络
难度:1

> 4
> 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]

知识点:中断、异常与系统调用
出处:网络
难度:1

> 4
> 如何识别elf格式？对应代码分析？？

- [x]

知识点:中断、异常与系统调用
出处:网络
难度:1

> 4
> 各种设备的中断初始化?

- [x]

知识点:中断、异常与系统调用
出处:网络
难度:1

> 4
> 中断描述符表IDT的排列顺序?

- [x]

知识点:中断、异常与系统调用
出处:网络
难度:1

> 4
> 中断号 CPU加电初始化后中断是使能的吗？为什么？1282.md

- [x]

知识点:中断、异常与系统调用
出处:网络
难度:1

> 4
> 中断服务例程的入口地址在什么地方设置的?

- [x]

知识点:中断、异常与系统调用
出处:网络
难度:1

> 4
> alltrap的中断号是在哪写入到trapframe结构中的?

- [x]

知识点:中断、异常与系统调用
出处:网络
难度:1

> 4
> trapframe结构?

- [x]

知识点:中断、异常与系统调用
出处:网络
难度:1

> 1
> 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()

- ( ) A.物理地址空间
- (x) B.逻辑地址空间
- ( ) C.外设地址空间
- ( ) D.都不是

知识点:物理内存管理:
出处:网络
难度:1

> B 用户进程访问的内存地址是虚拟地址
> 1
> 在使能分页机制的情况下，更合适的外碎片整理方法是()

- ( ) A.紧凑(compaction)
- ( ) B.分区对换(Swapping in/out)
- (x) C.都不是

知识点:物理内存管理:
出处:网络
难度:1

> C 分页方式不会有外碎片
> 2
> 操作系统中可采用的内存管理方式包括()

- [x] A.重定位(relocation)
- [x] B.分段(segmentation
- [x] C.分页(paging)
- [x] D.段页式（segmentation+paging）

知识点:物理内存管理:
出处:网络
难度:1

> ABCD 都有
> 2
> 连续内存分配的算法中，会产生外碎片的是()

- [x] A.最先匹配算法
- [x] B.最差匹配算法
- [x] C.最佳匹配算法
- [ ] D.都不会

知识点:物理内存管理:
出处:网络
难度:1

> ABC 三种算法都会有外碎片
> 2
> 描述伙伴系统(Buddy System)特征正确的是()

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

知识点:物理内存管理:
出处:网络
难度:1

> ABC 前三个是对的。
> 4
> 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)

- [x]

知识点:物理内存管理
出处:网络
难度:1

> - 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）
>
> 4
> 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
```

- [x]

知识点:物理内存管理
出处:网络
难度:1

> 4
> 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。

- [x]

知识点:物理内存管理
出处:网络
难度:1

> 4
> MMU的工作机理?

- [x]

知识点:物理内存管理
出处:网络
难度:1

> http://en.wikipedia.org/wiki/Memory_management_unit
> 4
> L1和L2高速缓存有什么区别?

- [x]

知识点:物理内存管理
出处:网络
难度:1

> 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
> 4
> 编译、链接和加载的过程了解?

- [x]

知识点:物理内存管理
出处:网络
难度:1

> 4
> 动态链接如何使用?

- [x]

知识点:物理内存管理
出处:网络
难度:1

> 4
> 什么是内碎片、外碎片?

- [x]

知识点:物理内存管理
出处:网络
难度:1

> 4
> 为什么最先匹配会越用越慢?

- [x]

知识点:物理内存管理
出处:网络
难度:1

> 4
> 为什么最差匹配会的外碎片少?

- [x]

知识点:物理内存管理
出处:网络
难度:1

> 4
> 在几种算法中分区释放后的合并处理如何做?

- [x]

知识点:物理内存管理
出处:网络
难度:1

> 4
> 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]

知识点:物理内存管理
出处:网络
难度:1

> 4
> 伙伴系统的空闲块如何组织?

- [x]

知识点:物理内存管理
出处:网络
难度:1

> 4
> 伙伴系统的内存分配流程?

- [x]

知识点:物理内存管理
出处:网络
难度:1

> 4
> 伙伴系统的内存回收流程?

- [x]

知识点:物理内存管理
出处:网络
难度:1

> 4
> struct list_entry是如何把数据元素组织成链表的?

- [x]

知识点:物理内存管理
出处:网络
难度:1

> 2
> 描述段管理机制正确的是() s2

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

知识点:物理内存管理
出处:网络
难度:1

> ABCD
> 2
> 描述页管理机制正确的是() s3

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

知识点:物理内存管理
出处:网络
难度:1

> ABC
> 2
> 页表项标志位包括() s4

- [x] A.存在位(resident bit)

- [x] B.修改位(dirty bit)
- [x] C.引用位(clock/reference bit)
- [x] D.只读位(read only OR read/write bit)

知识点:物理内存管理
出处:网络
难度:1

> ABCD
> 2
> 可有效应对大地址空间可采用的页表手段是() s7

- [x] A.多级页表
- [x] B.反置页表
- [ ] C.页寄存器方案
- [ ] D.单级页表

知识点:物理内存管理
出处:网络
难度:1

> AB
> 4
> (w3l2) 请简要分析64bit CPU体系结构下的分页机制是如何实现的.

- [x]

知识点:物理内存管理
出处:网络
难度:1

> - 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）
>
> 4
> (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]

知识点:物理内存管理
出处:网络
难度:1

> 500=0.9*150+0.1*x
> 4
> (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
```

- [x]

知识点:物理内存管理
出处:网络
难度:1

> 4
> 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。

- [x]

知识点:物理内存管理
出处:网络
难度:1

> 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。

- [x]

知识点:物理内存管理
出处:网络
难度:1

> 4
> 阅读64bit IBM Powerpc CPU架构是如何实现反置页表，给出分析报告。

- [x]

知识点:物理内存管理
出处:网络
难度:1

> 1
> 80386 CPU保护模式下的特权级个数是() s1

- ( ) A.1
- ( ) B.2
- ( ) C.3
- (x) D.4

知识点:物理内存管理
出处:网络
难度:1

> D ring0-ring3
> 2
> 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] A.0
- [ ] B.1
- [ ] C.2
- [x] D.3

知识点:物理内存管理
出处:网络
难度:1

> AD ring 0 for OS， ring3 for application
> 1
> 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

- ( ) A.产生外设中断
- (x) B.产生访存异常
- ( ) C.CPU继续正常执行
- ( ) D.系统重启

知识点:物理内存管理
出处:网络
难度:1

> B 将产生General Protection Fault异常
> 1
> 以下那种存储管理不可用于多道程序系统中

- ( ) A.固定分区存储管理
- (x) B.单一连续区存储管理
- ( ) C.可变分区存储管理
- ( ) D.段式存储管理

知识点:操作系统概述
出处:网络
难度:1

> B
> 1
> 段描述符中与特权级相关的一个组成部分的名称是（） s1

- (x) A.DPL
- ( ) B.AVL
- ( ) C.Base
- ( ) D.Limit

知识点:物理内存管理
出处:网络
难度:1

> A 是DPL
> 1
> CS段寄存器中的最低两位保存的是（） s1

- ( ) A.DPL
- (x) B.CPL
- ( ) C.RPL
- ( ) D.NPL

知识点:物理内存管理
出处:网络
难度:1

> C 是CPL
> 1
> DS段寄存器中的最低两位保存的是（） s1

- ( ) A.DPL
- ( ) B.CPL
- (x) C.RPL
- ( ) D.NPL

知识点:物理内存管理
出处:网络
难度:1

> C 是RPL
> 1
> 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) A.MAX(CPL, RPL) <= DPL[数据段]
- ( ) B.MIN(CPL, RPL) <= DPL[数据段]
- ( ) C.MAX(CPL, DPL) <= RPL[数据段]
- ( ) D.MIN(CPL, DPL) <= RPL[数据段]

知识点:物理内存管理
出处:网络
难度:1

> A 是MAX(CPL, RPL) <= DPL[数据段]
> 2
> 对于Task State Segment（TSS）而言，uCore OS可以利用它做（） s2

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

知识点:物理内存管理
出处:网络
难度:1

> AB 是保存ring 0的SS和ESP
> 1
> 页目录表的基址是保存在寄存器（） s3

- ( ) A.CR0
- ( ) B.CR1
- ( ) C.CR2
- (x) D.CR3

知识点:物理内存管理
出处:网络
难度:1

> D CR3
> 1
> 在启动页机制后，不可能进行的操作包括（） s3

- (x) A.取消段机制，只保留页机制
- ( ) B.取消页机制，只保留段机制
- ( ) C.取消页机制，也取消段机制
- ( ) D.保留页机制，也保留段机制

知识点:物理内存管理
出处:网络
难度:1

> A 不可能取消段机制，只保留页机制
> 2
> 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] A.给页目录表动态分配空间，给页表分配空间
- [x] B.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] C.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] D.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

知识点:物理内存管理
出处:网络
难度:1

> ABCD 都对，还要设置更多的一些属性。
> 2
> 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] A.内存寻址过程中
- [x] B.代码跳转过程中
- [x] C.中断处理过程中
- [ ] D.ALU计算过程中

知识点:物理内存管理
出处:网络
难度:1

> ABC 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。
> 4
> 请描述ucore OS建立页机制的准备工作包括哪些步骤？（w4l1）

- [x]

知识点:物理内存管理
出处:网络
难度:1

> - 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）
>
> 4
> 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]

知识点:物理内存管理
出处:网络
难度:1

> 4
> (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

```
1    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):

```
1    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]

知识点:物理内存管理
出处:网络
难度:1

> 4
> 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]

知识点:物理内存管理
出处:网络
难度:1

> 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]

知识点:物理内存管理
出处:网络
难度:1

> 4
> 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

- [x]

知识点:物理内存管理
出处:网络
难度:1

> - 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）
> 4
> 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]

知识点:物理内存管理
出处:网络
难度:1

> 4
> (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。

- [x]

知识点:虚拟内存管理
出处:网络
难度:1

> 4
> 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。

- [x]

知识点:虚拟内存管理
出处:网络
难度:1

> 4
> 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?

- [x]

知识点:虚拟内存管理
出处:网络
难度:1

> 4
> (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，页没有在硬盘上，这时页帧号为0x7F）。
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
```

- [x]

知识点:虚拟内存管理
出处:网络
难度:1

> 4
> 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。

- [x]

知识点:虚拟内存管理
出处:网络
难度:1

> 4
> 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？

- [x]

知识点:虚拟内存管理
出处:网络
难度:1

> 1
> 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

- ( ) A.1
- ( ) B.2
- ( ) C.3
- (x) D.4

知识点:虚拟内存管理
出处:网络
难度:1

> D
> 1
> 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

- ( ) A.1
- ( ) B.2
- ( ) C.3
- (x) D.4

知识点:虚拟内存管理
出处:网络
难度:1

> D
> 1
> 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

- ( ) A.1
- ( ) B.2
- ( ) C.4
- (x) D.6

知识点:虚拟内存管理
出处:网络
难度:1

> D
> 1
> 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

- ( ) A.8
- (x) B.9
- ( ) C.10
- ( ) D.11

知识点:虚拟内存管理
出处:网络
难度:1

> B
> 1
> 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

- ( ) A.8
- ( ) B.9
- (x) C.10
- ( ) D.11

知识点:虚拟内存管理
出处:网络
难度:1

> C
> 1
> 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) A.会
- ( ) B.不会

知识点:虚拟内存管理
出处:网络
难度:1

> D 3页时9次缺页，4页时10次缺页
> 2
> 下面哪些页面淘汰算法不会产生Belady异常现象 s4

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

知识点:虚拟内存管理
出处:网络
难度:1

> CD LRU和OPT属于一种栈算法
> 1
> 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

- ( ) A.2
- ( ) B.3
- ( ) C.4
- (x) D.5

知识点:虚拟内存管理
出处:网络
难度:1

> D
> 1
> 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

- ( ) A.2
- ( ) B.3
- ( ) C.4
- (x) D.5

知识点:虚拟内存管理
出处:网络
难度:1

> D
> 4
> 置换算法的功能?

- [x]

知识点:虚拟内存管理
出处:网络
难度:1

> 4
> 全局和局部置换算法的不同?

- [x]

知识点:虚拟内存管理
出处:网络
难度:1

> 4
> 最优算法、先进先出算法和LRU算法的思路?

- [x]

知识点:虚拟内存管理
出处:网络
难度:1

> 4
> 时钟置换算法的思路?

- [x]

知识点:虚拟内存管理
出处:网络
难度:1

> 4
> LFU算法的思路?

- [x]

知识点:虚拟内存管理
出处:网络
难度:1

> 4
> 什么是Belady现象?

- [x]

知识点:虚拟内存管理
出处:网络
难度:1

> 4
> 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?

- [x]

知识点:虚拟内存管理
出处:网络
难度:1

> 4
> 什么是工作集?

- [x]

知识点:虚拟内存管理
出处:网络
难度:1

> 4
> 什么是常驻集?

- [x]

知识点:虚拟内存管理
出处:网络
难度:1

> 4
> 工作集算法的思路?

- [x]

知识点:虚拟内存管理
出处:网络
难度:1

> 4
> 缺页率算法的思路?

- [x]

知识点:虚拟内存管理
出处:网络
难度:1

> 4
> 什么是虚拟内存管理的抖动现象?

- [x]

知识点:虚拟内存管理
出处:网络
难度:1

> 4
> 操作系统负载控制的最佳状态是什么状态?

- [x]

知识点:虚拟内存管理
出处:网络
难度:1

> 4
> （spoc）请证明为何LRU算法不会出现belady现象

- [x]

知识点:虚拟内存管理
出处:网络
难度:1

> 4
> （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

- [x]

知识点:虚拟内存管理
出处:网络
难度:1

> 4
> 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](http://www.ece.eng.wayne.edu/~sjiang/pubs/papers/jiang02_LIRS.pdf)
    [LIRS journal paper](http://www.ece.eng.wayne.edu/~sjiang/pubs/papers/jiang05_LIRS.pdf)
    [LIRS-replacement ppt1](http://dragonstar.ict.ac.cn/course_09/XD_Zhang/LIRS-replacement.pdf)
    [LIRS-replacement ppt2](http://www.ece.eng.wayne.edu/~sjiang/Projects/LIRS/sig02.ppt)
```

- [x]

知识点:虚拟内存管理
出处:网络
难度:1

> 2
> lab3中虚存管理需要直接借助的机制包括() s1

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

知识点:虚拟内存管理
出处:网络
难度:1

> ACD 段映射机制不直接需要
> 2
> lab3中实现虚存管理的过程包括() s2

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

知识点:虚拟内存管理
出处:网络
难度:1

> ABCD
> 1
> lab3中用于描述“合法”虚拟页的数据结构是（）s3

- (x) A.vma_struct
- ( ) B.trapframe
- ( ) C.gatedesc
- ( ) D.segdesc

知识点:虚拟内存管理
出处:网络
难度:1

> A
> 1
> lab3中访问“合法”虚拟页产生缺页异常的原因是（）s4

- (x) A.页表项的P bit为0
- ( ) B.页目录项的I/D bit为0
- ( ) C.页表项的U/S bit为0
- ( ) D.页目录项的W/R bit位0

知识点:虚拟内存管理
出处:网络
难度:1

> A 页表项的P bit为0，表示此页不存在
> 1
> lab3中把扇区索引信息放在（）s5

- (x) A.页表项中
- ( ) B.页目录项中
- ( ) C.内存中的Page结构中
- ( ) D.内存中的vma_struct结构中

知识点:虚拟内存管理
出处:网络
难度:1

> A 页表项中的高24位
> 4
> 缺页异常的处理流程?

- [x]

知识点:虚拟内存管理
出处:网络
难度:1

> 4
> 从外存的页面的存储和访问代码?

- [x]

知识点:虚拟内存管理
出处:网络
难度:1

> 4
> 缺页和页访问非法的返回地址有什么不同?

- [x]

知识点:虚拟内存管理
出处:网络
难度:1

> 4
> 虚拟内存管理中是否用到了段机制

- [x]

知识点:虚拟内存管理
出处:网络
难度:1

> 4
> ucore如何知道页访问异常的地址?

- [x]

知识点:虚拟内存管理
出处:网络
难度:1

> 4
> 中断处理例程的段表在GDT还是LDT?

- [x]

知识点:虚拟内存管理
出处:网络
难度:1

> 4
> 物理内存管理的数据结构在哪?

- [x]

知识点:虚拟内存管理
出处:网络
难度:1

> 4
> 页表项的结构?

- [x]

知识点:虚拟内存管理
出处:网络
难度:1

> 4
> 页表项的修改代码?

- [x]

知识点:虚拟内存管理
出处:网络
难度:1

> 4
> 如何设置一个虚拟地址到物理地址的映射关系?

- [x]

知识点:虚拟内存管理
出处:网络
难度:1

> 4
> 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

- [x]

知识点:虚拟内存管理
出处:网络
难度:1

> 4
> swap_init()做了些什么?

- [x]

知识点:虚拟内存管理
出处:网络
难度:1

> 4
> vmm_init()做了些什么?

- [x]

知识点:虚拟内存管理
出处:网络
难度:1

> 4
> vma_struct数据结构的功能?

- [x]

知识点:虚拟内存管理
出处:网络
难度:1

> 4
> mmap_list是什么列表?

- [x]

知识点:虚拟内存管理
出处:网络
难度:1

> 4
> 外存中的页面后备如何找到?

- [x]

知识点:虚拟内存管理
出处:网络
难度:1

> 4
> vma_struct和mm_struct的关系是什么?

- [x]

知识点:虚拟内存管理
出处:网络
难度:1

> 合法的连续虚拟地址区域、整个进程的地址空间
> 4
> 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；

- [x]

知识点:虚拟内存管理
出处:网络
难度:1

> 4
> 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?

- [x]

知识点:虚拟内存管理
出处:网络
难度:1

> 4
> find_vma()做了些什么?

- [x]

知识点:虚拟内存管理
出处:网络
难度:1

> 4
> swapfs_read()做了些什么?

- [x]

知识点:虚拟内存管理
出处:网络
难度:1

> 4
> 缺页时的页面创建代码在哪?

- [x]

知识点:虚拟内存管理
出处:网络
难度:1

> 4
> struct rb_tree数据结构的原理是什么？在虚拟管理中如何用它的?

- [x]

知识点:虚拟内存管理
出处:网络
难度:1

> 4
> 页目录项和页表项的dirty bit是何时，由谁置1的?

- [x]

知识点:虚拟内存管理
出处:网络
难度:1

> 4
> 页目录项和页表项的access bit是何时，由谁置1的?

- [x]

知识点:虚拟内存管理
出处:网络
难度:1

> 4
> 虚拟页与磁盘后备页面的对应有关系?

- [x]

知识点:虚拟内存管理
出处:网络
难度:1

> 4
> 如果在开始加载可执行文件时，如何改?

- [x]

知识点:虚拟内存管理
出处:网络
难度:1

> 4
> check_swap()做了些什么检查?

- [x]

知识点:虚拟内存管理
出处:网络
难度:1

> 4
> swap_entry_t数据结构做什么用的？放在什么地方?

- [x]

知识点:虚拟内存管理
出处:网络
难度:1

> 4
> 空闲物理页面的组织数据结构是什么?

- [x]

知识点:虚拟内存管理
出处:网络
难度:1

> 4
> 置换算法的接口数据结构?

- [x]

知识点:虚拟内存管理
出处:网络
难度:1

> swap_manager
> 4
> (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）。

- [x]

知识点:虚拟内存管理
出处:网络
难度:1

> 4
> (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?

- [x]

知识点:虚拟内存管理
出处:网络
难度:1

> 2
> 进程与程序的关系描述正确的是（） s1

- [x] A.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
- [ ] B.进程是一个具有一定独立功能的程序
- [ ] C.程序是一个动态执行的进程
- [x] D.进程包含了正在运行的一个程序的所有状态信息

知识点:进程和线程管理
出处:网络
难度:1

> AD
> 2
> 关于进程控制块的描述正确的是（） s2

- [x] A.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] B.进程控制块是进程存在的唯一标志
- [x] C.每个进程都在操作系统中有一个对应的进程控制块
- [x] D.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

知识点:进程和线程管理
出处:网络
难度:1

> ABCD
> 2
> 关于进程的生命周期的描述正确的是（） s3

- [x] A.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] B.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] C.进程执行的当前时间片用完了，此时进程处于就绪态
- [x] D.进程退出了，但还没被父进程回收，此时进程处于zombie态

知识点:进程和线程管理
出处:网络
难度:1

> ABCD
> 2
> 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] A.就绪态进程维护在进程就绪队列中
- [x] B.等待态进程维护在进程等待队列中
- [ ] C.运行态进程维护在进程运行队列中
- [ ] D.zombie态进程不在任何队列中

知识点:进程和线程管理
出处:网络
难度:1

> AB
> 2
> 有关线程或进程的描述正确的是（） s6

- [x] A.进程是资源分配单位，线程是CPU调度单位
- [x] B.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] C.线程能减少并发执行的时间和空间开销
- [x] D.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

知识点:进程和线程管理
出处:网络
难度:1

> ABCD
> 1
> 以下那种存储管理不可用于多道程序系统中

- ( ) A.固定分区存储管理
- (x) B.单一连续区存储管理
- ( ) C.可变分区存储管理
- ( ) D.段式存储管理

知识点:操作系统概述
出处:网络
难度:1

> B
> 2
> 常见的线程种类有() s7

- [x] A.用户线程

- [x] B.内核线程
- [x] C.轻量级进程

知识点:进程和线程管理
出处:网络
难度:1

> ABC
> 2
> 内核线程的描述正确的是() s8

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

知识点:进程和线程管理
出处:网络
难度:1

> A
> 4
> (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
```

- [x]

知识点:进程和线程管理
出处:网络
难度:1

> 4
> 什么是进程？什么是程序?

- [x]

知识点:进程和线程管理
出处:网络
难度:1

> 4
> 程序和进程联系和区别是什么?

- [x]

知识点:进程和线程管理
出处:网络
难度:1

> 4
> 进程控制块的功能是什么?

- [x]

知识点:进程和线程管理
出处:网络
难度:1

> 4
> 进程控制块中包括什么信息?

- [x]

知识点:进程和线程管理
出处:网络
难度:1

> 4
> 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？

- [x]

知识点:进程和线程管理
出处:网络
难度:1

> 4
> 进程切换过程中的几个关键代码分析

- [x]

知识点:进程和线程管理
出处:网络
难度:1

> 4
> 时钟中断触发调度函数的启动

- [x]

知识点:进程和线程管理
出处:网络
难度:1

> 4
> 当前进程的现场保存代码

- [x]

知识点:进程和线程管理
出处:网络
难度:1

> 4
> 进程切换代码 > 下一个运行进程的现场恢复

- [x]

知识点:进程和线程管理
出处:网络
难度:1

> 4
> 运行、就绪和等待三种状态的含义?

- [x]

知识点:进程和线程管理
出处:网络
难度:1

> 4
> 分析的4个相关状态转换代码和状态修改代码

- [x]

知识点:进程和线程管理
出处:网络
难度:1

> 4
> 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?

- [x]

知识点:进程和线程管理
出处:网络
难度:1

> 4
> 引入线程的目的是什么？什么是线程?

- [x]

知识点:进程和线程管理
出处:网络
难度:1

> 4
> 进程与线程的联系和区别是什么?

- [x]

知识点:进程和线程管理
出处:网络
难度:1

> 4
> 用户线程与内核线程的区别是什么?

- [x]

知识点:进程和线程管理
出处:网络
难度:1

> 2
> 关于进程切换描述正确的是（） s1

- [x] A.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] B.进程切换要保存当前进程的上下文
- [x] C.进程切换要恢复下一个进程的上下文
- [ ] D.进程切换的进程上下文不包括CPU的寄存器等硬件信息

知识点:进程和线程管理
出处:网络
难度:1

> ABC
> 2
> 关于创建新进程的描述正确的是（） s2

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

知识点:进程和线程管理
出处:网络
难度:1

> ABCD
> 2
> 关于进程加载执行的描述正确的是（） s3

- [x] A.系统调用exec( )加载新程序取代当前运行进程
- [x] B.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] C.exec调用成功时，它是相同的进程，但是运行了不同的程序
- [x] D.exec调用成功时，代码段、堆栈和堆(heap)等完全重写了

知识点:进程和线程管理
出处:网络
难度:1

> ABCD
> 2
> 有关管理进程等待的描述正确的是（） s 4

- [x] A.wait()系统调用用于父进程等待子进程的结束
- [x] B.子进程结束时通过exit()向父进程返回一个值
- [x] C.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] D.进程结束执行时调用exit()，完成进程的部分占用资源的回收

知识点:进程和线程管理
出处:网络
难度:1

> ABCD
> 2
> 下列叙述中正确的是() s2

- [ ] A.lab 5 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
- [ ] B.lab 5 建立了用户进程，且3GB ~ 4 GB都是内核可访问空间，内核可进行正常读写
- [x] C.lab5中的第一个用户进程是内核创建的。
- [x] D.lab5中的用户进程可通过fork创建新的用户进程。

知识点:进程和线程管理
出处:网络
难度:1

> CD
> 2
> lab5通过do_execve函数执行新的程序，为此需要完成（） s3

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

知识点:进程和线程管理
出处:网络
难度:1

> ABC
> 2
> lab5通过do_icode函数执行新的程序，为此需要完成（）s4

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

知识点:进程和线程管理
出处:网络
难度:1

> ABCD
> 2
> 关于进程管理的COW(Copy On Write)机制叙述正确的是（）s6

- [ ] A.父进程创建子进程需要复制父进程的内存空间
- [ ] B.父进程创建子进程需要给子进程分配内核堆栈
- [ ] C.父进程创建子进程需要给子进程分配用户堆栈
- [x] D.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

知识点:进程和线程管理
出处:网络
难度:1

> A
> 1
> 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) A.就绪
- ( ) B.等待
- ( ) C.运行
- ( ) D.完成

知识点:进程和线程管理
出处:网络
难度:1

> A
> 1
> 最高响应比优先算法的特点是（） s3

- ( ) A.有利于短作业但不利于长作业
- (x) B.有利于短作业又兼顾到长作业
- ( ) C.不利于短作业也不利于长作业
- ( ) D.不利于短作业但有利于长作业

知识点:进程和线程管理
出处:网络
难度:1

> B
> 1
> 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

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

知识点:进程和线程管理
出处:网络
难度:1

> C
> 1
> 时间片轮转调度算法是为了（） s4

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

知识点:进程和线程管理
出处:网络
难度:1

> A
> 1
> 下面关于硬时限（hard deadlines）和软时限（soft deadlines）的描述错误的是（）。s5

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

知识点:进程和线程管理
出处:网络
难度:1

> B
> 1
> 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) A.优先级反转
- ( ) B.优先级重置
- ( ) C.系统错误
- ( ) D.死循环

知识点:进程和线程管理
出处:网络
难度:1

> A
> 2
> lab6的调度过程包括() s2

- [x] A.触发：trigger scheduling
- [x] B.入队：‘enqueue’
- [x] C.选取：pick up
- [x] D.出队：‘dequeue’
- [x] E.切换：process switch

知识点:进程和线程管理
出处:网络
难度:1

> ABCDE
> 2
> lab6中涉及到的调度点包括（） s3

- [x] A.proc.c:do_exit　户线程执行结束，主动放弃CPU
- [x] B.proc.c:do_wait　用户线程等待子进程结束，主动放弃CPU
- [x] C.proc.c::cpu_idle　idleproc内核线程选取一个就绪进程并切换
- [x] D. 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

知识点:进程和线程管理
出处:网络
难度:1

> ABCD
> 2
> lab6调度算法支撑框架包括的函数指针有（）s4

- [x] A.(enqueue)(struct run_queue rq, ...);
- [x] B.(dequeue)(struct run_queue rq, ...);
- [x] C.(pick_next)(struct run_queue rq);
- [x] D.(proc_tick)(struct run_queue rq, ...);

知识点:进程和线程管理
出处:网络
难度:1

> ABCD
> 2
> lab6调度算法支撑框架中与时钟中断相关的函数指针有（）s4

- [ ] A.(enqueue)(struct run_queue rq, ...);
- [ ] B.(dequeue)(struct run_queue rq, ...);
- [ ] C.(pick_next)(struct run_queue rq);
- [x] D.(proc_tick)(struct run_queue rq, ...);

知识点:进程和线程管理
出处:网络
难度:1

> D
> 2
> 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

- [ ] A.等待进程结束
- [ ] B.进程退出
- [ ] C.进程睡眠

- [x] D.进程被时钟中断打断

知识点:进程和线程管理
出处:网络
难度:1

> D
> 1
> 临界资源是什么类型的共享资源（） s2

- ( ) A.临界资源不是共享资源
- ( ) B.用户共享资源
- (x) C.互斥共享资源
- ( ) D.同时共享资源

知识点:同步互斥
出处:网络
难度:1

> C
> 1
> 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（） s 2

- ( ) A.饥饿
- (x) B.死锁
- ( ) C.死机
- ( ) D.死循环

知识点:同步互斥
出处:网络
难度:1

> B
> 1
> 共享变量是指（）访问的变量 s2

- ( ) A.只能被系统进程
- ( ) B.只能被多个进程互斥
- ( ) C.只能被用户进程
- (x) D.可被多个进程

知识点:同步互斥
出处:网络
难度:1

> D
> 1
> 要想进程互斥地进入各自的同类资源的临界区，需要（） s3

- ( ) A.在进程间互斥使用共享资源
- ( ) B.在进程间非互斥使用临界资源
- (x) C.在进程间互斥地使用临界资源
- ( ) D.在进程间不使用临界资源

知识点:同步互斥
出处:网络
难度:1

> C
> 2
> 锁的实现方法有哪几种（） s4

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

知识点:同步互斥
出处:网络
难度:1

> ABD
> 2
> 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] A.一个进程释放一种资源
- [ ] B.系统新创建了一个进程
- [ ] C.一个进程从就绪队列进入阻塞队列
- [ ] D.一个在阻塞队列中的进程被系统取消了

知识点:同步互斥
出处:网络
难度:1

> A
> 1
> 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

- ( ) A.有四个进程进入了该程序段
- ( ) B.有一个进程在等待
- (x) C.有三个进程进入了程序段，有一个进程在等待
- ( ) D.有一个进程进入了该程序段，其余四个进程在等待

知识点:信号量与管程
出处:网络
难度:1

> C
> 1
> 2元信号量可以初始化为（） s2

- (x) A.0或1
- ( ) B.0或-1
- ( ) C.只能为1
- ( ) D.任意值

知识点:信号量与管程
出处:网络
难度:1

> A
> 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

- ( ) A.1个
- ( ) B.2个
- (x) C.3个
- ( ) D.4个

知识点:信号量与管程
出处:网络
难度:1

> C
> 1
> (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
```

- ( ) A.可能为-1或3
- ( ) B.只能为1
- (x) C.可能为0、1或2
- ( ) D.可能为-1、0、1、1或2

知识点:信号量与管程
出处:网络
难度:1

> C
> 2
> 管程的主要特点有（） s3

- [x] A.局部数据变量只能被管程的过程访问
- [x] B.一个进程通过调用管程的一个过程进入管程
- [ ] C.不会出现死锁
- [x] D.在任何时候，只能有一个进程在管程中执行

知识点:信号量与管程
出处:网络
难度:1

> ABD
> 1
> 关于管程的叙述正确的是（） s3

- (x) A.管程中的局部数据变量可以被外部直接访问
- (x) A.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
- (x) A.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) A.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

知识点:信号量与管程
出处:网络
难度:1

> D
> 2
> 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] A.处理器调度
- [x] B.屏蔽中断
- [x] C.等待队列
- [ ] D.动态内存分配

知识点:lab7
出处:网络
难度:1

> ABC 需用到前三个，动态内存分配不是必须的
> 2
> ucore实现的信号量机制被用于（） s3 信号量设计与实现

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

知识点:lab7
出处:网络
难度:1

> ABC 中断机制是支持信号量的，所以不选
> 2
> 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] A.管程中采用信号量用于互斥操作
- [x] B.管程中采用信号量用于同步操作
- [x] C.管程中采用条件变量用于同步操作
- [x] D.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

知识点:lab7
出处:网络
难度:1

> ABCD
> 2
> 死锁产生的必要条件包括（） s1

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

知识点:死锁
出处:网络
难度:1

> ABCD
> 2
> 死锁处理方法主要包括（） s 2

- [x] A.死锁预防(Deadlock Prevention):确保系统永远不会进入死锁状态
- [x] A.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] A.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] A.由应用进程处理死锁：通常操作系统忽略死锁

知识点:死锁
出处:网络
难度:1

> ABCD
> 1
> 可以使用银行家算法_ 死锁。s3

- ( ) A.预防
- ( ) B.检测
- ( ) C.解除
- (x) D.避免

知识点:死锁
出处:网络
难度:1

> D 是死锁避免
> 1
> 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) A.O(m*n^2)
- ( ) B.O(m^2*n)
- ( ) C.O(m^2*n^2)
- ( ) D.O(m*n)

知识点:死锁
出处:网络
难度:1

> A 是O(m*n^2)
> 2
> 关于进程通信原理的阐述正确的是（） s5

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

知识点:进程间通信
出处:网络
难度:1

> ABCD
> 2
> 关于信号和管道的进程通信机制的阐述正确的是（） s6

- [x] A.信号（signal）是一种进程间的软件中断通知和处理机制
- [x] B.信号的接收处理方式包括：捕获(catch)，忽略(Ignore)，屏蔽（Mask）
- [x] C.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
- [ ] D.管道（pipe）的实现需要在磁盘文件系统上创建一个文件

知识点:进程间通信
出处:网络
难度:1

> ABCD 管道（pipe）的实现只需基于内存即可。
> 2
> 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] A.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] A.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] A.消息队列机制可用于进程间的同步操作
- [x] A.共享内存机制可用于进程间的数据共享

知识点:进程间通信
出处:网络
难度:1

> ABCD
> 2
> 关于文件系统功能的阐述正确的是（） s1

- [x] A.负责数据持久保存
- [x] B.文件分配
- [x] C.文件管理
- [x] D.数据可靠和安全

知识点:文件系统
出处:网络
难度:1

> 2
> 打开文件时，文件系统要维护哪些信息（） s 2

- [x] A.文件指针
- [x] B.打开文件计数
- [x] C.文件访问权限
- [x] D.文件位置和数据缓存

知识点:文件系统
出处:网络
难度:1

> ABCD
> 2
> 关于目录和别名的阐述正确的是（）s3

- [x] A.目录是一类特殊的文件
- [x] B.目录的内容是文件索引表<文件名, 指向文件的指针>
- [x] C.可通过硬链接机制实现文件别名
- [x] D.可通过软链接机制实现文件别名

知识点:文件系统
出处:网络
难度:1

> ABCD
> 2
> 虚拟文件系统可支持的具体文件系统包括（） s4

- [x] A.磁盘文件系统
- [x] B.设备文件系统
- [x] C.网络文件系统
- [x] D.系统状态文件系统（proc...）

知识点:文件系统
出处:网络
难度:1

> ABCD
> 2
> 关于文件缓存和打开文件的阐述正确的是（） s5

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

知识点:文件系统
出处:网络
难度:1

> ABC 文件是共享资源，对于写操作需要互斥保护
> 2
> 关于文件分配的阐述正确的是（） s6

- [x] A.连续分配会产生外碎片
- [ ] B.链式分配会产生外碎片
- [ ] C.索引分配会产生外碎片
- [x] D.多级索引分配可支持大文件

知识点:文件系统
出处:网络
难度:1

> AD 链式分配和索引分配不会产生外碎片
> 2
> 关于冗余磁盘阵列(RAID, Redundant Array of Inexpensive Disks)的阐述正确的是（） s7

- [x] A.采用RAID机制可提高磁盘IO的吞吐量(通过并行)
- [x] B.采用RAID机制可提高磁盘IO的可靠性和可用性 (通过冗余)
- [ ] C.采用RAID-0可提高磁盘IO的可靠性和可用性
- [ ] D.采用RAID-1可提高磁盘IO的吞吐量

知识点:文件系统
出处:网络
难度:1

> AB RAID-0提高并行性，RAID-1提高可靠性
> 2
> ucore实现的文件系统抽象包括（） s1 总体介绍

- [x] A.文件
- [x] B.目录项
- [x] C.索引节点
- [ ] D.安装点

知识点:文件系统
出处:网络
难度:1

> ABC
> 1
> 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

- ( ) A.连续分配
- ( ) B.链式分配
- (x) C.索引分配
- ( ) D.位图分配

知识点:文件系统
出处:网络
难度:1

> C 索引分配
> 2
> 关于ucore实现的SFS阐述正确的是（） s3 Simple File System分析

- [x] A.SFS的超级块保存在硬盘上，在加载simple FS时会读入内存中
- [x] B.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] C.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
- [ ] D.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

知识点:文件系统
出处:网络
难度:1

> ABC 除了前三种结构，剩下的用于保存文件的inode, dir/file的data
> 2
> 关于ucore实现的Virtual FS（简称VFS）阐述正确的是() s4 Virtual File System分析

- [x] A.已支持磁盘文件系统
- [x] B.已支持设备文件系统
- [ ] C.已支持网络文件系统
- [ ] D.已支持系统状态文件系统

知识点:文件系统
出处:网络
难度:1

> AB 后两种可实现，但现在还没实现
> 2
> 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] A.串口设备
- [x] A.并口设备
- [x] A.CGA设备
- [x] A.键盘设备

知识点:文件系统
出处:网络
难度:1

> ABCD
> 2
> 字符设备包括（） s1

- [x] A.键盘
- [x] B.鼠标
- [x] C.并口
- [x] D.串口

知识点:I/O子系统
出处:网络
难度:1

> ABCD
> 2
> 块设备包括（） s1

- [x] A.硬盘
- [x] B.软盘
- [x] C.光盘
- [x] D.U盘

知识点:I/O子系统
出处:网络
难度:1

> ABCD
> 2
> 网络设备包括（） s1

- [x] A.以太网卡
- [x] B.wifi网卡
- [x] C.蓝牙设备
- [ ] D.网盘设备

知识点:I/O子系统
出处:网络
难度:1

> ABC 网盘在模拟实现上应该算块设备
> 2
> 关于CPU与设备的通信方式包括（） s2

- [x] A.轮询
- [x] B.设备中断
- [x] C.DMA
- [ ] D.PIPE

知识点:I/O子系统
出处:网络
难度:1

> ABC PIPE是用于进程间通信
> 2
> 关于IO数据传输的阐述正确的是（） s3

- [x] A.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] B.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
- [ ] C.DMA机制适合字符设备
- [ ] D.PIO机制适合块设备

知识点:I/O子系统
出处:网络
难度:1

> AB DMA机制适合块设备，PIO机制适合简单，低速的字符设备等
> 2
> 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] A.缓和CPU与I/O设备间速度不匹配的矛盾
- [x] B.减少对CPU的中断频率，放宽对CPU中断响应时间的限制
- [x] C.解决基本数据单元大小（即数据粒度）不匹配的问题
- [x] D.提高CPU和I/O设备之间的并行性

知识点:I/O子系统
出处:网络
难度:1

> ABCD
> 1
> 能及时处理由过程控制反馈的数据并作出响应的操作系统是

- ( ) A.分时系统
- ( ) B.网络系统
- (x) C.实时系统
- ( ) D.批处理系统

知识点:操作系统概述
出处:网络
难度:1

> C
> 3
> 多道程序的引入是为了提高CPU的利用率。

- (x) A.对
- ( ) B.错

知识点:操作系统概述
出处:网络
难度:1

> A 引入多道程序设计技术的根本目的是为了提高CPU的利用率，充分发挥计算机系统部件的并行性，现代计算机系统都采用了多道程序设计技术。
> 1
> UNIX系统是一个_ 操作系统。

- ( ) A.单用户
- ( ) B.单用户多任务
- ( ) C.多用户多任务
- (x) D.多用户单任务

知识点:操作系统概述
出处:网络
难度:1

> D
> 1
> 一般在哪种情况下发生从用户态到核心态的转换?

- ( ) A.使用特权指令
- ( ) B.发生子程序调用
- ( ) C.使用共享代码
- (x) D.进行系统调用

知识点:操作系统概述
出处:网络
难度:1

> D
> 1
> Windows NT属于哪一类操作系统?

- ( ) A.单用户单任务
- (x) B.单用户多任务
- ( ) C.单道批处理
- ( ) D.多用户

知识点:操作系统概述
出处:网络
难度:1

> B
> 1
> Windows NT 属于哪一类操作系统?

- ( ) A.单用户任务
- (x) B.单用户多任务
- ( ) C.多用户
- ( ) D.单道批处理

知识点:操作系统概述
出处:网络
难度:1

> B
> 3
> 在分时系统中，时间片越小，一个作业的总运行时间越短。

- ( ) A.对
- (x) B.错

知识点:操作系统概述
出处:网络
难度:1

> B
> 1
> 一个完整的计算机系统是由组成的。

- ( ) A.硬件
- ( ) B.软件
- (x) C.硬件和软件
- ( ) D.用户程序

知识点:操作系统概述
出处:网络
难度:1

> C
> 1
> 操作系统的基本职能是。

- (x) A.控制和管理系统内各种资源，有效地组织多道程序的运行
- ( ) B.提供用户界面，方便用户使用
- ( ) C.提供方便的可视化编辑程序
- ( ) D.提供功能强大的网络管理工具

知识点:操作系统概述
出处:网络
难度:1

> A
> 1
> 为了使系统中所有的用户都能得到及时的响应，该操作系统应该是_ 。

- ( ) A.多道批处理系统
- (x) B.分时系统
- ( ) C.实时系统
- ( ) D.网络系统

知识点:操作系统概述
出处:网络
难度:1

> B
> 1
> 在计算机系统中，控制和管理各种资源、有效地组织多道程序运行的系统软件称作_ 。

- ( ) A.文件系统
- (x) B.操作系统
- ( ) C.网络管理系统
- ( ) D.数据库管理系统

知识点:操作系统概述
出处:网络
难度:1

> B
> 1
> 以下著名的操作系统中，属于多用户、分时系统的是_ 。

- ( ) A.DOS系统
- ( ) B.Windows NT系统
- (x) C.UNIX系统
- ( ) D.OS/2系统

知识点:操作系统概述
出处:网络
难度:1

> C
> 1
> 操作系统是一种(　　)

- (x) A.系统软件
- ( ) B.系统硬件
- ( ) C.应用软件
- ( ) D.支援软件

知识点:操作系统概述
出处:网络
难度:1

> A
> 1
> MS—DOS的存贮管理采用了(　　)

- ( ) A.段式存贮管理
- ( ) B.段页式存贮管理
- (x) C.单用户连续存贮管理
- ( ) D.固定式分区存贮管理

知识点:操作系统概述
出处:网络
难度:1

> C
> 1
> MS—DOS中用于软盘整盘复制的命令是(　　)

- ( ) A.COMP
- (x) B.DISKCOPY
- ( ) C.SYS
- ( ) D.BACKUP

知识点:操作系统概述
出处:网络
难度:1

> B
> 3
> 早期批量处理解决了手工操作阶段的操作联机问题。

- (x) A.对
- ( ) B.错

知识点:操作系统概述
出处:网络
难度:1

> A
> 3
> 交互性是批处理系统的一个特征。

- ( ) A.对
- (x) B.错

知识点:操作系统概述
出处:网络
难度:1

> B 批处理操作系统不具有交互性，它是为了提高CPU的利用率而提出的一种操作系统。
> 1
> 按照操作系统提供的服务进行分类，_ 是基本的操作系统。

- ( ) A.批处理操作系统、分时操作系统、网络操作系统
- (x) B.批处理操作系统、分时操作系统、实时操作系统
- ( ) C.批处理操作系统、分时操作系统、分布式操作系统
- ( ) D.分时操作系统、网络操作系统、分布式操作系统

知识点:操作系统概述
出处:网络
难度:1

> B
> 1
> 在_ 操作系统的控制下，计算机能及时处理过程控制装置反馈的信息，并作出响应。

- ( ) A.网络
- ( ) B.分时
- (x) C.实时
- ( ) D.批处理

知识点:操作系统概述
出处:网络
难度:1

> C
> 1
> 在计算机系统中，通常把财务管理程序看作是_

- ( ) A.系统软件
- ( ) B.支援软件
- ( ) C.接口软件
- (x) D.应用软件

知识点:操作系统概述
出处:网络
难度:1

> D
> 1
> 对计算机系统起着控制和管理作用的是_ 。

- ( ) A.硬件
- (x) B.操作系统
- ( ) C.编译系统
- ( ) D.应用程序

知识点:操作系统概述
出处:网络
难度:1

> B
> 1
> 关于UNIX系统中设备的说明，正确的是_ 。

- ( ) A.UNIX系统是按设备和内存间交换的物理单位对设备进行分类的，有流设备、字符设备和块设备
- (x) B.常把块设备称为存储设备，把字符设备称为输入输出设备
- ( ) C.UNIX对每一个设备赋予一个编号，称为“绝对号”，驱动程序按绝对号控制设备
- ( ) D.UNIX为每一类设备赋予一个编号，称为“设备号”，驱动程序按设备号控制设备

知识点:操作系统概述
出处:网络
难度:1

> B
> 1
> （ ）不是批处理多道程序的性质。

- ( ) A.“多道作业并发工作”
- (x) B.“未采用 spooling 技术”
- ( ) C.“作业成批输入”
- ( ) D.“作业调度可合理选择作业投入运行”

知识点:操作系统概述
出处:网络
难度:1

> B
> 1
> 网络操作系统和分布式操作系统的主要区别是（ ）

- ( ) A.是否连接多台计算机
- (x) B.各台计算机有没有主次之分
- ( ) C.计算机之间能否通信
- ( ) D.网上资源能否共享

知识点:操作系统概述
出处:网络
难度:1

> B 两者共同点是均可共享资源及相互通信，主要区别在于分布式操作系统还能够共享运算处理能力。
> 1
> 关于Unix的用户接口处理的说明不正确的提法是（ ）

- ( ) A.用户注册成功后，即处于shell控制下
- ( ) B.shell 以交互方式为用户服务

- ( ) C.shell 以某个提示符（如$）表示等待用户输入命令
- (x) D.用户打入shell命令行后，当这个命令行执行完以后，才再次显示提示符，等待用户输入下一命令

知识点:操作系统概述
出处:网络
难度:1

> D
> 1
> 操作系统核心部分的主要特点是_ 。

- ( ) A.一个程序模块
- (x) B.主机不断电时常驻内存
- ( ) C.有头有尾的程序
- ( ) D.串行顺序执行

知识点:操作系统概述
出处:网络
难度:1

> B
> 1
> 操作系统中用得最多的数据结构是_ 。

- ( ) A.堆栈
- ( ) B.队列
- (x) C.表格
- ( ) D.树

知识点:操作系统概述
出处:网络
难度:1

> C 操作系统中应用最多的数据结构是表格，像页表，段表，设备控制表等等
> 1
> 在操作系统管理中，面向用户的管理组织机构称为_ 。

- ( ) A.用户结构
- ( ) B.实际结构
- ( ) C.物理结构
- (x) D.逻辑结构

知识点:操作系统概述
出处:网络
难度:1

> D
> 1
> 单机操作系统的共享资源主要是指_ 。

- (x) A.内存、CPU和基本软件
- ( ) B.键盘、鼠标、显示器
- ( ) C.打印机、扫描仪
- ( ) D.软盘、硬盘、光盘

知识点:操作系统概述
出处:网络
难度:1

> A
> 1
> 为方便用户，操作系统负责管理和控制计算机系统的_ 。

- ( ) A.软件资源
- (x) B.硬件和软件资源
- ( ) C.用户有用资源
- ( ) D.硬件资源

知识点:操作系统概述
出处:网络
难度:1

> B
> 5
> 多道批处理系统的特征为 _____ ；
> _____ ；
> _____ 。

- [x]

知识点:操作系统概述
出处:网络
难度:1

> 特征:⑴多道性。⑵无序性。⑶调度性
> 5
> 批量处理系统的缺点为 _____ ；
> _____ 。

- [x]

知识点:操作系统概述
出处:网络
难度:1

> 平均周转时间长 无交互能力
> 5
> 在操作系统控制下的多个程序的执行顺序和每个程序的执行时间是不确定的，这种现象称为操作系统的_____ 。

- [x]

知识点:操作系统概述
出处:网络
难度:1

> 不确定性
> 5
> 作业管理的基本功能包括_____ 。

- [x]

知识点:操作系统概述
出处:网络
难度:1

> 人机交互，图形界面和系统任务管理等。
> 5
> 一个用户的作业从开始进入系统到结束在计算机系统中经过的阶段为
> _____ 。

- [x]

知识点:操作系统概述
出处:网络
难度:1

> 3个阶段：收容 运行 完成
> 5
> 按资源分配的方式可将外设分为_____ 。

- [x]

知识点:操作系统概述
出处:网络
难度:1

> 独占、共享、虚拟设备
> 5
> _____ 系统的出现，标志着操作系统的形成。

- [x]

知识点:操作系统概述
出处:网络
难度:1

> 多道批处理
> 5
> 操作系统的基本类型有_____ 。

- [x]

知识点:操作系统概述
出处:网络
难度:1

> 批处理操作系统、分时系统和实时系统
> 5
> 分时系统的特征为_____ 。

- [x]

知识点:操作系统概述
出处:网络
难度:1

> 交互性、及时性、独立性
> 5
> 分时系统的特征为_____ 。

- [x]

知识点:操作系统概述
出处:网络
难度:1

> 交互性、及时性、独立性
> 5
> 操作系统的特征为_____ 。

- [x]

知识点:操作系统概述
出处:网络
难度:1

> 并发性 、共享性 、 虚拟性 、 异步性
> 5
> 计算机系统按用户指定的步骤，为用户一次上机解题所完成的工作的总和称为_____ 。

- [x]

知识点:操作系统概述
出处:网络
难度:1

> 作业
> 5
> 在手工操作阶段，当程序在输入或输出时， C P U处于空闲等待，我们称这种现象为_____ 。

- [x]

知识点:操作系统概述
出处:网络
难度:1

> 人机矛盾
> 5
> _____ 系统的出现，标志着操作系统的形成。

- [x]

知识点:操作系统概述
出处:网络
难度:1

> 多道批处理
> 5
> 操作系统的基本类型有_____ 。

- [x]

知识点:操作系统概述
出处:网络
难度:1

> 批处理操作系统、分时操作系统和实时操作系统
> 5
> 分时系统的特征为_____ 。

- [x]

知识点:操作系统概述
出处:网络
难度:1

> 同时性、交互性、独立性、及时性
> 5
> 操作系统的特征为_____ 。

- [x]

知识点:操作系统概述
出处:网络
难度:1

> 并发 共享 虚拟和异步
> 3
> 脱机批处理解决了手工操作阶段的操作联机和输入/输出联机的问题。

- (x) A.对
- ( ) B.错

知识点:操作系统概述
出处:网络
难度:1

> A
> 5
> 在手工操作阶段，操作员在进行装卸卡和磁带等手工操作时， C P U处于空闲等待，我们称这种现象为_____ _____ 。

- [x]

知识点:操作系统概述
出处:网络
难度:1

> 人机矛盾
> 5
> 多道批处理系统的特征为_____
> ；_____
> ；_____。

- [x]

知识点:操作系统概述
出处:网络
难度:1

> ⑴多道性。⑵无序性。⑶调度性
> 3
> 过载保护是分时系统的一个特征。

- ( ) A.对
- (x) B.错

知识点:操作系统概述
出处:网络
难度:1

> B
> 5
> 批量处理系统的缺点为_____；
> _____。

- [x]

知识点:操作系统概述
出处:网络
难度:1

> 平均周转时间长 无交互能力
> 5
> 在操作系统控制下的多个程序的执行顺序和每个程序的执行时间是不确定的，这种现象称为操作系统的_____。

- [x]

知识点:操作系统概述
出处:网络
难度:1

> 不确定性
> 5
> 作业管理的基本功能包括_____。

- [x]

知识点:操作系统概述
出处:网络
难度:1

> 人机交互，图形界面和系统任务管理等
> 5
> 一个用户的作业从开始进入系统到结束在计算机系统中经过的阶段为_____
> _____。

- [x]

知识点:操作系统概述
出处:网络
难度:1

> 3个阶段：收容、运行、完成
> 5
> 常用的多道处理系统的作业调度算法有_____
> __。

- [x]

知识点:操作系统概述
出处:网络
难度:1

> 先来先服务 轮转法 多级反馈队列列算法 优先级法 短作业优先法
> 3
> 操作系统的不确定性是指同一程序使用相同的输入、在相同的环境下，经过多次运行却可能获得完全不同的结果。

- ( ) A.对
- (x) B.错

知识点:操作系统概述
出处:网络
难度:1

> B
> 5
> 依据操作系统的用户服务方式，可把操作系统分为___ 系统、___ 系统和____ 系统三种基本类型。

- [x]

知识点:操作系统概述
出处:网络
难度:1

> 批处理、分时、实时
> 5
> 从资源管理的观点出发，可把操作系统分为__ 、__ 、__ 、__ 和__
> 五大部分。

- [x]

知识点:操作系统概述
出处:网络
难度:1

> 存储管理 、设备管理、 文件管理 、 处理机管理 和 作业管理
> 5
> 简单 以多道程序设计为基础的现代操作系统具有__ 、__ 、__ 和__ 四个基本特征。

- [x]

知识点:操作系统概述
出处:网络
难度:1

> 并发性 、共享性 、 虚拟性 、 异步性
> 5
> 操作系统通常可分为 _ _、 _ _、 _ _ 三种基本类型。

- [x]

知识点:操作系统概述
出处:网络
难度:1

> 批处理系统、分时系统和实时系统
> 5
> 用户和操作系统之间的接口可分为 _ _ 和 _ _ 两类。

- [x]

知识点:操作系统概述
出处:网络
难度:1

> 联机命令、系统调用
> 5
> 操作系统通常可分为三种基本类型，即_____ 、_____ 和_____ 。

- [x]

知识点:操作系统概述
出处:网络
难度:1

> 批处理系统、分时系统和实时系统
> 5
> 通用操作系统的衡量指标为_____ 。

- [x]

知识点:操作系统概述
出处:网络
难度:1

> 稳定性与安全性
> 5
> 操作系统的管理功能包括_____ 。

- [x]

知识点:操作系统概述
出处:网络
难度:1

> 处理器管理、 存储器管理、 设备管理、 文件管理、 作业管理
> 5
> 操作系统的基本特征是___ ， ____ ， _____ 和_____ 。

- [x]

知识点:操作系统概述
出处:网络
难度:1

> 并发、共享性、虚拟和异步性
> 5
> 从用户的源程序进入系统到相应程序在机器上运行，所经历的主要处理阶段有_____ ，_____ ，_____ ，_____ **和** _____ 。

- [x]

知识点:操作系统概述
出处:网络
难度:1

> 编辑阶段, 编译阶段 ,连接阶段, 装入阶段和运行阶段
> 5
> 一般说来，操作系统有三种结构，它们分别是___ 结构,_____ 结构和_____ **结构。传统的UNIX系统核心就采用**___ 结构。

- [x]

知识点:操作系统概述
出处:网络
难度:1

> 单块、层次、微内核、层次
> 5
> 操作系统一般为用户提供了三种界面，它们是_____ ,_____ 和_____ ；在UNIX系统中，_____ 只能在C程序中使用。

- [x]

知识点:操作系统概述
出处:网络
难度:1

> 命令界面 图形界面 系统调用界面 系统调用
> 5
> 分时系统必须为用户提供__ 以实现_____ 控制方式。

- [x]

知识点:操作系统概述
出处:网络
难度:1

> 操作控制命令 交互(或联机)
> 5
> MS—DOS中有三个文件：DOSIP.EXE，DOSIP.DAT和DOSZP.COM，若使用系统提供的替代符' * '和'?'，则这三个文件可统一表示为___ 。

- [x]

知识点:操作系统概述
出处:网络
难度:1

> DOS?P. * (或DOS?P.???)
> 5
> 无论哪种操作系统都执行同样的资源管理功能，它们的构架是一样的，都由____ 、____ 、设备管理和文件系统组成。

- [x]

知识点:操作系统概述
出处:网络
难度:1

> 存储管理 进程管理
> 5
> 分时操作系统采用的分时技术是，将____ 划分为很短的时间片，系统将时间片轮流地分配给各联机用户使用。

- [x]

知识点:操作系统概述
出处:网络
难度:1

> 处理机运行时间
> 5
> DOS操作系统是一种_____ 类型的操作系统。

- [x]

知识点:操作系统概述
出处:网络
难度:1

> 磁盘管理
> 5
> 现代计算机系统是分态的，当操作系统程序执行时，机器处于____ 态。

- [x]

知识点:操作系统概述
出处:网络
难度:1

> 内核
> 5
> 从资源管理的角度看操作系统，它具有四大管理功能，其中，___ 是对系统软件资源的管理。

- [x]

知识点:操作系统概述
出处:网络
难度:1

> 文件系统
> 5
> DOS系统是_____ 类型的操作系统。

- [x]

知识点:操作系统概述
出处:网络
难度:1

> 解释磁盘管理
> 5
> 常用的资源分配策略有___ 和___ 两种。

- [x]

知识点:操作系统概述
出处:网络
难度:1

> 先请求先服务和优先调度两种
> 5
> 操作系统的基本特征一般包括：_____ 、共享、虚拟、异步性。

- [x]

知识点:操作系统概述
出处:网络
难度:1

> 并发
> 5
> 程序的并发执行和顺序执行相比，表现出一些新的特征即：间断性、失去封闭性、 _____ 。

- [x]

知识点:操作系统概述
出处:网络
难度:1

> 不可再现性
> 5
> 多道程序环境下的各道程序，宏观上，它们是在（ ）运行，微观上则是在（ ）执行。

- [x]

知识点:操作系统概述
出处:网络
难度:1

> 并行、串行
> 4
> 简述操作系统的定义。

- [x]

知识点:操作系统概述
出处:网络
难度:1

> 操作系统是计算机系统的一种系统软件，它统一管理计算机系统的资源和控制程序的执行。
> 4
> 在多道程序设计技术的系统中，操作系统怎样才会占领中央处理器?

- [x]

知识点:操作系统概述
出处:网络
难度:1

> 只有当中断装置发现有事件发生时，它才会中断当前占用中央处理器的程序执行，让操作系统的处理服务程序占用中央处理器并执行之。
> 4
> 简述“删除文件”操作的系统处理过程。

- [x]

知识点:文件系统
出处:网络
难度:1

> 用户用本操作向系统提出删除一个文件的要求，系统执行时把指定文件的名字从目录和索引表中除去，并收回它所占用的存储区域，但删除一个文件前应先关闭该文件。
> 4
> 对相关临界区的管理有哪些要求?

- [x]

知识点:同步互斥
出处:网络
难度:1

> 为了使并发进程能正确地执行，对若干进程共享某一变量（资源）的相关临界区应满足以下三个要求： ①
> 一次最我让一个进程在临界区中执行，当有进程在临界区中时，其他想进入临界区执行的进程必须等待； ②
> 任何一个进入临界区执行的进程必须在有限的时间内退出临界区，即任何一个进程都不应该无限逗留在自己的临界区中； ③
> 不能强迫一个进程无限地等待进入它的临界区，即有进程退出临界区时应让下一个等待进入临界区的进程进入它的临界区。
> 4
> 简述解决死锁问题的三种方法。

- [x]

知识点:死锁
出处:网络
难度:1

> ① 死锁的防止。系统按预定的策略为进程分配资源，这些分配策略能使死锁的四个必要条件之一不成立，从而使系统不产生死锁。 ②
> 死锁的避免。系统动态地测试资源分配情况，仅当能确保系统安全时才给进程分配资源。 ③
> 死锁的检测。对资源的申请和分配不加限制，只要有剩余的资源就呆把资源分配给申请者，操作系统要定时判断系统是否出现了死锁，当有死锁发生时设法解除死锁。
> 4
> 从操作系统提供的服务出发，操作系统可分哪几类?

- [x]

知识点:操作系统概述
出处:网络
难度:1

> 批处理操作系统、分时操作系统、实时操作系统、网络操作系统、分布式操作系统。
> 4
> 简述计算机系统的中断机制及其作用。

- [x]

知识点:中断、异常与系统调用
出处:网络
难度:1

> 中断机制包括硬件的中断装置和操作系统的中断处理服务程序。 中断装置由一些特定的寄存器和控制线路组成，中央处理器和外围设备等识别到的事件保存在特定的寄存器中，
> 中央处理器每执行完一条指令，均由中断装置判别是否有事件发生。 若无事件发生，CPU继续执行；若有事件发生，则中断装置中断原占有CPU的程序的执行，让操作系统
> 的处理事件服务程序占用CPU，对出现的事件进行处理，事件处理完后，再让原来的程序继续占用CPU执行。
> 4
> 选择进程调度算法的准则是什么?

- [x]

知识点:处理机调度
出处:网络
难度:1

> 由于各种调度算法都有自己的特性，因此，很难评价哪种算法是最好的。一般说来，选择算法时可以考虑如下一些原则： ① 处理器利用率； ② 吞吐量； ③ 等待时间；
> ④ 响应时间。 在选择调度算法前，应考虑好采用的准则，当确定准则后，通过对各种算法的评估，从中选择出最合适的算法。
> 4
> 独占设备采用哪种分配方式?

- [x]

知识点:文件系统
出处:网络
难度:1

> 独占设备通常采用静态分配方式。 即在一个作业执行前，将作业要使用的这类设备分配给作业，在作业执行期间均归该作业占用，直到作业执行结束才归还。
> 4
> 产生死锁的原因是什么?

- [x]

知识点:死锁
出处:网络
难度:1

> ① 系统资源不足； ② 进程推进顺序不合适。 在早期的系统中，由于系统规模较小，结构简单，以及资源分配大多采用静态分配法，使得操作系统死锁问题的严重性未能充
> 分暴露出来。但今天由于多道程序系统，以至于数据系统的出现，系统中的共享性和并行性的增加，软件系统变得日益庞大和复杂等原因，使得系统出现死锁现象的可能性大大增
> 加。
> 4
> 何谓批处理操作系统?

- [x]

知识点:操作系统概述
出处:网络
难度:1

> 用户准备好要执行的程序、数据和控制作业执行的说明书，由操作员输入到计算机系统中等待处理。操作系统选择作业并按作业说明书的要求自动控制作业的执行。采用这种批量
> 化处理作业的操作系统称为批处理操作系统。
> 4
> 对特权指令的使用有什么限制?

- [x]

知识点:操作系统概述
出处:网络
难度:1

> 只允许操作系统使用特权指令，用户程序不能使用特权指令。
> 4
> 影响缺页中断率有哪几个主要因素?

- [x]

知识点:缺页中断
出处:网络
难度:1

> 影响缺页中断率的因素有四个： ① 分配给作业的主存块数多则缺页率低，反之缺页中断率就高。 ② 页面大，缺页中断率低；页面小缺页中断率高。 ③
> 程序编制方法。以数组运算为例，如果每一行元素存放在一页中，则按行处理各元素缺页中断率低；反之，按列处理各元素，则缺页中断率高。 ④
> 页面调度算法对缺页中断率影响很大，但不可能找到一种最佳算法。
> 4
> 磁盘移臂调度的目的是什么？常用移臂调度算法有哪些?

- [x]

知识点:操作系统概述
出处:网络
难度:1

> 磁盘移臂调度的目的是尽可能地减少输入输出操作中的寻找时间。 常用的移臂调度算法有： ① 先来先服务算法 ② 最短寻找时间优先算法 ③ 电梯调度算法 ④
> 单向扫描算法。
> 4
> 常用的作业调度算法有哪些?

- [x]

知识点:处理机调度
出处:网络
难度:1

> ① 先来先服务算法 ② 计算时间短的作业优先算法 ③ 响应比最高者优先算法 ④ 优先数调度算法 ⑤ 均衡调度算法
> 4
> 计算机系统的资源包括哪些?

- [x]

知识点:操作系统概述
出处:网络
难度:1

> 计算机系统的资源包括两大类：硬件资源和软件资源。 硬件资源主要有中央处理器、主存储器、辅助存储器和各种输入输出设备。软件资源有编译程序、编辑程序等各种程序以及有关数据。
> 4
> CPU在管态和目态下工作有何不同?

- [x]

知识点:操作系统概述
出处:网络
难度:1

> 当中央处理器处于管态时，可以执行包括特权指令在内的一切面器指令，而在目态下工作时不允许执行特权指令。
> 4
> 何为页表和快表？它们各起什么作用?

- [x]

知识点:物理内存管理实验
出处:网络
难度:1

> 页表指出逻辑地址中的页号与所占主存块号的对应关系。 作用：页式存储管理在用动态重定位方式装入作业时，要利用页表做地址转换工作。
> 快表就是存放在高速缓冲存储器的部分页表。它起页表相同的作用。
> 由于采用页表做地址转换，读写内存数据时CPU要访问两次主存。有了快表，有时只要访问一次高速缓冲存储器，一次主存，这样可加速查找并提高指令执行速度。
> 4
> 作业在系统中有哪几种状态?

- [x]

知识点:进程状态与控制
出处:网络
难度:1

> 一个作业进入系统到运行结束，一般要经历进入、后备、运行和完成四个阶段，相应地，作业亦有进入、后备、运行和完成四种状态。 ①
> 进入状态：作业的信息从输入设备上预输入到输入井，此时称为作业处于进入状态。 ② 后备状态：当作业的全部信息都已输入，且由操作系统将其存放在输入井中，此时称作
> 业处于后备状态。系统将所有处于后备状态的作业组成后备作业队列，等待作业调度程序的调度。 ③
> 运行状态：一个后备作业被作业调度程序选中，分配了必要的资源，调入内存运行，称作业处于运行状态。 ④
> 完成状态：当作业正常运行完毕或因发生错误非正常终止时，作业进入这完成状态。
> 4
> 用fork创建新进程，它要做哪些工作?

- [x]

知识点:进程状态与控制
出处:网络
难度:1

> 由fork创建新进程的主要工作有： ① 在进程表proc[ ]中为子进程找一个空闲的表项，用来存放子进程的proc结构； ② 为子进程分配一个唯一的标识号；
> ③ 把父进程中的字段复制到子进程的proc中，并把p – pid置为分配到的进程标识号，把p-pid置为父进程的标识号，把p-stat置为创建状态； ④
> 按父进程中p-size所示的长度为子进程申请分配内存。若有足够的内存，则把父进程的user结构、栈和用户数据区全部复制到子进程的空间中；若无足够的内存，则在
> 磁盘对换区中分配存储空间，然后复制到对换区中，置于进程状态为就绪状态。
> 4
> 为什么说批处理多道系统能极大地提高计算机系统的工作效率?

- [x]

知识点:操作系统概述
出处:网络
难度:1

> ① 多道作业并行工作，减少了处理器的空闲时间。 ② 作业调度可以合理选择装入主存储器中的作业，充分利用计算机系统的资源。 ③
> 作业执行过程中不再访问低速设备，而直接访问高速的磁盘设备，缩短执行时间。 ④ 作业成批输入，减少了从操作到作业的交接时间。
> 4
> 操作系统为用户提供哪些接口?

- [x]

知识点:操作系统概述
出处:网络
难度:1

> 操作系统为用户提供两种类型的使用接口： 一是操作员级的，它为用户提供控制作业执行的途径； 二是程序员级的，它为用户程序提供服务功能。
> 4
> 什么是线程？多线程技术具有哪些优越性?

- [x]

知识点:进程状态与控制
出处:网络
难度:1

> 线程是进程中可独立执行的子任务，一个进程可以有一个或多个线程，每个线程都有一个惟一的标识符。线程与进程有许多相似之处，往往把线程又称为"轻型进程"，线程与进
> 程的根本区别是把进程作为资源分配单位，而线程是调度和执行单位。 多线程技术具有多个方面的优越性： ① 创建速度快、系统开销小：创建线程不需要另行分配资源；
> ② 通信简洁、信息传送速度快：线程间的通信在统一地址空间进程，不需要额外的通信机制； ③
> 并行性高：线程能独立执行，能充分利用和发挥处理器与外围设备并行工作的能力。
> 4
> UNIX系统中的优先权和优先数有什么关系？如何确定进程的优先权和优先数?

- [x]

知识点:进程状态与控制
出处:网络
难度:1

> UNIX中每个进程都有一个优先数，就绪进程能否占用处理器的优先权取决于进程的优先数，优先数越小则优先权越高。 UNIX以动态方式确定优先权，如核心的进程优先
> 权高于进入用户态的进程；降低用完一个时间片的进程的优先权；对进入睡眠的进程，其等待事件越急优先数越高；降低使用处理器时间较长的进程的优先权。 UNIX中确定
> 进程优先数的方法有两种：设置方法和计算方法。前者对要进入睡眠状态的进程设置优先数，若等待的事件急迫，则设置较小的优先数；后者用户进程正在或即将转入用户状态运
> 行时确定优先数。
> 4
> 主存空间信息保护有哪些措施?

- [x]

知识点:进程管理实验
出处:网络
难度:1

> 保存主存空间中的信息一般采用以下措施： ① 程序执行时访问属于自己主存区域的信息，允许它既可读，又可写； ② 对共享区域中的信息只可读，不可修改； ③
> 对非共享区域或非自己的主存区域中的信息既不可读，也不可写。
> 4
> 共享设备允许多个作业同时使用，这里的“同时使用”的含义是什么?

- [x]

知识点:文件系统
出处:网络
难度:1

> “同时使用”的含义是多个作业可以交替地启动共享设备，在某一时刻仍只有一个作业占有。
> 4
> 简述“打开文件”操作的系统处理过程。

- [x]

知识点:文件系统
出处:网络
难度:1

> 用户要使用一个已经存放在存储介质上的文件前，必须先提出“打开文件”要求。这时用户也必须向系统提供参数：用户名、文件名、存取方式、存储设备类型、口令等。系统在
> 接到用户的“打开文件”要求后，找出该用户的文件目录，当文件目录不在主存储器中时还必须把它读到主存储器中；然后检索文件目录，指出与用户要求相符合的目录项，取出
> 文件存放的物理地址。 对索引文件还必须把该文件的索引表存放在主存储器中，以便后继的读写操作能快速进行。
> 4
> 什么是“前台”作业、“后台”作业？为什么对“前台”作业要及时响应?

- [x]

知识点:操作系统概述
出处:网络
难度:1

> 批处理操作系统实现自动控制无需人为干预，分时操作系统实现了人机交互对话，这两种操作系统具有各自的优点。为了充分发挥批处理系统和分时系统的优点，在一个计算机系
> 统上配置的操作系统往往既具有批处理能力，又有提供分时交互的能力。这样，用户可以先在分时系统的控制下，以交互式输入、调试和修改自己的程序；然后，可以把调试好的
> 程序转交给批处理系统自动控制其执行而产生结果。这些由分时系统控制的作业称为“前台”作业，而那些由批处理系统控制的作业称为“后台”作业。
> 在这样的系统中，对前台作业应该及时响应，使用户满意；对后台作业可以按一定的原则进行组合，以提高系统的效率。
> 4
> 存储型设备和输入输出型设备的输入输出操作的信息传输单位有何不同?

- [x]

知识点:I/O子系统
出处:网络
难度:1

> 存储型设备输入输出操作的信息传输单位是“块”，而输入输出型设备输入输出操作的信息传输单位是“字符”。
> 4
> 简述信号量S的物理含义。

- [x]

知识点:信号量
出处:网络
难度:1

> S > 0时，S表示可使用的资源数；或表示可使用资源的进程数； S = 0时，表示无资源可供使用；或表示不允许进程再进入临界区；
> S < 0时， - S表示等待使用资源的进程个数；或表示等待进入临界区的进程个数；
> 当S > 0时，调用P（S）的进程不会等待；调用V（S）后使可用资源数加1或使可用资源的进程数加1；
> 当S < 0时，调用P（S）的进程必须等待；调用V（S）后将释放一个等待使用资源者或释放一个等待进入临界区者。

> 4
> 什么是计算机系统？它由哪几部分组成？

- [x]

知识点:操作系统概述
出处:网络
难度:1

> 计算机系统是按用户的要求接收和存储信息，自动进行数据处理并输出结果信息的系统。
> 计算机系统由硬件系统和软件系统组成。硬件系统是计算机系统赖以工作的实体，软件系统保证计算机系统按用户指定的要求协调地工作。
> 4
> 计算机系统怎样实现存储保护？

- [x]

知识点:操作系统概述
出处:网络
难度:1

> 一般硬件设置了基址寄存器和限长寄存器。 中央处理器在目态下执行系统中，对每个访问主存的地址都进行核对，若能满足：基址寄存器值≤访问地址≤基址寄存器值＋限长寄
> 存值，则允许访问，否则不允许访问。并且不允许用户程序随意修改这两个寄存器的值。这就实现了存储保护。
> 4
> 给出系统总体上的中断处理过程。

- [x]

知识点:中断、异常与系统调用
出处:网络
难度:1

> CPU每执行完一条指令就去扫描中断寄存器，检查是否有中断发生，若没有中断就继续执行下条指令；若有中断发生就转去执行相应的中断处理程序。中断处理过程可粗略的分
> 为以下四个过程： ① 保护当前正在运行程序的现场； ② 分析是何种中断，以便转去执行相应的中断处理程序； ③ 执行相应的中断处理程序； ④
> 恢复被中断程序的现场。
> 4
> 死锁发生的必要条件有哪些？

- [x]

知识点:死锁
出处:网络
难度:1

> 发生死锁的必要条件有四点：互斥条件、不可抢占条件、部分分配条件和循环等待条件。 ① 互斥条件：系统中存在一个资源一次只能被一个进程所使用； ②
> 非抢占条件：系统中存在一个资源仅能被占有它的进程所释放，而不能被别的进程强行抢占。 ③
> 占有并等待条件：系统中存在一个进程已占有了分给它的资源，但仍然等待其他资源。 ④
> 循环等待条件：在系统中存在一个由若干进程形成的环形请求链，其中的每一个进程均占有若干种资源中的某一种，同时每个进程还要求（链上）下一个进程所占有的资源。
> 4
> 用户程序中通常用什么方式指定要使用的设备？为什么？

- [x]

知识点:文件系统
出处:网络
难度:1

> 用户程序中通常用“设备类、相对号”请求要使用的设备，即不具体指定要哪一台设备，而是提出要申请哪类设备多少台。 这种方式使设备分配适应性好、灵活性强。
> 否则若用绝对号来指定设备，如果这台设备已被占用或有故障时，该作业就无法装入主存中。
> 4
> 进程调度中“可抢占”和“非抢占”两种方式，哪一种系统的开销更大？为什么？

- [x]

知识点:处理机调度
出处:网络
难度:1

> 可抢占式会引起系统的开销更大。 可抢占式调度是严格保证任何时刻，让具有最高优先数（权）的进程占有处理机运行，因此增加了处理机调度的时机，引起为退出处理机的进
> 程保留现场，为占有处理机的进程恢复现场等时间（和空间）开销增大。
> 4
> 一个含五个逻辑记录的文件，系统把它以链接结构的形式组织在磁盘上，每个记录占用一个磁盘块，现要求在第一记录和第二记录之间插入一个新记录，简述它的操作过程。

- [x]

知识点:文件系统
出处:网络
难度:1

> 从文件目录中找到该文件，按址读出第一个记录； 取出第一个记录块中指针，存放到新记录的指针位置； 把新记录占用的物理块号填入第一个记录的指针位置；
> 启动磁盘把第一个记录和新记录写到指字的磁盘块上。
> 4
> 在SPOOL系统中设计了一张“缓输出表”，请问哪些程序执行时要访问缓输出表，简单说明之。

- [x]

知识点:文件系统
出处:网络
难度:1

> 井管理写程序把作业执行结果文件登记在缓输出表中； 缓输出程序从缓输出表中查找结果文件并打印输出。
> 4
> 试比较进程调度与作业调度的不同点。

- [x]

知识点:处理机调度
出处:网络
难度:1

> ① 作业调度是宏观调度，它决定了哪一个作业能进入主存。进程调度是微观调度，它决定各作业中的哪一个进程占有中央处理器。 ② 作业调度是选符合条件的收容态作业装入主存。进程调度是从就绪态进程中选一个占用处理器。
> 4
> 试说明资源的静态分配策略能防止死锁的原因。

- [x]

知识点:死锁
出处:网络
难度:1

> 资源静态分配策略要求每个过程在开始执行前申请所需的全部资源，仅在系统为之分配了所需的全部资源后，该进程才开始执行。这样，进程在执行过程中不再申请资源，从而破坏了死锁的四个必要条件之一“占有并等待条件”，从而防止死锁的发生。
> 4
> 简述操作系统提供的服务功能。

- [x]

知识点:操作系统概述
出处:网络
难度:1

> 处理用户命令；读/写文件；分配/回收资源； 处理硬件/软件出现的错误；及其他控制功能。
> 4
> 简述中断装置的主要职能。

- [x]

知识点:中断、异常与系统调用
出处:网络
难度:1

> 中断装置的职能主要有三点： ① 检查是否有中断事件发生； ② 若有中断发生，保护好被中断进程的断点及现场信息，以便进程在适当时候能恢复执行； ③
> 启动操作系统的中断处理程序。
> 4
> 实现虚拟设备的硬件条件是什么？操作系统应设计哪些功能程序?

- [x]

知识点:文件系统
出处:网络
难度:1

> 硬件条件是：配置大容量的磁盘，要有中断装置和通道。 操作系统应设计好“预输入”程序，“井管理”程序，“缓输出”程序。
> 4
> 一个具有分时兼批处理功能的操作系统应怎样调度和管理作业?

- [x]

知识点:处理机调度
出处:网络
难度:1

> ① 优先接纳终端作业，仅当终端作业数小于系统可以允许同时工作的作业数时，可以调度批处理作业； ② 允许终端作业的批处理作业混合同时执行； ③
> 把终端作业的就绪进程排成一个就绪队列，把批处理作业的就绪进程排入另外的就绪队列中； ④
> 有终端作业进程就绪时，优先让其按“时间片轮转”法先运行。没有终端作业时再按确定算法选批处理作业就绪进程运行。
> 4
> 简述死锁的防止与死锁的避免的区别。

- [x]

知识点:死锁
出处:网络
难度:1

> 死锁的防止是系统预先确定一些资源分配策略，进程按规定申请资源，系统按预先规定的策略进行分配从而防止死锁的发生。
> 而死锁的避免是当进程提出资源申请时系统测试资源分配仅当能确保系统安全时才把资源分配给进程，使系统一直处于安全状态之中，从而避免死锁。
> 1
> 计算机操作系统是一个（ ）

- ( ) A.应用软件
- ( ) B.硬件的扩展
- ( ) C.用户软件
- (x) D.系统软件

知识点:操作系统概述
出处:网络
难度:1

> D
> 1
> （ ）不是分时系统的基本特征

- ( ) A.同时性
- ( ) B.独立性
- (x) C.多路性
- ( ) D.交互性

知识点:操作系统概述
出处:网络
难度:1

> C
> 5
> 操作系统是计算机系统的一种系统软件，它以尽量合理、有效的方式组织和管理计算机 的____，并控制程序的运行，使整个计算机系统能高效地运行。

- [x]

知识点:操作系统概述
出处:网络
难度:1

> 软硬件资源
> 5
> 进程主要由__ 、 ____ 、__三部分内容组成，其中是进程存在的
> 唯一标志。而__部分也可以为其他进程共享。

- [x]

知识点:进程状态与控制
出处:网络
难度:1

> 程序段；数据段；PCB；PCB；程序段
> 5
> 死锁是指在系统中的多个_____无限期地等待永远不会发生的条件。

- [x]

知识点:死锁
出处:网络
难度:1

> 进程
> 5
> 进程调度负责__的分配工作。

- [x]

知识点:处理机调度
出处:网络
难度:1

> 处理机
> 5
> 把__地址转换为__和地址的工作称为地址映射。

- [x]

知识点:操作系统概述
出处:网络
难度:1

> 逻辑；物理
> 5
> 重定位的方式有__和__两种。

- [x]

知识点:操作系统概述
出处:网络
难度:1

> 静态重定位；动态重定位
> 3
> 分时操作系统必然建立在多道程序技术的基础之上。

- (x) A.对
- ( ) B.错

知识点:操作系统概述
出处:网络
难度:1

> A
> 4
> 什么是操作系统？它有什么功能都有哪些?

- [x]

知识点:操作系统概述
出处:网络
难度:1

> 操作系统是计算机系统中的一个系统软件，是一些程序模块的集合。它们能以尽量有效、
> 合理的方式组织和管理计算机的软硬件资源。合理的组织计算机的工作流程，控制程序的 执行并向用户提供各种服务功能。
> 使得用户能够灵活、方便、有效的使用计算机，使整 个计算机系统能高效地运行 。
> 它的基本功能有：处理机管理、存储器管理、设备管理和软件资源的管理
> 4
> 进程与程序是两个完全不同的概念，但又有密切的联系，试写出两者的区别。

- [x]

知识点:进程状态与控制
出处:网络
难度:1

> 1. 动态性和静态性：进程是一个动态概念，程序是一个静态概念。程序可以作为一种软 件资源长期保存进程是把程序作为它的运行实体，没有程序，也就没有进程。
> 2. 进程控制块：进程由：程序+数据+PCB构成
> 3. 一对多的关系：一个程序可对应多个进程，一个进程为多个程序服务
> 4. 并发性：多个进程实体，能在一段时间内同时执行；而程序无法描述并发执行
> 5. 进程具有创建其他进程的功能，而程序没有
> 6. 操作系统中的每一个程序都是在一个进程现场中运行的
>
> 4
> 设有A、B、C三组进程，它们互斥地使用某一独占型资源R，使用前申请，使用后释放。资源 分配原则如下：
> 当只有一组申请进程时，该组申请进程依次获得R；
> 当有两组申请进程时，各组申请进程交替获得R，组内申请进程交替获得R；
> 当有三组申请进程时，各组申请进程轮流获得R，组内申请进程交替获得R。
> 试用信号灯和PV操作分别给出各组进程的申请活动程序段和释放活动程序段。

- [x]

知识点:信号量
出处:网络
难度:1

> ```
> Int Free=1;                  // 设备状态标志
> semaphore mutex=1;
> semaphore qa=qb=qc=0;         // 各组等待队列
> Int counta=countb=countc= 0; // 等待队列长度
> ```
>
> A组申请： P(mutex); if Free==1 then begin Free=0; V(mutex); end else begin
> counta++; V(mutex); P(qa); end
> A组释放： if countb>0 then begin countb--; V(qb); end else begin if countc>0 then
> begin countc--; V(qc); end else begin if counta > 0 then begin counta--;
> V(qa); end else begin Free=1; end end end
> A组进程活动可以给出B组和C组进程活动。
> 4
> 进程A1、A2、...、An1通过m个缓冲区向进程B1、B2、...、Bn2不断发送消息。发送和接收 工作遵循下列规则：
> 每个发送进程一次发送一个消息，写入一个缓冲区，缓冲区大小等于消息长度；
> 对每个消息，B1，B2，Bn2都须各接收一次，读入各自的数据区内；
> m个缓冲区都满时，发送进程等待，没有可读消息时，接收进程等待。
> 试用P、V操作组织正确的发送和接收工作。

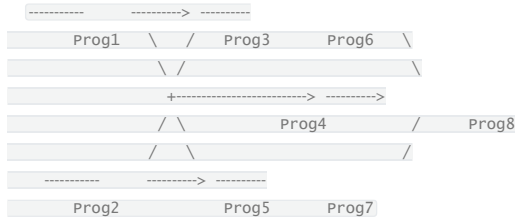
- [x]

知识点:信号量
出处:网络
难度:1

> 每个缓冲区只要写一次但要读n2次，因此，可以看成n2组缓冲区，每个发送者要同时写n2 个缓冲区，而每个接收者只要读它自己的缓冲区。
>
> ```
> Sin[n2]=m Sout[n2]=0;
>     cobegin
>         procedure Aj:
>             while (1) begin
> ```

```
                for(i=1;i<=n2;i++)
                    P(Sin[i]);
                P(mutex);
                // 将数据放入缓冲区
                V(mutex);
                for(i=1;i<=n2;i++)
                    V(Sout[2]);
            end
        procedure Bi:
            while (1) begin
                P(Sout[i]);
                P(mutex);
                // 从缓冲区取数据
                V(mutex);
                V(Sin[i]);
            end
    coend
```

4
设有8个程序prog1，prog2，…，prog8。它们在并发系统中执行时有如下所示的制约关系， 使用P、V操作实现这些程序间的同步。



- [x]

知识点:信号量
出处:网络
难度:1

本题目是用来检查考生对使用P、V操作实现进程间同步的掌握情况。一般地，若要求进程B在进程A之后方可执行时，只需在进程P操作，而在进程A执行完成时对同一信号量进行V操作即可。本题要求列出8个进程（程序）的控制关系，使题目显得较为复杂。但当对进程间的同步理解透彻后，应不难写出对应的程序。解这一类问题还应注意的一点是，要看 清图示的制约关系，不要漏掉或多处制约条件。

```
BEGIN
var s13, s14, s15, s23, s24, s25,s36, s48, s57, s68, s78: semaphore;
s13 :=0; s14 :=0; s15 :=0; s23 :=0; s24 :=0; s25 :=0; s36 :=0;
s48 :=0; s57 :=0; s68 :=0; s78 :=0;
COBEGIN
  prog1:         prog2:        prog3:        prog4:
    BEGIN          BEGIN         BEGIN         BEGIN
      do work;       do work;      V(S13);       P(S14);
      V(s13);        V(s23);       V(S23);       P(S24);
      V(s14);        V(s24);       do work;      do work;
      V(s15);        V(s25);       V(s36);       V(s48);
    END            END           END           END
  prog5:         prog6         prog7         prog8
    BEGIN          BEGIN         BEGIN         BEGIN
      P(s15);        P(s36);       P(s57);       P(s48);
      P(s25);        do work;      do work;      P(s68);
      do work;       V(s68);       V(S78);       P(s78);
      V(57);       END           END             do work;
    END                                          END
COEND
END
```

4
把学生和监考老师都看做进程，学生有N个人，教师1人，考场门口每次只能进出一个人，进考场原则是先来先进，当N个学生都进入考场后，教师才能发试卷。学生交卷后可以离开考场，教师要等收上来全部试卷并封装试卷后才能离开考场。问共需设置几个进程？使用P，V 操作解决上述问题中的同步和互斥关系。

- [x]

知识点:信号量
出处:网络
难度:1

```
var mutex,Beginready,Testready,Endready:semaphore;
    //mutex用以标示教室门这个临界资源
    //beginready等待考生来全，标示考试开始
    mutex:=1;
    Beginready:=-(N-1);
    Testready:=0;
    Endready:=-(N-1);
    cobegin
        Procedure Student       Procedure Teacher
            P(mutex);               P(mutex);
```

```
                // Enter;                // Enter;
            V(mutex);               V(mutex);
                // Waiting;
            P(Beginready);
-------------------------------------------------
                                    P(Beginready);
                                    // Hand Out;
                                    V(Beginready);
-------------------------------------------------
            P(Testready);
            V(Testready);
            // 答题
            // 交卷
            // 离开
            V(Endready);
-------------------------------------------------
                                    P(Endready);
                                    // 封卷离开
```

4
我们将只读数据的进程称为“读者”进程，而写或者修改数据的进程称为“写者”进程，允许多
个“读者”同时读数据，但不运行写者与其它读者或者写者进程同时访问数据。另外，要保证：
一旦有写者等待，新到达的读者必须等待，直到该写者完成数据访问为止，用P,V 操作实现 读者，写者同步。

- [x]

知识点:信号量
出处:网络
难度:1

互斥资源：读写者问题，隐含一个互斥资源-读写的问件
互斥锁：读文件时不能写，写文件时不能读文件
读进程：允许多个文件读，读进程时> 0时，锁定文件，读文件进程< 1时，解锁；读进程 数> 0时，说明读进程拥有锁
写进程:拥有锁时写文件

```
增加一个信号量w：=1，用以在写进程到达时封锁后续进程
  cobegin
  procedure Reader            procedure Writer
    begin                       begin
                                P(w);
      P(rmutex);                P(wmutex);
      if rcount==0 then         // 写数据：
        P(wmutex);              V(wmutex);
        V(rmutex);              V(w)
        V(w);                 end
        // 读数据：
        P(rmutex);
        rcount:=rcount-1;
        if rcount==0 then
         V(rmutex);
          V(rmutex);
          end
  coend
```

4
三个吸烟者在一间房间内，还有一个香烟供应者。为了制造并抽掉香烟，每个吸烟者需要三
样东西：烟草、纸和火柴。供应者有丰富的货物提供。三个吸烟者中，第一个有自己的烟草，
第二个有自己的纸，第三个有自己的火柴。供应者将两样东西放在桌子上，允许一个吸烟者
进行对健康不利的吸烟。当吸烟者完成吸烟后唤醒供应者，供应者再放两样东西（随机地） 在桌面上，然后唤醒另一个吸烟者。试为
吸烟者和供应者编写程序解决问题。

- [x]

知识点:信号量
出处:网络
难度:1

每个吸烟者需要一个进程，分别和经销商进行同步
互斥资源：桌子
A,B,C,D四个进程，A表示烟草拥有者，B是纸拥有者，C火柴拥有者，D经销商
S实现互斥，表示桌子上是否放有东西
Sad,Sbd,Scd分别表示进程AD,BD,CD之间的同步

```
cobegin
经销商              烟草拥有者          纸拥有者              火柴拥有者
  begin               begin               begin                 begin
    P(s);               P(Sad);             P(Sbd);               P(Scd);
    // 放原料：          // 取纸和火柴：      // 取烟草和火柴：      // 取纸和烟草：
    if(纸和火柴)        V(s);               V(s);                 V(s);
      V(Sad);           // 吸烟：           // 吸烟：              // 吸烟：
    else              end                 end                   end
      if(烟草和火柴)
        V(Sbd);
     else
       V(Scd);
```

```
      end
  coend
```

4
有n+1个进程A1，A2，... ,An和B：
A1,A2,... ,An通过同一个缓冲池各自不断地向B发送消息，B不断地取消息，它必须取走 发来的每个消息，刚开始时缓冲区为空，使用P,V操作实现之。
若缓冲区个数增至M个，试用P,V实现正确通讯。

- [x]

知识点:信号量
出处:网络
难度:1

```
var full,empty,mutex:semaphore;
    full=0;
    empty=1;
    mutex=1;
  cobegin
    procedure A_i(i=1,...,n)          procedure  B:
      begin                             begin
        P(empty);                         P(full);
        P(mutex);                         P(mutex);
        // put message to the buffer;     // Get the message;
        V(mutex);                         V(mutex);
        V(full);                          V(empty);
    end                               end
  coend
```

4
阅览室问题
有一个阅览室，共有100个座位，读者进入时必须先在一张登记表上登记，该表为每一个座位列一表目，包括座号和读者姓名等，读者离开时要消掉登记的信息，试问；
(1)为描述读者的动作，应编写几个程序，设置几个进程?
(2)试用PV操作描述各个进程之间的同步互斥关系。

- [x]

知识点:信号量
出处:网络
难度:1

读者动作有两个，一个时填表进入阅览室，这时要考虑阅览室里是否有空位；一是读者阅读完毕，离开阅览室，这时的操作要考虑阅览室里是否有读者。读者在阅览室读书时，由
于没有引起资源的变动，不算动作变化。 算法的信号量有三个：seats-表示阅览室时否有座位(初值为100)；readers-表示阅览室里的读者数，初值为0；用于互斥的mutex，初值为1。

```
var seats, raaders, mutex:semaphore;
    seats:=100;
    readers:=0;
    mutex:=1;
cobegin
   procedure Enter
   begin
       while TRUE
       begin
           p(seats);   //没有座位则离开
           p(mutex);   //进入临界区
           填写登记表;
           进入阅览室阅读;
           v(mutex);   //离开临界区 v(readers);
       end
   end
   procedure Leave
   begin
       while TRUE
       begin
           p(readers);
           p(mutex);
           消掉登记;
           离开阅览室;
           v(mutex);
           v(seats);
       end
   end
coend。
```

4
P,V改错(2001)
设有两个优先级相同的进程P1，P2如下。令信号S1，S2的初值为0，已知z=2，试问P1，P2并发运行结束后x=? y=? z=?

进程P1	进程P2
y:=1;	x:=1;
y:=y+2;	x:=x+1;
V(S1);	P(S1);
z:=y+1;	x:=x+y;
P(S2);	V(S2);
y:=z+y;	z:=x+z;

- [x]

知识点:信号量
出处:网络
难度:1

> 受信号量S1和S2的控制，进程P1和P2中P,V操作的顺序应明确。但当进程P2执行V(S2)调用后，可能会产生这种情形，即P2中的语句“z:=x+z”可以在
> P1中的语句“y:=z+y”前面或后面执行，因而P1和P2并发运行结束后，有两种可能的结果。即： x=5、y=12、z=9或x=5、y=7、z=9。
> 4
> 打印机问题
> 设系统中有5台类型相同的打印机，依次编号为1～5。又设系统中有n个使用打印机的进程，使用前申请，使用后释放。每个进程有一个进程标识，用于区别不同的进程。每个
> 进程还有一个优先数，不同进程的优先数各异。当有多个进程同时申请时，按照进程优先数由高到低的次序实施分配。试用信号灯和PV操作实现对于打印机资源的管理，即要求
> 编写如下函数和过程:
> (1)函数require(pid，pri): 申请一台打印机。参数pid为进程标识，其值为1到n的整数; pri为进程优先数，其值为正整数;
> 函数返回值为所申请到打印机的编号，其值为1到5的整数;
> (2)过程return(prnt): 释放一台打印机。参数prnt为所释放打印机的编号，其值为1到5的整数。

- [x]

知识点:信号量
出处:网络
难度:1

```
#define N 5
Int flag[N+1]; //flag[0]表示可用打印机数,
//flag表示第i号打印机的状态（1<=i<=N），0表示占用，1表示空闲
PCB queue=NULL;//进程阻塞队列
semaphore mutex_flag=1;//用于对flag数组的互斥操作
semaphore mutex_queue=1;//用于对阻塞队列的互斥操作
int require(int pid,int priority)
{
   P(mutex_flag);
   if(flag[0]>0)
   {
       flag[0]--;
       for(int  i=1;i<N+1;i++)
           if(flag=  =1)
           {
               flag=0;
               break;
           }
       V(mutex_flag);
       return i;
   }
   else
   {
       V(mutex_flag);
       p(mutex_queue);
       将进程pid按其优先数插入到等待队列queue中;
       V(mutex_queue);
   }
}
return(int print)
{
   P(mutex_flag);
   if(queue==NULL)
   {
       flag[0]++;
       flag[print]=1;
       V(mutex_flag);
   }
   else
   {
       V(mutex_flag);
       p(mutex_queue);
       将print分配给queue队首进程;
       queue下移;
       V(mutex_queue);
   }
```

```
}
```

4
批处理系统问题
设某个批处理系统中，有三个进程：卡片输入进程、作业调度进程、作业控制进程。他们之间的合作关系是：
只要卡片输入机上有作业信息输入，进程把作业逐个输入至输出井并为每个作业建立一个JCB块并把它插入至后备作业队列(JCB链表)中。
当内存中无作业运行时，作业调度进程从JCB中选一个作业，把该作业装入内存。
作业控制进程负责处理已调入内存的作业。
(1)P,V写出输入和调度进程的同步。
(2)用消息缓冲痛惜，写出调度进程与作业控制进程间的同步算法。

- [x]

知识点:信号量
出处:网络
难度:1

```
procedure 输入:
begin
   L1:
   如果有卡片 then L2
   等待卡片;
   L2:
   把作业输入至输出井并建立JCB块;
   p(s);
   把JCB插入链中;
   v(mutex);
   v(s);
   Goto L1;
end
procedure 调度:
begin
   M:
   P(s);
   p(mutex);
   查JCB;
   v(s);
   send();//向控制进程发信息
   receive();//接受信息
   Goto M;
end
procedure 作业控制:
begin
   N:
   receive();
   处理; send();//向调度发信息
   Goto N;
end
```

4
保管员问题
有一材料保管员，他保管纸和笔若干。有A 、B 两组学生，A 组学生每人都备有 纸，B 组学生每人都备有笔．任一学生只要能得到其他一种材料就可以写信。有一个可
以放一张纸或一支笔的小盒，当小盒中无物品时，保管员就可任意放一张纸或一支笔 供学生取用，每次允许一个学生从中取出自己所需的材料，当学生从盒中取走材料后
允许保管员再存放一件材料，请用信号量与P 、v 操作

- [x]

知识点:信号量
出处:网络
难度:1

```
var
s, Sa.Sb, mutexa, mutexb: semaphore;
s: = mutexa : =mutexb: = 1;
sa: = sb: = 0;
box: (PaPer, Pen);
cobegin
   process 保管员
       begin
           repeat
           P(S);
           take a material intobox ;
           if (box)=Paper then V(Sa);
           else V(Sb);
           untile false ;
       end
Process A组学生
   begin
       repeat
       P(Sa);
       P(mutexa);
       take the pen from box ;
```

```
        V(mutexa);
        V(S);
        write a letter;
        untile false ;
    end
 Process B组学生
    begin
        repeat
        P(Sb);
        P(mutexb);
        take the paper from box ;
        V(mutexb);
        V(S);
        wnte a letter ;
        untile false ;
    end
 Coend.
```

4
招聘问题
现有100名毕业生去甲、乙两公司求职，两公司合用一间接待室，其中甲公司招 收10 人，乙公司准备招收10人，招完为止。两公司各有一位人事主管在接待毕业生，
每位人事主管每次只可接待一人，其他毕业生在接待室外排成一个队伍等待。试用信 号量和P 、v 操作实现人员招聘过程。

- [x]

知识点:信号量
出处:网络
难度:1

由于毕业生仅排成一队，故用如图的一个队列数据结构表示。在队列中不含甲、乙公司.
A| B| A| A| B| Sm
A| B| Sn
B| A| ...| | |
---|---|---|---|---|---|---|---|---|---|---|---
都接待过的毕业生和已被录用的毕业生。只含标识为A （被甲接待过）或只含标识为B （被乙接待过）及无标识的毕业生队列。此外，sm 和Sn
分别为队列中甲、乙正在面试的毕业生i (i = 1 , 2 ， ...，100 ）标识、即此刻另一方不得面试该毕业生i 。K1和K2
为甲、乙所录取的毕业生数，C1 、C2 为互斥信号量。注意，如果甲录取了一人，且该生没有被乙面试的话，则乙面试的毕业生将减1 。办法是：如果甲录取了一人，且该生没有被乙面试可把乙的面试计数器C2加1 （相当于乙己面试了他），从而，保证乙面试的人数值为100 。反之对甲亦然。

```
var Sa,Sb,mutex:semaphore;
Sa:=Sb:=mnutex:=1;
C1,C2,K1,K2: integer;
C1:=C2:=K1:=K2:=0;
cobegin
   process 甲公司
       begin
       L1: P(mutex);
       P(Sa);
       C1:==C1+1 ;
       V(Sa);
       If C1≤100 then
        {
            从标识为B 且不为Sn 或
            无标识的毕业生队列中选
            第i 个学生，将学生i 标
            识为A 和Sm
        }
       V(mutex) ;
       面试:
       P(mutex);
       if 合格then
       {
           K1:=K1+1;
           if 学生i 的标识不含B then
           {
               P (Sb);
               C2:=C2+1;
               V(Sb);
               将学生i 从队列摘除;
           }
           else 将学生i 从队列摘除;
       }
       else if 学生i 的标识含B then
                将学生i 从队列摘除;
           else
               取消学生i 的Sm 标识;
       V(mutex);
       If(K1<10)&(C2<100) then
           goto L1;
end
process 乙公司
begin
```

```
    L2:P(mutex);
    P(Sb);
    C2:=C2+1;
    V(Sb);
    if C2≤100 then
        从标识为A 且不为sm 或无标识的
        毕业生队列中选第i个学生将学生i
        标识为B和Sn
    V(mutex);
    面试;
    P(mutex);
    if 合格then
    {
        K2:=K2+1;
        if 学生i 的标识不含A then
        {
        P(Sa)
        C1:=C1+1;
        V(Sa);
        将学生i 从队列摘除;
        }
        else 将学生i 从队列摘除;
    }
    else if 学生i 的标识含A then
             将学生i 从队列摘除;
        else
             取消学生i 的Sn 标识;
    V(mutex);
    if(K2<10)&(c1<100）then
        goto L2;
end
coend
```

4
博物馆-公园问题
Jurassic公园有一个恐龙博物馆和一个花园，有m 个旅客租卫辆车，每辆车仅能乘
一个一旅客。旅客在博物馆逛了一会，然后，排队乘坐旅行车，挡一辆车可用喊飞它载 入一个旅客，再绕花园行驶任意长的时间。若n 辆车都己被旅客乘坐游玩，则想坐车的
旅客需要等待。如果一辆车己经空闲，但没有游玩的旅客了，那么，车辆要等待。试用 信号量和P 、V 操作同步m 个旅客和n 辆车子。

- [x]

知识点:信号量
出处:网络
难度:1

这是一个汇合机制，有两类进程：顾客进程和车辆进程，需要进行汇合、即顾客要坐进车辆后才能游玩，开始时让车辆进程进入等待状态
解答:

```
var sc1 , sck , sc , Kx,xc , mutex : semaphore ;
sck:=kx:=sc:=xc:=0;
sc1:=n ; mutex : = 1 ;
sharearea : 一个登记车辆被服务乘客信息的共享区;
cobegin
   process 顾客i ( i = 1 , 2 , … )
   begin
       P (sc1) ; /车辆最大数量信号量
       P (mutex) ; /封锁共享区，互斥操作
       在共享区sharearea登记被服务的顾客的信息;
       起始和到达地点，行驶时间
       V (sck) ; / 释放一辆车 ,即顾客找到一辆空车
       P(Kx); / 待游玩结束之后，顾客等待下车
       V(sc1) ; /空车辆数加1
   End
   Process 车辆j(j=1,2,3…)
   Begin
       L:P(sck); /车辆等待有顾客来使用
       在共享区sharearea登记一辆车被使用，并与顾客进程汇合;
       V(mutex); /这时可开放共享区，让另一顾客雇车
       V(kx); /允许顾客用此车辆
       车辆载着顾客开行到目的地;
       V(xc); /允许顾客下车
       Goto L;
   End
coend
```

4
生产流水线问题
设自行车生产线上有一只箱子，其中有N 个位置(N ≥3)，每个位置可存放一个车架或一个车轮; 又设有三个工人，其活动分别为:
工人1活动: do{ 加工一个车架; 车架放入箱中; }while(1) | 工人2活动： do{ 加工一个车轮; 车轮放入箱中; }while(1)
| 工人1活动： do{ 箱中取一个车架; 箱中取两个车轮; 组装为一台车; }while(1)
---|---|---
试分别用信号灯与PV 操作实现三个工人的合作，要求解中不含死锁。

- [x]

知识点:信号量
出处:网络
难度:1

> 问题分析:
>
> 用信号灯与PV 操作实现三个工人的合作首先不考虑死锁问题，工人1与工人3、工 人2与工人3构成生产者与消费者关系，这两对生产/消费关系通过共同的缓冲区相联
> 系。从资源的角度来看，箱字中的空位置相当于工人1和工人2的资源，而车架和车轮相 当于工人3的资源。定义三个信号灯如下:
> ----------The P,V code Using Pascal----------
> semaphore empty=N; semaphore wheel=0; semaphore frame=0; 三位工人的活动分别为: procedure
> 工人1: do{ 加工一个车架; P(empty); 车架放入箱中; V(frame); }while(1) | 工人2活动: do{ 加工一个车轮;
> P(empty); 车轮放入箱中; V(wheel); }while(1) | 工人1活动: do{ P(frame); 箱中取一个车架;
> V(empty); P(wheel); P(wheel); 箱中取两个车轮; V(empty); V(empty); 组装为一台车; }while(1)
> ---|---|---
> 分析上述解法易见，当工人1推进速度较快时，箱中空位置可能完全被车架占满或只留 有一个存放车轮的位置，而当此时工人3同时取2个车轮时将无法得到，而工人2又无法
> 将新加工的车轮放入箱中；当工人2推进速度较快时，箱中空位置可能完全被车轮占满，
> 而当此时工人3同取车架时将无法得到，而工人1又无法将新加工的车架放入箱中。上述
> 两种情况都意味着死锁。为防止死锁的发生，箱中车架的数量不可超过N-2，车轮的数
> 量不可超过N-1，这些限制可以用两个信号灯来表达。如此，可以给出不含死锁的完整 解法如下:
> ----------The P,V code Using Pascal----------
> semaphore s1=N-2; semaphore s2=N-1; procedure 工人1: do{ 加工一个车架; P(s1);
> P(empty); 车架放入箱中; V(frame); }while(1) | 工人2活动: do{ 加工一个车轮; P(s2); P(empty);
> 车轮放入箱中; V(wheel); }while(1) | 工人1活动: do{ P(frame); 箱中取一个车架; V(empty); V(s1);
> P(wheel); P(wheel); 箱中取两个车轮; V(empty); V(empty); V(s2); V(s2); 组装为一台车;
> }while(1)
> ---|---|---
> 详细描述还应考虑对箱子单元的描述以及访问互斥问题。建议车架放在箱子的一端，车 轮放在箱子的另一端，车架与车轮都采用后进先出的管理方式。
> ----------The P,V code Using Pascal----------
> Semaphore s1=N-2，s2=N-1，mutex=1; int in1=0，in2=N-1; int buf[N]; procedure 工人1:
> do{ 加工一个车架; P(s1); P(empty); P(mutex); Buf[in1] = 车架; in1 = in1 + 1;
> V(mutex); V(frame); }while(1) | 工人2活动: do{ 加工一个车轮; P(s2); P(empty);
> P(mutex); Buf[in2] = 车轮; in2 = in2 - 1; V(mutex); V(wheel); }while(1) |
> 工人1活动: do{ P(frame); P(mutex); Temp1 = Buf[in1-1]; in1 = in1 - 1; V(mutex);
> V(empty); V(s1); P(wheel); P(wheel); P(mutex); Temp2 = Buf[in2+1]; in2 = in2 +
> 1; Temp3 = Buf[in2 + 1]; in2 = in2 + 1; V(mutex); V(empty); V(empty); V(s2);
> V(s2); 组装为一台车; }while(1)
> ---|---|---
> 4
> 知错能改
> 进程p0,p1共享变量flag,turn;他们进入临界区的算法如下:

```
var flag:array[0..1] of boolean;//初值为false
   turn:01
   process i (0或1)
       while true
       do begin
           flag[i] =true;
           while turn!=i
           do begin
               while flag[j]==false
               do skip;//skip为空语句
               turn = i
           end
           临界区;
           flag[i] = false;
           出临界区;
       end
```

> 该算法能否正确地实现互斥?若不能,应该如何修改(假设flag,turn单元内容的修改和访问 是互斥的).

- [x]

知识点:信号量
出处:网络
难度:1

> 不能正确实现互斥.考虑如下情况:process0先执行到flag[0]
> =true,process1开始执行,进入内循环时,将turn设置为1;此时进程调度转到process0,
> process0可以进入内循环,由于flag[1]的值为true,所以process0再次将turn的值设置为0,重复上述操作,两个进程谁也不能进入临界区.

```
var flag:array[0..1] of boolean;//初值为false
   turn:0 1
cobegin
   process 0
       while true
       do begin
       flag[0] =true;
       turn = 1
```

```
        while flag[1]==true and turn = 1
        do skip;//skip为空语句
        临界区;
        flag[0] = false;
        出临界区;
        end
    process 1
        while true
        do begin
        flag[1] =true;
        turn = 0
        while flag[0]==true and turn = 0
        第四章 福尔摩斯探案之网络搜捕 73
        do skip;//skip为空语句
        临界区;
        flag[1] = false;
        出临界区;
        end
coend
```

容易证明这种方法保证了互斥,对于进程0,一旦它设置flag[0]为true,进程1就不能进入其临界段.若进程1已经在其临界段中，那么flag[1]=tru
e并且进程0被阻塞进入临界段.另一方面,防止了相互阻塞,假设进程0阻塞于while循环,这意味着flag[1]为true,而且turn=1,当flag[1]为false或turn为0时,进程0就可进入自己的临界段了.
4
在虚拟存储系统中，当由虚拟地址找不到对应的物理地址时，会产生缺页故障。请完成如下任务。 1）描述缺页故障（page_fault）的处理流程；
2）补全下面缺页处理中所缺代码。

kern/trap/trap.c

```
    ------------------------------
    ...
    static int
    pgfault_handler(struct trapframe tf) {
        extern struct mm_struct check_mm_struct;
        if(check_mm_struct !=NULL) { //used for test check_swap
                print_pgfault(tf);
            }
        struct mm_struct mm;
        if (check_mm_struct != NULL) {
            assert(current == idleproc);
            mm = check_mm_struct;
        }
        else {
            if (current == NULL) {
                print_trapframe(tf);
                print_pgfault(tf);
                panic("unhandled page fault.
");
            }
            mm = current->mm;
        }
        return do_pgfault(mm, tf->tf_err, rcr2());
              ____(1)____
    }
    ...
    static void
    trap_dispatch(struct trapframe tf) {
        char c;
        int ret=0;
        switch (tf->tf_trapno) {
        case T_DEBUG:
        case T_BRKPT:
            debug_monitor(tf);
            break;
        case T_PGFLT:
            if ((ret = pgfault_handler(tf)) != 0) {
             ___(2)___
                print_trapframe(tf);
                if (current == NULL) {
                    panic("handle pgfault failed. ret=%d
", ret);
                }
                else {
                    if (trap_in_kernel(tf)) {
                        panic("handle pgfault failed in kernel mode. ret=%d
", ret);
                    }
```

```
                    cprintf("killed by kernel.
");
                    panic("handle user mode pgfault failed. ret=%d
", ret);
                    do_exit(-E_KILLED);
                }
            }
            break;
        case T_SYSCALL:
            syscall();
            break;
        case IRQ_OFFSET + IRQ_TIMER:
    //    LAB3 : If some page replacement algorithm need tick to change the priority of pages,
    //    then you can add code here.
            ticks ++;
            assert(current != NULL);
            run_timer_list();
            break;
        case IRQ_OFFSET + IRQ_COM1:
        case IRQ_OFFSET + IRQ_KBD:
            if ((c = cons_getc()) == 13) {
                debug_monitor(tf);
            }
            else {
                cprintf("%s [%03d] %c
",
                        (tf->tf_trapno != IRQ_OFFSET + IRQ_KBD) ? "serial" : "kbd", c, c);
            }
            break;
        case IRQ_OFFSET + IRQ_IDE1:
        case IRQ_OFFSET + IRQ_IDE2:
            / do nothing /
            break;
        default:
            print_trapframe(tf);
            if (current != NULL) {
                cprintf("unhandled trap.
");
                do_exit(-E_KILLED);
            }
            // in kernel, it must be a mistake
            panic("unexpected trap in kernel.
");
        }
    }
    kern/mm/vmm.c
    ------------------------------
    // do_pgfault - interrupt handler to process the page fault execption
    int
    do_pgfault(struct mm_struct mm, uint32_t error_code, uintptr_t addr) {
        int ret = -E_INVAL;
        struct vma_struct vma = find_vma(mm, addr);
        pgfault_num++;
        if (vma == NULL ) {
            cprintf("not valid addr %x, and  can not find it vma %x
", addr, vma);
            goto failed;
        }
        else if (vma->vm_start > addr) {
            cprintf("not valid addr %x, and  can not find it vma range[%x, %x]
", addr, vma->vm_start, vma->vm_end);
            goto failed;
        }
        cprintf("valid addr %x, and find it in vma range[%x, %x]
", addr, vma->vm_start, vma->vm_end);
        switch (error_code & 3) {
        default:
                / default is 3: write, present /
        case 2: / write, not present /
            if (!(vma->vm_flags & VM_WRITE)) {
                cprintf("write, not present in do_pgfault failed
");
                goto failed;
            }
```

```
            break;
        case 1: / read, present /
            cprintf("read, present in do_pgfault failed
");
            goto failed;
        case 0: / read, not present /
            if (!(vma->vm_flags & (VM_READ | VM_EXEC))) {
                cprintf("read, not present in do_pgfault failed
");
                goto failed;
            }
        }
        uint32_t perm = PTE_U;
        if (vma->vm_flags & VM_WRITE) {
            perm |= PTE_W;
        }
        addr = ROUNDDOWN(addr, PGSIZE);
        ret = -E_NO_MEM;
        pte_t ptep;
        // try to find a pte, if pte's PT(Page Table) isn't existed, then create a PT.
        // (notice the 3th parameter '1')
        if ((ptep = get_pte(mm->pgdir, addr, 1)) == NULL) {
            cprintf("get_pte in do_pgfault failed
");
            goto failed;
        }
        if (ptep == 0) { // if the phy addr isn't exist, then alloc a page & map the phy addr with logical
addr
            if (pgdir_alloc_page(mm->pgdir, addr, perm) == NULL) {
             ___(3)___
                cprintf("pgdir_alloc_page in do_pgfault failed
");
                goto failed;
            }
        }
        else {
            struct Page page=NULL;
            cprintf("do pgfault: ptep %x, pte %x
",ptep, ptep);
            if (ptep & PTE_P) {
                page = pte2page(ptep);
             ___(4)___
            } else{
               // if this pte is a swap entry, then load data from disk to a page with phy addr
               // and call page_insert to map the phy addr with logical addr
               if(swap_init_ok) {
                   if ((ret = swap_in(mm, addr, &page;)) != 0) {
             ___(5)___
                       cprintf("swap_in in do_pgfault failed
");
                       goto failed;
                   }
               }
               else {
                cprintf("no swap_init_ok but ptep is %x, failed
",ptep);
                goto failed;
               }
           }
           page_insert(mm->pgdir, page, addr, perm);
           swap_map_swappable(mm, addr, page, 1);
       }
       ret = 0;
    failed:
        return ret;
    }
    kern/mm/swap.c
    -----------------------------
    ...
    int
    swap_out(struct mm_struct mm, int n, int in_tick)
    {
         int i;
         for (i = 0; i != n; ++ i)
```

```
        {
            uintptr_t v;
            //struct Page ptr_page=NULL;
            struct Page page;
            // cprintf("i %d, SWAP: call swap_out_victim
",i);
            int r = sm->swap_out_victim(mm, &page;, in_tick);
            if (r != 0) {
                    cprintf("i %d, swap_out: call swap_out_victim failed
",i);
                    break;
            }
            //assert(!PageReserved(page));
            //cprintf("SWAP: choose victim page 0x%08x
", page);
            v=page->pra_vaddr;
            pte_t ptep = get_pte(mm->pgdir, v, 0);
            assert((ptep & PTE_P) != 0);
            if (swapfs_write( (page->pra_vaddr/PGSIZE+1)<<8, page) != 0) {
            ___(6)___
                    cprintf("SWAP: failed to save
");
                    sm->map_swappable(mm, v, page, 0);
                    continue;
            }
            else {
                    cprintf("swap_out: i %d, store page in vaddr 0x%x to disk swap entry %d
", i, v, page->pra_vaddr/PGSIZE+1);
                    ptep = (page->pra_vaddr/PGSIZE+1)<<8;
                    free_page(page);
            }
            tlb_invalidate(mm->pgdir, v);
        }
        return i;
    }
    int
    swap_in(struct mm_struct mm, uintptr_t addr, struct Page ptr_result)
    {
        struct Page result = alloc_page();
        assert(result!=NULL);
        pte_t ptep = get_pte(mm->pgdir, addr, 0);
        // cprintf("SWAP: load ptep %x swap entry %d to vaddr 0x%08x, page %x, No %d
", ptep, (ptep)>>8, addr, result, (result-pages));
        int r;
        if ((r = swapfs_read((ptep), result)) != 0)
            ___(7)___
        {
           assert(r!=0);
        }
        cprintf("swap_in: load disk swap entry %d with swap_page in vadr 0x%x free_area.nr_free %d
", (ptep)>>8, addr, free_area.nr_free);
        ptr_result=result;
        return 0;
    }
    kern/mm/pmm.h
    ------------------------------
    ...
    #define alloc_page() alloc_pages(1)
    #define free_page(page) free_pages(page, 1)
    kern/mm/pmm.c
    ------------------------------
    ...
    // pgdir_alloc_page - call alloc_page & page_insert functions to
    //                  - allocate a page size memory & setup an addr map
    //                  - pa<->la with linear address la and the PDT pgdir
    struct Page 
    pgdir_alloc_page(pde_t pgdir, uintptr_t la, uint32_t perm) {
        struct Page page = alloc_page();
        if (page != NULL) {
            if (page_insert(pgdir, page, la, perm) != 0) {
                free_page(page);
                return NULL;
            }
            if (swap_init_ok){
```

```
                if(check_mm_struct!=NULL) {
                    swap_map_swappable(check_mm_struct, la, page, 0);
                    page->pra_vaddr=la;
                    assert(page_ref(page) == 1);
                    //cprintf("get No. %d  page: pra_vaddr %x, pra_link.prev %x, pra_link_next %x in
pgdir_alloc_page
", (page-pages), page->pra_vaddr,page->pra_page_link.prev, page->pra_page_link.next);
                }
                else  {  //now current is existed, should fix it in the future
                    //swap_map_swappable(current->mm, la, page, 0);
                    //page->pra_vaddr=la;
                    //assert(page_ref(page) == 1);
                    //panic("pgdir_alloc_page: no pages. now current is existed, should fix it in the future
");
                }
            }
        }
        return page;
    }
    kern/fs/swapfs.c
    -----------------------------
    ...
    int
    swapfs_read(swap_entry_t entry, struct Page page) {
        return ide_read_secs(SWAP_DEV_NO, swap_offset(entry)  PAGE_NSECT, page2kva(page), PAGE_NSECT);
    }
    int
    swapfs_write(swap_entry_t entry, struct Page page) {
        return ide_write_secs(SWAP_DEV_NO, swap_offset(entry)  PAGE_NSECT, page2kva(page), PAGE_NSECT);
    }
    kern/mm/swap_fifo.c
    -----------------------------
    ...
    struct swap_manager swap_manager_fifo =
    {
         .name            = "fifo swap manager",
         .init            = &_fifo_init,
         .init_mm         = &_fifo_init_mm,
         .tick_event      = &_fifo_tick_event,
         .map_swappable   = &_fifo_map_swappable,
         .set_unswappable = &_fifo_set_unswappable,
         .swap_out_victim = &_fifo_swap_out_victim,
         .check_swap      = &_fifo_check_swap,
    };
```

- [x]

知识点:缺页中断
出处:网络
难度:1

> 1）缺页故障的处理流程：每点2分，共8分； 中断、权限检查、分配空闲页面并加载、没有空闲页面时的转换 2）7个填空，每个1分，变量不对扣0.5分，共7分；
> 4
> 某计算机系统中有M个同类型共享资源，有N个进程竞争使用，每个进程最多需要K个共享资源。该系统不会发生死锁的K的最大值是多少？要求给出计算过程，并说明理由。

- [x]

知识点:死锁
出处:网络
难度:1

> K<=M/N+1的取整； 说明：出现死锁时占用最多资源的情况是(K-1)N
> 4
> 请求分页管理系统中，假设某进程的页表内容如下表所示。
> 页号| 页框号| 有效位

（存在位）	0	101H	1
1	--	0	
2	254H	1	

页面大小为4KB，一次内存的访问时间是100ns，一次快表（TLB）的访问时间是10ns，处理一次缺页的平均时间为108ns（已含更新TLB和页表的时间），
进程的驻留集大小固定为2，采用最近最少使用置换算法（LRU）和局部淘汰策略。假设
①TLB初始为空；
②地址转换时先访问TLB，若TLB未命中，再访问页表（忽略访问页表之后的TLB更新时间）；

③有效位为0表示页面不在内存，产生缺页中断，缺页中断处理后，返回到产生缺页中断的指令处重新执行。设有虚地址访问序列2362H、1565H、25A5H
请问：
1) 依次访问上述三个虚地址，各需多少时间？给出计算过程。
2) 基于上述访问序列，虚地址1565H的物理地址是多少？请说明理由。

- [x]

知识点:置换算法
出处:网络
难度:1

> （1）根据页式管理的工作原理，应先考虑页面大小，以便将页号和页内位移分解出来。页面大小为4KB，即212，则得到页内位移占虚地址的低12位，页号占剩余高位。
> 可得三个虚地址的页号P如下（十六进制的一位数字转换成4位二进制，因此，十六进制的低三位正好为页内位移，最高位为页号）：
> 2362H：P=2，访问快表10ns
> ，因初始为空，访问页表100ns得到页框号，合成物理地址后访问主存100ns，共计10ns+100ns+100ns=210ns。 1565H：P=1，访问快
> 表10ns，落空，访问页表100ns落空，进行缺页中断处理108ns，合成物理地址后访问主存100ns，共计
> 10ns+100ns+108ns+100ns≈1
> 08ns。
> 25A5H：P=2，访问快表，因第一次访问已将该页号放入快表，因此花费10ns便可合成物理地址，访问主存100ns，共计
> 10ns+100ns=110ns。
> （2）当访问虚地址1565H时，产生缺页中断，合法驻留集为2，必须从页表中淘汰一个页面，根据题目的置换算法，应淘汰0号页面，因此1565H的对应页框号为10
> 1H。由此可得1565H的物理地址为101565H。
> 4
> 三个进程P1、P2、P3互斥使用一个包含N（N>0）个单元的缓冲区。
> P1每次用produce()生成一个正整数并用put)送入缓冲区某一空单元中；
> P2每次用getodd()从该缓冲区中取出一个奇数并用countodd()统计奇数个数；
> P3每次用geteven()从该缓冲区中取出一个偶数并用counteven()统计偶数个数。
> 请用信号量机制实现这三个进程的同步与互斥活动，并说明所定义的信号量的含义。要求用伪代码描述。

- [x]

知识点:
出处:网络
难度:1

> 定义信号量S1控制P1与P2之间的同步；S2控制P1与P3之间的同步；empty控制生产者与消费者之间的同步；mutex控制进程间互斥使用缓冲区。程序如下：
> Var s1=0,s2=0,empty=N,mutex=1; Parbegin P1:begin X=produce(); P(empty);
> P(mutex); Put(); If x%2==0 V(s2); else V(s1); V(mutex); end. P2:begin P(s1);
> P(mutex); Getodd(); Countodd():=countodd()+1; V(mutex); V(empty); end.
> P3:begin P(s2) P(mutex); Geteven(); Counteven():=counteven()+1; V(mutex);
> V(empty); end. Parend.
> 4
> 设某计算机的逻辑地址空间和物理地址空间均为64KB，按字节编址.某进程最多需要6页数据存储空间，页的大小为1KB，操作系统采用固定分配局部置换策略为此进程分
> 配4个页框.

页号	页框号	装入时间	访问位	0	7	130	1
1	4	230	1				
2	2	200	1				
3	9	160	1				

当该进程执行到时刻260时，要访问逻辑地址为17CAH的数据.请回答下列问题:
(1)该逻辑地址对应的页号时多少?
(2)若采用先进先出(FIFO)置换算法，该逻辑地址对应的物理地址?要求给出计算过程.
(3)采用时钟(Clock)置换算法，该逻辑地址对应的物理地址是多少?要求给出计算过程.
(设搜索下一页的指针按顺时针方向移动，且指向当前2号页框，示意图如下) TODO

- [x]

知识点:置换算法
出处:网络
难度:1

> （1）因为 17CAH =0001 0111 1100 1010 B ，表示的页号的位为左边 6 位，即 00101B，所以页号为 5.
> （2）根据 FIFO 算法，需要替换装入时间最早的页，故需要置换装入时间最早的 0 号页，即将 5 号页装入到 7 号页框中，所以对应的物理地址为 0001
> 1111 1100 1010 B = 1FCAH.
> （3）根据 CLOCK 算法，如果当前指针所指页框的使用位为 0 时，则替换该页；否则将使用位清
> 0，并将指针指向下一个页框，继续查找。根据题设和示意图，将从 2 号页框开始查找，前 4 次查找页框号的顺序为 2->4->7->9，并将对应页框使用位清
> 0.在第 5 次查找中，指针指向 2 号页框，这时 2 号页框的使用位为 0，故置换 2 号页框对应的 2 号页，将 5 号页转入 2
> 号页框中，并将对应使用位设置为 1，所以对应的物理地址为 0000 1011 1100 1010 B = 0BCAH。
> 【分析】 45、46 题的几个知识点：空闲外存储空间的管理方法（考题中位图表 bitmap），磁盘调度算法（考题中的
> CSCAN），磁盘的结构（考题中的平均旋转延时的计算），逻辑地址到物理地址的映射（考题中的物理地址计算），页面置换算法（考题中的 clock

> 算法）。这些知识点都包含到操作系统原理的本科教学大纲中，要求考生必须掌握的。但是在实际考试中，考生这两题的得分少，且得 0
> 分考生多。在学习操作系统原理时，考生需要充分理解和掌握操作系统的概念、原理和算法，并且能够灵活应用。
> 3
> 多道程序技术可将一台物理CPU虚拟为多台逻辑CPU。

- (x) A.对
- ( ) B.错

知识点:操作系统概述
出处:网络
难度:1

> A
> 1
> 本地用户通过键盘登陆系统是，首先获得键盘输入信息的程序时

- ( ) A.命令解释程序
- ( ) B.中断处理程序
- ( ) C.系统调用程序
- (x) D.用户登录程序

知识点:操作系统概述
出处:网络
难度:1

> D
> 1
> (20150309*操作系统试题*doc)操作系统中采用多道程序设计技术提高CPU和外设的【 】。

- ( ) A.可靠性
- ( ) B.兼容性
- (x) C.利用率
- ( ) D.稳定性

知识点:操作系统概述
出处:网络
难度:1

> C
> 1
> (20150309*操作系统试题*doc)若把操作系统看作系统资源的管理者，下列的【 】不属于操作系统所管理的资源。

- ( ) A.程序
- ( ) B.CPU
- (x) C.中断
- ( ) D.内存

知识点:操作系统概述
出处:网络
难度:1

> D 解析：可以从资源管理的角度来描述操作系统。资源管理之一是操作系统的主要作用。资源主要是指计算机系统为了进行数值计算和数据处理所需要的各种物质基础，通常分硬件资源和软件资源。就本题来讲，CPU和内存储器属于硬件资源，程序属于软件资源，所以只有中断不属于硬件资源，也不属于软件资源，当然也不在操作系统管理的资源范围之内。
> 5
> (20150309*操作系统试题*doc)操作系统的五大主要功能：__、_、__、文件管理以及用户接口管理。

- [x]

知识点:操作系统概述
出处:网络
难度:1

> 处理器管理、 存储器管理、 设备管理
> 5
> (20150309*操作系统试题*doc)从系统的角度看，作业由程序、_和_组成。

- [x]

知识点:操作系统概述
出处:网络
难度:1

> 数据和作业说明书
> 5
> (20150309*操作系统试题*doc)现代操作系统的特点是程序的_执行、系统所拥有的资源被_和系统的用户_的使用。

- [x]

知识点:操作系统概述
出处:网络
难度:1

> 并发 共享 随机
> 3
> (20150309*操作系统试题*doc)在计算机系统运行过程中，系统开销越大，系统运行效率越高。【 】

- ( ) A.对
- (x) B.错

知识点:操作系统概述
出处:网络
难度:1

> B
> 3
> (20150309*操作系统试题*doc)实时系统的主要特点式提供即时响应和高可靠性。【 】

- (x) A.对
- ( ) B.错

知识点:操作系统概述
出处:网络
难度:1

> A
> 3
> (20150309*操作系统试题*doc)最高响应比优先法是FCFS方式和SJF方式的一种综合平衡。【 】

- (x) A.对
- ( ) B.错

知识点:操作系统概述
出处:网络
难度:1

> A
> 4
> (20150309*操作系统试题*doc)名词解释：抖动

- [x]

知识点:缺页中断
出处:网络
难度:1

> 4
> (20150309*操作系统试题*doc)名词解释：中断

- [x]

知识点:中断、异常与系统调用
出处:网络
难度:1

> 4
> (20150309*操作系统试题*doc)DMA方式与中断方式相比，其主要优点是什么?

- [x]

知识点:I/O子系统
出处:网络
难度:1

> 4
> (20150309*操作系统试题*doc)进程调度的功能有那些?

- [x]

知识点:进程状态与控制
出处:网络
难度:1

> 4
> (20150309*操作系统试题*doc)什么是页式管理？静态页式管理可以实现虚存吗?

- [x]

知识点:非连续内存分配
出处:网络
难度:1

> 4
> (20150309*操作系统试题*doc)假设有四个作业的单道系统，它们的提交、运行时间如下表所示（时间单位：小时，以十进制进行计算）。若采用响应比高者优先
> 调度的非抢占式调度算法，试回答：
> （1）作业应以怎样的顺序调度？给出分析过程。
> （2）计算平均周转时间。
> |作业号|到达时间|运行时间|
> | A | 8.0 | 2 |
> | B | 8.5 | 0.5 |
> | C | 9.0 | 1.0 |
> | D | 9.5 | 0.2 |

- [x]

知识点:处理机调度
出处:网络
难度:1

> 4
> (20150309*操作系统试题*doc)假如某银行的营业大厅内只有10个座位，唯一的管理员发现有空位时按流水号通知门外等候的顾客进入大厅，顾客没有被通知时
> 不得擅自进入大厅。如果把管理员看作进程Supervisor，把顾客看作进程Client，用P、V原语描述管理员和顾客之间合作关系，回答以下问题：
> （1）设置信号量empty，用于查看是否有空座位，其初值应为多少？设置信号量enter，用于通知门外等候的顾客，其初值应为多少？
> （2）根据定义的信号量，将适当的P、V原语填入以下程序使并发进程能够正确执行。
> `COBEGIN`
> `Supervisor：Repeat`
> `Begin`
> `查看座位情况；`
> `① ；`
> `走出大厅；`
> `通知顾客；`
> `② ；`
> `返回大厅；`
> `End`
> `Until False；`
> `Client：Repeat`
> `Begin`
> `③ ；`
> `进入大厅；`
> `办理业务；`
> `④ ；`
> `离开大厅；`
> `End`
> `Until False；`
> `COEND`

- [x]

知识点:信号量
出处:网络
难度:1

> 4
> (20150309*操作系统试题*doc)设进程A（30K）、B（70K）和C（50K）依次请求内存分配，内存采用可变分区管理。现有两个空闲分区F1（150
> K）和F2（90K），如下图所示。若采用最佳适应算法，画出内存分配情况示意图。
> | 已用 |
> |F1(150K)|
> | 已用 |
> |F2(90K)|
> | 已用 |

- [x]

知识点:连续内存分配
出处:网络
难度:1

> 4
> (20150309*操作系统试题*doc)在一个分页存储管理系统中，已知页面大小L=1024KB。现有一进程，其页表如下：
> |页号|块号|
> | 0 | 5 |
> | 1 | 7 |
> | 2 | 3 |
> | 3 | 2 |
> | 4 | 8 |
> 计算与给定逻辑地址LA对应的物理地址PA，给出计算过程：
> （1）逻辑地址LA=2548；
> （2）逻辑地址LA=6000。

- [x]

知识点:缺页中断
出处:网络
难度:1

> 4
> (20150309*操作系统试题*doc)在一个请求分页存储管理系统中，某进程的页面走向为4、3、2、1、4、3、5、4、3、2，当分配给该进程的物理块数为
> 3 时（假设开始执行时内存中没有页面），试回答以下问题：
> （1）计算采用先进先出FIFO置换算法的缺页率；
> （2）FIFO置换算法选择什么样的页面淘汰？

- [x]

知识点:置换算法
出处:网络
难度:1

> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)操作系统是计算机系统的一种_____。

- ( ) A.应用软件
- (x) B.系统软件
- ( ) C.通用软件
- ( ) D.工具软件

知识点:操作系统概述
出处:网络
难度:1

> B
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)操作系统是一种系统软件，它_____。

- ( ) A.控制程序的执行
- ( ) B.管理计算机系统的资源
- ( ) C.方便用户使用计算机
- (x) D.管理计算机系统的资源和控制程序的执行

知识点:操作系统概述
出处:网络
难度:1

> D
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)下列选择中，_____不是操作系统关心的主要问题，

- ( ) A.管理计算机裸机
- ( ) B.设计、提供用户程序与计算机硬件系统的界面
- ( ) C.管理计算机系统资源
- (x) D.高级程序设计语言的编译器

知识点:操作系统概述
出处:网络
难度:1

> D
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)操作系统的主要功能是管理计算机系统中的资源，其中包括_____管理和存储器管理，以及设备管理
> 和文件管理。这里的_____管理主要是对进程进行管理。

- ( ) A.存储器
- ( ) B.虚拟存储器
- ( ) C.运算器
- (x) D.处理机
- ( ) E.控制器

知识点:操作系统概述
出处:网络
难度:1

> D
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)实现不同的作业处理方式（如：批处理、分时处理、实时处理等），主要是基于操作系统对
> 管理采用了不同的策略。

- (x) A.处理机
- ( ) B.存储器
- ( ) C.设备
- ( ) D.文件

知识点:操作系统概述
出处:网络
难度:1

> A
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)在操作系统中采用多道程序设计方式能提高CPU和外部设备的_____。

- (x) A.利用效率
- ( ) B.可靠性
- ( ) C.稳定性
- ( ) D.兼容性

知识点:操作系统概述
出处:网络
难度:1

> A
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)为了实现多道程序设计，计算机需要有_____。

- (x) A.更大的内存
- ( ) B.更快的外部设备
- ( ) C.更快的CPU
- ( ) D.更先进的终端

知识点:操作系统概述
出处:网络
难度:1

> A
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)多道程序设计系统中，让多个计算问题同时装入计算机系统的主存储器_____。

- (x) A.并发执行
- ( ) B.顺序执行
- ( ) C.并行执行
- ( ) D.同时执行

知识点:操作系统概述
出处:网络
难度:1

> A
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)从总体上说，多道程序设计技术可_____单位时间的算题量。

- (x) A.增加
- ( ) B.减少
- ( ) C.维持

知识点:操作系统概述
出处:网络
难度:1

> A
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)为了提高计算机的处理机和外部设备的利用率，把多个程序同时放入主存储器，在宏观上并行运行是_____。

- ( ) A.分时操作系统
- ( ) B.实时操作系统
- ( ) C.批处理系统
- (x) D.多道程序设计
- ( ) E.并发程序设计

知识点:操作系统概述
出处:网络
难度:1

> D
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)有一类操作系统的系统响应时间的重要性超过系统资源的利用率，它被广泛地应用于卫星控制、导弹发
> 射、飞机飞行控制、飞机订票业务等领域是_____。

- ( ) A.分时操作系统
- (x) B.实时操作系统
- ( ) C.批处理系统
- ( ) D.网络操作系统

知识点:操作系统概述
出处:网络
难度:1

> B
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)操作系统有多种类型：允许多用户将若干个作业提交给计算机系统集中处理的操作系统称为_____。

- (x) A.批处理操作系统
- ( ) B.分时操作系统
- ( ) C.实时操作系统
- ( ) D.网络操作系统

知识点:操作系统概述
出处:网络
难度:1

> A
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)操作系统有多种类型：允许多个用户以交互方式使用计算机的操作系统，称为_____。

- ( ) A.批处理操作系统
- (x) B.分时操作系统
- ( ) C.实时操作系统
- ( ) D.网络操作系统

知识点:操作系统概述
出处:网络
难度:1

> B
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)UNIX操作系统是著名的_____。

- ( ) A.多道批处理系统
- (x) B.分时系统
- ( ) C.实时系统
- ( ) D.分布式系统

知识点:操作系统概述
出处:网络
难度:1

> B
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)在设计批处理系统时，首先要考虑的是_____。

- ( ) A.灵活性和可适应性
- ( ) B.交互性和响应时间
- (x) C.周转时间和系统吞吐量
- ( ) D.实时性和可靠性

知识点:操作系统概述
出处:网络
难度:1

> C
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)在设计分时操作系统时，首先要考虑的是_____。

- ( ) A.灵活性和可适应性
- (x) B.交互性和响应时间
- ( ) C.周转时间和系统吞吐量
- ( ) D.实时性和可靠性

知识点:操作系统概述
出处:网络
难度:1

> B
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)在设计实时操作系统时，首先要考虑的是_____。

- ( ) A.灵活性和可适应性
- ( ) B.交互性和响应时间
- ( ) C.周转时间和系统吞吐量
- (x) D.实时性和可靠性

知识点:操作系统概述
出处:网络
难度:1

> D
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)分时操作系统的主要特征之一是提高_____。

- ( ) A.计算机系统的可靠性
- (x) B.计算机系统的交互性
- ( ) C.计算机系统的实时性
- ( ) D.计算机系统的安全性

知识点:操作系统概述
出处:网络
难度:1

> B
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)批处理系统的主要缺点是_____。

- ( ) A.CPU的利用率不高
- (x) B.失去了交互性
- ( ) C.不具备并行性
- ( ) D.以上都不是

知识点:操作系统概述
出处:网络
难度:1

> B
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)分时系统中，为使多个用户能够同时与系统交互，最关键的问题是_____。

- ( ) A.计算机具有足够的运行速度
- ( ) B.内存容量应足够大
- ( ) C.系统能及时地接收多个用户输入
- (x) D.能在一短的时间内，使所有用户程序都能运行
- ( ) E.能快速进行内外存对换

知识点:操作系统概述
出处:网络
难度:1

> D
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)实时操作系统对可靠性和安全性要求极高，它_____。

- ( ) A.十分注重系统资源的利用率
- ( ) B.不强调响应速度
- (x) C.不强求系统资源的利用率
- ( ) D.不必向用户反馈信息

知识点:操作系统概述
出处:网络
难度:1

> C
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)分时系统的响应时间（及时性）主要是根据_____确定的。

- ( ) A.时间片大小
- ( ) B.用户数目
- ( ) C.计算机运行速度
- (x) D.用户所能接受的等待时间
- ( ) E.控制对象所能接受的时延

知识点:操作系统概述
出处:网络
难度:1

> D
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)实时系统的响应时间则是由_____确定的。

- ( ) A.时间片大小
- ( ) B.用户数目
- ( ) C.计算机运行速度
- ( ) D.用户所能接受的等待时间
- (x) E.控制对象所能接受的时延

知识点:操作系统概述
出处:网络
难度:1

> E
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)分时系统中，当用户数目为100时，为保证响应不超过2秒；此时的时间片最大应为_____。

- ( ) A.10ms
- (x) B.20ms
- ( ) C.50ms
- ( ) D.100ms
- ( ) E.200ms

知识点:操作系统概述
出处:网络
难度:1

> B
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)假设就绪队列中有10个进程，系统将时间片设为200ms，CPU进行进程切换要花费10ms。
> 则系统开销所占的比率约为_____

- ( ) A.0.5%
- ( ) B.1%
- (x) C.5%
- ( ) D.10%

知识点:操作系统概述
出处:网络
难度:1

> C
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)在操作系统中进程是一个具有一定独立功能程序在某个数据集合上的一次_____。

- ( ) A.并发活动
- (x) B.运行活动
- ( ) C.单独操作
- ( ) D.关联操作

知识点:进程状态与控制
出处:网络
难度:1

> B
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)在操作系统中，进程是一个_____概念，而程序是一个静态的概念。

- ( ) A.组合态
- ( ) B.关联态
- ( ) C.运行态
- ( ) D.等待态
- ( ) E.静态
- (x) F.动态

知识点:进程状态与控制
出处:网络
难度:1

> F
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)操作系统中，进程与程序的重要区别之一是_____。

- ( ) A.程序有状态而进程没有
- (x) B.进程有状态而程序没有
- ( ) C.程序可占有资源而进程不可
- ( ) D.进程能占有资源而程序不能

知识点:进程状态与控制
出处:网络
难度:1

> B
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)从静态角度看，进程由程序、数据和_____三部分组成。

- ( ) A.JCB
- ( ) B.DCB
- (x) C.PCB
- ( ) D.PMT

知识点:进程状态与控制
出处:网络
难度:1

> C
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)在一单处理机系统中，若有5个用户进程，在非管态的某一时刻，处于就绪态的用户进程最多有_____个。

- ( ) A.1
- ( ) B.2
- ( ) C.3
- (x) D.4
- ( ) E.5
- ( ) F.0

知识点:进程状态与控制
出处:网络
难度:1

> D
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)在一单处理机系统中，若有5个用户进程，在非管态的某一时刻，处于阻塞态的用户进程最多有_____个。

- ( ) A.1
- ( ) B.2
- ( ) C.3
- ( ) D.4

- (x) E.5
- ( ) F.0

知识点:进程状态与控制
出处:网络
难度:1

> E
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)用户可通过系统调用建立和撤消进程，通常用户进程被建立后，_____。

- ( ) A.便一直存在于系统中，直到被操作人员撤消
- (x) B.随着作业运行正常或不正常结束而撤消
- ( ) C.随着时间片轮转而撤消与建立
- ( ) D.随着进程的阻塞或唤醒而撤消与建立

知识点:进程状态与控制
出处:网络
难度:1

> B
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)一个进程释放一种资源将有可能导致一个或几个进程_____。

- ( ) A.由就绪变运行
- ( ) B.由运行变就绪
- ( ) C.由阻塞变运行
- (x) D.由阻塞变就绪

知识点:进程状态与控制
出处:网络
难度:1

> D
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)正在执行的进程由于其时间片完而被暂停执行，此时进程应从运行态变为_____状态。

- ( ) A.静止阻塞
- ( ) B.活动阻塞
- ( ) C.静止就绪
- (x) D.活动就绪
- ( ) E.执行

知识点:进程状态与控制
出处:网络
难度:1

> D
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)处于静止阻塞状态的进程，在进程等待的事件出现后，应转变为_____状态。

- ( ) A.静止阻塞
- ( ) B.活动阻塞
- (x) C.静止就绪
- ( ) D.活动就绪
- ( ) E.执行

知识点:进程状态与控制
出处:网络
难度:1

> C
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)若进程正处于运行态时，应终端的请求而暂停下来以便研究其运行情况(执行挂起进程原语)，这时进
> 程应转变为_____状态。

- ( ) A.静止阻塞
- ( ) B.活动阻塞
- (x) C.静止就绪
- ( ) D.活动就绪
- ( ) E.执行

知识点:进程状态与控制
出处:网络
难度:1

> C
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)执行解除挂起进程原语后，如挂起进程处于阻塞状态，则应转变为_____态。

- ( ) A.静止阻塞
- (x) B.活动阻塞
- ( ) C.静止就绪

- ( ) D.活动就绪
- ( ) E.执行

知识点:进程状态与控制
出处:网络
难度:1

> B
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)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。

- ( ) A.大于0
- (x) B.小于0
- ( ) C.大于等于0
- ( ) D.小于等于0

知识点:信号量
出处:网络
难度:1

> B
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)对于记录型信号量，在执行V操作(signal操作)时，信号量的值应当加1；当其值为_____
> 时，应唤醒阻塞队列中的进程。

- ( ) A.大于0
- ( ) B.小于0
- ( ) C.大于等于0
- (x) D.小于等于0

知识点:信号量
出处:网络
难度:1

> D
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)设两个进程共用一个临界段的互斥信号量mutex，当mutex=-1时表示：_____
> _____。

- (x) A.一个进程入了临界段，另一个进程等待
- ( ) B.没有一个进程进入了临界段
- ( ) C.只有一个进程进入了临界段
- ( ) D.两个进程都在等待

知识点:信号量
出处:网络
难度:1

> A
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)设两个进程共用一个临界段的互斥信号量mutex，当mutex=0时表示：_____
> ____。

- ( ) A.一个进程入了临界段，另一个进程等待
- ( ) B.没有一个进程进入了临界段
- (x) C.只有一个进程进入了临界段
- ( ) D.两个进程都在等待

知识点:信号量
出处:网络
难度:1

> C
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)计算机操作系统中有3个用户进程，若P、V操作的信号量S初值为2，当前值为-1，则表示当前有
> _____进程在等待。

- (x) A.1个
- ( ) B.2个
- ( ) C.3个
- ( ) D.0个

知识点:信号量
出处:网络
难度:1

> A
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)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_____
> ____。

- ( ) A.[-3,2]
- ( ) B.[-2,2]

- (x) C.[-1,2]
- ( ) D.[0,2]
- ( ) E.[-2,1]

知识点:信号量
出处:网络
难度:1

> C
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)如果有四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的取值范围是_____。

- ( ) A.4，3，2，1，0
- (x) B.3，2，1，0，-1
- ( ) C.2，1，0，-1，-2
- ( ) D.1，0，-1，-2，-3

知识点:信号量
出处:网络
难度:1

> B
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)进程从阻塞状态进入就绪状态可能是由于_____.

- ( ) A.现运行进程运行结束
- ( ) B.现运行进程执行了P操作
- (x) C.现运行进程执行了V操作
- ( ) D.现运行进程时间片用完

知识点:信号量
出处:网络
难度:1

> C
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)进程从运行态进入阻塞态可能是由于_____.

- ( ) A.现运行进程运行结束
- (x) B.现运行进程执行了P操作
- ( ) C.现运行进程执行了V操作
- ( ) D.现运行进程时间片用完

知识点:信号量
出处:网络
难度:1

> B
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)实现进程互斥时，用_____对应，对同一个信号量调用PV操作实现互斥。

- ( ) A.一个信号量与一个临界区
- ( ) B.一个信号量与一个相关临界区
- (x) C.一个信号量与一组相关临界区
- ( ) D.一个信号量与一个消息

知识点:信号量
出处:网络
难度:1

> C
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)实现进程同步时，每一个（类）消息与一个信号量对应，进程_____可把不同的消息发送出去。

- ( ) A.在同一信号量上调用P操作
- ( ) B.在不同信号量上调用P操作
- ( ) C.在同一信号量上调用V操作
- (x) D.在不同信号量上调用V操作

知识点:信号量
出处:网络
难度:1

> D
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)在直接通信时，用send(N,M)原语发送信件，其中N表示_____。

- ( ) A.发送信件的进程名
- (x) B.接收信件的进程名
- ( ) C.信箱名
- ( ) D.信件内容

知识点:进程间通信
出处:网络
难度:1

> B
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)操作系统的主要性能参数：_____指的是单位时间内系统处理的作业量。

- ( ) A.周转时间
- ( ) B.处理时间
- ( ) C.消逝时间
- ( ) D.利用率
- ( ) E.生产率
- (x) F.吞吐量

知识点:操作系统概述
出处:网络
难度:1

> F
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)操作系统的主要性能参数：_____指的是从作业或命令的输入到其结束的间隔时间，在分析性能时常
> 用其倒数。

- (x) A.周转时间
- ( ) B.处理时间
- ( ) C.消逝时间
- ( ) D.利用率
- ( ) E.生产率
- ( ) F.吞吐量

知识点:操作系统概述
出处:网络
难度:1

> A
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)操作系统主要性能参数:
> _____指的是在一个给定的时间内，系统的一个指定成份被使用的时间比例。

- ( ) A.周转时间
- ( ) B.处理时间
- ( ) C.消逝时间
- (x) D.利用率
- ( ) E.生产率
- ( ) F.吞吐量

知识点:操作系统概述
出处:网络
难度:1

> D
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)在所学的调度算法中，能兼顾作业等待时间和作业执行时间调度算法是_____。

- ( ) A.FCFS调度算法
- ( ) B.短作业优先调度算法
- ( ) C.时间片轮转法
- ( ) D.多级反馈队列调度算法
- (x) E.高响应比优先算法
- ( ) F.基于优先权的剥夺调度算法

知识点:处理机调度
出处:网络
难度:1

> E
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)在所学的调度算法中，最有利于提高资源的使用率、能使短作业、长作业及交互作业用户都比较满意的
> 调度算法是_____。

- ( ) A.FCFS调度算法
- ( ) B.短作业优先调度算法
- ( ) C.时间片轮转法
- (x) D.多级反馈队列调度算法
- ( ) E.高响应比优先算法
- ( ) F.基于优先权的剥夺调度算法

知识点:处理机调度
出处:网络
难度:1

> D
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)在所学的调度算法中，对所有进程和作业都是公平合理的调度算法是_____。

- (x) A.FCFS调度算法
- ( ) B.短作业优先调度算法
- ( ) C.时间片轮转法
- ( ) D.多级反馈队列调度算法
- ( ) E.高响应比优先算法
- ( ) F.基于优先权的剥夺调度算法

知识点:处理机调度
出处:网络
难度:1

> A
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)在所学的调度算法中，最有利于提高系统吞吐量的作业调度算法是_____。

- ( ) A.FCFS调度算法
- (x) B.短作业优先调度算法
- ( ) C.时间片轮转法
- ( ) D.多级反馈队列调度算法
- ( ) E.高响应比优先算法
- ( ) F.基于优先权的剥夺调度算法

知识点:处理机调度
出处:网络
难度:1

> B
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)在所学的调度算法中，为实现人机交互作用应采用调度算法是_____。

- ( ) A.FCFS调度算法
- ( ) B.短作业优先调度算法
- (x) C.时间片轮转法
- ( ) D.多级反馈队列调度算法
- ( ) E.高响应比优先算法
- ( ) F.基于优先权的剥夺调度算法

知识点:处理机调度
出处:网络
难度:1

> C
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)在所学的调度算法中，能对紧急作业进行及时处理的调度算法是_____。

- ( ) A.FCFS调度算法
- ( ) B.短作业优先调度算法
- ( ) C.时间片轮转法
- ( ) D.多级反馈队列调度算法
- ( ) E.高响应比优先算法
- (x) F.基于优先权的剥夺调度算法

知识点:处理机调度
出处:网络
难度:1

> F
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)在调度算法中，有二种调度算法是照顾短作业用户，其中_____调度算法中采用作业估计运行时间。

- ( ) A.FCFS调度算法
- (x) B.短作业优先调度算法
- ( ) C.时间片轮转法
- ( ) D.多级反馈队列调度算法
- ( ) E.高响应比优先算法
- ( ) F.基于优先权的剥夺调度算法

知识点:处理机调度
出处:网络
难度:1

> B
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)关于优先权大小的论述中，第_____条是正确的论述。

- ( ) A.计算型作业的优先权，应高于I/O型作业的优先权。
- ( ) B.用户进程的优先权，应高于系统进程的优先权。
- ( ) C.长作业的优先权，应高于短作业的优先权。
- ( ) D.资源要求多的作业，其优先权应高于资源要求少的作业。
- ( ) E.在动态优先权中，随着作业等待时间的增加，其优先权将随之下降。

- (x) F.在动态优先权中，随着进程执行时间的增加，其优先权降低。

知识点:处理机调度
出处:网络
难度:1

> F
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)在采用抢占式优先权进程调度算法的系统中，正在运行进程的优先权是_____。

- ( ) A.系统中优先权最高的进程
- (x) B.比就绪队列中进程优先权高的进程
- ( ) C.比就绪队列中进程优先权不一定高的进程

知识点:处理机调度
出处:网络
难度:1

> B
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)操作系统中,死锁"的概念是指_____。"

- ( ) A.程序死循环
- ( ) B.硬件发生故障
- (x) C.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
- ( ) D.系统停止运行

知识点:死锁
出处:网络
难度:1

> C
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)产生死锁的基本原因是系统资源不足和_____。

- (x) A.进程推进顺序非法
- ( ) B.进程调度不当
- ( ) C.系统中进程太多
- ( ) D.CPU运行太快

知识点:死锁
出处:网络
难度:1

> A
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)预防死锁的论述中，_____条是正确的论述。

- ( ) A.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。
- ( ) B.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。
- ( ) C.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) D.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。

知识点:死锁
出处:网络
难度:1

> D
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)对资源采用按序分配策略能达到_____的目的。

- (x) A.防止死锁
- ( ) B.避免死锁
- ( ) C.检测死锁
- ( ) D.解除死锁

知识点:死锁
出处:网络
难度:1

> A
> 3
> 多道程序的引入主要是为了提高资源利用率。

- (x) A.对
- ( ) B.错

知识点:操作系统概述
出处:网络
难度:1

> A
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)把逻辑地址转变为内存的物理地址的过程称作_____。

- ( ) A.编译
- ( ) B.连接

- ( ) C.运行
- (x) D.重定位

知识点:进程状态与控制
出处:网络
难度:1

> D
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)动态重定位是在作业的_____中进行的。

- ( ) A.编译过程
- ( ) B.装入过程
- ( ) C.修改过程
- (x) D.执行过程

知识点:进程状态与控制
出处:网络
难度:1

> D
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)在可变分区存储管理方案中需要一对界地址寄存器，其中_____作为重定位（地址映射）使用。

- ( ) A.逻辑地址寄存器
- ( ) B.长度寄存器
- ( ) C.物理地址寄存器
- (x) D.基址寄存器

知识点:非连续内存分配
出处:网络
难度:1

> D
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)分页系统中信息的逻辑地址到物理地址的变换是由_____决定。

- ( ) A.段表
- (x) B.页表
- ( ) C.物理结构
- ( ) D.重定位寄存器

知识点:非连续内存分配
出处:网络
难度:1

> B
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)分段系统中信息的逻辑地址到物理地址的变换是由_____决定。

- (x) A.段表
- ( ) B.页表
- ( ) C.物理结构
- ( ) D.重定位寄存器

知识点:非连续内存分配
出处:网络
难度:1

> A
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)在最佳适应算法中是按_____顺序形成空闲分区链。

- ( ) A.空闲区首址递增
- ( ) B.空闲区首址递减
- (x) C.空闲区大小递增
- ( ) D.空闲区大小递减

知识点:连续内存分配
出处:网络
难度:1

> C
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)在首次适应算法中，要求空闲分区按_____顺序链接成空闲分区链。

- (x) A.空闲区首址递增
- ( ) B.空闲区首址递减
- ( ) C.空闲区大小递增
- ( ) D.空闲区大小递减

知识点:连续内存分配
出处:网络
难度:1

> A
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)在可变分区式内存管理中，倾向于优先使用低址部分空闲区的算法是_____。

- ( ) A.最佳适应算法
- ( ) B.最坏适应算法
- (x) C.首次适应算法
- ( ) D.循环适应算法

知识点:连续内存分配
出处:网络
难度:1

> C
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)在可变分区式内存管理中，能使内存空间中空闲区分布较均匀的算法是_____。

- ( ) A.最佳适应算法
- ( ) B.最坏适应算法
- ( ) C.首次适应算法
- (x) D.循环适应算法

知识点:连续内存分配
出处:网络
难度:1

> D
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)在可变式分区分配方案中，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，造成空闲区表项数减1的情况是_____。

- ( ) A.无上邻（前邻、低址）空闲区，也无下邻（后邻、高址）空闲区
- ( ) B.有上邻（前邻、低址）空闲区，但无下邻（后邻、高址）空闲区
- ( ) C.有下邻（后邻、高址）空闲区，但无上邻（前邻、低址）空闲区
- (x) D.有上邻（前邻、低址）空闲区，也有下邻（后邻、高址）空闲区
- ( ) E.不可能的

知识点:连续内存分配
出处:网络
难度:1

> D
> 3
> 交互性是批处理系统的一个特征。

- ( ) A.对
- (x) B.错

知识点:操作系统概述
出处:网络
难度:1

> B
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)在可变式分区分配方案中，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，造成空闲区表项数不变、某项的始址改变、长度增加的情况是_____。

- ( ) A.无上邻（前邻、低址）空闲区，也无下邻（后邻、高址）空闲区
- ( ) B.有上邻（前邻、低址）空闲区，但无下邻（后邻、高址）空闲区
- (x) C.有下邻（后邻、高址）空闲区，但无上邻（前邻、低址）空闲区
- ( ) D.有上邻（前邻、低址）空闲区，也有下邻（后邻、高址）空闲区
- ( ) E.不可能的

知识点:连续内存分配
出处:网络
难度:1

> C
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)在可变式分区分配方案中，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，造成空闲区表项数增1的情况是_____。

- (x) A.无上邻（前邻、低址）空闲区，也无下邻（后邻、高址）空闲区
- ( ) B.有上邻（前邻、低址）空闲区，但无下邻（后邻、高址）空闲区
- ( ) C.有下邻（后邻、高址）空闲区，但无上邻（前邻、低址）空闲区
- ( ) D.有上邻（前邻、低址）空闲区，也有下邻（后邻、高址）空闲区
- ( ) E.不可能的

知识点:连续内存分配
出处:网络
难度:1

> A
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)在可变式分区分配方案中，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，造成空闲区表项数不变、某项的始址不变、长度增加的情况是_____。

- ( ) A.无上邻（前邻、低址）空闲区，也无下邻（后邻、高址）空闲区
- (x) B.有上邻（前邻、低址）空闲区，但无下邻（后邻、高址）空闲区
- ( ) C.有下邻（后邻、高址）空闲区，但无上邻（前邻、低址）空闲区
- ( ) D.有上邻（前邻、低址）空闲区，也有下邻（后邻、高址）空闲区
- ( ) E.不可能的

知识点:连续内存分配
出处:网络
难度:1

> B
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)当存储器采用段页式管理时，主存被划分为定长的_____。

- ( ) A.段
- ( ) B.页
- ( ) C.区域
- (x) D.块

知识点:非连续内存分配
出处:网络
难度:1

> D
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)当存储器采用段页式管理时，程序按逻辑被划分成_____。

- (x) A.段
- ( ) B.页
- ( ) C.区域
- ( ) D.块

知识点:非连续内存分配
出处:网络
难度:1

> A
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)在存储器采用段页式管理的多道程序环境下，每道程序都有对应的_____。

- ( ) A.一个段表和一个页表
- (x) B.一个段表和一组页表
- ( ) C.一组段表和一个页表
- ( ) D.一组段表和一组页表

知识点:非连续内存分配
出处:网络
难度:1

> B
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)在分页式存储管理系统中时，每次从主存中取指令或取操作数，至少要访问_____主存。

- ( ) A.1次
- (x) B.2次
- ( ) C.3次
- ( ) D.4次
- ( ) E.0次

知识点:非连续内存分配
出处:网络
难度:1

> B
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)在分段式存储管理系统中时，每次从主存中取指令或取操作数，至少要访问_____主存。

- ( ) A.1次
- (x) B.2次
- ( ) C.3次
- ( ) D.4次
- ( ) E.0次

知识点:非连续内存分配
出处:网络
难度:1

> B
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)在段页式存储管理系统中时，每次从主存中取指令或取操作数，至少要访问_____主存。

- ( ) A.1次
- ( ) B.2次
- (x) C.3次
- ( ) D.4次
- ( ) E.0次

知识点:非连续内存分配
出处:网络
难度:1

> C
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)使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。

- ( ) A.内存分配
- (x) B.内存保护
- ( ) C.地址映射
- ( ) D.对换
- ( ) E.内存扩充

知识点:缺页中断
出处:网络
难度:1

> B
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)在可变分区存储管理方案中需要一对界地址寄存器，其中_____是作为存贮保护使用。

- ( ) A.逻辑地址寄存器
- (x) B.长度寄存器
- ( ) C.物理地址寄存器
- ( ) D.基址寄存器

知识点:连续内存分配
出处:网络
难度:1

> B
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)在分页式存储管理中用作存贮保护的是_____。

- (x) A.页表长度
- ( ) B.页表始址
- ( ) C.页长(大小)
- ( ) D.重定位寄存器

知识点:非连续内存分配
出处:网络
难度:1

> A
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)在分段式存储管理中用作存贮保护的首先是_____。

- (x) A.段表长度
- ( ) B.段表始址
- ( ) C.段长
- ( ) D.重定位寄存器

知识点:非连续内存分配
出处:网络
难度:1

> A
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)用外存换内存是以牺牲程序运行时间为代价的。为提高CPU有效利用率，避免内外存的频繁交换，虚
> 拟存储技术常用某种页面淘汰策略来选择换出内存的页面，它的基础是程序的_____。

- ( ) A.完整性
- (x) B.局部性
- ( ) C.递归性
- ( ) D.正确性

知识点:置换算法
出处:网络
难度:1

> B
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)在下面关于虚拟存储器的叙述中，正确的是：_____。

- ( ) A.要求程序运行前必须全部装入内存且在运行过程中一直驻留在内存
- (x) B.要求程序运行前不必全部装入内存且在运行过程中不必一直驻留在内存
- ( ) C.要求程序运行前不必全部装入内存但是在运行过程中必须一直驻留在内存
- ( ) D.要求程序运行前必须全部装入内存但在运行过程中不必一直驻留在内存

知识点:缺页中断
出处:网络
难度:1

> B
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)虚拟存储器的作用是允许_____。

- ( ) A.直接使用外存代替内存
- ( ) B.添加比地址字长允许的更多内存容量
- (x) C.程序直接访问比内存更大的地址空间
- ( ) D.提高内存的访问速度

知识点:缺页中断
出处:网络
难度:1

> C
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)由于内存大小有限，为使得一个或多个作业能在系统中运行，常需要用外存来换取内存。其中在作业内
> 部对内外进行交换的技术称为 _____技术。

- ( ) A.SPOOLING
- ( ) B.SWAPPING
- (x) C.虚拟存储
- ( ) D.虚拟机
- ( ) E.进程管理
- ( ) F.设备管理

知识点:缺页中断
出处:网络
难度:1

> C
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)在请求分页内存管理的页表表项中，其中访问位供_____时参考。

- ( ) A.分配页面
- (x) B.置换算法
- ( ) C.程序访问
- ( ) D.换出页面
- ( ) E.调入页面

知识点:缺页中断
出处:网络
难度:1

> B
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)在请求分页内存管理的页表表项中，其中修改位供_____时参考。

- ( ) A.分配页面
- ( ) B.置换算法
- ( ) C.程序访问
- (x) D.换出页面
- ( ) E.调入页面

知识点:缺页中断
出处:网络
难度:1

> D
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)在请求分页内存管理的页表表项中，其中状态位供_____时参考。

- ( ) A.分配页面
- ( ) B.置换算法
- (x) C.程序访问
- ( ) D.换出页面
- ( ) E.调入页面

知识点:缺页中断
出处:网络
难度:1

> C
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)在请求分页内存管理的页表表项中，其中外存始址供_____时参考。

- ( ) A.分配页面
- ( ) B.置换算法

- ( ) C.程序访问
- ( ) D.换出页面
- (x) E.调入页面

知识点:缺页中断
出处:网络
难度:1

> E
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)在请求调页系统中有着多种置换算法：选择在以后不再使用的页面予以淘汰的算法称为----------。

- ( ) A.FIFO算法
- (x) B.OPT算法
- ( ) C.LRU算法
- ( ) D.NRU算法
- ( ) E.LFU算法

知识点:置换算法
出处:网络
难度:1

> B
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)在请求调页系统中有着多种置换算法：选择自上次访问以来所经历时间最长的页面予以淘汰的算法称为
> ----------。

- ( ) A.FIFO算法
- ( ) B.OPT算法
- (x) C.LRU算法
- ( ) D.NRU算法
- ( ) E.LFU算法

知识点:置换算法
出处:网络
难度:1

> C
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)在请求调页系统中有着多种置换算法：选择最先进入内存的页面予以淘汰的算法称为 ----------。

- (x) A.FIFO算法
- ( ) B.OPT算法
- ( ) C.LRU算法
- ( ) D.NRU算法
- ( ) E.LFU算法

知识点:置换算法
出处:网络
难度:1

> A
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)在请求调页系统中有着多种置换算法：选择自某时刻开始以来，访问次数最少的页面予以淘汰的算法称
> 为----------。

- ( ) A.FIFO算法
- ( ) B.OPT算法
- ( ) C.LRU算法
- ( ) D.NRU算法
- (x) E.LFU算法

知识点:置换算法
出处:网络
难度:1

> E
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)下面----------种页面置换算法会产生Belady异常现象?

- (x) A.先进先出页面置换算法（FIFO）
- ( ) B.最近最久未使用页面置换算法（LRU）
- ( ) C.最不经常使用页面置换算法（LFU）
- ( ) D.最佳页面置换算法（OPT）
- ( ) E.最近未用页面置换算法（NRU）(Clock)

知识点:置换算法
出处:网络
难度:1

> A
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)在请求分页管理中，若采用先进先出（FIFO）页面置换算法，可能会产生“Belady异常”，
> “Belady异常”指的是_____。

- ( ) A.频繁地出页入页的现象
- (x) B.分配的页面数增加，缺页中断的次数也可能增加
- ( ) C.进程交换的信息量过大，导致系统工作区不足
- ( ) D.分配给进程的内存空间不足使进程无法正常工作

知识点:置换算法
出处:网络
难度:1

> B
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)在虚拟存储器系统中常使用联想存储器进行管理，它是_____寻址的。

- ( ) A.按地址
- (x) B.按内容
- ( ) C.寄存器
- ( ) D.计算

知识点:缺页中断
出处:网络
难度:1

> B
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)下列关于虚拟存储器的论述中，正确的论述_____。

- ( ) A.在请求段页式系统中，以页为单位管理用户的虚空间，以段为单位管理内存空间。
- (x) B.在请求段页式系统中，以段为单位管理用户的虚空间，以页为单位管理内存空间。
- ( ) C.为提高请求分页系统中内存的利用率，允许用户使用不同大小的页面。
- ( ) D.实现虚拟存储器的最常用的算法是最佳适应算法OPT。

知识点:缺页中断
出处:网络
难度:1

> B
> 1
> (20150309*操作系统原理习题集及答案*笔试必需品_doc)在虚拟分页存贮管理系统中，若进程访问的页面不在主存，且主存中没有可用的空闲块时，系统正确的
> 处理顺序为_____。

- ( ) A.决定淘汰页/页面调出/缺页中断/页面调入
- ( ) B.决定淘汰页/页面调入/缺页中断/页面调出
- (x) C.缺页中断/决定淘汰页/页面调出/页面调入
- ( ) D.缺页中断/决定淘汰页/页面调入/页面调出

知识点:缺页中断
出处:网络
难度:1

> C
> 4
> (20140606-期末考试试题&参考答案.docx)信号
> (12分)在Linux/Unix中，一个用户从shell中执行了一个运行时间较长且不知何时能够结束的程序，Linux /UNIX可以让用户根据个人需求随时通过敲击Ctrl-C组合键来终止这个程序的执行。请回答如下问题。要求设计应该具有通用性，列出的设计实现不超过6点，每点不超过4行。问题的执行流程描述不超过8行。
> 1）如果要在ucore中实现Linux/UNIX同样的功能，请问应该如何修改ucore来支持此功能?
> 2）uCore的shell也是一个程序，我们希望避免这个shell在执行中被用户敲入的Ctrl-C所终止，请问在保证1）的要求请看下，如何修改ucore和shell来支持此功能?
> 3）说明在你的设计下，shell和某一可被终止程序在执行过程中，用户敲击Ctrl-C后，uCore 和shell的执行流程。

- [x]

知识点:中断、异常与系统调用
出处:网络
难度:1

> 1）（4分，信号和杀死进程各2分）
> 需要修改ucore，在中断处理中增加信号（signal）处理机制，为此需要做如下设计：
> 1. 在中断处理例程中增加对Ctrl+C的按键的识别和处理，一旦收到Ctrl+C按键，则调用do_kill（current pid）来杀死当前进程current。
> 备注：如果没有说出do_kill函数，但表达了这个处理过程，可以给分。
> 2） 6分
> 1. 增加系统调用 signal(SIGID, function addr),让ucore知道用户进程有专门的函数处理信号Ctrl+C对应的信号。（1分）
> 2. 并在进程控制块中，增加相关field，记录要处理的signal id和对应的用户处理函数地址,目前待处理的signal id。（1分）
> 3. 在中断处理例程中，如果发现有Ctrl+C按键，则在当前进程的进程控制块中，设置挂起的信号id。（2分）
> 4. 当前进程在返回用户态时，判断是否有挂起的signal id，如果有，则进一步判断当前的进程是否有要专门处理的signal id，如果没有，则调用do_kill来杀死当前进程；如果有，则修改返回用户态的堆栈（返回地址是要signal

id对应的处理函数地址），让用户进程在返回后执行处理信号的函数。（2分）
3）（2分）用户敲Ctrl+C后，ucore的中断处理例程会判断收到了此敲键组合，并在当前进程的进程控制块的记录挂起的singal id的域中记录此信息；当当前进程返回用户态前，会执行2）中描述的第4个步骤，完成相关处理。
4
(20140606-期末考试试题&参考答案.docx)IPC
(15分)在具备了执行用户态进程的能力之后，uCore要为这些进程提供的一个重要服务，是用户进程之间的消息传递机制（Inter-Process
Communication，简写为IPC）。现在，我们要为uCore实现以下两个系统调用，以实现一种同步的IPC机制（暂不考虑超时等功能）：
int sys_send_event(int pid, int event);
参数：pid - 该消息的目标进程的进程号；
event – 消息内容，用一个整型表示。
返回值：消息成功发送时，返回0；否则，返回相应的错误代码。
int sys_recv_event(int pid, int event);
参数：pid - 函数返回时，pid保存发出消息的进程的进程号，可以为NULL；
event – 函数返回时，event保存消息内容，可以为NULL。
返回值：消息成功接收时，返回0；否则，返回相应的错误代码。
1）以下是一个基于上述IPC机制求质数的用户程序：

```
  #include
    #include
    #include
    #include
    const int total = 1000;
    void primeproc(void)
    {
            int index = 0, this, num, pid = 0;
    top:
            recv_event(NULL, &this;);
            cprintf("%d is a primer.
", this);
            while (recv_event(NULL, &num;) == 0) {
                    if ((num % this) == 0) {
                            continue;
                    }
                    if (pid == 0) {
                            if (index + 1 == total) {
                                    goto out;
                            }
                            if ((pid = fork()) == 0) {
                                    index++;
                                    goto top;
                            }
                            if (pid < 0) {
                                    goto out;
                            }
                    }
                    if (send_event(pid, num) != 0) {
                            goto out;
                    }
            }
    out:
            cprintf("[%04d] %d quit.
", getpid(), index);
    }
    int main(void)
    {
            int i, pid;
            unsigned int time = gettime_msec();
            if ((pid = fork()) == 0) {
                    primeproc();
                    exit(0);
            }
            assert(pid > 0);
            for (i = 2;; i++) {
                    if (send_event(pid, i) != 0) {
                            break;
                    }
            }
            cprintf("use %d msecs.
", gettime_msec() - time);
            cprintf("primer3 pass.
");
            return 0;
    }
```

> 简述这个程序是如何判断并输出前五个质数的。
> 2）给出一种基于等待队列的上述IPC机制的实现方案。

- [x]

知识点:进程间通信
出处:网络
难度:1

> 1）（6分）每个运行primeproc的进程有一个自身编号index，其最多有一个子进程，子进程的编号号为index+1;这样形成了一个子进程链。
> 1. main函数生成第一个primeproc进程后，将从2开始的所有整数发给primeproc进程；
> 2. 每个primeproc第一次接受到消息时，会把消息输出。如果是再接受到消息，则会把第一次收到的数除这次收到的数，若不能整除，则把num交给下一个pr
> imeproc进行整除尝试；如果不存在下一个primeproc，这创建一个子进程，然后发送。
> 3. 若所有的primeproc第一次收到的均是质数，这一个新的primeproc第一次收到的数必不能被之前所有质数整除，故必定也为质数。而第一个prim
> eproc收到的是2,为质数，所以有归纳法可知，接下来的4个primeproc收到的也都是质数。
> 4. primerproc进程结束的条件是达到上限。
> 2) 可对每个进程创建一个"消息"队列（1分），send_event的操作就是把自身pid和event插入到要发送的进程pid对应的队列的尾部（2分），而r
> ecv_event的操作就是从自身的"消息"队列的头取出（2分）发送进程的pid和event.
> 由于进程是同步的：
> 1. 所以当队列为空时，recv_event应该阻塞（1分）睡眠，让接收进程睡眠在对应的信号量上；而send_event的操作在插入元素后，需要检查是否有
> sleep的进程，如果有，需要唤醒（1分）。
> 2. 所以当队列为满时，send_event应该阻塞（1分）睡眠，让发送进程睡眠在对应的信号量上；而recv_event的操作在取出元素后，需要检查是否有
> sleep的进程，如果有，需要唤醒（1分）。
> 4
> (20140606-期末考试试题&参考答案.docx)信号量实现
> (10分)在uCore中，信号量的定义如下
> ```
> typedef struct {
>     int value;
>     wait_queue_t wait_queue;
> } semaphore_t;
> ```
> **up函数是信号量V操作的具体实现函数**
> ```
> static noinline void up(semaphore_t sem, uint32_t wait_state) {
>     bool intr_flag;
>     local_intr_save(intr_flag);
>     {
>         wait_t wait;
>         if((wait=wait_queue_first(&(sem->wait_queue)))==NULL){
>             __;
>         } else {
>             wakeup_wait(&(sem->wait_queue), wait, wait_state, 1);
>         }
>     }
>     local_intr_restore(intr_flag);
> }
> ```
> 1）补全程序中的空行__。
> **2）信号量的value值>0时，表示**_的数量;value值<0时，表示___的数量。
> 3）local_intr_save和local_intr_restore这两个函数的功能分别是什么？为什么要调用这两个函数?

- [x]

知识点:信号量
出处:网络
难度:1

> 1) sem->value++（2分）
> 2) 空闲共享资源（2分）
> 信号量等待队列中的进程（2分）
> 3）关闭中断（1分）
> 打开中断（1分）
> 通过关闭中断，防止当前的操作被打断，保证了读写内存的原子性，实现了对临界区的互斥操作。（2分）
> 4
> (20140606-期末考试试题&参考答案.docx)信号量应用
> (15分)假设一个MOOC网站有1、2、3三种不同的课程视频可由学生选择学习，网站播放课程视频的规则为：
> 1）任一时刻最多只能播放一种课程视频，正在播放的课程视频是自动循环播放的，最后一个学生主动离开时结束当前课程视频的播放；
> 2）选择当前正在播放的课程视频的学生可立即进入播放页面，允许同时有多位选择同一种课程视频的学生观看，同时观看的学生数量不受限制；
> 3）等待观看其它课程视频的学生按到达顺序排队，当一种新的课程视频开始放映时，所有等待观看该课程视频的学生可依次序进入播放页面同时观看。
> 用一个进程代表一个学生，要求：用信号量的P、V操作实现上述规则，并给出信号量的定义和初始值。

- [x]

知识点:信号量
出处:网络
难度:1

> 信号量和变量：（6分，信号量数组，每个2分，其他每个量1分）
> i取值范围为0～2
> m[3]是三个信号量，初始值为1,用于保护对count[i]的读写。
> sem_g是一个信号量，初始值为1,代表当前是否可以播放视频
> count[3]是三个统计值，初始值为0,代表看视频i的人数
> 代码：（9分，其中三个PV对，每个2分；两个计数修改，每个计数1分；条件1分；）

```
Procedure student(i)
{
    down(m[i]);
    if count[i]==0 {
        down（sem_g);
    }
    count[i]++;
    up(m[i]);
    // into Critical Section
    // watch video
    // out Critical Section
    down(m[i]);
    count[i]--;
    if count[i]==0{
        up(sem_g);
    }
    up(m[i]);
}
```

> 4
> (20140606-期末考试试题&参考答案.docx)调度算法
> (12分)在lab6中，我们实现了Stride
> Scheduling调度算法，并声称它对“进程的调度次数正比于其优先级”。对于优先级为2、3、5、7的4个进程，选取210为
> MAX_STRIDE，则：
> 1）简要描述Stride Scheduling调度算法。
> 2）四个进程的步长分别为：_、_、_、_。
> 3）假设四个进程的初始stride值均为0，证明：总有一个时刻，四个进程的stride值都是210，且此时四个进程被调度的次数正比于其优先级。

- [x]

知识点:处理机调度实验
出处:网络
难度:1

> 1）要点：
> a. 每个进程设置一个当前stride和步长；（2分）
> b. 每次进程被调度之后，stride增加其步长值（优先级）；（1分）
> c. 每次调度，选择当前stride最小的进程。（1分）
> 2）105、70、42、30（4分，每个1分）
> 3）由stride算法，可知每个进程在多次调度之后，其各自的stride都会恰好达到210；
> 假设某一时刻，有一个进程A的stride为210加其步长，同时存在一个stride小于210的进程B，那么在A上一次被调度时，A的stride为210，而B
> 的stride仍然小于210，与stride算法总选择stride最小的规则矛盾；（2分）
> 此时，四个进程被调度的次数为2、3、5、7，与其优先级相同。
> 每个进程的stride都是210的因子，于是会有时刻，它的stride为210（1分）。第一个进程大于210时（1分），其他进程不可能小于210。否则，与选
> 择stride最小的进程矛盾（1分）。步长值的定义可知，这时的调度次数是步长值。（1分）
> 4
> (20140606-期末考试试题&参考答案.docx)死锁
> (12分)死锁是操作系统中资源共享时面临的一个难题。请回答下列与死锁相关的问题。
> 1）设系统中有下述解决死锁的方法：
> a)银行家算法；
> b)检测死锁，终止处于死锁状态的进程，释放该进程占有的资源；
> c)资源预分配。
> 简述哪种办法允许最大的并发性，即哪种办法允许更多的进程无等待地向前推进？请按“并发性”从大到小对上述三种办法进行排序。
> 2）假设一个使用银行家算法的系统，当前有5个进程P0, P1, P2, P3, P4，系统中有三类资源A、B、C，假设在某时刻有如下状态：
> | |Allocation| Max |Available|
> | | A | B | C|A|B|C| A | B| C|
> |P0| 0 | 0 | 3|0|0|4| 1 | 4| 0|
> |P1| 1 | 0 | 0|1|7|5| | | |
> |P2| 1 | 3 | 5|2|3|5| | | |
> |P3| 0 | 0 | 2|0|6|4| | | |
> |P4| 0 | 0 | 1|0|6|5| | | |
> 请问当前系统是否出于安全状态？如果系统中的可利用资源为（0, 6,
> 2），系统是否安全？如果系统处在安全状态，请给出安全序列；如果系统处在非安全状态，请简要说明原因。

- [x]

知识点:死锁
出处:网络
难度:1

> 1）（4分，b最大2分；顺序2分） b)方法具有最大的并行性。 b>a>c
> 2) 当前系统处于安全状态（2分）。因为 p2,p0,p1,p3,p4是一个安全的执行序列。（2分）
> 如果系统中的可利用资源为（0, 6, 2），则找不到一个安全执行序列（2分），因为p1需要B资源数量7,p2需要A资源数量1,形成了死锁（2分）。
> 4
> (20140606-期末考试试题&参考答案.docx)文件系统
> (12分)uCore实现了一个简单的文件系统Simple FS，假设该文件系统现已经装载到一个硬盘中（disk0），该硬盘的大小为20M，目前有三个文件A.txt,B.txt和C.txt存放在该硬盘中，三个文件的大小分别是48K, 1M和4M。
> 1）简要描述SFS文件系统中文件数据的组织结构（即：SFS文件的数据的存放位置组织方式）。
> 2）请根据Simple FS的设计实现情况，画出该文件系统当前在disk0上的布局情况，需要给出相应结构的名称和起始块号。

- [x]

知识点:文件系统实验
出处:网络
难度:1

> 1) 一个superblock维护基本信息（1'），多个freemap（数量由分区大小确定，1'），一个根目录inode（1'）；
> 目录和文件均由一个inode和具体数据块组成，其中inode包含文件的基本属性、12个直接索引和一级/二级索引表的块地址（1'），目录的数据块中存放（文件名
> 、inode地址）的数组（1'），文件的数据块中存放文件的具体内容（1'）。
> 2）（除了0、1、2以外，其它块地址均可变，言之有理即可）
> （2分）
> 0 superblock
> 1 根目录inode
> 2 freemap（640K，只需要1块）
> 3 根目录的数据块（包含A.txt、B.txt、C.txt的inode的地址）
> （1分）
> 4 A.txt的inode（包含12个直接索引块的地址）
> 5-16 A.txt的数据块
> （2分）
> 17 B.txt的inode（包含12个直接索引块和1个一级间接索引）
> 18-29 B.txt的直接索引数据
> 30 B.txt的一级间接索引（包含244个数据块地址）
> 31-274 B.txt的一级间接索引块
> （1分）
> 275 C.txt的inode（包含12个直接索引块和1个一级间接索引）
> 276-287 C.txt的一级间接索引块
> 288 C.txt的一级间接索引（包含1012个数据块地址）
> 289-1300 C.txt的一级间接索引块
> 4
> (20140606-期末考试试题&参考答案.docx)VFS
> (12分)uCore的文件管理主要由以下四个部分组成：通用文件系统访问接口层, 文件系统抽象层(VFS), 具体文件系统层以及外设接口层，其中VFS层的作用
> 是用来管理不同的文件系统并向上提供一致的接口给内核其他部分访问，在ucore中我们已经实现了一个具体的文件系统：Simple FS，并将该文件系统装载到了disk0上，假设ucore又实现了一个文件系统FAT32，并将这个新的文件系统装载到了disk1上。
> 1）请简单描述一下如何修改VFS层的数据结构使其可以有效的管理上述已安装的具体文件系统。 涉及VFS层的数据结构如下：

```
  struct file {
        enum {
            FD_NONE, FD_INIT, FD_OPENED, FD_CLOSED,
        } status;
        bool readable;
        bool writable;
        int fd;
        off_t pos;
        struct inode node;
        atomic_t open_count;
    };
    struct inode {
        union {
            struct device __device_info;
            struct sfs_inode __sfs_inode_info;
        } in_info;
        enum {
            inode_type_device_info = 0x1234,
            inode_type_sfs_inode_info,
        } in_type;
        atomic_t ref_count;
        atomic_t open_count;
        struct fs in_fs;
        const struct inode_ops in_ops;
    };
    struct fs {
        union {
```

```
            struct sfs_fs __sfs_info;
        } fs_info;
        enum {
            fs_type_sfs_info,
        } fs_type;
        int (fs_sync)(struct fs fs);
        struct inode (fs_get_root)(struct fs fs);
        int (fs_unmount)(struct fs fs);
        void (fs_cleanup)(struct fs fs);
    };
    struct inode_ops {
        unsigned long vop_magic;
        int (vop_open)(struct inode node, uint32_t open_flags);
        int (vop_close)(struct inode node);
        int (vop_read)(struct inode node, struct iobuf iob);
        int (vop_write)(struct inode node, struct iobuf iob);
        int (vop_getdirentry)(struct inode node, struct iobuf iob);
        int (vop_create)(struct inode node, const char name, bool excl, struct inode node_store);
        int (vop_lookup)(struct inode node, char path, struct inode node_store);
        ……
    };
```

2）两个具体文件系统均已实现了对数据文件的4种基本操作。现在有某个用户态进程执行了一个copy（source_path, dest_path,...）函数，
该函数是把disk1根目录下的一个文件A.txt拷贝到了disk0的根目录下（不用考虑文件的大小），请结合ucore中对数据文件的操作流程描述一下这个函数的
执行过程。

- [x]

知识点:文件系统实验
出处:网络
难度:1

第一问7分，第二问5分；
1）
（2分）在inode中的in_info和in_type增加fat32相关表示；
（2分）在fs中的fs_info和fs_type增加fat32相关信息，并在创建inode时将新的fs赋给in_fs；
（1分）实现fs中用函数指针定义的所有操作；
（2分）实现inode_ops中定义的所有操作，并在创建inode时将其赋给in_ops。
2）打开（2分）、查找、复制和关闭各1分
a. vop_lookup在disk1上查找该文件
b. vop_open在disk1上打开A.txt；
c. vop_lookup在disk0上找到待写文件的父目录；
d. vop_getdirentry在该目录中查找待写文件是否存在，若存在则用vop_open打开，否则用vop_create创建；
e. vop_read（disk1）和vop_write（disk0）在两个文件之间复制数据；
f. vop_close分别关闭两个文件。
4
(20140410-2-期中考试试题v4a答案.docx)页表
（20分）内存管理（Memory Management）是操作系统的重要职能之一，现代操作系统基于硬件所提供的内存管理单元（Memory Management Unit），可以为应用程序提供相互隔离的虚拟地址空间，同时对物理内存进行高效的管理。在32位x86架构提供的MMU中，除了传统的段模式
以外，也同时包括页管理机制。页管理所需要的硬件支持包括两个部分：一是完成虚拟地址到物理地址映射的页表，二是页异常（Page Fault）。
1) 在32位的x86系统中，一般使用二级页表，分别用虚拟地址的31-22位和21-12位作为页表内相应页表项的偏移，此时一个物理页的大小为***(1a)***
_K；实际上，x86系统同样允许只使用一级页表，页表项偏移仍然取虚拟地址的31-22位，此时一个物理页的大小为**(1b)**K。
2) 发生页异常时，硬件会保存执行时的上下文并关闭中断，然后跳转到操作系统设置好的页错误处理例程，这里的“上下文”应该包括（在你认为需要保存的寄存器前打勾，
并简述如果不保存会产生什么问题）：
(2a)（ ）指令计数器（CS, EIP）

(2b)（ ）堆栈指针（SS, ESP）

(2c)（ ）通用寄存器（EAX、EBX、……）

(2d)（ ）执行时标志位寄存器（EFLAGS）

(2e)（ ）控制寄存器（CR0、CR3、……）

---

3) 页异常处理完毕后，返回用户程序继续执行，此时执行的第一条用户指令为（ ）
A. 触发页异常指令的上一条指令
B. 触发页异常的指令
C. 触发页异常指令的下一条指令
4) 除了维护基本的地址映射关系外，x86页表的每一个页表项还包括一些其它配置和信息位，例如该页是否可写（W），是否可以在Ring 3访问（U），是否曾被访

问过（A），以及是否曾被写过（D）。请根据x86页表对这些位的定义，在下表中填写在页表项的几种情况下进行各种操作时，页表项的内容会如何变化。（只需写出会变化
的位和/或会产生的事件，如缺页异常，形式可参考表中已填入的部分内容）
	W=0 U=0 A=0 D=0	W=1 U=0 A=0 D=0	W=0 U=1 A=1 D=0	W=1 U=1 A=1 D=1
在ring0读				无变化
在ring0写		A→1，D→1		
在ring3读				
在ring3写			缺页异常	

- [x]

知识点:缺页中断
出处:网络
难度:1

> 4
> (20140410-2-期中考试试题v4a答案.docx)进程管理
> （20分）进程/线程管理（Process/Thread
> Management）是操作系统的重要职能之一，现代操作系统基于硬件所提供的内存管理单元（Memory Management
> Unit），可以为进程提供相互隔离的虚拟地址空间，为线程提供共享的虚拟地址空间。以下我们对32位x86架构的uCore所实现的进程管理机制进行讨论。
> 1) 在ucore中，管理一个用户进程的进程控制块数据结构为proc_struct，管理一个内核线程的线程控制块数据结构为**(1a)**。在proc_struct中，为了有效管理用户进程，请问可唯一标识一个进程的field是**(1b)**；用户进程控制块与内核线程控制块相比，数据内容肯定不同的field包括
> **(1c)**、**(1d)**、**(1e)**。
> 2) 在ucore中，一个用户进程具有"自己"的用户栈，当用户进程通过系统调用进入到内核态开始继续执行ucore指令时，进程的页表起始地址是否会改变？**(**
> **2a)**。当用户进程在用户态执行时，硬件产生了一个中断，打断了用户进程的执行，这时CPU将开始执行中断服务例程，，这个时候的页表起始地址是否已经不是被打断
> 的用户进程的页表起始地址了？**(2b)_**。
> **3) 在ucore中，当用户进程访问的某个虚拟地址的映射关系不在TLB中时，是否一定会产生异常？**(3a)**。当用户进程访问的某个虚拟地址在其页表中没有**
> **valid的页表项，是否一定回产生异常？**(3b)**。**
> **4) 在ucore中，当用户进程A与用户进程B进行进程上下文切换时，需要保存相关的寄存器内容，请问是否需要保存CS？**(4a)**。是否需要保存EIP？** _
> ***(4b)*****。是否需要保存SS？**(4c)**。是否需要保存ESP？**(4d)__。
> 5) 在ucore中，如果当父进程创建子进程时，如果没有COW机制，则fork系统调用会创建新的子进程的进程控制块，创建新的子进程的页表，并把父进程的代码段
> 和数据段所占的物理内存空间复制一份到新的物理内存空间，并更新子进程页表。如果采用了COW机制，则fork系统调用的处理过程是？（用不超过6行文字进行描述）

- [x]

知识点:线程管理实验
出处:网络
难度:1

> （20分）进程/线程管理（Process/Thread
> Management）是操作系统的重要职能之一，现代操作系统基于硬件所提供的内存管理单元（Memory Management
> Unit），可以为进程提供相互隔离的虚拟地址空间，为线程提供共享的虚拟地址空间。以下我们对32位x86架构的uCore所实现的进程管理机制进行讨论。
> 每空1分
> 1) 在ucore中，管理一个用户进程的进程控制块数据结构为proc_struct，管理一个内核线程的线程控制块数据结构为**proc_struct__**
> *。在proc_struct中，为了有效管理用户进程，请问可唯一标识一个进程的field是**pid_；用户进程控制块与内核线程控制块相比，数据内容*
> ***肯定不同的field包括**_pid、**kstack_**、mm_。*
> *2) 在ucore中，一个用户进程具有"自己"的用户栈，当用户进程通过系统调用进入到内核态开始继续执行ucore指令时，进程的页表起始地址是否会改变？* _
> 不会_。当用户进程在用户态执行时，硬件产生了一个中断，打断了用户进程的执行，这时CPU将开始执行中断服务例程，，这个时候的页表起始地址是否已经不是被
> 打断的用户进程的页表起始地址了？***不是***。
> *3) 在ucore中，当用户进程访问的某个虚拟地址的映射关系不在TLB中时，是否一定会产生异常？**否**_。当用户进程访问的某个虚拟地址在其页表中没有*
> *valid的页表项，是否一定回产生异常？是_。*
> ***4) 在ucore中，当用户进程A与用户进程B进行进程上下文切换时，需要保存相关的寄存器内容，请问是否需要保存CS？**否_。是否需要保存EIP？*
> ***是__**。是否需要保存SS？***否_***。是否需要保存ESP？***是_***。
> 5) 在ucore中，如果当父进程创建子进程时，如果没有COW机制，则fork系统调用会创建新的子进程的进程控制块，创建新的子进程的页表，并把父进程的代码段
> 和数据段所占的物理内存空间复制一份到新的物理内存空间，并更新子进程页表。如果采用了COW机制，则fork系统调用的处理过程是？（用不超过6行文字进行描述）
> 1 创建新的子进程的进程控制块 1分
> 2 分配内核堆栈kstack 2分
> 3 共享进程控制块中的mm field 2分
> 4 在子进程的内核堆栈上设置trapframe 如果答出，也给1分
> 5 把用户空间的进程页表项设置只读 （这样对页写会触发异常，从而进一步完成COW机制） 2分
> 6 设置父进程返回值为子进程的pid，子进程的返回值为0 如果答出，也给1分
> 4
> (20140410-2-期中考试试题v4a答案.docx)系统启动
> （15分）Bootloader是ucore操作系统启动中的很重要的一部分，Bootloader是由BIOS代码读入内存，然后跳转到它开始执行的。请参考boo

tasm.S和bootmain.c的源代码，回答下列问题：
1) Bootloader包含在硬盘主引导扇区中，硬盘主引导扇区的主要特征有哪些?
2) Bootloader执行的第一条指令是哪一行？Bootloader从实模式进入保护模式后执行的第一条指令是哪一行？为什么要转换到保护模式?
3) Bootloader在完成从硬盘扇区读入ucore内核映像后是如何跳转到ucore内核代码的?

```
// =========/libs/elf.h=========
#ifndef __LIBS_ELF_H__
#define __LIBS_ELF_H__
#include
#define ELF_MAGIC    0x464C457FU            // "\x7FELF" in little endian
/ file header /
struct elfhdr {
    uint32_t e_magic;     // must equal ELF_MAGIC
    uint8_t e_elf[12];
    uint16_t e_type;      // 1=relocatable, 2=executable, 3=shared object, 4=core image
    uint16_t e_machine;   // 3=x86, 4=68K, etc.
    uint32_t e_version;   // file version, always 1
    uint32_t e_entry;     // entry point if executable
    uint32_t e_phoff;     // file position of program header or 0
    uint32_t e_shoff;     // file position of section header or 0
    uint32_t e_flags;     // architecture-specific flags, usually 0
    uint16_t e_ehsize;    // size of this elf header
    uint16_t e_phentsize; // size of an entry in program header
    uint16_t e_phnum;     // number of entries in program header or 0
    uint16_t e_shentsize; // size of an entry in section header
    uint16_t e_shnum;     // number of entries in section header or 0
    uint16_t e_shstrndx;  // section number that contains section name strings
};
/ program section header /
struct proghdr {
    uint32_t p_type;   // loadable code or data, dynamic linking info,etc.
    uint32_t p_offset; // file offset of segment
    uint32_t p_va;     // virtual address to map segment
    uint32_t p_pa;     // physical address, not used
    uint32_t p_filesz; // size of segment in file
    uint32_t p_memsz;  // size of segment in memory (bigger if contains bss)
    uint32_t p_flags;  // read/write/execute bits
    uint32_t p_align;  // required alignment, invariably hardware page size
};
#endif / !__LIBS_ELF_H__ /
//=========/boot/bootasm.S=========
#include
# Start the CPU: switch to 32-bit protected mode, jump into C.
# The BIOS loads this code from the first sector of the hard disk into
# memory at physical address 0x7c00 and starts executing in real mode
# with %cs=0 %ip=7c00.
.set PROT_MODE_CSEG,        0x8                     # kernel code segment selector
.set PROT_MODE_DSEG,        0x10                    # kernel data segment selector
.set CR0_PE_ON,             0x1                     # protected mode enable flag
# start address should be 0:7c00, in real mode, the beginning address of the running bootloader
.globl start
start:
.code16                                             # Assemble for 16-bit mode
    cli                                             # Disable interrupts
    cld                                             # String operations increment
    # Set up the important data segment registers (DS, ES, SS).
    xorw %ax, %ax                                   # Segment number zero
    movw %ax, %ds                                   # -> Data Segment
    movw %ax, %es                                   # -> Extra Segment
    movw %ax, %ss                                   # -> Stack Segment
    # Enable A20:
    #  For backwards compatibility with the earliest PCs, physical
    #  address line 20 is tied low, so that addresses higher than
    #  1MB wrap around to zero by default. This code undoes this.
seta20.1:
    inb $0x64, %al                                  # Wait for not busy
    testb $0x2, %al
    jnz seta20.1
    movb $0xd1, %al                                 # 0xd1 -> port 0x64
    outb %al, $0x64
seta20.2:
    inb $0x64, %al                                  # Wait for not busy
    testb $0x2, %al
    jnz seta20.2
```

```
        movb $0xdf, %al                                 # 0xdf -> port 0x60
        outb %al, $0x60
        # Switch from real to protected mode, using a bootstrap GDT
        # and segment translation that makes virtual addresses
        # identical to physical addresses, so that the
        # effective memory map does not change during the switch.
        lgdt gdtdesc
        movl %cr0, %eax
        orl $CR0_PE_ON, %eax
        movl %eax, %cr0
        # Jump to next instruction, but in 32-bit code segment.
        # Switches processor into 32-bit mode.
        ljmp $PROT_MODE_CSEG, $protcseg
    .code32                                             # Assemble for 32-bit mode
    protcseg:
        # Set up the protected-mode data segment registers
        movw $PROT_MODE_DSEG, %ax                       # Our data segment selector
        movw %ax, %ds                                   # -> DS: Data Segment
        movw %ax, %es                                   # -> ES: Extra Segment
        movw %ax, %fs                                   # -> FS
        movw %ax, %gs                                   # -> GS
        movw %ax, %ss                                   # -> SS: Stack Segment
        # Set up the stack pointer and call into C. The stack region is from 0--start(0x7c00)
        movl $0x0, %ebp
        movl $start, %esp
        call bootmain
        # If bootmain returns (it shouldn't), loop.
    spin:
        jmp spin
    # Bootstrap GDT
    .p2align 2                                          # force 4 byte alignment
    gdt:
        SEG_NULLASM                                     # null seg
        SEG_ASM(STA_X|STA_R, 0x0, 0xffffffff)           # code seg for bootloader and kernel
        SEG_ASM(STA_W, 0x0, 0xffffffff)                 # data seg for bootloader and kernel
    gdtdesc:
        .word 0x17                                      # sizeof(gdt) - 1
        .long gdt                                       # address gdt
```

```
//=========/boot/bootmain.c=========
#include
#include
#include
/
  This a dirt simple boot loader, whose sole job is to boot
  an ELF kernel image from the first IDE hard disk.
```

```
1   DISK LAYOUT
2     This program(bootasm.S and bootmain.c) is the bootloader.
3      It should be stored in the first sector of the disk.
4
5     The 2nd sector onward holds the kernel image.
6
7     The kernel image must be in ELF format.
8
9   BOOT UP STEPS
10    when the CPU boots it loads the BIOS into memory and executes it
11
12    the BIOS intializes devices, sets of the interrupt routines, and
13     reads the first sector of the boot device(e.g., hard-drive)
14     into memory and jumps to it.
15
16    Assuming this boot loader is stored in the first sector of the
17     hard-drive, this code takes over...
18
19    control starts in bootasm.S -- which sets up protected mode,
20     and a stack so C code then run, then calls bootmain()
21
22    bootmain() in this file takes over, reads in the kernel and jumps to it.
23   /
24  #define SECTSIZE        512
25  #define ELFHDR          ((struct elfhdr )0x10000)      // scratch space
26  / waitdisk - wait for disk ready /
27  static void
28  waitdisk(void) {
29      while ((inb(0x1F7) & 0xC0) != 0x40)
30          / do nothing /;
31  }
32  / readsect - read a single sector at @secno into @dst /
```

```
static void
readsect(void dst, uint32_t secno) {
    // wait for disk to be ready
    waitdisk();
    outb(0x1F2, 1);                         // count = 1
    outb(0x1F3, secno & 0xFF);
    outb(0x1F4, (secno >> 8) & 0xFF);
    outb(0x1F5, (secno >> 16) & 0xFF);
    outb(0x1F6, ((secno >> 24) & 0xF) | 0xE0);
    outb(0x1F7, 0x20);                      // cmd 0x20 - read sectors
    // wait for disk to be ready
    waitdisk();
    // read a sector
    insl(0x1F0, dst, SECTSIZE / 4);
}
/
  readseg - read @count bytes at @offset from kernel into virtual address @va,
  might copy more than asked.
  /
static void
readseg(uintptr_t va, uint32_t count, uint32_t offset) {
    uintptr_t end_va = va + count;
    // round down to sector boundary
    va -= offset % SECTSIZE;
    // translate from bytes to sectors; kernel starts at sector 1
    uint32_t secno = (offset / SECTSIZE) + 1;
    // If this is too slow, we could read lots of sectors at a time.
    // We'd write more to memory than asked, but it doesn't matter --
    // we load in increasing order.
    for (; va < end_va; va += SECTSIZE, secno ++) {
        readsect((void )va, secno);
    }
}
/ bootmain - the entry of bootloader /
void
bootmain(void) {
    // read the 1st page off disk
    readseg((uintptr_t)ELFHDR, SECTSIZE  8, 0);
    // is this a valid ELF?
    if (ELFHDR->e_magic != ELF_MAGIC) {
        goto bad;
    }
    struct proghdr ph, eph;
    // load each program segment (ignores ph flags)
    ph = (struct proghdr )((uintptr_t)ELFHDR + ELFHDR->e_phoff);
    eph = ph + ELFHDR->e_phnum;
    for (; ph < eph; ph ++) {
        readseg(ph->p_va & 0xFFFFFF, ph->p_memsz, ph->p_offset);
    }
    // call the entry point from the ELF header
    // note: does not return
    ((void ()(void))(ELFHDR->e_entry & 0xFFFFFF))();
bad:
    outw(0x8A00, 0x8A00);
    outw(0x8A00, 0x8E00);
    / do nothing /
    while (1);
}
```

- [x]

知识点:启动和中断处理实验
出处:网络
难度:1

> MBR由三部分构成：
> 1．主引导程序代码，占446字节
> 2．硬盘分区表DPT，占64字节
> 3．主引导扇区结束标志AA55H
> 4
> (20140410-2-期中考试试题v4a答案.docx)Fork
> （15分）进程管理是操作系统提供给应用程序的一种用于进程控制的服务。下面是一个用fork系统调用完成进程创建的程序。试回答下面问题：
> 1) 描述fork系统调用的功能、调用接口。
> 2) 补全程序的输出信息。
> `//=========fork.c=========`
> `#include`
> `#include`
> `#include`
> `#include`
> `#include`
> `#define DEFAULT_TIME 5`
> `#define DEFAULT_STATUS 0`

```
    int main (int argc, char argv) {
        int child_id;
        int seconds;
        int status;
        pid_t whodied;
        status = DEFAULT_STATUS;
        if (argc == 1)
            seconds = DEFAULT_TIME;
        else
            seconds = atoi (argv[1]);
        printf ("Here I am in the program!  Time to wait = %d
", seconds);
        system ("ps -l");
        child_id = fork();
        if (child_id) {
            printf ("I'm the parent at Line 33.  My parent's process ID is %d, My process ID is %d, status =
%d.
", getpid(), getppid(), status);
            whodied = wait (&status;);
            printf ("Child %d exited ", whodied);
    / WIFEXITED  evaluates  to  true  when  the  process  exited  by using an exit(2V) call.
     If WIFEXITED(status) is non-zero, WEXITSTATUS evaluates to  the low-order byte of the
    argument that the child process passed to _exit() (see exit(2V)) or exit(3),  or  the  value  the  child
process returned from main() (see execve(2V)).
    /
            if (! WIFEXITED(status)) {
                printf ("abnormally!
");
                }
            else {
                printf ("with status %d.
", WEXITSTATUS(status));
                }
            printf ("I'm the parent at Line 43.  My parent's process ID is %d, My process ID is %d, status =
%d.
", getpid(), getppid(), WEXITSTATUS(status));
            return status;
            }
        else {
            status = 17;
            sleep(seconds);
            printf ("I'm the child.  My parent's process ID is %d, My process ID is %d, status = %d.
", getpid(), getppid(), status);
            printf ("Bye now!
");
            return status;
            }
        }
```

fork程序的两次执行时的输出信息

```
  xyong@portal:~/work$ ./a.out
    Here I am in the program!  Time to wait = __(1)__
    F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
    0 S  1000 11739 11738  0  80   0 -  6926 wait   pts/0    00:00:00 bash
    0 S  1000 11862 11739  0  80   0 -  1041 wait   pts/0    00:00:00 a.out
    0 S  1000 11863 11862  0  80   0 -  1101 wait   pts/0    00:00:00 sh
    0 R  1000 11864 11863  0  80   0 -  2433 -      pts/0    00:00:00 ps
    I'm the parent at Line 33.  My parent's process ID is __(2)__, My process ID is __(3)__, status =
__(4)__.
    I'm the child.  My parent's process ID is __(5)__, My process ID is __(6)__, status = __(7)__.
    Bye now!
    Child 11865 exited with status __(8)__.
    I'm the parent at Line 43.  My parent's process ID is __(9)__, My process ID is __(10)__, status =
__(11)__.
    xyong@portal:~/work$ ./a.out 3
    Here I am in the program!  Time to wait = __(12)__
    F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
    0 S  1000 11739 11738  0  80   0 -  6926 wait   pts/0    00:00:00 bash
    0 S  1000 11866 11739  0  80   0 -  1041 wait   pts/0    00:00:00 a.out
    0 S  1000 11867 11866  0  80   0 -  1101 wait   pts/0    00:00:00 sh
    0 R  1000 11868 11867  0  80   0 -  2433 -      pts/0    00:00:00 ps
    I'm the parent at Line 33.  My parent's process ID is __(13)__, My process ID is __(14)__, status =
__(15)__.
    I'm the child.  My parent's process ID is __(16)__, My process ID is __(17)__, status = __(18)__.
    Bye now!
```

```
    Child __(19)__ exited with status __(20)__.
    I'm the parent at Line 43.  My parent's process ID is 11866, My process ID is __(21)__, status =
__(22)__.
    xyong@portal:~/work$
```

- [x]

知识点:进程状态与控制
出处:网络
难度:1

fork程序的两次执行时的输出信息

```
xyong@portal:~/work$ ./a.out
Here I am in the program!  Time to wait = 5
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  1000 11739 11738  0  80   0 -  6926 wait   pts/0    00:00:00 bash
0 S  1000 11862 11739  0  80   0 -  1041 wait   pts/0    00:00:00 a.out
0 S  1000 11863 11862  0  80   0 -  1101 wait   pts/0    00:00:00 sh
0 R  1000 11864 11863  0  80   0 -  2433 -      pts/0    00:00:00 ps
I'm the parent at Line 33.  My parent's process ID is 11862, My process ID is 11739, status = 0.
I'm the child.  My parent's process ID is 11865, My process ID is 11862, status = 17.
Bye now!
Child 11865 exited with status 17.
I'm the parent at Line 43.  My parent's process ID is 11862, My process ID is 11739, status = 17.
xyong@portal:~/work$ ./a.out 3
Here I am in the program!  Time to wait = 3
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  1000 11739 11738  0  80   0 -  6926 wait   pts/0    00:00:00 bash
0 S  1000 11866 11739  0  80   0 -  1041 wait   pts/0    00:00:00 a.out
0 S  1000 11867 11866  0  80   0 -  1101 wait   pts/0    00:00:00 sh
0 R  1000 11868 11867  0  80   0 -  2433 -      pts/0    00:00:00 ps
I'm the parent at Line 33.  My parent's process ID is 11866, My process ID is 11739, status = 0.
I'm the child.  My parent's process ID is 11869, My process ID is 11866, status = 17.
Bye now!
Child 11869 exited with status 17.
I'm the parent at Line 43.  My parent's process ID is 11866, My process ID is 11739, status = 17.
xyong@portal:~/work$
```

4
(20140410-2-期中考试试题v4a答案.docx)函数调用
（15分）为实现函数的调用和返回功能，X86指令集中提供了call和ret两条指令。为在操作系统内核执行过程中分析了解函数函数的嵌套调用关系，ucore中实
现了函数print_stackframe，用于跟踪函数调用堆栈中记录的返回地址。如果能够正确实现此函数，它将在qemu模拟器中得到类似如下的输出：

```
……
    ebp:0x00007b28 eip:0x00100992 args:0x00010094 0x00010094 0x00007b58 0x00100096
        kern/debug/kdebug.c:305: print_stackframe+22
    ebp:0x00007b38 eip:0x00100c79 args:0x00000000 0x00000000 0x00000000 0x00007ba8
        kern/debug/kmonitor.c:125: mon_backtrace+10
    ebp:0x00007b58 eip:0x00100096 args:0x00000000 0x00007b80 0xffff0000 0x00007b84
        kern/init/init.c:48: grade_backtrace2+33
    ebp:0x00007b78 eip:0x001000bf args:0x00000000 0xffff0000 0x00007ba4 0x00000029
        kern/init/init.c:53: grade_backtrace1+38
    ebp:0x00007b98 eip:0x001000dd args:0x00000000 0x00100000 0xffff0000 0x0000001d
        kern/init/init.c:58: grade_backtrace0+23
    ebp:0x00007bb8 eip:0x00100102 args:0x0010353c 0x00103520 0x00001308 0x00000000
        kern/init/init.c:63: grade_backtrace+34
    ebp:0x00007be8 eip:0x00100059 args:0x00000000 0x00000000 0x00000000 0x00007c53
        kern/init/init.c:28: kern_init+88
    ebp:0x00007bf8 eip:0x00007d73 args:0xc031fcfa 0xc08ed88e 0x64e4d08e 0xfa7502a8
    : -- 0x00007d72 –
    ……
```

请回答如下问题。
1) 描述函数调用和返回指令的执行过程。
2) ucore中的函数调用参数是如何从调用函数（caller）传递给被调用函数（callee）的。
3) 补全函数调用堆栈跟踪函数print_stackframe。

```
//=========/kern/debug/kdebug.c=========
#include
#include
#include
#include
#include
#include
#define STACKFRAME_DEPTH 20
extern const struct stab STAB_BEGIN[];  // beginning of stabs table
extern const struct stab STAB_END[];    // end of stabs table
extern const char STABSTR_BEGIN[];      // beginning of string table
extern const char STABSTR_END[];        // end of string table
```

```
/ debug information about a particular instruction pointer /
struct eipdebuginfo {
    const char eip_file;              // source code filename for eip
    int eip_line;                     // source code line number for eip
    const char eip_fn_name;           // name of function containing eip
    int eip_fn_namelen;               // length of function's name
    uintptr_t eip_fn_addr;            // start address of function
    int eip_fn_narg;                  // number of function arguments
};
/
  stab_binsearch - according to the input, the initial value of
  range [@region_left, @region_right], find a single stab entry
  that includes the address @addr and matches the type @type,
  and then save its boundary to the locations that pointed
  by @region_left and @region_right.
```

```
1   Some stab types are arranged in increasing order by instruction address.
2   For example, N_FUN stabs (stab entries with n_type == N_FUN), which
3   mark functions, and N_SO stabs, which mark source files.
4
5   Given an instruction address, this function finds the single stab entry
6   of type @type that contains that address.
7
8   The search takes place within the range [@region_left, @region_right].
9   Thus, to search an entire set of N stabs, you might do:
10
11       left = 0;
12       right = N - 1;    (rightmost stab)
13       stab_binsearch(stabs, &left;, &right;, type, addr);
14
15  The search modifies region_left and region_right to bracket the @addr.
16  @region_left points to the matching stab that contains @addr,
17  and @region_right points just before the next stab.
18  If @region_left > region_right, then @addr is not contained in any
19  matching stab.
20
21  For example, given these N_SO stabs:
22       Index  Type   Address
23       0      SO     f0100000
24       13     SO     f0100040
25       117    SO     f0100176
26       118    SO     f0100178
27       555    SO     f0100652
28       556    SO     f0100654
29       657    SO     f0100849
30  this code:
31       left = 0, right = 657;
32       stab_binsearch(stabs, &left;, &right;, N_SO, 0xf0100184);
33  will exit setting left = 118, right = 554.
34  /
35 static void
36 stab_binsearch(const struct stab stabs, int region_left, int region_right,
37            int type, uintptr_t addr) {
38     ……
39 }
40 /
41   debuginfo_eip - Fill in the @info structure with information about
42   the specified instruction address, @addr.  Returns 0 if information
43   was found, and negative if not.  But even if it returns negative it
44   has stored some information into 'info'.
45   /
46 int
47 debuginfo_eip(uintptr_t addr, struct eipdebuginfo info) {
48 …..
49 }
50 /
51   print_kerninfo - print the information about kernel, including the location
52   of kernel entry, the start addresses of data and text segements, the start
53   address of free memory and how many memory that kernel has used.
54   /
55 void
56 print_kerninfo(void) {
57     extern char etext[], edata[], end[], kern_init[];
58     cprintf("Special kernel symbols:
59 ");
60     cprintf("  entry  0x%08x (phys)
61 ", kern_init);
62     cprintf("  etext  0x%08x (phys)
63 ", etext);
64     cprintf("  edata  0x%08x (phys)
65 ", edata);
66     cprintf("  end    0x%08x (phys)
67 ", end);
68     cprintf("Kernel executable memory footprint: %dKB
69 ", (end - kern_init + 1023)/1024);
```

```
    }
    /
      print_debuginfo - read and print the stat information for the address @eip,
      and info.eip_fn_addr should be the first address of the related function.
      /
    void
    print_debuginfo(uintptr_t eip) {
        struct eipdebuginfo info;
        if (debuginfo_eip(eip, &info;) != 0) {
            cprintf("    : -- 0x%08x --
", eip);
        }
        else {
            char fnname[256];
            int j;
            for (j = 0; j < info.eip_fn_namelen; j ++) {
                fnname[j] = info.eip_fn_name[j];
            }
            fnname[j] = '\u0000';
            cprintf("    %s:%d: %s+%d
", info.eip_file, info.eip_line,
                    fnname, eip - info.eip_fn_addr);
        }
    }
    static __noinline uint32_t
    read_eip(void) {
        uint32_t eip;
        asm volatile("movl 4(%%ebp), %0" : "=r" (eip));
        return eip;
    }
    /
      print_stackframe - print a list of the saved eip values from the nested 'call'
      instructions that led to the current point of execution

      The x86 stack pointer, namely esp, points to the lowest location on the stack
      that is currently in use. Everything below that location in stack is free. Pushing
      a value onto the stack will invole decreasing the stack pointer and then writing
      the value to the place that stack pointer pointes to. And popping a value do the
      opposite.

      The ebp (base pointer) register, in contrast, is associated with the stack
      primarily by software convention. On entry to a C function, the function's
      prologue code normally saves the previous function's base pointer by pushing
      it onto the stack, and then copies the current esp value into ebp for the duration
      of the function. If all the functions in a program obey this convention,
      then at any given point during the program's execution, it is possible to trace
      back through the stack by following the chain of saved ebp pointers and determining
      exactly what nested sequence of function calls caused this particular point in the
      program to be reached. This capability can be particularly useful, for example,
      when a particular function causes an assert failure or panic because bad arguments
      were passed to it, but you aren't sure who passed the bad arguments. A stack
      backtrace lets you find the offending function.

      The inline function read_ebp() can tell us the value of current ebp. And the
      non-inline function read_eip() is useful, it can read the value of current eip,
      since while calling this function, read_eip() can read the caller's eip from
      stack easily.

      In print_debuginfo(), the function debuginfo_eip() can get enough information about
      calling-chain. Finally print_stackframe() will trace and print them for debugging.

      Note that, the length of ebp-chain is limited. In boot/bootasm.S, before jumping
      to the kernel entry, the value of ebp has been set to zero, that's the boundary.
      /
    void
    print_stackframe(void) {
         / LAB1 YOUR CODE : STEP 1 /
        / (1) call read_ebp() to get the value of ebp. the type is (uint32_t);
          (2) call read_eip() to get the value of eip. the type is (uint32_t);
          (3) from 0 .. STACKFRAME_DEPTH
          (3.1) printf value of ebp, eip
          (3.2) (uint32_t)calling arguments [0..4] = the contents in address (unit32_t)ebp +2 [0..4]
          (3.3) cprintf("
");
          (3.4) call print_debuginfo(eip-1)to print the C calling function name and line number, etc.
          (3.5) popup a calling stackframe
                    NOTICE: the calling funciton's return addr eip  = ss:[ebp+4]
                            the calling funciton's ebp = ss:[ebp]
         /
    }
    ```
```

- [x]
```

知识点:启动和中断处理实验
出处:网络
难度:1

```
 void
 print_stackframe(void) {
      / LAB1 YOUR CODE : STEP 1 /
      / (1) call read_ebp() to get the value of ebp. the type is (uint32_t);
        (2) call read_eip() to get the value of eip. the type is (uint32_t);
        (3) from 0 .. STACKFRAME_DEPTH
           (3.1) printf value of ebp, eip
           (3.2) (uint32_t)calling arguments [0..4] = the contents in address (unit32_t)ebp +2
[0..4]
           (3.3) cprintf("
");
           (3.4) call print_debuginfo(eip-1) to print the C calling function name and line number,
etc.
           (3.5) popup a calling stackframe
                  NOTICE: the calling funciton's return addr eip  = ss:[ebp+4]
                          the calling funciton's ebp = ss:[ebp]
       /
     uint32_t ebp = read_ebp(), eip = read_eip();
     int i, j;
     for (i = 0; ebp != 0 && i < STACKFRAME_DEPTH; i ++) {
         cprintf("ebp:0x%08x eip:0x%08x args:", ebp, eip);
         uint32_t args = (uint32_t )ebp + 2;
         for (j = 0; j < 4; j ++) {
             cprintf("0x%08x ", args[j]);
         }
         cprintf("
");
         print_debuginfo(eip - 1);
         eip = ((uint32_t )ebp)[1];
         ebp = ((uint32_t )ebp)[0];
     }
 }
```

4
(20140410-2-期中考试试题v4a答案.docx)中断
（15分）中断（Interrupt）是操作系统为处理意外事件而提供的一种响应机制，中断可分为硬件中断（Hardware interrupt）和软件中断（software interrupt）。中断响应需要硬件和软件的协调合作来完成。在虚拟机中的中断响应需要宿主机（Host
OS）、虚拟机（Guest OS）和硬件的协调合作来完成。试回答下面问题。
1) 描述硬件中断、软件中断和系统调用（system call）的区别。
2) 简要描述外部中断的响应处理过程，并说明各处理操作的执行者。
3) 简要描述虚拟机中客户操作系统对硬件中断的响应处理过程。

- [x]

知识点:中断、异常与系统调用
出处:网络
难度:1

4
(操作系统之PV金典)生产者一消费者问题 (producer-consumer problem)，也称有限缓冲问题 (Bounded-buffer problem)，是指若干进程通过有限的共享缓冲区交换数据时的缓冲区资源使用问题。
假设“生产者”进程不断向共享缓冲区写人数据(即生产数据)，而“消费者”进程不断从共享缓冲区读出数据(即消费数据)；共享缓冲区共有n个；任何时刻
只能有一个进程可对共享缓冲区进行操作。所有生产者和消费者之间要协调，以完成对共享缓冲区的操作。

- [x]

知识点:信号量
出处:网络
难度:1

Semaphore 方法
可把共享缓冲区中的n个缓冲块视为共享资源，生产者写人数据的缓冲块成为消费者可用资源，而消费者读出数据后的缓冲块成为生产者的可用资源。为此，可设
置三个信号量：itemCounter、vacancyCounter和mutex。其中：
itemCounter表示有数据的缓冲块数目，初值是0;
vacancyCounter表示空的缓冲块数初值是n;
mutex用于访问缓冲区时的互斥，初值是1。
producer 伪码

```
procedure producer() {
    while (true) {
        item = produceItem();
        vacancyCounter->P();
            mutex->P();
                Add item to buffer;
            mutex->V();
        itemCounter->V();
    }
}
```

consumer 伪码

```
procedure consumer() {
    while (true) {
        itemCounter->P();
            mutex->P();
                Remove from buffer;
            mutex->V();
        vacancyCounter->V();
    }
}
```

Monitor 方法
设置一个管程，内有两个condition variable：notFull和notEmpty。其中，notFull表示缓存满，notEmpty表示缓存空
producer 伪码

```
procedure producer() {
    lock->Acquire();
    while (count == n)
        notFull.Wait(&lock;);
    Add c to the buffer;
    count++;
    notEmpty.Signal();
    lock->Release();
}
```

consumer 伪码

```
procedure consumer() {
    lock->Acquire();
    while (count == 0)
        notEmpty.Wait(&lock;);
    Remove c from buffer;
    count--;
    notFull.Signal();
    lock->Release();
}
```

4
(操作系统之PV金典)有一个许多进程共享的数据区，有一些只读这个数据区的进程(reader)和一些只往数据区中写数据的进程(writer)；此外还必须满足以
下条件
任意多的读进程可以同时读这个文件
一次只有一个写进程可以往文件中写
如果一个写进程正在往文件中写时，则禁止任何读进程和其他写进程。
读者写者问题又分为"读者优先"和"写者优先"
读者优先：要求指一个读者试图进行读操作时，如果这时正有其他读者在进行操作，他可以直接开始读操作，直到某个时刻没有任何读者。读者之间不互斥，写者之间互斥，只能一个写，可以多个读；读者写者之间互斥，有写者则不能有读者。所以只需要当前第一个读者和写者竞争，竞争成功则后面的读者因为已经有读者在读，可以直接读。
写者优先：一个读者试图进行读操作时，如果有其他写者在等待进行写操作或者正在进行写操作，他要等待写者完成写操作后才开始读操作

- [x]

知识点:信号量
出处:网络
难度:1

信号量实现：
读者优先：两个信号量sem_wsem和sem_x。信号量sem_wsem用于实施互斥，只要一个写进程正在访问共享数据区，其他的写进程和读进程都不能访问它。读进程也使用sem_wsem实施互斥，但是只需要第一个读进程在sem_wsem上等待;全局变量readcount用于记录读进程的数目,信号量sem_x用于确保readcount被正确更新。
写者优先：除了上述两个sem_wsem和sem_x两个信号量外，增加三个新的信号量:sem_rsem, sem_y, sem_z。sem_rsem用于当至少有一个写进程准备访问数据区时，禁止其他所有的读进程；sem_y用于控制writecount被正确更新；sem_z用于读者竞争sem_rsem失败后，后续读者在此信号上排队。
管程实现：
两种问题实现方法基本类似，以读者优先为例：定义条件变量r表示可以对缓冲区读，条件变量w表示可以对缓冲区写;布尔类型变量IsWriting表示当前有写者进程在
缓冲区写数据；整型变量read_count表示读数据的个数；
4
(操作系统之PV金典)5个哲学家围绕一张圆桌而坐，桌子上放着5支筷子，每两个哲学家之间放一支；哲学家的动作包括思考和进餐，

> 进餐时需要同时拿起他左边和右边的筷
> 子，思考时则同时将两支筷子放回原处。如何保证哲学家们的动作有序进行?

- [x]

知识点:信号量
出处:网络
难度:1

> 信号量实现：
> 每个哲学家都有一个信号量与之对应，同时有一个实现临界区互斥的信号量，还有一个状态数组来标记每个哲学家的当前状态：思考、饥饿或者吃面。每次一个哲学家想要吃面时
> ，首先进入互斥区，不让其他哲学家进入，然后标记为饥饿状态；接着检查两边的筷子是否可用，如果可用的话就标记为吃面状态，然后把自己的信号量加一，开始吃面；吃完面
> 后离开互斥区，然后把自己的信号量减一。当然如果拿不到筷子，那么直接离开互斥区，然后把自己的信号量减一，进入阻塞状态。当吃完面后，哲学家会放回自己的筷子，这时
> 仍需要进入互斥区，改为思考状态，然后检查两边的哲学家是否还在饥饿，如果有的哲学家还在饥饿中而且筷子可用，那么就让该哲学家修改自己的信号量变为可执行，等待当前
> 哲学家离开互斥区后再执行。
> 管程实现：
> 管程与信号量类似，它实现了信号量的封装。monitor的成员变量cv会对每个哲学家建立一个信号量，mutex信号量是一个二值信号量，每次只允许一个进程进入管
> 程,确保了互斥访问性质。next保存的是因为唤醒其他进程而进入睡眠状态的进程，next_count保存next链表的长度。信号量sem用于让发出wait(程
> 序中为down)操作的等待某个条件的为真的进程睡眠，而让发出signal(程序中为up)的进程来唤醒睡眠进程。count表示等待在这个条件上的睡眠进程个数，
> owner表示此条件变量宿主是哪个管程。
> 4
> (操作系统之PV金典)理发店理有一位理发师、一把理发椅和n把供等候理发的顾客坐的椅子。如果没有顾客，理发师便在理发椅上睡觉一个顾客到来时，它必须叫醒理发师，
> 如果理发师正在理发时又有顾客来到，则如果有空椅子可坐，就坐下来等待，否则就离开。
> 要求：
> 1. 每个顾客进入理发室后，即时显示“Entered” 及其线程标识，还同时显示理发室共有几名顾客及其所坐的位置；
> 2. 至少有10个顾客，每人理发至少3秒钟；
> 3. 多个顾客须共享操作函数代码

- [x]

知识点:信号量
出处:网络
难度:1

> 总体设计：需要两类进程Barber ()和Customer()分别描述理发师和顾客的行为。当理发师睡觉时顾客进来需要唤醒理发师为其理发，当有顾客时理发师为其
> 理发，没有的时候理发师睡觉。因此理发师和顾客之间是同步的关系，由于每次理发师只能为一个人理发，且可供等侯的椅子有限只有n个，即理发师和椅子是临界资源，所以顾
> 客之间是互斥的关系。
> 信号量及控制变量：引入三个信号量和一个控制量，如下： 控制变量waiting用来记录等候理发的顾客数，初值均为0；
> 信号量customers用来记录等候理发的顾客数，并用作阻塞理发师进程，初值为0；
> 信号量barbers用来记录正在等候顾客的理发师数，并用作阻塞顾客进程，初值为0； 信号量mutex用于互斥，初值为1。
> 椅子定义为5个，用waitingID[0-4]来表示，用两个变量first和last来记录下一个理发的顾客和下一个空闲椅子(目前为止最后一个理发的顾客)。
> 三个随机函数flat，normal，bursty是用来控制顾客到来的随机情况的，这更符合实际生活中的实际情况。根据个人理解，flat和normal情况下顾客
> 到来的时间比较平均，而bursty随机函数下顾客会比较集中地到来，然后一段时间空闲，又会来比较密集的一批。
> 时间设定：本次实验中，我设定的程序运行总时间为10s，理发师理一次发的时间为4s,故在程序运行期间内，顾客到来的时间因为由随机数控制，所以不同次运行程序所得
> 到的顾客数及到达理发店的时间都不定，但是平均下来程序一次运行中共可到达约20位顾客，即平均1人/0.5s。并且在每位顾客到达以及理发师开始理发时都会输出系统
> 时间来进行对比观察 .
> 4
> (操作系统之PV金典)三个吸烟者在一间房间内，还有一个香烟供应者。为了制造并抽掉香烟，每个吸烟者需要三样东西：烟草、纸和火柴。供应者有丰富的货物提供。三个吸
> 烟者中，第一个有自己的烟草，第二个有自己的纸，第三个有自己的火柴。供应者将两样东西放在桌子上，允许一个吸烟者进行对健康不利的吸烟。当吸烟者完成吸烟后唤醒供应
> 者，供应者再放两样东西（随机地）在桌面上，然后唤醒另一个吸烟者。试为吸烟者和供应者编写程序解决问题。

- [x]

知识点:信号量
出处:网络
难度:1

> 用信号量和P、V操作：
> 一个供应者和三个吸烟者各有一个信号量，供应者的信号量初始化为1，吸烟者为0。他们各起一个进程，对于供应者进程，若其信号量为1则随机提供两种材料，并把供应者信
> 号量置0，需要该材料的吸烟者信号量置1，唤醒该吸烟者进程；否则，供应者进程阻塞。对于吸烟者进程，若其信号量为1则表明该吸烟者得到了想要的材料，可以制烟并吸烟
> ，然后要将该吸烟者信号量置0，供应者信号量置1，唤醒供应者进程。
> 用管程：
> 一个供应者和三个吸烟者各有一个条件量，然后用一标志位flag来标记桌上是否有物品，初始化为0。对于供应者进程，如果桌上有物品，则等待；否则，供应两样物品，置

> flag为1，并唤醒需要该材料的吸烟者进程；对于吸烟者进程，如果桌上无物品，则等待；否则，吸烟者吸烟，置flag为0，再唤醒供应者进程。
> 4
> (操作系统之PV金典)设有一个可以装A、B两种物品的仓库，其容量无限大，但要求仓库中A、B两种物品的数量满足下述不等式： -M≤A物品数量-B物品数量≤N
> 其中M和N为正整数。试用信号量和PV操作描述A、B两种物品的入库过程。

- [x]

知识点:信号量
出处:网络
难度:1

> Semaphore：一共需要2个Semaphore，A、B各一个，表示A、B在差值满足要求的情况下各还可以放入多少个。同时用depot保证往仓库里面放置物品
> 是互斥的。
> Monitor：首先用1个变量表示A与B的差值，然后判断A-
> B是否满足-M<=A-B<=N。在达到右临界值时就开始等待条件变量，在B往仓库里面加的时候就发送signal。
> 4
> (操作系统之PV金典)设有一个可以装A、B两种物品的仓库,其容量有限(分别为N),但要求仓库中A、B两种物品的数量满足下述不等式: -M≤A物品数量-
> B物品数量≤N 其中M和N为正整数。另外,还有一个进程消费A,B,一次取一个A,B组装成C。 试用信号量和PV操作描述A、B两种物品的入库过程。

- [x]

知识点:信号量
出处:网络
难度:1

```
semaphore mutex=1,a,empty1=m,b,empty2=N,full1,full2=0;
cobegin
    process(A);
    process(B);
    process(C)
coend
// A物品入库
process A
begin
    while(TRUE)
    begin
        p(empty1);
        P(a);
        p(mutex);
        A物品入库;
        v(mutex);
        V(b);
        v(full1);
    end
end
// B物品入库:
process B
begin
    while(TRUE)
    begin
        p(empty2);
        P(b);
        p(mutex);
        B物品入库;
        v(mutex);
        V(a);
        p(full2);
    end
end
// process C
begin
    while(TRUE)
    begin
        p(full1);
        p(full2);
        p(a);
        P(b);
        组装;
        V(a);
        v(b);
        v(empty1);
        v(empty2);
    end
end
```

> 4
> "1）试说明硬中断（hardware interrupt）、异常（exception）和系统调用（system call）的相同点和不同点。
> 2）下面代码完成在进入trap()函数前的准备工作。其中pushal完成包括esp在内的CPU寄存器压栈。试说明“pushl %esp”的作用是什么?
> ==========trapentry.S (kern\trap)========== #include # vectors.S sends

> all traps here. .text .globl **alltraps** alltraps: # push registers to build a trap frame # therefore make the stack look like a struct trapframe pushl %ds pushl %es pushl %fs pushl %gs pushal # load GD_KDATA into %ds and %es to set up data segments for kernel movl Missing open brace for subscript 0x8, %esp iret ==========Trap.c (kern\trap)========== ..... / trap - handles or dispatches an exception/interrupt. if and when trap() returns, the code in kern/trap/trapentry.S restores the old CPU state saved in the trapframe and then uses the iret instruction to return from the exception. / void trap(struct trapframe tf) { // dispatch based on what type of trap occurred trap_dispatch(tf); } ....."

- [x]

知识点:中断、异常与系统调用
出处:网络
难度:1

> Hardware interrupt Interruption based on an external hardware event external to the CPU An interrupt is generally initiated by an I/O device, and causes the CPU to stop what it's doing Exception an exceptional condition in the processor (Illed program) an interrupt that is caused by software (by executing an instruction) System call a programmer initiated in user mode and expected transfer of control to the kernel an interrupt that is caused by software (by executing an instruction) 共同：中断当前执行/保存现场（3分） 不同：产生原因(每个2分) 2)3分 给trap函数传参数，汇编调用C时如何传参。 # push %esp to pass a pointer to the trapframe as an argument to trap() pushl %esp # call trap(tf), where tf=%esp call trap 4
> "1）系统调用的参数传递有几种方式？各有什么特点?
> 2）sys_exec是一个加载和执行指定可执行文件的系统调用。请说明在下面的ucore实现中，它的三个参数分别是以什么方式传递的。
> ==========Proc.c (kern\process)========== ..... // do_execve - call exit_mmap(mm)&pug;_pgdir(mm) to reclaim memory space of current process //

- call load_icode to setup new memory space accroding binary prog. int do_execve(const char name, int argc, const char argv) { static_assert(EXEC_MAX_ARG_LEN >= FS_MAX_FPATH_LEN); struct mm_struct mm = current->mm; if (!(argc >= 1 && argc <= EXEC_MAX_ARG_NUM)) { return -E_INVAL; } char local_name[PROC_NAME_LEN + 1]; memset(local_name, 0, sizeof(local_name)); char kargv[EXEC_MAX_ARG_NUM]; const char path; int ret = -E_INVAL; lock_mm(mm); if (name == NULL) { snprintf(local_name, sizeof(local_name), " %d", current->pid); } else { if (!copy_string(mm, local_name, name, sizeof(local_name))) { unlock_mm(mm); return ret; } } if ((ret = copy_kargv(mm, argc, kargv, argv)) != 0) { unlock_mm(mm); return ret; } path = argv[0]; unlock_mm(mm); files_closeall(current->filesp); / sysfile_open will check the first argument path, thus we have to use a user-space pointer, and argv[0] may be incorrect / int fd; if ((ret = fd = sysfile_open(path, O_RDONLY)) < 0) { goto execve_exit; } if (mm != NULL) { lcr3(boot_cr3); if (mm_count_dec(mm) == 0) { exit_mmap(mm); put_pgdir(mm); mm_destroy(mm); } current->mm = NULL; } ret= -E_NO_MEM;; if ((ret = load_icode(fd, argc, kargv)) != 0) { goto execve_exit; } put_kargv(argc, kargv); set_proc_name(current, local_name); return 0; execve_exit: put_kargv(argc, kargv); do_exit(ret); panic("already exit: %e. ", ret); } ..... ==========Syscall.c (kern\syscall)========== ..... static int sys_exec(uint32_t arg[]) { const char name = (const char)arg[0]; int argc = (int)arg[1]; const char argv = (const char)arg[2]; return do_execve(name, argc, argv); } ..... static int (syscalls[])(uint32_t arg[]) = { [SYS_exit] sys_exit, [SYS_fork] sys_fork, [SYS_wait] sys_wait, [SYS_exec] sys_exec, [SYS_yield] sys_yield, [SYS_kill] sys_kill, [SYS_getpid] sys_getpid, [SYS_putc] sys_putc, [SYS_pgdir] sys_pgdir, }; #define NUM_SYSCALLS ((sizeof(syscalls)) / (sizeof(syscalls[0]))) void syscall(void) { struct trapframe tf = current->tf; uint32_t arg[5]; int num = tf->tf_regs.reg_eax; if (num >= 0 && num < NUM_SYSCALLS) { if (syscalls[num] != NULL) { arg[0] = tf->tf_regs.reg_edx; arg[1] = tf->tf_regs.reg_ecx; arg[2] = tf->tf_regs.reg_ebx; arg[3] = tf->tf_regs.reg_edi; arg[4] = tf->tf_regs.reg_esi; tf->tf_regs.reg_eax = syscallsnum; return ; } } print_trapframe(tf); panic("undefined syscall %d, pid = %d, name = %s. ", num, current->pid, current->name); } ..... ==========libs-user-ucore/syscall.c========== ..... int sys_exec(const char filename, const char argv, const char envp) { return syscall(SYS_exec, filename, argv, envp); } ..... ==========libs-user-ucore/arch/i386/syscall.c========== ..... uint32_t syscall(int num, ...) { va_list ap; va_start(ap, num); uint32_t a[MAX_ARGS]; int i; for (i = 0; i < MAX_ARGS; i++) { a[i] = va_arg(ap, uint32_t); } va_end(ap); uint32_t ret; asm volatile ("int %1;":"=a" (ret) :"i"(T_SYSCALL), "a"(num), "d"(a[0]), "c"(a[1]), "b"(a[2]), "D"(a[3]), "S"(a[4]) :"cc", "memory"); return ret; }"
- [x]

知识点:中断、异常与系统调用
出处:网络
难度:1

> 1）Three general methods used to pass parameters to the OS Simplest: pass the parameters in registers. （2分） In some cases, may be more parameters than registers（2分） Parameters stored in a block, or table, in memory（2分）, and address of block passed as a parameter in a register（1分） Parameters placed, or pushed, onto the stack（2分） by the program and popped off the stack by the operating system (ucore methord) Block and stack methods do not limit the number or length（1分） of parameters being passed 2）三个参数都是通过堆栈来传递的。（3分）
> 从用户态到内核态时参数是在寄存器中的； name可理解为是在内存块中来传递的； 综合而言，言之有理即可。
> 4
> "1）描述伙伴系统（Buddy System）中对物理内存的分配和回收过程。2）假定一个操作系统内核中由伙伴系统管理的物理内存有1MB，试描述按下面顺序进行
> 物理内存分配和回收过程中，每次分配完成后的分配区域的首地址和大小，或每次回收完成后的空闲区域队列（要求说明，每个空闲块的首地址和大小）。建议给出分配和回收的
> 中间过程。 a) 进程A申请50KB； b) 进程B申请100KB； c) 进程C申请40KB； d) 进程D申请70KB； e) 进程B释放100KB；
> f) 进程E申请127KB； g) 进程D释放70KB； h) 进程A释放50KB； i) 进程E释放127KB； j) 进程C释放40KB；"

- [x]

知识点:连续内存分配
出处:网络
难度:1

> 整个空间被分成2U大小；（2分） 分配：找到2U大小的块， 满足2U-1 < s <= 2U（2分） 如果比它大，就划分成两个等大小的块（2分） 释放：相邻且大小相同2U-1的两块中第一块起始地址为2U倍数（2分）时，合并（2分）； a) 进程A申请50KB；
> Addr:0,Size:64KB b) 进程B申请100KB；Addr:128K,Size:128KB c) 进程C申请40KB；Addr:64K,Size:64KB d) 进程D申请70KB；
> Addr:256K,Size:128KB e) 进程B释放100KB；Addr:128K,Size:128KB f) 进程E申请127KB；Addr:128K,Size:128KB g) 进程D释放
> 70KB；Addr:256K,Size:512KB h) 进程A释放50KB；Addr:0,Size:64KB i) 进程E释放127KB；Addr:128K,Size:128KB j) 进程C释放
> 40KB；Addr:0,Size:1024KB
> 4
> "1）试用图示描述32位X86系统在采用4KB页面大小时的虚拟地址结构和地址转换过程。2）在采用4KB页面大小的32位X86的ucore虚拟存储系统中，进程
> 页面的起始地址由宏VPT确定。 #define VPT 0x0D000000
> 请计算：2a)试给出页目录中自映射页表项的虚拟地址；2b)虚拟地址0X87654321对应的页目录项和页表项的虚拟地址。"

- [x]

知识点:非连续内存分配
出处:网络
难度:1

> 1）（12分）地址划分：10 + 10 + 12（6分）
> 地址转换过程关键点：两级页面（2分）、缺页处理（2分）（分配物理页面、更新页表项、重新访问）（有一个就给2分） 2a）（4分） 自映射页表项地址4分
> 每个地址3分，每个地址中的三段，二进制每段1分；（结果对了，就给全分） 0D00 0000 0000 1101 0000 0000 0000 0000
> 0000 0000 0000 1101 0000 0011 0100 0000 1101 0000 0X0D0340D0 2b）
> 虚拟地址0X87654321对应的页目录项和页表项的虚拟地址（4分,每个2分，二进制对，就给全分） 87654321 1000 0111 0110 0101
> 0100 0011 0010 0001 PDE: 0000 1101 0000 0011 0100 1000 0111 01 00 0X0D034874
> PTE: 0000 1101 00 10 00 01 11 01 10 01 01 01 00 00 0X0D21 D950
> 4
> "试描述FIFO页面替换算法的基本原理，并swap_fifo.c中未完成FIFA页面替换算法实验函数map_swappable()和swap_out_vic
> tim()。 ==========Defs.h (libs)========== / to_struct - get the
> struct from a ptr @ptr: a struct pointer of member @type: the type of the
> struct this is embedded in @member: the name of the member within the struct
> / #define to_struct(ptr, type, member) \ ((type)((char)(ptr) -
> offsetof(type, member))) ==========Memlayout.h (kern\mm)========== //
> convert list entry to page #define le2page(le, member) \ to_struct((le),
> struct Page, member) ==========List.h (libs)========== #ifndef
> **LIBS_LIST_H** #define **LIBS_LIST_H** #ifndef **ASSEMBLER** #include /
> Simple doubly linked list implementation. Some of the internal functions
> ("**xxx**") **are useful when manipulating whole lists rather than single
> entries, as sometimes we already know the next/prev entries and we can
> generate better code by using them directly rather than using the generic
> single-entry routines. / struct list_entry { struct list_entry prev,
> next; }; typedef struct list_entry list_entry_t; static inline void
> list_init(list_entry_t elm)** attribute**((always_inline)); static inline
> void list_add(list_entry_t listelm, list_entry_t elm)**
> attribute**((always_inline)); static inline void
> list_add_before(list_entry_t listelm, list_entry_t elm)**
> attribute**((always_inline)); static inline void list_add_after(list_entry_t
> listelm, list_entry_t elm)** attribute**((always_inline)); static inline
> void list_del(list_entry_t listelm)** attribute**((always_inline)); static
> inline void list_del_init(list_entry_t listelm)**
> attribute**((always_inline)); static inline bool list_empty(list_entry_t
> list)** attribute**((always_inline)); static inline list_entry_t
> list_next(list_entry_t listelm)** attribute**((always_inline)); static
> inline list_entry_t list_prev(list_entry_t listelm)**
> attribute**((always_inline)); static inline void** list_add(list_entry_t
> elm, list_entry_t prev, list_entry_t next) **attribute**((always_inline));
> static inline void **list_del(list_entry_t prev, list_entry_t next)**
> attribute__((always_inline)); / list_init - initialize a new entry

@elm: new entry to be initialized / static inline void list_init(list_entry_t elm) { elm->prev = elm->next = elm; } / list_add

- add a new entry @listelm: list head to add after @elm: new entry to be added Insert the new element @elm after the element @listelm which is already in the list. / static inline void list_add(list_entry_t listelm, list_entry_t elm) { list_add_after(listelm, elm); } / list_add_before - add a new entry @listelm: list head to add before @elm: new entry to be added Insert the new element @elm before the element @listelm which is already in the list. / static inline void list_add_before(list_entry_t listelm, list_entry_t elm) { **list_add(elm, listelm->prev, listelm); } / list_add_after - add a new entry @listelm: list head to add after @elm: new entry to be added Insert the new element @elm after the element @listelm which is already in the list. / static inline void list_add_after(list_entry_t listelm, list_entry_t elm) {** list_add(elm, listelm, listelm->next); } / list_del - deletes entry from list @listelm: the element to delete from the list Note: list_empty() on @listelm does not return true after this, the entry is in an undefined state. / static inline void list_del(list_entry_t listelm) { **list_del(listelm->prev, listelm->next); } / list_del_init - deletes entry from list and reinitialize it. @listelm: the element to delete from the list. Note: list_empty() on @listelm returns true after this. / static inline void list_del_init(list_entry_t listelm) { list_del(listelm); list_init(listelm); } / list_empty - tests whether a list is empty @list: the list to test. / static inline bool list_empty(list_entry_t list) { return list->next == list; } / list_next - get the next entry @listelm: the list head / static inline list_entry_t list_next(list_entry_t listelm) { return listelm->next; } / list_prev - get the previous entry @listelm: the list head / static inline list_entry_t list_prev(list_entry_t listelm) { return listelm->prev; } / Insert a new entry between two known consecutive entries. This is only for internal list manipulation where we know the prev/next entries already! / static inline void** list_add(list_entry_t elm, list_entry_t prev, list_entry_t next) { prev->next = next->prev = elm; elm->next = next; elm->prev = prev; } / Delete a list entry by making the prev/next entries point to each other. This is only for internal list manipulation where we know the prev/next entries already! / static inline void **list_del(list_entry_t prev, list_entry_t next) { prev->next = next; next->prev = prev; } #endif / !**ASSEMBLER_ */ #endif / !***LIBS_LIST_H*** */ ========== *Swap_fifo.c (kern\mm)========== #include #include #include #include #include #include #include / [wikipedia]The simplest Page Replacement Algorithm(PRA) is a FIFO algorithm. (1) Prepare: In order to implement FIFO PRA, we should manage all swappable pages, so we can link these pages into pra_list_head according the time order. At first you should be familiar to the struct list in list.h. struct list is a simple doubly linked list implementation. You should know howto USE: list_init, list_add(list_add_after), list_add_before, list_del, list_next, list_prev. Another tricky method is to transform a general list struct to a special struct (such as struct page). You can find some MACRO: le2page (in memlayout.h), (in future labs: le2vma (in vmm.h), le2proc (in proc.h),etc. / list_entry_t pra_list_head; / (2) _fifo_init_mm: init pra_list_head and let mm->sm_priv point to the addr of pra_list_head. Now, From the memory control struct mm_struct, we can access FIFO PRA / static int _fifo_init_mm(struct mm_struct mm) { list_init(&pra;*list_head); mm->sm_priv = &pra;*list_head; //cprintf(" mm->sm_priv %x in fifo_init_mm ",mm->sm_priv); return 0; } / (3)*fifo_map_swappable: According FIFO PRA, we should link the most recent arrival page at the back of pra_list_head qeueue / static int *fifo_map_swappable(struct mm_struct mm, uintptr_t addr, struct Page page, int swap_in) { list_entry_t head=(list_entry_t) mm->sm_priv; list_entry_t entry=&(page->pra_page_link); assert(entry != NULL && head != NULL); //record the page access situlation /LAB3 EXERCISE 2: YOUR CODE/ //(1)link the most recent arrival page at the back of the pra_list_head qeueue. ===Your code 2=== return 0; } / (4)*fifo_swap_out_victim: According FIFO PRA, we should unlink the earliest arrival page in front of pra_list_head qeueue, then set the addr of addr of this page to ptr_page. / static int *fifo_swap_out_victim(struct mm_struct mm, struct Page ptr_page, int in_tick) { list_entry_t head=(list_entry_t) mm->sm_priv; assert(head != NULL); assert(in_tick==0); / Select the victim / /LAB3 EXERCISE 2: YOUR CODE/ //(1) unlink the earliest arrival page in front of pra_list_head qeueue //(2) set the addr of addr of this page to ptr_page / Select the tail / ===Your code 3=== return 0; } static int _fifo_check_swap(void) { cprintf("write Virt Page c in fifo_check_swap "); (unsigned char)0x3000 = 0x0c; assert(pgfault_num==4); cprintf("write Virt Page a in fifo_check_swap "); (unsigned char)0x1000 = 0x0a; assert(pgfault_num==4); cprintf("write Virt Page d in*

> *fifo_check_swap*
> *"); (unsigned char)0x4000 = 0x0d; assert(pgfault_num==4);*
> *cprintf("write Virt Page b in fifo_check_swap*
> *"); (unsigned char)0x2000 =*
> *0x0b; assert(pgfault_num==4); cprintf("write Virt Page e in*
> *fifo_check_swap*
> *"); (unsigned char)0x5000 = 0x0e; assert(pgfault_num==5);*
> *cprintf("write Virt Page b in fifo_check_swap*
> *"); (unsigned char)0x2000 =*
> *0x0b; assert(pgfault_num==5); cprintf("write Virt Page a in*
> *fifo_check_swap*
> *"); (unsigned char)0x1000 = 0x0a; assert(pgfault_num==6);*
> *cprintf("write Virt Page b in fifo_check_swap*
> *"); (unsigned char)0x2000 =*
> *0x0b; assert(pgfault_num==7); cprintf("write Virt Page c in*
> *fifo_check_swap*
> *"); (unsigned char)0x3000 = 0x0c; assert(pgfault_num==8);*
> *cprintf("write Virt Page d in fifo_check_swap*
> *"); (unsigned char)0x4000 =*
> *0x0d; assert(pgfault_num==9); return 0; } static int _fifo_init(void) { return*
> *0; } static int _fifo_set_unswappable(struct mm_struct mm, uintptr_t addr) {*
> *return 0; } static int _fifo_tick_event(struct mm_struct mm) { return 0; }*
> *struct swap_manager swap_manager_fifo = { .name = "fifo swap manager", .init =*
> *&*fifo_init, .init_mm = &*fifo_init_mm, .tick_event = &*fifo_tick_event,
> .map_swappable = &*fifo_map_swappable, .set_unswappable =*
> *&*fifo_set_unswappable, .swap_out_victim = &*fifo_swap_out_victim, .check_swap*
> *=* &fifo_check_swap, }; "

- [x]

知识点:置换算法
出处:网络
难度:1

> 算法：（4分）占用页面按置换时间先后排序；缺页时置换最先进入内存的页面； 实现： map_swappable() //record the page
> access situlation /LAB3 EXERCISE 2: YOUR CODE/ //(1)link the most recent
> arrival page at the back of the pra_list_head qeueue. list_add(head, entry);//
> （3分） swap_out_victim() / Select the victim / /LAB3 EXERCISE 2: YOUR CODE/
> //(1) unlink the earliest arrival page in front of pra_list_head qeueue //(2)
> set the addr of addr of this page to ptr_page / Select the tail /
> list_entry_t le = head->prev; // 找到链表尾（2分） assert(head!=le); struct Page p =
> le2page(le, pra_page_link); //找到物理页面数据结构，并保存 （2分） list_del(le); //
> 从链表中取出页面（2分） assert(p !=NULL); ptr_page = p; //返回被置换的物理页面数据结构指针 （2分） return
> 0;
> 4
> "描述int fork(void)系统调用的功能和接口，给出程序fork.c的输出结果，并用图示给出所有进程的父子关系。注：1）getpid()和
> getpp
> id()是两个系统调用，分别返回本进程标识和父进程标识。2）你可以假定每次新进程创建时生成的进程标识是顺序加1得到的；在进程
> 标识为1000的命令解释程序sh
> ell中启动该程序的执行。 #include #include / getpid() and fork() are system calls
> declared in unistd.h. They return / / values of type pid_t. This pid_t is a
> special type for process ids. / / It's equivalent to int. / int main(void)
> { pid_t childpid; int x = 5; int i; childpid = fork(); for (i = 0; i < 3;
> i++) { printf("This is process %d; childpid = %d; The parent of this process
> has id %d; i = %d; x = %d
> ", getpid(), childpid, getppid(), i, x); sleep(1);
> x++; } return 0; }"

- [x]

知识点:进程状态与控制
出处:网络
难度:1

> 功能：复制当前进程，生成一个子进程（2分），并从当前位置继续执行（2分）； 接口：没有输入，父进程返回子进程标识（2
> 分）； 子进程返回零（2分）； 输出：
> 三次循环（3分）； i的值输出正确（2分）； x的值输出正确（2分）； 父子进程标识正确（2分）； xyong@ubuntu:~/work$ ./a.out
> This is process 13724; childpid = 13725; The parent of this process has id
> 9917; i = 0; x = 5 This is process 13725; childpid = 0; The parent of this
> process has id 13724; i = 0; x = 5 This is process 13724; childpid = 13725;
> The parent of this process has id 9917; i = 1; x = 6 This is process 13725;
> childpid = 0; The parent of this process has id 13724; i = 1; x = 6 This is
> process 13724; childpid = 13725; The parent of this process has id 9917; i =
> 2; x = 7 This is process 13725; childpid = 0; The parent of this process has
> id 13724; i = 2; x = 7 父子关系图：1分
> 4
> "设P,Q,R共享一个缓冲区,P,Q构成一对生产者-消费者,R既为生产者又为消费?者。使用P,V 实现其同步。"

- [x]

知识点:信号量
出处:网络
难度:1

> Semaphore 方法 设置三个信号量：full(itemCounter)、empty(vacancyCounter)和mutex。
> full表示有数据的缓冲块数目，初值是0；empty表示空的缓冲块数初值是n；mutex用于访问缓冲区时的互斥，初值是1。
> 三种进程，consumer,producer,both，both表示既是producer又是consumer。 producer 伪码 while true
> p(empty); P(mutex); produce one; v(mutex); v(full); end while consumer 伪码
> while true p(full); P(mutex); consume one; v(mutex); v(empty); end while both
> 伪码 if empty>=1 then begin p(empty); p(mutex); product one; v(mutex); v(full);
> end if full>=1 then begin p(full); p(mutex); consume one; v(mutex); v(empty);
> end Monitor 方法
> 设置一个monitor，内有两个条件变量：notFull和notEmpty。其中，notFull表示缓存满，notEmpty表示缓存空 producer
> 伪码 lock.Acquire(); while (count == n) notFull.Wait(&lock;); produce one;
> count++; notEmpty.Signal(); end while lock.Release(); consumer 伪码
> lock.Acquire(); while (count == 0); notEmpty.Wait(&lock;); consume one;
> count--; notFull.Signal(); end while lock.Release(); both 伪码 lock.Acquire();
> notEmpty.Wait(&lock;); consume one; count--; notFull.Signal();
> notFull.Wait(&lock;); produce one; count++; notEmpty.Signal();
> lock.Release();
> 4
> "此问题是对读者-写者问题的一个扩展，既如果读者写者均是平等的即二者都不优先情况下。
> 此问题的一个更高的版本是说，每个资源可以同时读取的人的个数也是有限的（限制数RN）。"

- [x]

知识点:信号量
出处:网络
难度:1

> "为了达到公平的目的，即在读者进行读取的时候，如果有写者在排队，后面的读者不能够加入到读取的队列中来，应该等待写者执行完写操作之后再进行读取。
> 针对上面一种情况引入一个排队信号量q,每次有操作必须等待这个信号量释放再进行操作（如果有写操作在排队，q没有释放，下一个读操作没有办法进入并进行读操作）
> 算法流程 q,s, mutex <=1, ReadCount <= 0 Reader: while True: wait(q) wait(mutex) if
> ReadCount ==0 wait(s) ReadCount++ signal(mutex) signal(q) READING.....
> signal(mutex) ReadCount-- if ReadCount==0 signal(s) signal(mutex) end while
> Writer: While True: wait(q) wait(s) WRITING..... singal(s) singal(w)
> 问题二使用一个计数器计算当前还有几个剩下的读者名额，当写者掌控时，直接进行0/RN级别的替换。 代码无需修改。"
> 4
> "有一个许多进程共享的数据区，有一些只读这个数据区的进程(reader)和一些只往数据区中写数据的进程(writer)；此外还需满足如下条件：
> 1.任意多的读进程可以同时读这个文件。 2.一次只有一个写进程可以往文件中写。 3.如果一个写进程正在往文件中写时，则禁止任何读进程和其他写进程。
> 实现基于先来先服务策略的读者 - 写者的问题，具体要求描述如下： 1.存在m个读者和n个写者，共享同一个缓冲区。
> 2.当没有读者在读，写者在写时，读者写者均可进入读或写。 3.当有读者在读时：(1) 写者来了，则写者等待。(2)
> 读者来了，则分两种情况处理：无写者等待，则读者可以直接进入读操作，如果有写者等待，则读者必须依次等待。 4.当有写者在写时，写者或读者来了，均需等待。
> 5.当写者写完后，如果等待队列中第一个是写者，则唤醒该写者；如果等待队列中第一个是读者，则唤醒该队列中从读者开始连续的所有读者。
> 6.当最后一个读者读后，如果有写者在等待，则唤醒第一个等待的写者。"

- [x]

知识点:信号量
出处:网络
难度:1

> 前面的实现方法中可能出现多个写和读同时等待同一个锁打开，一旦锁打开，会随机挑选一个操作执行，但我们知道在写操作之后加入的读操作是不能在写操作之前执行的，所以
> 上述的方法会 有错误产生。 可以考虑建立一个读写操作队列，给队列设置两个队列锁（read锁锁定read操作，write锁锁定write操作），每次挑选队列中
> 最早加入的操作执行，由于数组删除很复杂，所以采用循环数组。以信号量实现为例，管程的实现方法也是对前一位同学的代码做出相应类似的修改即可。贴出主要代码(读写队
> 列操作部分，monitor不再赘述，跟很多人是一样的)： 变量定义 #define OP_NUM 200; //操作队列上限 int op_num = 0;
> //队列当前等待数目 int op_list[OP_NUM]; //等待队列，奇数为读，偶数为写 int start=0;//队首位置 int
> end=-1;//队尾位置 semaphore_t op_sem;//队首和队尾位置,等待数目锁 semaphore_t
> list_read_sem;//队列读互斥锁 semaphore_t list_write_sem;//队列写互斥锁 读操作 int read_op(int
> id){ down(&list;*write_sem);//只锁写操作 cprintf("No.%d Reader is*
> *reading*
> *",i); do_sleep(50); cprintf("No.%d Reader finished reading*
> *",i);*
> *up(&list;*write_sem); cprintf("No.%d Reader Sem Proc Quit
> ",i); return 0;
> } 写操作 int write_op(int id){ down(&list;*write_sem);*
> *down(&list;*read_sem);//同时锁定读写操作 cprintf("No.%d Writer is writing
> ",i);
> do_sleep(50); cprintf("No.%d Writer finished writing
> ",i);
> up(&list;*write_sem); up(&list;*read_sem);//同时解锁 cprintf("No.%d Writer
> Sem Proc Quit

",i); return 0; } 加入操作 int add_op(int id){
down(&op;*sem);//锁定队列信息 if(op_num>OP_NUM) return -1;//队列已满
end=(end+1)%OP_NUM; op_list[end]=id; op_num_sem++; up(&op;*sem); return 0;
} 队列执行操作 int run_op(){ if(op_num==0) return -1;//队列为空
if(op_list[start]%2==1){//读操作 read_op(op_list[start]); } else{
write_op(op_list[start]); } down(&op;*sem);//锁住队列信息
start=(start+1)%OP_NUM; op_num--; up(&op;*sem); return 0; }
4
"在一间酒吧里有三个音乐爱好者队列，第一队的音乐爱好者只有随身听，第二队的只有音乐磁带，第三队只有电池。而要听音乐就必须随身听，音乐磁带和电池这三种物品俱全
。酒吧老板依次出售这三种物品中的任意两种。当一名音乐爱好者得到这三种物品并听完一首乐曲后，酒吧老板才能再一次出售这三种物品中的任意两种。于是第二名音乐爱好者
得到这三种物品，并开始听乐曲。全部买卖就这样进行下去。试用P，V操作正确解决这一买卖。"

- [x]

知识点:信号量
出处:网络
难度:1

```
#include  #include  #include  #include  #include  #define ROUND 10 const char
GOODS[3][20] = { "Walkman", "Tape", "Battery" }; const char WANT[3][20] = {
"Tape&Battery;", "Walkman&Battery;", "Walkman&Tape;" }; int
sema_flag; int condvar_flag; semaphore_t listener[3]; semaphore_t seller;
struct proc_struct listener_sema_proc[3]; struct proc_struct
seller_sema_proc; void listener_sema(void arg){ int i = (int) arg;
while(sema_flag){ down(&listener;[i]); if (sema_flag){ cprintf("No %d
listener has %s, and bought %s. sema
",i,GOODS[i],WANT[i]);
up(&seller;); } } cprintf("No %d listener quit! sema
",i); } void
seller_sema(void arg){ int i; int pos; for(i=0;i<ROUND;i ++){ pos = rand() %
3; cprintf("Iter %d : Seller is selling: %s. sema
",i,WANT[pos]);
up(&listener;[pos]); down(&seller;); } sema_flag = 0; for(i = 0;i <
3;i ++) up(&listener;[i]); cprintf("Seller quit! sema
"); } monitor_t
lmt, mtp2= &lmt; struct proc_struct listener_condvar_proc[3]; struct
proc_struct seller_condvar_proc; void seller_condvar(void arg){ int i; int
pos; for(i = 0;i < ROUND;i ++){ down(&mtp2-;>mutex); pos = rand() % 3;
cprintf("Iter %d : Seller is selling: %s. condvar
",i,WANT[pos]);
cond_signal(&mtp2-;>cv[pos + 1]); cond_wait(&mtp2-;>cv[0]); if
(mtp2->next_count > 0) up(&mtp2-;>next); else up(&mtp2-;>mutex); }
condvar_flag = 0; down(&mtp2-;>mutex); for(i = 0;i < 3;i ++)
cond_signal(&mtp2-;>cv[i + 1]); cprintf("Seller_condvar quit!
"); if
(mtp2->next_count > 0) up(&mtp2-;>next); else up(&mtp2-;>mutex); }
void listener_condvar(void arg){ int num = (int)arg; down(&mtp2-;>mutex);
cprintf("No %d listener is waiting
", num); cond_wait(&mtp2-;>cv[num+1]);
if (mtp2->next_count > 0) up(&mtp2-;>next); else up(&mtp2-;>mutex);
while(condvar_flag){ down(&mtp2-;>mutex); if(condvar_flag){ cprintf("No %d
listener has %s, and bought %s and is listening music now.condvar
",num,GOODS[num],WANT[num]); cond_signal(&mtp2-;>cv[0]);
cond_wait(&mtp2-;>cv[num + 1]); } if (mtp2->next_count > 0)
up(&mtp2-;>next); else up(&mtp2-;>mutex); } cprintf("No %d listener
quit! condvar
",num); } void check_sync(void) {/ 吸烟者问题拓展一（北大1999) / int i,
pid; //check semaphore sem_init(&seller;, 0); pid =
kernel_thread(seller_sema, NULL, 0); if (pid <= 0) { panic("create seller_sema
failed.
"); } seller_sema_proc = find_proc(pid);
set_proc_name(seller_sema_proc, "seller_sema_proc"); sema_flag = 1; for(i = 0;
i < 3; ++i){ sem_init(&listener;[i], 0); pid =
kernel_thread(listener_sema, (void )i, 0); if (pid <= 0) { panic("create
No.%d listener_sema failed.
", i); } listener_sema_proc[i] = find_proc(pid);
set_proc_name(listener_sema_proc[i], "listener_sema_proc"); } //check
condition variable monitor_init(&lmt;, 4); pid =
kernel_thread(seller_condvar, NULL, 0); if (pid <= 0) { panic("create
seller_condvar failed.
"); } seller_condvar_proc = find_proc(pid);
set_proc_name(seller_condvar_proc, "seller_condvar_proc"); condvar_flag = 1;
for(i = 0; i < 3; ++i){ pid = kernel_thread(listener_condvar, (void )i, 0);
if (pid <= 0) { panic("create No.%d listener_condvar failed.
"); }
listener_condvar_proc[i] = find_proc(pid);
set_proc_name(listener_condvar_proc[i], "listener_condvar_proc"); } }
```

> 4
> "假设一个录像厅有0,1，2三种不同的录像片可由观众选择放映，录像厅的放映规则为:
> 任一时刻最多只能放映一种录像片，正在放映的录像片是自动循环放映的，最后一个观众主动离开时结束当前录像片的放映；
> 选择当前正在放映的录像片的观众可立即进入，允许同时有多位选择同一种录像片的观众同时观看，同时观看的观众数量不受限制；
> 等待观看其他录像片的观众按到达顺序排队，当一种新的录像片开始放映时，所有等待观看该录像片的观众可依次序进入录像厅同时观看。用一个进程代表一个观众。
> 要求:用信号量方法PV实现，并给出信号量定义和初始值。（最好也能写出录像厅的进程）"

- [x]

知识点:信号量
出处:网络
难度:1

> #include #include #include #include #include int cinema=-1; int people=0;
> semaphore_t mov[num]; / 每个电影一个信号量 / int wait[3]; void semaphore_test(i) /
> i：影片编号 / { if(cinema==-1 || (cinema==i && people>0)) { cinema=i;
> up(&mov;[i]); } } void semaphore_movie_play(int i) { down(&mutex;);
> semaphore_test(i); int ifwait=0; if (i!=cinema) ifwait=1; wait[i]+=ifwait;
> //cprintf("testing %d %d %d
> ",cinema,i,mov[i].value); up(&mutex;);
> down(&mov;[i]); down(&mutex;); wait[i]-=ifwait; people++; cinema=i;
> cprintf("No.%d movie_sema is playing,remain people num:%d
> ",i,people);
> /电影放映/ //cprintf("testING %d %d %d %d
> ",cinema,i,mov[i].value,wait[i]); if
> (wait[i]!=0) up(&mov;[i]); up(&mutex;); //if (bf==people)
> down(&mov;[i]); } void semaphore_cinema_end(int i) / i：影片编号从0到N-1 / {
> down(&mutex;); / 进入临界区 / people--; cprintf("No.%d movie_sema quit,remain
> people num: %d
> ",i,people); if(people==0) cinema=-1; semaphore_test(left);
> semaphore_test(right); / 看一下其他影片可否播放 / up(&mutex;); / 离开临界区 / } int
> semaphore_movie(void arg) / i：电影编号，从0到N-1 / { int i, iter=0; i=(int)arg;
> cprintf("I am No.%d movie_sema
> ",i); cprintf("Iter %d, No.%d movie_sema is
> ready
> ",iter,i); do_sleep(SLEEP_TIME); semaphore_movie_play(i); / 开始电影放映 /
> do_sleep(SLEEP_TIME); semaphore_cinema_end(i); / 结束放映 / cprintf("No.%d
> movie_sema quit
> ",i); return 0; }
> 4
> "银行有n个柜员,每个顾客进入银行后先取一个号,并且等着叫号,当一 个柜员空闲后,就叫下一个号."

- [x]

知识点:信号量
出处:网络
难度:1

> 将顾客号码排成一个队列,顾客进入银行领取号码后,将号码由队尾插入;柜员空闲 时,从队首取得顾客号码,并且为这个顾客服务,由于队列为若干进程共享, 所以需要互
> 斥.柜员空闲时,若有顾客,就叫下一个顾客为之服务.因此,需要设置一个信号量来记录等 待服务的顾客数. begin var
> mutex=1,customer_count=0:semaphore; cobegin process customer begin repeat 取号码；
> p(mutex); 进入队列； v(mutex); v(customer_count); end process serversi(i=1,...,n)
> begin repeat p(customer_count); p(mutex); 从队列中取下一个号码； v(mutex); 为该号码持有者服务； end
> 4
> "假设缓冲区buf1和缓冲区buf2无限大，进程p1向buf1写数据，进程p2向buf2写数据，
> 要求buf1数据个数和buf2数据个数的差保持在(m,n)之间(m<n,m,n都是正数)."

- [x]

知识点:信号量
出处:网络
难度:1

> 题中没有给出两个进程执行顺序之间的制约关系，只给出了一个数量上的制约 关系，即m≤|buf1数据个数 - buf2数据个数≤n．不需要考虑缓冲区的大小，只需要考
> 虑两个进程的同步和互斥．p2向buf2写数据比p1向buf1写数据的次数最少不超过m次，
> 最多不能超过n次，反之也成立．所以是一个生产者和消费者问题。将等式展开得：(1)m≤(buf1数据个数 - buf2数据个数)≤n;
> (2)m≤(buf2数据个数 - buf1数据个数)≤n;由 于m,n都是正数，等式只有一个成立，不妨设(1)成立．在进程p1和p2都没有运行时，
> 两个缓冲区数据个数之差为0,因此，p1必须先运行，向buf1至少写m+1个数据后再唤
> 醒p2运行．信号量s1表示p1一次写入的最大量,初值为n，s2表示p2一次写入的最大量,初 值为-m. begin var
> mutex1=1,mutex2=1,s1=n,s2=-m:semaphore; cobegin process p1 begin repeat get
> data; p(s1); p(mutex1); 写数据到buf1; v(mutex1); v(s2); end process p2 begin
> repeat; get data; p(s2); p(mutex2); 写数据到buf2; v(mutex2); v(s1); end
> 1
> 操作系统是（）。

- ( ) A.硬件
- (x) B.系统软件
- ( ) C.应用软件
- ( ) D.虚拟机

知识点:操作系统概述
出处:网络
难度:1

> B
> 1
> 下面关于SPOOL的叙述错误的是()

- ( ) A.SPOOL又称"斯普林"，是Simultaneous Peripheral Operation On Line的缩写
- (x) B.SPOOL处理方式只是方便操作员，不能直接提高系统效率
- ( ) C.SPOOL是把磁盘作为巨大缓冲器的技术
- ( ) D.SPOOL处理方式不仅方便操作员，而且还提高系统效率

知识点:操作系统概述
出处:网络
难度:1

> B
> 1
> 对于下列文件的物理结构，()只能采用顺序存取方式

- ( ) A.顺序文件
- (x) B.链接文件
- ( ) C.索引文件
- ( ) D.Hash文件

知识点:连续内存分配
出处:网络
难度:1

> B
> 1
> 设备分配问题中，算法实现时，同样要考虑安全性问题，防止在多个进程进行设备请求时，因相互等待对方释放所占设备所造成的()现象

- ( ) A.瓶颈
- ( ) B.碎片
- ( ) C.系统抖动
- (x) D.死锁

知识点:死锁
出处:网络
难度:1

> D
> 1
> 下面有关可变分区管理中采用的主存分配算法说法错误的是（）

- ( ) A.可变分区管理常采用的主存分配算法包括首次适应、最优适应和循环首次适应等算法
- ( ) B.首次适应算法实现简单，但碎片过多使主存空间利用率降低
- (x) C.最优适应算法是最好的算法，但后到的较大作业很难得到满足
- ( ) D.循环首次适应算法能使内存中的空闲分区分布得更均匀

知识点:非连续内存分配
出处:网络
难度:1

> C
> 1
> 如下表所示，虚拟段页式存储管理方案的特性为() 地址空间 空间浪费 存储共享 存储保护 动态扩充 动态连接

- ( ) A.一维 大 不易 易 不可 不可
- ( ) B.一维 小 易 不易 可以 不可
- ( ) C.二维 大 不易 易 可以 可以
- (x) D.二维 小 易 易 可以 可以

知识点:非连续内存分配
出处:网络
难度:1

> D
> 1
> 执行一次磁盘输入输出操作所花费的时间包括

- ( ) A.寻道时间、旋转延迟时间、传送时间和等待时间
- ( ) B.寻道时间、等待时间、传送时间
- ( ) C.等待时间、寻道时间、旋转延迟时间和读写时间
- (x) D.寻道时间、旋转延迟时间、传送时间

知识点:I/O子系统
出处:网络
难度:1

> D
> 1
> 在下列操作系统的各个功能组成部分中,哪一个不需要有硬件的支持

- (x) A.进程调度

- ( ) B.时钟管理
- ( ) C.地址映射
- ( ) D.中断系统

知识点:操作系统概述
出处:网络
难度:1

> A
> 1
> 一个正在访问临界资源的进程由于申请等待I/O操作而被中断时

- ( ) A.可以允许其他进程进入与该进程相关的临界区
- ( ) B.不允许其他进程进入任何临界区
- (x) C.可以允许其他就绪进程抢占处理器，继续运行
- ( ) D.不允许任何进程抢占处理器

知识点:同步互斥
出处:网络
难度:1

> C
> 1
> 批处理操作系统的特点不包括

- ( ) A.提高了系统资源的利用率
- (x) B.用户可以直接干预作业的运行，具有交互性
- ( ) C.提高了单位时间内的处理能力
- ( ) D.提高了系统的吞吐率

知识点:操作系统概述
出处:网络
难度:1

> B
> 1
> 下面不属于操作系统提供虚拟设备技术原因的是

- ( ) A.独占设备可以作为共享设备来使用
- ( ) B.独占设备使用的静态分配技术既不能充分利用设备，又不利于提高系统效率
- ( ) C.在一定硬件和软件条件的基础上共享设备可以部分或全部地模拟独占设备的工作，提高独占设备的利用率和系统效率
- (x) D.计算机系统具有多道处理功能，允许多道作业同时执行

知识点:I/O子系统
出处:网络
难度:1

> D
> 1
> 采用多道程序设计的实质之一是

- (x) A.以空间换取时间
- ( ) B.将独享设备改造为共享设备
- ( ) C.提高内存和I/O设备利用率
- ( ) D.虚拟设备

知识点:操作系统概述
出处:网络
难度:1

> A
> 1
> 访管指令的作用是

- ( ) A.嵌套调用
- ( ) B.用户使用的命令
- (x) C.用户态转换为核心态
- ( ) D.保证运行在不同状态

知识点:中断、异常与系统调用
出处:网络
难度:1

> C
> 1
> 不属于I/O控制方式的是

- ( ) A.程序查询方式
- (x) B.复盖方式
- ( ) C.DMA方式
- ( ) D.中断驱动方式

知识点:I/O子系统
出处:网络
难度:1

> B
> 1
> 软件共享的必要性是为了

- ( ) A.节约内存空间
- ( ) B.缩短运行时间
- ( ) C.减少内外存对换信息量
- (x) D.A和C

知识点:非连续内存分配
出处:网络
难度:1

> D
> 1
> 下面软件系统中完全属于系统软件的一组是

- (x) A.操作系统、编译系统、windowsNT
- ( ) B.接口软件、操作系统、软件开发工具
- ( ) C.专用程序、财务管理软件、编译系统、操作系统
- ( ) D.操作系统、接口软件、Office 2000

知识点:操作系统概述
出处:网络
难度:1

> A
> 1
> 主存储器是

- ( ) A.以“字”为单位进行编址的
- (x) B.是中央处理机能够直接访问的惟一的存储空间
- ( ) C.与辅助存储器相比速度快、容量大、价格低的一类存储器
- ( ) D.只能被CPU访问的存储器

知识点:连续内存分配
出处:网络
难度:1

> B
> 1
> 特权指令

- (x) A.是可能影响系统安全的一类指令
- ( ) B.既允许操作系统程序使用，又允许用户程序使用
- ( ) C.是管态和目态运行的基本单位
- ( ) D.是一种存储保护方法

知识点:操作系统概述
出处:网络
难度:1

> A
> 1
> 下面有关选择进程调度算法的准则错误的是

- ( ) A.尽量提高处理器利用率
- ( ) B.尽可能提高系统吞吐量
- (x) C.适当增长进程在就绪队列中的等待时间
- ( ) D.尽快响应交互式用户的请求

知识点:处理机调度
出处:网络
难度:1

> C
> 1
> 下面是关于重定位的有关描述，其中错误的是

- ( ) A.绝对地址是主存空间的地址编号
- ( ) B.用户程序中使用的从0地址开始的地址编号是逻辑地址
- ( ) C.动态重定位中装入主存的作业仍保持原来的逻辑地址
- (x) D.静态重定位中装人主存的作业仍保持原来的逻辑地址

知识点:非连续内存分配
出处:网络
难度:1

> D
> 3
> 操作系统的所有程序都必须常驻内存

- ( ) A.对
- (x) B.错

知识点:操作系统概述
出处:网络
难度:1

> B
> 3
> 虚拟存储系统可以在每一台计算机上实现

- ( ) A.对
- (x) B.错

知识点:缺页中断
出处:网络
难度:1

> B
> 3
> 执行系统调用时可以被中断

- (x) A.对
- ( ) B.错

知识点:中断、异常与系统调用
出处:网络
难度:1

> A
> 3
> 选择通道主要用于连接低速设备

- ( ) A.对
- (x) B.错

知识点:操作系统概述
出处:网络
难度:1

> B
> 3
> 在请求分页存储管理中，从主存中刚刚移走某一页面后，根据请求马上又调进该页，这种反复调进调出的现象，称为系统颠簸，也叫系统抖动

- (x) A.对
- ( ) B.错

知识点:置换算法
出处:网络
难度:1

> A
> 3
> 通道程序解决了I / O操作的独立性和各部件工作的并行性，采用通道技术后，能实现CPU与通道的并行操作

- (x) A.对
- ( ) B.错

知识点:I/O子系统
出处:网络
难度:1

> A
> 1
> 下列哪一条是在操作系统设计中引入多道程序技术的好处?

- (x) A.使并发执行成为可能
- ( ) B.简化操作系统的实现
- ( ) C.减少对内存容量的需求
- ( ) D.便于实施存储保护

知识点:操作系统概述
出处:网络
难度:1

> A
> 3
> 程序的顺序执行具有顺序性，封闭性和不可再现性

- ( ) A.对
- (x) B.错

知识点:操作系统概述
出处:网络
难度:1

> B
> 3
> 快表是高速缓存，是内存的一部分区域

- ( ) A.对
- (x) B.错

知识点:非连续内存分配
出处:网络
难度:1

> B
> 3
> 磁盘上物理结构为链接结构的文件只能顺序存取

- (x) A.对
- ( ) B.错

知识点:I/O子系统
出处:网络
难度:1

> A
> 3
> 一旦出现死锁, 所有进程都不能运行

- ( ) A.对
- (x) B.错

知识点:死锁
出处:网络
难度:1

> B
> 4
> "什么叫进程同步和互斥?举例说明"

- [x]

知识点:同步互斥
出处:网络
难度:1

> 进程同步是在几个进程合作完成一项任务时，体现各进程相互联系相互协调的关系。例如：A、B两个进程合作通过缓存区输出数据。
> 把两个以上进程不能同时访问临界区的工作
> 规则称为进程互斥。例如：两个进程同时使用打印机
> 4
> "什么是动态链接"

- [x]

知识点:非连续内存分配
出处:网络
难度:1

> 指用户程序中的各程序段，不是在程序开始运行前就链接好，而是在程序装入或运行过程中，当发现要调用的程序段未链接时，才进行链接。
> 4
> "在下面的条件下，若用一个位图来实现空闲表，那么存储空闲表需要多少位? （a） 共有500000个块，有200000个空闲块 （b） 共有500000个块，有0个空闲块"

- [x]

知识点:置换算法
出处:网络
难度:1

> 在任何一种情况下，每个地址所用的位数和空闲块数目无关。在500000个块中，需要500000位。
> 4
> "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。若缺页率是10%，为使有效访问时间达到0.5ms,求不在内存的页面的平均访问时间
> 。"

- [x]

知识点:置换算法
出处:网络
难度:1

> 4.99865ms
> 1
> 下面( )种页面置换算法会产生Belady异常现象?

- (x) A.先进先出页面置换算法（FIFO）
- ( ) B.最近最久未使用页面置换算法（LRU）
- ( ) C.最不经常使用页面置换算法（LFU）
- ( ) D.最佳页面置换算法（OPT）

知识点:置换算法
出处:网络
难度:1

> A
> 1
> 在请求分页管理中，若采用先进先出（FIFO）页面置换算法，可能会产生“Belady异常”，“Belady异常”指的是( )。

- ( ) A.频繁地出页入页的现象
- (x) B.分配的页面数增加，缺页中断的次数也可能增加
- ( ) C.进程交换的信息量过大，导致系统工作区不足
- ( ) D.分配给进程的内存空间不足使进程无法正常工作

知识点:置换算法
出处:网络
难度:1

> B
> 1
> Windows 属于下列哪一类操作系统?

- ( ) A.单用户单任务
- (x) B.单用户多任务
- ( ) C.多用户
- ( ) D.批处理

知识点:操作系统概述
出处:网络
难度:1

> B
> 1
> 在虚拟存储器系统中常使用联想存储器进行管理，它是( )寻址的。

- ( ) A.按地址
- (x) B.按内容
- ( ) C.寄存器
- ( ) D.计算

知识点:非连续内存分配
出处:网络
难度:1

> B
> 1
> 下列关于虚拟存储器的论述中，正确的论述( )。

- ( ) A.在请求段页式系统中，以页为单位管理用户的虚空间，以段为单位管理内存空间。
- (x) B.在请求段页式系统中，以段为单位管理用户的虚空间，以页为单位管理内存空间。
- ( ) C.为提高请求分页系统中内存的利用率，允许用户使用不同大小的页面。
- ( ) D.实现虚拟存储器的最常用的算法是最佳适应算法OPT。

知识点:缺页中断
出处:网络
难度:1

> B
> 1
> 在虚拟分页存贮管理系统中，若进程访问的页面不在主存，且主存中没有可用的空闲块时，系统正确的处理顺序为( )。

- ( ) A.决定淘汰页->页面调出->缺页中断->页面调入
- ( ) B.决定淘汰页->页面调入->缺页中断->页面调出
- (x) C.缺页中断->决定淘汰页->页面调出->页面调入
- ( ) D.缺页中断->决定淘汰页->页面调入->页面调出

知识点:缺页中断
出处:网络
难度:1

> C
> 1
> 在I/O设备控制的发展过程中，最主要的推动因素是_____、提高I/O速度和设备利用率。

- ( ) A.提高资源利用率
- ( ) B.提高系统吞吐量
- (x) C.减少主机对I/O控制的干预
- ( ) D.提高CPU与I/O设备的并行操作程度

知识点:I/O子系统
出处:网络
难度:1

> C
> 1
> 下面关于设备属性的论述中，正确的是 _ _ _ _。

- ( ) A.字符设备的基本特征是可寻址到字节，即能指定输入的源地址或输出的目标地址
- (x) B.共享设备必须是可寻址的和可随机访问的设备
- ( ) C.共享设备是指同一时间内允许多个进程同时访问的设备
- ( ) D.在分配共享设备和独占设备时都可能引起进程死锁

知识点:I/O子系统
出处:网络
难度:1

> B
> 1
> 使用户所编制的程序与实际使用的物理设备无关是由_____功能实现的。

- ( ) A.设备分配
- ( ) B.缓冲管理
- ( ) C.设备管理
- (x) D.设备独立性

知识点:I/O子系统
出处:网络
难度:1

> D
> 1
> 通道是一种( )。

- ( ) A.I/O设备
- ( ) B.设备控制器
- (x) C.I/O处理机
- ( ) D.I/O控制器

知识点:I/O子系统
出处:网络
难度:1

> C
> 1
> 通道具有_____能力。

- (x) A.执行I/O指令集
- ( ) B.执行CPU指令集
- ( ) C.传输I/O命令
- ( ) D.运行I/O进程

知识点:I/O子系统
出处:网络
难度:1

> A
> 1
> 实现CPU和外部设备并行工作的硬件支持是：( )。

- ( ) A.中断机构
- ( ) B.外部设备接口（通道、控制器等）
- (x) C.通道和中断
- ( ) D.多总线

知识点:I/O子系统
出处:网络
难度:1

> C
> 1
> 在具有通道处理机的系统中，用户进程请求启动外设时，由 ( )根据I/O要求构造通道程序及通道状态字，并将通道程序保存在内存，然后执行启动“I/O”命令。

- ( ) A.用户进程
- ( ) B.应用程序
- ( ) C.通道
- (x) D.操作系统

知识点:I/O子系统
出处:网络
难度:1

> D
> 1
> 在具有通道处理机的系统中，用户进程请求启动外设时，由操作系统根据I/O要求构造通道程序及通道状态字，并将通道程序保存在( )，然后执行启动“I/O”命令。

- (x) A.内存
- ( ) B.硬盘
- ( ) C.通道
- ( ) D.外部设备

知识点:I/O子系统
出处:网络
难度:1

> A
> 1
> 不通过CPU进行主存与I/O设备间大量的信息交换方式，可以是( )方式。

- (x) A.DMA
- ( ) B.中断
- ( ) C.查询等待

- ( ) D.程序控制

知识点:I/O子系统
出处:网络
难度:1

> A
> 1
> 从下面关于设备独立性的论述中，第( )条是正确的论述。

- ( ) A.设备独立性是I/O设备具有独立执行I/O功能的一种特性。
- (x) B.设备独立性是指用户程序独立于具体使用的物理设备的一种特性。
- ( ) C.设备独立性是指能独立实现设备共享的一种特性。
- ( ) D.设备独立性是指设备驱动独立于具体使用的物理设备的一种特性。

知识点:I/O子系统
出处:网络
难度:1

> B
> 1
> 为了实现设备的独立性、系统中的逻辑设备表必须包含：( )。

- ( ) A.逻辑设备名和物理设备名
- ( ) B.逻辑设备名和驱动程序入口地址
- ( ) C.物理设备名和驱动程序入口地址
- (x) D.逻辑、物理设备名和驱动程序入口地址

知识点:I/O子系统
出处:网络
难度:1

> D
> 1
> 使用编制的程序与实际使用的物理设备无关是由( )功能实现的。

- ( ) A.设备分配
- ( ) B.设备驱动
- ( ) C.虚拟设备
- (x) D.设备独立性

知识点:I/O子系统
出处:网络
难度:1

> D
> 1
> 下面关于虚拟设备的论述中，第( )条是正确的论述。

- ( ) A.虚拟设备是指允许用户使用比系统中具有的物理设备更多的设备。
- ( ) B.虚拟设备是指允许用户以标准化方式来使用物理设备。
- (x) C.虚拟设备是把一个物理设备变换成多个对应的逻辑设备。
- ( ) D.虚拟设备是指允许用户程序不必全部装入内存便可使用系统中的设备。

知识点:I/O子系统
出处:网络
难度:1

> C
> 1
> 通过硬件和软件的功能扩充，把原来独占的设备改造成能为若干用户共享的设备，这种设备称为( )。

- ( ) A.存储设备
- ( ) B.系统设备
- (x) C.虚拟设备
- ( ) D.用户设备

知识点:I/O子系统
出处:网络
难度:1

> C
> 1
> 如果I/O所花费的时间比CPU处理时间短得多，则缓冲区( )。

- ( ) A.最有效
- (x) B.几乎无效
- ( ) C.均衡
- ( ) D.都不是

知识点:I/O子系统
出处:网络
难度:1

> B
> 1
> 在现代操作系统中采用缓冲技术的主要目的是( )

- ( ) A.改善用户编程环境
- ( ) B.提高CPU的处理速度
- (x) C.提高CPU和设备之间的并行程度
- ( ) D.实现与设备无关性

知识点:I/O子系统
出处:网络
难度:1

> C
> 1
> 下列有关SPOOLing系统的论述中第( )条是正确的论述。

- ( ) A.构成SPOOLing系统的基本条件，是具有外围输入机与外围输出机。
- ( ) B.只要操作系统中采用了多道程序设计技术，就可以构成SPOOLing系统。
- ( ) C.SPOOLing系统是虚拟存储技术的体现。
- ( ) D.当输出设备忙时，SPOOLing系统中的用户程序暂停执行，待I/O 空闲时再被唤醒，去执行输出操作。
- (x) E.在SPOOLing系统中，用户程序可以随时将输出数据送到输出井中，待输出设备空闲时再执行数据输出操作。

知识点:I/O子系统
出处:网络
难度:1

> E
> 1
> 下列有关SPOOLing系统的论述中第( )条是正确的论述。

- ( ) A.构成SPOOLing系统的基本条件，是只要具有大容量、高速硬盘作为输入井与输出井。
- ( ) B.SPOOLing系统是建立在分时系统中。
- ( ) C.SPOOLing系统是在用户程序要读取数据时起动输入进程输入数据。
- ( ) D.当输出设备忙时，SPOOLing系统中的用户程序暂停执行，待I/O空闲时再被唤醒，去执行输出操作。
- (x) E.SPOOLing系统实现了对I/O设备的虚拟，只要输入设备空闲，SPOOLing可预先将输入数据从设备传输到输入井中供用户程序随时读取。

知识点:I/O子系统
出处:网络
难度:1

> E
> 1
> 在采用SPOOLing技术的系统中，用户作业的打印输出结果首先被送到( )。

- (x) A.磁盘固定区域
- ( ) B.内存固定区域
- ( ) C.终端
- ( ) D.打印机

知识点:I/O子系统
出处:网络
难度:1

> A
> 1
> 在操作系统中SPOOLing技术是一种并行机制，它可以使 ( )。

- ( ) A.不同进程同时运行
- ( ) B.应用程序和系统软件同时运行
- ( ) C.不同系统软件同时运行
- (x) D.程序执行与打印同时进行

知识点:I/O子系统
出处:网络
难度:1

> D
> 1
> 在设备管理中，虚拟设备的引入和实现是为了充分利用设备，提高系统效率，采用( )来模拟低速设备（输入机或打印机）的工作。

- ( ) A.Spooling技术，利用磁带设备
- (x) B.Spooling技术，利用磁盘设备
- ( ) C.脱机批处理系统

知识点:I/O子系统
出处:网络
难度:1

> B
> 1
> SPOOLing是对脱机I/O工作方式的模拟，SPOOLing系统中的输入井是对脱机输入中的( )进行模拟。

- ( ) A.内存输入缓冲区
- (x) B.磁盘
- ( ) C.外围控制机
- ( ) D.输入设备

知识点:I/O子系统
出处:网络
难度:1

> B
> 1
> SPOOLing是对脱机I/O工作方式的模拟，SPOOLing系统中的输出进程是对脱出输出中的( )进行模拟。

- ( ) A.内存输入缓冲区
- ( ) B.磁盘
- (x) C.外围控制机
- ( ) D.输入设备

知识点:I/O子系统
出处:网络
难度:1

> C
> 1
> 从下列关于驱动程序的论述中，选出一条正确的论述。

- ( ) A.驱动程序与I/O设备的特性紧密相关，因此应为每一I/O设备配备一个驱动程序。
- ( ) B.驱动程序与I/O控制方式紧密相关，因此对DMA方式应是以字节为单位去启动设备及进行中断处理。
- ( ) C.由于驱动程序与I/O设备（硬件）紧密相关，故必须全部用汇编语言书写。
- (x) D.对于一台多用户机，配置了相同的8个终端，此时可以只配置一个由多个终端共享的驱动程序。

知识点:I/O子系统
出处:网络
难度:1

> D
> 1
> 操作系统设备管理功能的内部设计一般是基于分层的思想，因此通常将I/O软件组成4个层次，用户应用软件、终端应用层软件、中断应用层程序、中断处理程序、独立于设备
> 的软件和设备驱动程序，采用分层思想的主要目的是( )。

- ( ) A.便于即擦即用
- ( ) B.减少系统占用的空间
- (x) C.便于系统修改、扩充和移植
- ( ) D.提高处理速度

知识点:I/O子系统
出处:网络
难度:1

> C
> 1
> 操作系统设备管理功能的内部设计一般是基于分层的思想，因此通常将I/O软件组成4个层次，用户应用软件、终端应用层软件、中断应用层程序、中断处理程序、独立于设备
> 的软件和设备驱动程序，当进程提出I/O请求访问硬件时，需要按( )的层次结构进行。

- ( ) A.进程请求I/O->独立于设备的软件— > 中断处理程序— > 设备驱动程序— > 硬件
- (x) B.进程请求I/O->独立于设备的软件— > 设备驱动程序— > 中断处理程序— > 硬件
- ( ) C.进程请求I/O->设备驱动程序— > 中断处理程序— > 独立于设备的软件— > 硬件
- ( ) D.进程请求I/O->设备驱动程序— > 独立于设备的软件— > 中断处理程序— > 硬件

知识点:I/O子系统
出处:网络
难度:1

> B
> 1
> 对磁盘进行移臂调度时，既考虑了减少寻找时间，又不频繁改变动臂的移动方向的调度算法是( )。

- ( ) A.先来先服务
- ( ) B.最短寻找时间优先
- (x) C.电梯调度
- ( ) D.优先级高者优先

知识点:I/O子系统
出处:网络
难度:1

> C
> 1
> 对移动臂磁盘的一次信息传输所花费的时间由三部分组成，它们是( )。

- (x) A.传送时间，延迟时间和寻找时间
- ( ) B.旋转等待时间，延迟时间和寻找时间
- ( ) C.磁头移动时间，延迟时间和寻找时间
- ( ) D.延迟时间，移动时间和等待时间

知识点:I/O子系统
出处:网络
难度:1

> A
> 1
> ( )调度算法总是从等待访问者中挑选等待时间最短的那个请求先执行。

- (x) A.先来先服务
- ( ) B.最短寻找时间优先
- ( ) C.电梯
- ( ) D.单向扫描

知识点:I/O子系统
出处:网络
难度:1

> A
> 1
> 对磁盘进行移臂调度的目的是缩短( )。

- ( ) A.启动时间
- ( ) B.传送时间
- (x) C.寻找定位时间
- ( ) D.旋转延迟时间

知识点:I/O子系统
出处:网络
难度:1

> C
> 1
> 下列第( )项不是文件系统的功能?

- ( ) A.文件系统实现对文件的按名存取
- ( ) B.负责实现数据的逻辑结构到物理结构的转换
- (x) C.提高磁盘的读写速度
- ( ) D.提供对文件的存取方法和对文件的操作

知识点:文件系统
出处:网络
难度:1

> C
> 1
> 文件系统的主要目的是( )。

- (x) A.实现对文件的按名存取
- ( ) B.实现虚拟存贮器
- ( ) C.提高外围设备的输入输出速度
- ( ) D.用于存贮系统文档

知识点:文件系统
出处:网络
难度:1

> A
> 1
> 按逻辑结构划分，文件主要有两类，UNIX中的文件系统采用_____。

- ( ) A.网状文件
- ( ) B.只读文件
- ( ) C.读写文件
- ( ) D.记录式文件
- ( ) E.索引文件
- (x) F.流式文件

知识点:文件系统
出处:网络
难度:1

> F
> 1
> 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。

- ( ) A.堆文件
- (x) B.流式文件
- ( ) C.索引文件
- ( ) D.直接（Hash）文件

知识点:文件系统
出处:网络
难度:1

> B
> 1
> 通常，文件的逻辑结构中( )组织方式，既适合于交互方式应用，也适合于批处理方式应用。

- ( ) A.堆文件
- ( ) B.流式文件
- (x) C.索引顺序文件

- ( ) D.顺序文件

知识点:文件系统
出处:网络
难度:1

> C
> 1
> 下面关于索引文件的论述中，第( )条是正确的论述。

- ( ) A.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) B.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。
- ( ) C.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。
- ( ) D.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。

知识点:文件系统
出处:网络
难度:1

> B
> 1
> 下面关于顺序文件和链接文件的论述中错误的论述是( )。

- (x) A.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。
- ( ) B.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。
- ( ) C.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。
- ( ) D.在MS-DOS中采用的是链接文件结构。

知识点:文件系统
出处:网络
难度:1

> A
> 1
> 文件信息的逻辑块号到磁盘物理块号的变换是由( )决定。

- ( ) A.逻辑结构
- ( ) B.页表
- (x) C.物理结构
- ( ) D.重定位寄存器

知识点:文件系统
出处:网络
难度:1

> C
> 1
> 对于下列文件的物理结构，( )只能采用顺序存取方式。

- ( ) A.顺序文件
- (x) B.链接文件
- ( ) C.索引文件
- ( ) D.Hash文件

知识点:文件系统
出处:网络
难度:1

> B
> 1
> 一个采用一级索引文件系统，存取一块盘块信息通常要访问()次磁盘。

- ( ) A.1
- (x) B.2
- ( ) C.3
- ( ) D.4

知识点:文件系统
出处:网络
难度:1

> B
> 1
> 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。

- ( ) A.1
- ( ) B.2
- (x) C.3
- ( ) D.5

知识点:文件系统
出处:网络
难度:1

> C
> 1
> 一个采用三级索引文件系统，存取一块盘块信息通常要访问()次磁盘。

- ( ) A.1
- ( ) B.2
- ( ) C.3
- (x) D.6

知识点:文件系统
出处:网络
难度:1

> D
> 1
> 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，试问该文件至少应
> 该建立( )级索引〔假定一级索引占用一个物理块〕？

- ( ) A.1
- ( ) B.2
- (x) C.3
- ( ) D.7

知识点:文件系统
出处:网络
难度:1

> C
> 1
> 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，试问索引应占(
>)个物理块?

- ( ) A.1
- ( ) B.11
- (x) C.111
- ( ) D.1111

知识点:文件系统
出处:网络
难度:1

> C
> 1
> 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?

- ( ) A.1000
- ( ) B.1001
- ( ) C.1011
- (x) D.1111

知识点:文件系统
出处:网络
难度:1

> D
> 1
> 文件管理实际上是对_____的管理。

- ( ) A.主存空间
- (x) B.辅助存储空间
- ( ) C.逻辑地址空间
- ( ) D.物理地址空间

知识点:文件系统
出处:网络
难度:1

> B
> 1
> 在文件系统中设置一张( )表，它是利用二进制的一位来表示磁盘中一个块的使用情况。

- ( ) A.文件描述符表
- ( ) B.链接指针表
- ( ) C.文件表
- ( ) D.空闲区表
- (x) E.位示图

知识点:文件系统
出处:网络
难度:1

> E
> 1
> 文件系统中用_____管理文件。

- ( ) A.堆栈结构
- ( ) B.指针

- (x) C.目录
- ( ) D.页表

知识点:文件系统
出处:网络
难度:1

> C
> 1
> 为了允许不同用户的文件具有相同的文件名，通常在文件系统中采用_____。

- ( ) A.重名翻译
- (x) B.多级目录
- ( ) C.约定
- ( ) D.路径

知识点:文件系统
出处:网络
难度:1

> B
> 1
> 文件系统的按名存取主要是通过( )实现的。

- ( ) A.存储空间管理
- (x) B.目录管理
- ( ) C.文件安全性管理
- ( ) D.文件读写管理

知识点:文件系统
出处:网络
难度:1

> B
> 1
> Windows操作系统的一个文件的绝对路径名是从( )开始的整个通路上所有子目录 名组成的一个有序组合。

- ( ) A.当前目录
- ( ) B.根目录
- ( ) C.家目录(home directory)
- (x) D.磁盘驱动器编号

知识点:文件系统
出处:网络
难度:1

> D
> 1
> 在UNIX (linux) 操作系统中文件的绝对路径名首先是( )。

- ( ) A.盘符
- (x) B.根目录
- ( ) C.盘符或根目录
- ( ) D.以上都不是

知识点:文件系统
出处:网络
难度:1

> B
> 1
> 文件系统中每个文件有( )个文件控制块FCB。

- ( ) A.1
- (x) B.1或多于1
- ( ) C.1或0
- ( ) D.0

知识点:文件系统
出处:网络
难度:1

> B
> 1
> 将文件描述信息从目录项中分离出来（将文件控制块FCB分离为文件名和文件描述信息） 的好处是：( )

- ( ) A.减少读文件时的I/O信息量
- ( ) B.减少写文件时的I/O信息量
- (x) C.减少查文件时的I/O信息量
- ( ) D.减少复制文件时的I/O信息量

知识点:文件系统
出处:网络
难度:1

> C
> 1
> 允许多个用户同时使用同一个共享文件时，下列( )做法是不对的。

- ( ) A.允许多个用户同时打开共享文件执行读操作
- (x) B.允许读者和写者同时使用共享文件
- ( ) C.不允许读者和写者同时使用共享文件
- ( ) D.不允许多个写者同时对共享文件执行写操作

知识点:文件系统
出处:网络
难度:1

> B
> 1
> 操作系统为保证未经文件拥有者授权，任何其它用户不能使用该文件所提供的解决方法 是( ).

- (x) A.文件保护
- ( ) B.文件保密
- ( ) C.文件转储
- ( ) D.文件共享

知识点:文件系统
出处:网络
难度:1

> A
> 1
> 在linux/Unix中对文件有条件的共享是指：( )

- ( ) A.给不同身份的用户赋予不同的访问权限
- ( ) B.给不同身份的用户赋予相同的访问权限
- (x) C.给不同身份的用户赋予相同或不同的访问权限
- ( ) D.给相同身份的用户赋予不同的访问权限

知识点:文件系统
出处:网络
难度:1

> C
> 1
> 假设在一个系统中一个文件有二个名字，它与一个文件保存为二个副本的区别是_____ ___。

- ( ) A.前者比后者所占的存储空间大
- ( ) B.前者需二个目录项，后者只需一个目录项
- ( ) C.前者存取文件的速度快，后者存取速度慢
- (x) D.前者改变与某个名字相联系的文件时，另一个名字相联的文件也改变，后者的另一个副本不改变。

知识点:文件系统
出处:网络
难度:1

> D
> 1
> 用户请求使用一个已存在的文件时，其操作次序为( )

- ( ) A.读 / 写→关闭
- ( ) B.打开→读 / 写
- (x) C.打开→读 / 写→关闭

知识点:文件系统
出处:网络
难度:1

> C
> 1
> 打开文件操作的使用是( )。

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

知识点:文件系统
出处:网络
难度:1

> B
> 5
> "桌上有一个空盒，盒内只允许放一个水果。妈妈轮流向盒内放桔子和苹果，儿子专等吃盒中的桔子，女儿专等吃盒中的苹果。若盒内已有水果，放者必须等待，若盒内没有自己
> 吃的水果，吃者必需等待。试在下述类PASCAL程序中虚线位置分别填上信号量、信号量初值和P、V操作实现三个进程正确的并发执行。 var
> (信号量)_____：semaphore：= (信号量初值) _____;
> begin parbegin 妈:begin repeat 準備 _____ 向盒内放桔子 _____ 準備 _____ 向盒内放苹果 _____ until false end 儿：begin repeat _____ 拿盒中的桔子 _____ 吃桔子 until false end 女：begin repeat _____ 拿盒中的苹果 _____ 吃苹果 until false end parend end "

- [x]

知识点:信号量
出处:网络
难度:1

> var (信号量)_____S , S1 , S2 _____: semaphore：= (信号量初值) _____1 , 0 , 0
> _____； begin parbegin 妈:begin repeat 准备 _____ P (S) _____ 向盒内放桔子 _____ V (S1)
> _____ 准备 _____ P (S) _____ 向盒内放苹果 _____ V (S2) _____ until false end 儿：begin repeat _____ P (S1
>) _____ 拿盒中的桔子 _____ V (S) _____ 吃桔子 until false end 女：begin repeat _____ P (S2) _____
> 拿盒中的苹果 _____ V (S) _____ 吃苹果 until false end parend end
> 4
> "假定在一个处理机上执行以下五个作业： 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
> (1)画出采用FCFS调度算法时调度图，并计算每个作业的周转时间和计算平均周转时间。
> (2)画出采用SJF调度算法时调度图，并计算每个作业的周转时间和计算平均周转时间。
> (3)写出采用HRN（响应比高者优先）调度算法时选择作业号的次序和选择作业的依据（各作业的响应比）。 "

- [x]

知识点:处理机调度
出处:网络
难度:1

> 1. 先来先服务调度算法FCFS作业调度次序的计算：
> FCFS按照作业到达的先后次序来选择作业，按作业到达时间的先后次序五个作业调度次序为A、B、C、D、E。 2.
> 短作业优先调度算法SJF作业调度次序的计算： SJF在到达的作业中挑选所需运行时间最短的作业进入主存先运行，调度次序如下：
> T=0：只有作业A已到达，调度作业A运行。
> T=4：作业A完成，作业B、C、D、E已全部到达，比较作业B、C、D、E的运行时间，按运行时间短的作业先运行，则调度次序为D、B、E、C。 调度图： T 0
> 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 FCFS A A A A B B B C C C C C D D
> E E E E SJF A A A A D D B B B E E E E C C C C C 进 程 A B C D E 平均 到达时间 Ta 0 1 2
> 3 4 运行时间 TS 4 3 5 2 4 3.高响应比优先(HRRN)(作业)调度算法作业调度次序的计算： T=0：只有作业A已到达，调度作业A运行。
> T=4：作业A完成，作业B、C、D、E已到达，计算作业B、C、D、E响应比RP分别为：
> 1+3/3、1+2/5、1+1/2、1+0/4，作业B响应比最大调度运行。 T=7：作业B完成，作业C、D、E已到达，计算作业C、D、E响应比RP分别为：
> 1+5/5、1+4/2、1+3/4，作业D响应比最大调度运行。 T=9：作业D完成，作业C、E已到达，计算作业C、E响应比RP分别为：
> 1+7/5、1+5/4，作业C响应比最大调度运行。 T=14：作业C完成，作业E已到达，调度作业E运行。 T=18：作业E完成。
> 4
> "试描述避免死锁的银行家算法，若系统运行中出现下述资源分配情况 进程 ALLOCATION NEED AVAILABLE A B C D A B C D A
> B C D P0 0 0 3 2 0 0 1 2 1 6 2 2 P1 1 0 0 0 1 7 5 0 P2 1 3 5 4 2 3 5 6 P3 0 3
> 3 2 0 6 5 2 P4 0 0 1 4 0 6 5 6
> 该系统是否安全？如果进程P2此时提出资源申请（1，2，2，2），系统能否将资源分配给它？为什么？ "

- [x]

知识点:死锁
出处:网络
难度:1

> 进程 Allocation Need Availabe Avelable+ Allocation No A B C D A B C D A B C D A
> B C D P0 0 0 3 2 0 0 1 2 1 6 2 2 1 6 5 4 1 P1 1 0 0 0 1 7 5 0 1 9 8 6 2 9 8 6
> 3 P2 1 3 5 4 2 3 5 6 2 9 8 6 3 12 13 10 4 P3 0 3 3 2 0 6 5 2 1 6 5 4 1 9 8 6 2
> P4 0 0 1 4 0 6 5 6 3 12 13 10 3 12 14 14 5 可以找到一个安全序列{P0、P3、P1、P2、P4}，系统是安全的。
> 进程P2此时提出资源申请（1，2，2，2），
> 如系统实施此次分配使系统可用资源减到（0，4，0，0），再也无法满足各进程对资源的需求，系统进入一个不安全状态，系统不能将资源分配给进程P2。 进程
> Allocation Need Availabe Avelable+ Allocation No A B C D A B C D A B C D A B C
> D P0 0 0 3 2 0 0 1 2 0 4 0 0 P1 1 0 0 0 1 7 5 0 P2 2 5 7 6 1 1 3 4 P3 0 3 3 2
> 0 6 5 2 P4 0 0 1 4 0 6 5 6
> 4
> "1. 某虚拟存储器的用户空间共有32个页面，每页IKB，主存为16KB。假定某时刻系统为用户的第0、l、2、3页分别分配到物理块号为5、10、4、7中，试
> 分别写出虚拟地址1234和2345的页号和页内地址，并将虚拟地址变换为物理地址。"

- [x]

知识点:缺页中断
出处:网络
难度:1

> (1)将虚地址分离成页号和页内地址d： 页号P = （虚地址 / 页大小）取整 = （1234/1024）取整 = 1 页内地址d = 虚地址 - 页号P×每页大小
> = 1234 - 1×1024 = 1234 - 1024 = 210 根据页号查页表，由页表项读出物理页号： 由页号 P = 1查页表得物理页号为10
> 将物理页号和页内地址构成物理地址： 物理地址 = 物理页号×页大小 + 页内地址 = 10×1024 + 210 = 10450 (2)将虚地址分离成页号和页内地址d：
> 页号P = （虚地址 / 页大小）取整 = （2345/1024）取整 = 2 页内地址d = 虚地址 - 页号P×每页大小 = 2345 - 2×1024 = 2345 - 2048 = 297
> 根据页号查页表，由页表项读出物理页号： 由页号 P = 2查页表得物理页号为4 将物理页号和页内地址构成物理地址： 物理地址 = 物理页号×页大小 + 页内地址
> = 4×1024 + 297 = 4397
> 4
> "考虑一个分页系统，其页表存放在内存，如果内存读写周期为1.0us，快表的访问时间为0.2us。如果设立一个可存放64个页表表项的快表，90%的地址变换可通
> 过快表完成，问内存平均存取周期为多少？"

- [x]

知识点:缺页中断
出处:网络
难度:1

> 当快表命中时CPU存取内存一个数据的时间为T1 = 检索快表时间 + 访问内存数据时间=T(快表)+T(内存) = 20 + 100 = 120ns。
> 当快表不命中时CPU存取内存一个数据的时间为T2 = 检索快表时间 + 检索内存中的页表时间 + 访问内存数据时间=T(快表)+T(内存)
> +T(内存) = 20 + 100 + 100 = 220ns。 则CPU存取内存一个数据的平均时间为 T = T1命中率+T2（1 - 命中率） =
> T1ρ+T2(1-ρ)= 1200.9+2200.1 = 130ns。
> 4
> "在一个请求分页系统中，分别采用FIFO和 LRU页面置换算法时，假如一个作业的页面访问顺序为4，3，2，1，4，3，5，4，3，2，l，5，当分配给该作业的物理块数M为4时，试试写出页面访问的过程，并计算访问中所发生的缺页次数和缺页率？"

- [x]

知识点:置换算法
出处:网络
难度:1

> (1) FIFO置换算法 页面走向 4 3 2 1 4 3 5 4 3 2 1 5 物理块 4 3 2 1 1 1 5 4 3 2 1 5 4 3 2 2
> 2 1 5 4 3 2 1 4 3 3 3 2 1 5 4 3 2 4 4 4 3 2 1 5 4 3 缺页中断 √ √ √ √ √ √ √ √ √ √
> 用FIFO置换算法产生缺页次数10次 (2) LRU置换算法 页面走向 4 3 2 1 4 3 5 4 3 2 1 5 物理块 4 3 2 1 4 3 5
> 4 3 2 1 5 4 3 2 1 4 3 5 4 3 2 1 4 3 2 1 4 3 5 4 3 2 4 3 2 1 1 1 5 4 3 缺页中断 √ √
> √ √ √ √ √ 用LRU置换算法产生缺页次数8次
> 4
> "一个文件系统中有一个20MB大文件和一个15KB小文件,当分别采用二级索引和UNIX Sytem V
> 分配方案时(每块大小为2048B,每块地址用4B表示)，问: (1)各文件系统管理的最大的文件是多少?
> (2)每种方案对大、小二文件各需要多少专用块来记录文件的物理地址(说明各块的用途)？
> (3)如需要读大文件前面第5.5KB和后面（16M + 5.5KB）信息，则每个方案各需要多少次盘I/O操作? "

- [x]

知识点:文件系统
出处:网络
难度:1

> (1)各种分配方案的文件系统可管理的最大文件 ① 二级索引：由于盘块大小为2KB，每个地址用4B表示，一个盘块可存0.5K个索引表目，二级索引可管理的最大文
> 件容量为2KB×0.5K×0.5K = 0.5GB。（三级索引可管理的最大文件容量为2KB×0.5K×0.5K×0.5K = 0.25TB。） ②
> UNIX混合分配：可管理的最大文件为2KB×(10+0.5K+0.5K×0.5K+0.5K×0.5K×0.5K)=
> 20KB + 1MB+0.5GB + 0.25TB。 (2)每种分配方案对20MB大文件和15KB小文件各需要多少专用块来记录文件的物理地址? ① 二级索引：对大小
> 文件都固定要用二级索引，对15KB小文件，用一块作第一级索引，用另一块作二级索引，共用二块专用物理块作索引块，对于20MB大文件，用一块作第一级索引，用20
> 块作第二级索引，共用21块专用物理块作索引块。 ② UNIX的混合分配：对15KB小文件只需在文件控制块FCB的i_addr[13]中使用前8个表目存放文件
> 的物理块号，不需专用物理块。对20MB大文件，FCB的i_addr[13]中使用前10个表目存放大文件前10块物理块块号，用一级索引块一块保存大文件接着的0
> .5K块块号，还要用二级索引存大文件以后的块号，二级索引使用第一级索引1块，第二级索引19块。总共也需要21块专用物理块来存放文件物理地址。
> (3)为读大文件前面第5.5KB和后面第（16M + 5.5KB）信息需要多少次盘I/O操作? ①二级索引：为读大文件前面和后面信息的操作相同，首先进行一次盘I
> / O读第一级索引块，然后根据它的相对逻辑块号计算应该读第二级索引的那块，再化一次盘I / O读出信息所在盘块，这样读取大文件前面或后面信息都只需要3次盘I / O操
> 作。 ②UNIX混合分配：为读大文件前面5.5KB信息，先根据它的相对逻辑块号，在内存文件控制块FCB的i_addr[13]第二个表目中读取信息所在块块号，
> 而只化费一次盘I / O操作即可读出该块信息。为读大文件后在（16MB + 5。5KB）信息，先根据它的相对逻辑块号判断它是在UNIX二级索引管理范围，先根据i_a
> ddr[11]内容化一次盘I / O操作读出第一级索引块取得二级索引表项所在盘块号，第二次读出第二级索引块，就可以得到信息所在块块号，最后化一次盘I / O读出信息
> 所在盘块，这样总共需要3次盘I / O操作才能读出文件后面的信息。 二级索引 UNIX 管理最大文件 0.5GB 20KB + 1MB+0.5GB +
> 0.25TB
> 管理用的 专用块数 15KB文件 2 0 20MB文件 21 21 读20MB文件某处信息 5.5KB 2+1 1 16MB+5.5KB 2+1 2+1
> 4
> "试述在设有快表的分页存贮管理系统的地址变换机构和地址变换过程。"

- [x]

知识点:非连续内存分配
出处:网络
难度:1

> 在CPU给出有效地址（逻辑地址）后，系统将有效地址分离为页号和页内地址。系统将页号与页表长度进行比较，如果页号大于页表寄存器中的页表长度，则访问越界，产生越
> 界中断。 地址变换机构又自动地将页号送入高速缓存，确定所需要的页是否在快表中。若是，则直接读出该页所对应的物理块号，送入物理地址寄存器；与此同时，将有效地址
> （逻辑地址）寄存器中页内地址直接装入物理地址寄存器的块内地址字段中，这样便完成了从逻辑地址到物理地址的变换。 若在快表中未找到对应的页表项，则根据页表寄存器
> 中的页表始址和页号计算出该页在页表项中的位置，通过查找页表，得到该页的物理块号，将此物理块号装入物理地址寄存器中，与有效地址寄存器中页内地址组合成物理地址；
> 同时，把从页表中读出的页表项存入快表中的一个寄存器单元中，以取代一个旧的页表项。

> 4
> "试比较段式存贮管理与页式存贮管理异同？"

- [x]

知识点:非连续内存分配
出处:网络
难度:1

> 分页和分段系统有许多相似之处。两者都采用离散分配方式，且都要通过地址映射机构来实现地址变换。 分页和分段的主要区别：
> （1）页是信息的物理单位，分页仅仅是由于系统管理的需要；段是信息的逻辑单位,分段的目的是为了能更好地满足用户的需要。
> （2）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，决定于用户所编写的程序。 （3）分页的作业地址空间是一维的，分段的作业地址空间是二维的。 （4
> ）分页以页架为单位离散分配，无外碎片，所以也无紧缩问题；分段以段为单位离散分配，类同可变分区，会产生许多分散的小自由分区——外碎片，造成主存利用率低，需采用
> 紧缩解决碎片问题，但紧缩需化机时。 （4）分段便于处理变化的数据结构段，可动态增长；分页不能动态增长。
> （5）分段便于共享段逻辑上完整信息共享有价值提高主存利用率；分页共享困难。
> （6）分段提供动态连接的便利，运行中不用的模块可以不连接调入，节省内存空间；分页不能动态连接。
> （7）分段便于控制存取访问，段是逻辑上完整信息可根据各段信息决定存取访问权；分页存取访问控制困难。
> 4
> "画出段式存贮管理系统地址变换机构。"

- [x]

知识点:非连续内存分配
出处:网络
难度:1

> (2)在进行地址变换时，系统将逻辑地址截成段号S与段内地址d，将逻辑地址中的段号S与段表长度TL进行比较。若 S≥TL，表示段号太大，访问越界，于是产生越界
> 中断信号；若未越界，则根据段表的始址和该段的段号，计算出该段对应段表项的位置，从中读出该段在内存中的起始地址，然后再检查段内地址d是否超过该段的段长SL。若
> 超过，即d≥SL，同样发出越界中断信号；若未越界，则将该段的基址与段内地址d相加，得要访问的内存物理地址。
> 4
> "试述段页式存贮管理系统地址变换机构和地址变换过程。"

- [x]

知识点:非连续内存分配
出处:网络
难度:1

> (2)地址变换过程 在段页式系统中必需同时配置段表和页表，段表中的内容是页表始址和页表长度。
> 在进行地址变换时，系统将逻辑地址截成段号S、段内页号P与页内地址W，首先利用段号S，将它与段长TL进行比较，若S>=TL，表示越界。 若S < TL，表示未越界
> ，于是利用段表寄存器的段表始址和段号求出该段对应的段表项在段表中的位置，从中得到该段的页表始址，并利用逻辑地址中的段内页号P来获得对应页的页表项位置，从中读
> 出该页所在的物理块号b，再用块号 b和页内地址构成物理地址。
> 4
> "试述动态分区、分页和分段三种存储管理方案中如何实现信息的存储保护。"

- [x]

知识点:非连续内存分配
出处:网络
难度:1

> 1. 越界保护 在动态分区的保护的常用方法是由系统提供硬件：一对界限寄存器。这可以是上界限寄存器、下界限寄存器，或者是基址寄存器、限长寄存器。基址寄存器存
> 放起始地址，作为重定位（地址映射）使用；限长寄存器存放程序长度，作为存贮保护使用。 在分页存储管理方案中，在CPU给出有效地址（逻辑地址）后，系统将有效地址
> 分离为页号和页内地址。系统将页号与页表寄存器中的页表长度进行比较，如果页号大于页表长度，则访问越界，产生越界中断。 在段式系统存储管理方案中，在CPU给出有
> 效地址（逻辑地址）后，系统将有效地址分离为段号S和段内地址。系统将逻辑地址中的段号S与段表寄存器中的段表长度TL进行比较，若S≥TL访问越界，产生越界中断信
> 号。未越界，根据段表的始址和段长SL，计算出该段对应段表项的位置，从中读出该段在内存中的起始地址。如增补位为0，再检查段内地址d是否超过该段的段长SL，超过
> ，产生越界中断，否则，将该段的基址d与段内地址相加，得到要访问的内存物理地址。 2. 存取控制检查:存取权(R、W、E)
> 在页表项中增设“存取控制”字段，用来规定对该页的存取方式，用于标识本页的存取属性是只执行、只读，还是允许读 / 写。
> 在段表项中增设“存取控制”字段，用来规定对该段的存取方式，用于标识本分段的存取属性是只执行、只读，还是允许读 / 写。 3．环保护机构 处理器状态分为多个环，分
> 别具有不同的存储访问特权级别，通常是级别高的在内环，编号小（如0环）级别最高；可访问同环或更低级别环的数据；可调用同环或更高级别环的服务。
> 4
> "用户在使用配置UNIX/Linux 操作系统的计算机时不能将用户软盘随便插进和拿出，试从UNIX/Linux子文件系统的使用原理说明它需要一定的操作的依据
> 和操作的步骤。（写出在配置Linux/UNIX OS的计算机上使用软盘的操作步骤和简要依据。）"

- [x]

知识点:文件系统
出处:网络
难度:1

> UNIX系统只有一个安装UNIX操作系统的根设备的文件系统常驻系统，在硬盘上的其它盘区和软盘上的文件系统被安装前UNIX
> OS不知道，系统要使用其它文件系统，必须先用mount命令将其安装到系统，被安装的子文件系统的根安装到根设备树形目录的某一节点上。 子文件系统在安装时将该子
> 系统的管理块（superblock）和有关目录信息拷贝到系统缓冲区和活动索引节点表，管理块中存放该子文件系统所对应盘区的管理信息，如即将分配的空闲块号和空闲
> 索引节点号等。 子文件系统安装后进行文件读写增删，文件创建和删除等操作，其变化要记录在系统缓冲区中管理块和活动索引节点表中。 子文件系统使用完毕后要使用um
> ount拆卸命令拆卸安装上去的文件系统，在拆卸时系统将内存系统缓冲区中的管理块和活动索引节点表信息拷贝到将拆卸的子文件系统盘中，保证信息的完整性。
> 软盘的子文件系统，它需按规定使用，步骤如下：(1)插入软盘 (2)使用安装命令安装软盘文件系统 (3)读 / 写盘中文件 (4)使用拆卸命令拆卸软盘文件系统
> (5)取出软盘 如使用软盘时随便插进和拿出软盘，就可能造成软盘信息的丢失。
> 4
> "在某些系统中有这样的情况，假如某用户打开文件，向该文件中增加了若干新记录之后没有关闭文件就关机了，当他下次开机再打开文件时却无法读出所增加的新记录。试从打
> 开文件和关闭文件的作用出发，分析产生这种现象的原因，并说明编程时使用文件的操作的步骤。（写出在编程时使用文件的操作步骤和简要依据。）"

- [x]

知识点:文件系统
出处:网络
难度:1

> 由于文件的控制块（目录）存于磁盘中，有些系统为了减少在多次读写同一文件查目录时盘I/O操作次数，在读写文件前需先执行打开文件操作，它的作用是将要用到的文件目
> 录从磁盘拷到内存，在内存建立内存文件目录表。UNIX系统磁盘上的目录分成索引节点和目录文件目录项，内存文件目录表是活动索引节点表（或称内存索引节点表）和系统
> 打开文件表，分别保存已打开文件的索引节点和文件管理内容，同时在每个进程控制块的User区中设置一张用户文件描述表（又称进程打开文件表），每个打开文件在相应的
> 用户文件描述表目中存储一个指向在系统打开文件表中相应表目位置偏移的指针f。以后的读写文件操作只与内存文件目录表打交道，避免读写盘上文件目录所需的盘I/O操作
> 。同时在读写文件结束后再执行关闭文件操作，它的作用是将内存中修改过的内存文件目录表中的信息写回到磁盘中的文件目录中，避免已修改文件目录信息的丢失。 如果用户
> 打开文件后向该文件中增加了若干新记录，这时文件可能增加存储记录的物理块，相应的内存的文件目录中增加了记录数和新增物理块的地址，而这时磁盘中相应文件的文件目录
> 内容还未改变，它需在执行关闭文件后，将内存文件目录写入磁盘文件目录后才能改变。如没有关闭文件就关机，磁盘中该文件的文件目录中未记入新增的物理块的地址和新增的
> 记录数，当他下次开机再打开文件时，无法找到的新增记录的物理块的地址，也无法读出新增的新记录。
> 4
> "什么是文件共享？试述UNIX系统中文件共享的实现方法和命令的使用。"

- [x]

知识点:文件系统
出处:网络
难度:1

> 文件共享是允许不同的用户使用不同的名字名存取同一文件。 UNIX的文件共享方式有二种：(1)基于索引节点的共享方式--文件硬连接 UNIX系统将文件控制块
> FCB中文件名和文件说明分开。文件说明为索引节点，各文件索引节点集中存放在索引节点区。而文件名与索引节点号构成目录，同一级目录构成目录文件，在文件区存放。
> 为了共享文件，只是在二个不同子目录下取了不同的文件名，但它们具有相同的索引节点号。在文件的索引节点中有一个量di_nlink表示连接到该索引节点上的连接数；
> 使用命令“ln”可给一已存在文件增加一个新文件名，即文件链接数增加1。此种链接不能跨越文件系统，文件硬连接不利于文件主删除它拥有的文件。
> 命令的使用例：
> $ln/bin/ls/usr/lx20/dir(2)$利用符号连接实现文件共享$7$分系统为共享的用户创建一个$link$类型的新文件，将这新文件登录在该用户共享目录项中，这个$link$型文件包含连接文件
> ln -s
> /bin/ls /usr/lx20/dir
> 4
> "试述UNIX (Linux) 树型带勾连的目录结构和查询方法。"

- [x]

知识点:文件系统
出处:网络
难度:1

> UNIX为了加快目录的寻找速度，UNIX将文件控制块FCB中文件名和文件说明分开。文件说明为索引节点，各文件索引节点集中存放在索引节点区，索引节点按索引节点
> 号排序。而文件名与索引节点号构成目录，UNIX S V
> 操作系统的文件名14个字节，索引节点2个字节，共16个字节构成目录项。同一级目录构成目录文件，在文件区存放。
> UNIX采用文件名和文件说明分离的目录结构如下图所示： 采用文件名和文件说明分离的目录结构有利于实现文件共享，如上图所示。为了共享文件，只是在二个不同子目录
> 下取了不同的文件名ls和dir，但它们具有相同的索引节点。UNIX这种文件的结构称为树形带勾连的目录结构。
> 下面以图所示UNIX树型目录中查找文件/bin/ls为例介绍线性检索法。
> 首先系统读入根索引节点（其索引节点号为1），从文件地址项查找根目录文件所在物理块号读入内存。同时从用户提供的文件名中读入根目录下第一个文件分量bin
> ,用它与根目录文件中各个目录项的文件名顺序地进行比较，从中找到匹配号，得到匹配项的索引节点号为2。 然后将磁盘第2个索引节点读入内存，从中找出bin目录文件
> 所在物理块号，并将它读入内存。同时从用户提供的文件名中读入第二个文件分量ls，用它与bin目录文件中目录项的文件名顺序地进

> 行比较，从中找出匹配号，得到匹配项
> 的索引号为10。尔后，将磁盘第10号节点读入内存，从中判断ls文件所在的物理块号。目录查询操作到此结束，如果顺序查找过程中发现一个文件分量名未能找到，则仃止
> 查找并送回"文件未找到"信息。
> 4
> "简述UNIX对文件实现存取控制的方法和命令的使用。"

- [x]

知识点:文件系统
出处:网络
难度:1

> UNIX系统使用文件存取控制表来实现对文件存取控制，它把用户分成三类：文件主、同组用户和其它用户，每类用户的存取权限为可读、可写、可执行以及它们的组合。不同
> 类的用户对文件的访问规定不同的权限，以防止文件被未经文件主同意的用户访问。文件存取控制表存放在每个文件的文件控制块(即目录表目)中，对UNIX它只需9位二进
> 制来表示三类用户对文件的存取权限，它存在文件索引节点的di_mode中。
> Linux/UNIX可使用命令chmod改变文件或目录的存取控制权限，改变存取控制权限的操作有增加、删除某些权限和绝对地赋予某些权限。 格式： chmod
> mode filename 例：chmod 660 dante 表示使文件dante的文件主和同组用户具有读写的权限。
> 5
> 用户与操作系统的接口有 ， 两种。

- [x]

知识点:中断、异常与系统调用
出处:网络
难度:1

> 命令接口，系统调用
> 5
> 用户程序调用操作系统有关功能的途径是 。

- [x]

知识点:中断、异常与系统调用
出处:网络
难度:1

> 利用系统调用命令
> 5
> UNIX系统是 ① 操作系统，DOS系统是 ② 操作系统。

- [x]

知识点:操作系统概述
出处:网络
难度:1

> ①分时（或多用户、多任务），②单用户（或单用户、单任务）
> 5
> 现代计算机中，CPU工作方式有目态和管态两种。目态是指运行 ① 程序，管态是指运行 ② 程序。执行编译程序时，CPU处于 ③ 。

- [x]

知识点:操作系统概述
出处:网络
难度:1

> ①用户，②操作系统，③目态
> 5
> 从资源分配的角度讲，计算机系统中的资源分为 、 、 和 。操作系统相应的组成部分是 、 、 和 。

- [x]

知识点:操作系统概述
出处:网络
难度:1

> 处理机、存储器、输入 / 输出设备和文件资源；处理机管理、存储器管理、设备管理和文件系统
> 5
> 根据服务对象不同，常用的单处理机OS可以分为如下三种类型： 允许多个用户在其终端上同时交互地使用计算机的OS称为 ① ，它通常采用 ② 策略为用户服务；
> 允许用户把若干个作业提交计算机系统集中处理的OS，称为 ③ ，衡量这种系统性能的一个主要指标是系统的 ④ ； 在 ⑤
> 的控制下，计算机系统能及时处理由过程控制反馈的数据并作出响应。设计这种系统时，应首先考虑系统的 ⑥ 。

- [x]

知识点:操作系统概述
出处:网络
难度:1

> ①分时OS，②时间片轮转，③批处理OS，④吞吐率，⑤实时OS，⑥实时性和可靠性
> 5
> 实时系统通常采用 方法来提高可靠性。

- [x]

知识点:操作系统概述
出处:网络
难度:1

> 双工体制
> 1
> 计算机的操作系统是一种 。

- ( ) A.应用软件
- (x) B.系统软件
- ( ) C.工具软件
- ( ) D.字表处理软件

知识点:操作系统概述
出处:网络
难度:1

> B
> 1
> UNIX属于一种 操作系统。

- (x) A.分时系统
- ( ) B.批处理系统
- ( ) C.实时系统
- ( ) D.分布式系统

知识点:操作系统概述
出处:网络
难度:1

> A
> 1
> 操作系统是一组 程序。

- ( ) A.文件管理
- ( ) B.中断处理
- (x) C.资源管理
- ( ) D.设备管理

知识点:操作系统概述
出处:网络
难度:1

> C
> □