往年题

2018 秋-期中

一、关于 CPU 状态 (8 分)

1. **为什么要将指令集合划分为不同的特权级别?**
   答：将指令集划分为不同特权级别是为了保护系统安全和稳定运行。高特权级（内核态）执行关键操作，低特权级（用户态）执行普通应用程序，防止用户程序直接访问硬件资源或执行危险操作，避免系统崩溃和安全漏洞。
2. **下列指令哪些指令只能在内核态（管态）下执行？（请在你要选择的指令上划勾）**
   算术运算、内存读写、读日期时间、✅ 启动 I/O、✅ 修改程序状态字、访管指令、陷入指令（如 sysenter）、✅ 允许/禁止中断、取数指令
   答：只能在内核态下执行的指令：启动 I/O、修改程序状态字、允许/禁止中断。

二、关于中断、异常和系统调用 (15 分)

1. **在实现系统调用时，需要硬件体系结构提供哪些支持？在操作系统中应完成哪些工作？请以 IA32 体系结构为例，描述执行系统调用时硬件和软件的工作流程。**
   中断描述符格式：

段选择符(高16位)	DPL	P	门类型	000	空闲	段选择符	段内偏移(低16位)

   - IDTR -> IDT
   - GDTR -> GDT

   段描述符格式：

段基址	段界限	属性(包括 DPL)	段基址

   答：
   - **硬件支持**：特权级机制、中断门机制、寄存器保存现场。IA32 提供特殊指令（如 `int` 、 `sysenter` ）、IDT 和 GDT 表、特权级切换机制。
   - **操作系统工作**：系统调用表维护、参数传递接口、内核态处理逻辑实现。
   - **工作流程**：用户程序将系统调用号放入 EAX，参数放入其他寄存器（EBX/ECX/EDX 等），执行 `int 0x80` 指令；CPU 检查特权级，切换到内核态，保存现场；根据 IDTR 找到 IDT，根据中断号找到对应处理程序；系统调用处理程序从 EAX 获取调用号，从寄存器获取参数；执行内核服务；返回用户态，通过 EAX 返回结果。
2. **系统调用与函数调用有什么区别？系统调用与 API 是什么关系？**
   答：
   - **区别**：系统调用涉及特权级切换（用户态 → 内核态），函数调用在同一特权级；系统调用使用特殊指令（如 `int 0x80` ），函数调用使用 `call` 指令；系统调用开销较大，需保存更多现场信息。
   - **关系**：API 是应用程序编程接口，为应用程序提供标准化功能；系统调用是操作系统内核提供的服务；API 通常封装了系统调用，提供更友好的接口，一个 API 可能涉及多个系统调用。

三、关于进程线程模型 (18 分)

1. **进程控制块（PCB）的作用是什么？它与进程控制操作（创建、撤销、阻塞、唤醒）、进程切换、进程调度的关联是什么？它是怎么描述进程地址空间的?**
   答：
   - **作用**：记录进程状态信息，是进程存在的唯一标志，包含进程标识、状态、优先级、程序计数器、寄存器值、内存信息等。
   - **与控制操作关联**：创建进程时分配 PCB 并初始化；撤销时回收 PCB 及资源；阻塞/唤醒时修改 PCB 状态字段。
   - **与切换关联**：进程切换时保存当前进程寄存器等上下文信息到 PCB，并从下一个进程 PCB 恢复上下文；PCB 是上下文切换的核心数据结构。
   - **与调度关联**：调度器根据 PCB 中优先级、状态等信息决定调度顺序；PCB 中包含调度所需的时间片、优先级等信息。
   - **描述地址空间**：PCB 记录进程页表基址或段表基址；记录代码段、数据段、堆栈段等内存区域信息；维护虚拟地址到物理地址的映射关系。
2. **操作系统对用户级线程和内核级线程的支持有什么不同点？各列举一个支持用户级线程和内核级线程的实例操作系统名字。**
   答：
   - **用户级线程**：由用户空间线程库管理，内核不感知；切换开销小但一个线程阻塞会导致整个进程阻塞；无法利用多处理器。
   - **内核级线程**：由内核直接支持和管理；线程阻塞不影响其他线程；可利用多处理器；切换开销较大。
   - **实例系统**：支持用户级线程的系统：早期的 Solaris（Green Threads）；支持内核级线程的系统：Linux（NPTL）。

四、关于进程调度 (18 分)

1. **应用哪一种进程调度算法会导致优先级反转（置）问题？请举例描述"优先级反转（置）"的现象。如何解决这一问题?**
   答：优先级调度算法（尤其是抢占式优先级调度）会导致优先级反转问题。
   - **现象**：低优先级进程 L 持有某资源，高优先级进程 H 需要该资源而被阻塞，但中优先级进程 M 可以继续执行，间接导致 H < M，违反原本优先级设计。
   - **解决方法**：
     - 优先级继承（低优先级进程临时继承被阻塞的高优先级进程的优先级）
     - 优先级天花板（持有资源时提升到可能访问该资源的最高优先级）
     - 避免共享资源（使用无锁算法或复制资源）
2. **请叙述多级反馈队列调度算法的设计思想。说明这一定义如何处理 I/O 密集型和 CPU 密集型进程？是否会带来"饥饿"问题？如何解决?**
   答：
   - **设计思想**：根据进程行为动态调整优先级；设置多个就绪队列，不同队列优先级和时间片不同；新进程先进入高优先级队列，用完时间片后降级；优先调度高优先级队列进程。
   - **I/O vs CPU 密集型**：I/O 密集型进程在 I/O 操作前往往用不完时间片即主动让出 CPU，下次调度时仍在高优先级队列；CPU 密集型进程会用完时间片被降级到低优先级队列，减少对系统的影响。
   - **饥饿问题**：高优先级队列任务过多时，低优先级队列任务可能长时间得不到执行。
   - **解决方法**：设置老化机制（等待时间超过阈值自动提升优先级）；定期将所有进程重新放入最高优先级队列；保证每个队列都有最小执行时间份额。

2019 秋-期中

一、关于系统调用与中断异常机制 (10 分)

1. **系统调用在实现上依赖于硬件支持。请从 IA-32 硬件支持的角度，简要说明用户态程序执行系统调用的过程。**
   答：
   - 用户态程序通过软件中断（如 `int 0x80` ）或快速系统调用指令（如 `sysenter` ）进入内核态。
   - CPU 检查调用指令的目标向量（IDT 表项），找到对应的中断门或系统调用门。
   - CPU 自动完成权限检查（CPL ≤ DPL）、堆栈切换（SS:ESP → 内核栈），保存用户态现场（EIP、CS、EFLAGS 等）。
   - 控制权转移到内核态处理例程，读取系统调用号（如 EAX）、参数（如 EBX、ECX）。
   - 执行内核功能逻辑。
   - 执行 `iret` 或 `sysexit` 返回用户态，恢复用户态上下文。
2. **简述系统调用与异常（如缺页异常）两者的异同点。**
   - **相同点**：
     - 都会触发特权级切换，从用户态进入内核态；
     - 都通过中断/异常机制进入内核处理程序；
     - 都需要保存用户态现场，处理后再返回。
   - **不同点**：
     - 触发方式不同：系统调用是主动触发（指令），异常是被动触发（由错误或事件引发）；
     - 系统调用可由用户程序预期控制，异常不可预期；
     - 系统调用执行特定服务，异常则通常处理错误或特殊事件（如缺页异常引发页面调度）。

二、关于进程管理与调度 (12 分)

1. **简述进程控制块（PCB）的内容与作用。说明操作系统是如何通过 PCB 实现进程切换的?**
   答：
   - **内容**：进程标识符（PID）、进程状态、程序计数器、寄存器上下文、优先级、时间片、内存地址空间信息、打开的文件列表等。
   - **作用**：PCB 是操作系统调度和管理进程的核心数据结构。
   - **进程切换流程**：
     - 当前运行进程执行完时间片或发生阻塞，保存其 CPU 上下文至 PCB；
     - 选择下一个就绪进程；
     - 从下一个进程的 PCB 中恢复上下文；
     - 更新状态并切换到新的进程执行。
2. **请简要说明何为抢占式调度？请说明其与非抢占式调度的区别。**
   答：
   - **抢占式调度**：当有更高优先级的进程进入就绪状态或当前进程时间片用尽时，操作系统可中断当前进程，将 CPU 分配给其他进程。
   - **非抢占式调度**：当前进程一直运行到阻塞、终止或主动让出 CPU，系统才进行调度。
   - **区别**：
     - 抢占式响应更快，有利于实时性；
     - 非抢占式更简单，切换次数少，但可能导致低优先级进程饥饿。

三、关于线程实现 (10 分)

比较用户级线程（User-Level Threads, ULT）和内核级线程（Kernel-Level Threads, KLT）的实现方式、优缺点和典型操作系统支持。

项目	用户级线程 (ULT)	内核级线程 (KLT)
实现位置	用户空间线程库	内核空间
切换效率	高（不涉及系统调用）	低（需要内核调度）
阻塞影响	一个线程阻塞 → 整个进程阻塞	一个线程阻塞不影响其他线程
多核支持	不支持	支持
系统支持	GNU Pthreads（部分实现）	Linux (NPTL)、Windows

四、关于调度策略与饥饿问题 (8 分)

1. **调度策略中常出现饥饿问题。请指出一种可能导致饥饿的调度策略，并说明其原理。**
   答：优先级调度策略可能导致饥饿问题。高优先级进程持续到达，低优先级进程一直得不到 CPU 资源，长期处于就绪队列而不被调度执行。
2. **简述一种缓解饥饿问题的机制。**
   答：**优先级老化机制（Priority Aging）**：随着等待时间增加，进程的优先级逐渐提升，最终可以获得调度，避免无限期等待。

五、简答题 (10 分)

请比较系统调用接口与库函数接口的区别。

项目	系统调用接口	库函数接口
执行层级	内核态（需要切换）	用户态
开销	高（需要陷入内核）	低
依赖	操作系统提供	编译器/标准库提供
作用	调用操作系统核心服务（如文件、进程管理）	提供常用函数封装（如 `printf` 、 `malloc` ）
示例	`read()` , `write()`	`printf()` , `strcpy()`

2021 春-期中考试整理

一、环境和运行机制、系统调用 (30 分)

1. 请列举五条需要在保护模式下运行的指令。"访管指令"需要在保护模式下执行吗？为什么？(10 分)

需要在保护模式下运行的指令：

- `LGDT` （加载全局描述符表寄存器）
- `LIDT` （加载中断描述符表寄存器）
- `MOV CR0` （修改控制寄存器）
- `CLI` （清中断标志）
- `LTR` （加载任务寄存器）

访管指令必须在保护模式下执行。因为访管指令涉及特权级转换，需要保护模式提供的特权级机制来确保安全，实模式下无法提供这种保护。

2. 描述从接收中断/异常到中断处理程序运行结束的处理过程，并指出哪些是硬件实现的？哪些是操作系统实现的？中断处理程序执行结束后需要执行哪一模块？操作系统初始化和中断有什么关系？(10 分)

处理中断/异常的过程如下：

1. **硬件**保存当前上下文（保存标志寄存器、CS、IP 等）
2. **硬件**根据中断向量查找 IDT，获取中断处理程序入口地址
3. **硬件**切换到内核态，跳转到中断处理程序
4. **操作系统**保存其他现场（如寄存器等）
5. **操作系统**执行具体中断处理逻辑
6. **操作系统**恢复现场
7. **硬件**执行 `IRET` 指令，恢复标志寄存器、CS、IP 等，返回被中断的程序

中断处理程序结束后，执行 **中断返回例程**。

操作系统初始化时需要设置中断描述符表，初始化中断控制器，为系统提供中断处理的基础设施。

3. 为了实现系统调用机制，系统要做哪些工作？(10 分)

实现系统调用机制需要：

1. 定义系统调用接口和编号
2. 实现用户态到内核态的特权级切换（通常通过软中断实现）
3. 提供系统调用表，将调用编号映射到对应的内核函数
4. 实现参数传递机制（通过寄存器或栈）
5. 实现返回值传递机制
6. 提供用户库封装系统调用（如 `libc` ）
7. 实现权限检查和安全控制

二、进程线程模型 (20 分)

1. 请画出五状态进程模型，包括状态的名称以及转移条件。(10 分)

五状态进程模型：

- **新建（New）**：进程刚被创建
- **就绪（Ready）**：等待 CPU 调度
- **运行（Running）**：正在执行
- **阻塞（Blocked）**：等待某事件发生
- **终止（Terminated）**：执行完毕

状态转移：

- 新建 → 就绪：创建完成
- 就绪 → 运行：调度器选择
- 运行 → 阻塞：等待资源/I/O
- 阻塞 → 就绪：事件发生
- 运行 → 就绪：时间片用完或被抢占
- 运行 → 终止：执行完毕或被终止

2. 线程和进程有什么区别？线程有什么属性？为什么线程要有自己的栈？线程的实现方式有哪些？典型的操作系统采用哪种方式实现线程？(10 分)

区别：

- 进程是资源分配的基本单位；线程是 CPU 调度的基本单位
- 同一进程内线程共享资源；进程间相互独立
- 线程切换开销小于进程切换
- 线程没有独立地址空间；进程有

线程的属性：

- 线程 ID
- 程序计数器
- 寄存器集合
- 栈
- 状态
- 优先级

线程需要自己的栈，因为每个线程执行不同函数调用序列，需要独立的栈保存局部变量、返回地址、函数参数，保证函数调用的正确性。

实现方式：

- 用户级线程（在用户空间实现）
- 内核级线程（在内核中实现）
- 混合实现（结合两者）

典型操作系统（如 Linux、Windows）采用 **内核级线程**。

三、进程调度 (25 分)

1. 三个进程 A、B、C 的调度分析。A、B 为 CPU 密集，运行时间 1000ms；C 为 I/O 密集，每次执行 1ms CPU，10ms I/O。(10 分)

先来先服务（FCFS）：

顺序 A → B → C

- A：运行 1000ms
- B：运行 1000ms
- C：等待 2000ms 后开始运行

评价： C 响应性差，对 I/O 密集型进程不友好。

时间片轮转（RR），时间片 100ms：

- A、B 轮流执行，每轮 100ms
- C 每轮获得 1ms 后进入 I/O

评价： 公平性好，C 响应性提升，A、B 总运行时间约 2000ms。

时间片轮转（RR），时间片 1ms：

- A、B、C 每轮执行 1ms，频繁切换

评价： 切换开销大，但 C 响应性最佳。

SRTN（最短剩余时间优先）：

- C 因 CPU 使用短，总被优先执行
- I/O 时 A、B 执行
- A、B 可能饥饿

评价： C 响应性最强，A、B 长期无法完成。

2. 请设计一个多级反馈队列调度算法并回答相关问题。(15 分)

设计：

- 三个队列：Q0、Q1、Q2，优先级逐渐降低
- 时间片：Q0 = 4ms，Q1 = 8ms，Q2 = 16ms
- 新进程进入 Q0
- 若未用完时间片被抢占 → 重新排队
- 用完时间片 → 进入下一队列
- Q2 采用 FCFS
- 每隔 100ms 所有进程重置回 Q0

(1) 是否抢占？为什么?
是抢占式的。新进入高优先级队列的进程可抢占低优先级的，提高响应性。

(2) 更适合哪类进程?
更友好于 I/O 密集型进程。因为它们通常在时间片用完前进入阻塞，返回后仍有高优先级。

(3) 等待 I/O 后时间片如何处理?
返回时重进 Q0，重新获得最短时间片与最高优先级。

(4) 是否会造成饥饿？怎么解决?
可能造成 CPU 密集型进程长期在低优先级等待。解决方法是 **定期提升其优先级**（如每 100ms 回到 Q0）。