What is an operating system?

What is an interrupt? How does operating system handle interrupt?

In what cases will interrupt happen?

What is caching? How is it used?

What is direct memory access?

What are the advantages of multiprocessors systems?

Briefly describe how modern computer works.

What is multiprogramming and multitasking? What’s their difference?

What is a system call. How does OS switch between user mode and kernel mode?

What are the activities of process management?

What are the activities of memory management?

What are the activities of file management?

钟教授上课说过最多的设计原则是什么？

What services do operating system provide?

Why use APIs rather than system calls?

What kinds of system calls do OS provide? Process, file, device, memory.

What kind of system programs are there? ls, mv, cp, rm, ps, top, kill, bash, lsblk, mount, tar…

What’s the user goals and system goals to design an operating system

What is a microkernel system. Describe its features and draw the diagram of it.

**What is Kernighan’s Law?**

**“每个程序员在编写代码时，所写的代码行数与所需的代码行数成正比。”**

What are the parts of the process? What are their usages?

What is context switch?

What are the zombie and orphans in the process?

What is the producer-consumer problem?

涉及到进程之间的同步和资源管理。通过使用信号量**Semaphore**和互斥锁**Mutex**等同步机制，可以有效地解决这一问题，确保生产者和消费者能够安全、高效地访问共享缓冲区。

What are the two methods for inter-process communication? Briefly describe them.

What are the blocking and non-blocking send/receive? What are the possible combinations of them?

What are pipes? What are the two kinds of pipes?

What’s the difference between process and thread?

What are the advantages of thread over process?

**What is the Amdahl’s Law? (云计算联动)**

What are the 3 kinds of thread models? Draw a diagram for each of them. List at least one real-case operating system example for each case.

What is the thread pool? What’s the advantages of it?

What is TLS? What’s the difference between TLS and local variables?

What are the scheduling criteria for process scheduling?

**What is the convoy effect?**

Which scheduling algorithm gives the shortest average waiting time?

预测题型:Process scheduling 不同算法的次序,平均等待时间计算

What is starvation and its solution in process scheduling?

What are the algorithms used in foreground and background modes?

What’s the difference between soft and hard real-time system?

What is the deterministic modeling in algorithm evaluation?

What’s Little’s Law? If the average wait time per process is 3 seconds and normally 18 processes in queue, what’s the average arrival rate ?

What time can address binding happen?

What are the logical address and physical address? What’s the difference between them?

What is stub in dynamic linking in memory management? What’s its usage?

**存根（Stub）是一个代理或占位符，它代表了一个实际的函数或方法。在动态链接的上下文中，存根通常是一个小的代码片段，用于在程序运行时调用动态链接库中的实际实现。**

What is swapping? Briefly describe the mechanism and draw the diagram.

What are the strategies to allocate the memory in **a contiguous allocatio**n?

First fit, best fix, worst fit…

What kinds of fragmentation are there? What is 50-percent rule?

Inter & outer

What are STBR and STLR in segmentation?

**STBR（Segment Table Base Register）**

**STLR（Segment Table Limit Register）**

Access control

What is TLB? What’s its usage and structure?

What is lazy swapper? (never swaps a page into memory unless page will be needed)

When will an abort happen? (invalid reference)

What are the steps in handling a page fault?

错误识别 => save context => check page table => load / update page => continue / abort

How to solve the instruction restart problem?

* + 1. the microcode computes and attempts to access both ends of both blocks.
    2. The other solution uses temporary registers to hold the values of overwritten locations

### 1. 微代码计算并尝试访问两个块的两端

这种方法涉及到微代码的设计，具体步骤如下：

- \*\*微代码执行\*\*：在执行指令时，微处理器的微代码可以被设计为在执行过程中同时计算并访问指令涉及的两个数据块的两端。这意味着在执行指令的过程中，微代码会预先计算出可能需要的所有数据，确保在发生中断时，能够快速恢复到一个一致的状态。

- \*\*双重访问\*\*：通过同时访问两个块的两端，微处理器可以在发生异常时，确保所有相关数据都已经被处理或准备好。这种方法可以减少因中断导致的数据不一致性，从而降低指令重启的复杂性。

- \*\*状态保存\*\*：在发生中断时，微代码可以保存当前的状态，包括所有相关寄存器和内存位置的值，以便在恢复时能够准确地重启指令。

### 2. 使用临时寄存器保存被覆盖位置的值

这种方法通过使用临时寄存器来解决指令重启问题，具体步骤如下：

- \*\*临时寄存器\*\*：在执行指令时，微处理器可以使用额外的临时寄存器来保存即将被覆盖的内存位置的值。这些寄存器可以在指令执行过程中存储关键数据，以防止在中断发生时丢失这些数据。

- \*\*数据保护\*\*：当指令执行需要修改内存时，首先将要被修改的值存储到临时寄存器中。这样，即使在执行过程中发生中断，原始数据仍然可以通过临时寄存器进行恢复。

- \*\*恢复机制\*\*：在中断处理完成后，操作系统可以检查临时寄存器的内容，并根据需要恢复被覆盖的内存位置。这种方法确保了数据的一致性，并允许指令在中断后安全地重启。

### 总结

这两种方法都提供了有效的解决方案来处理指令重启问题：

1. \*\*微代码计算并访问两个块的两端\*\*：通过预先计算和访问，减少了中断时的数据不一致性。

2. \*\*使用临时寄存器保存被覆盖位置的值\*\*：通过保护关键数据，确保在中断后能够安全恢复。

结合这两种方法，可以提高系统的稳定性和可靠性，确保在发生中断或异常时，指令能够正确地重启并继续执行。

How to optimize demand page?

* + 1. Copy entire process image to swap space at process load time
    2. Demand page in from program binary on disk, but discard rather than paging out when freeing frame

在操作系统中，\*\*需求分页\*\*是一种内存管理技术，允许系统在需要时才加载页面。以下是两种优化需求分页的方法的简要介绍：

### 1. 在进程加载时将整个进程映像复制到交换空间

- \*\*预加载\*\*：在进程启动时，将整个进程的映像复制到交换空间。

- \*\*快速恢复\*\*：当进程被换出时，可以快速从交换空间恢复，减少缺页中断。

- \*\*提高性能\*\*：减少运行过程中的缺页中断，提高系统响应速度。

### 2. 按需加载页面并在释放帧时丢弃

- \*\*按需加载\*\*：仅在需要时从程序的二进制文件中加载页面。

- \*\*丢弃页面\*\*：释放内存帧时直接丢弃不再使用的页面，而不是写回交换空间。

- \*\*提高效率\*\*：减少磁盘I/O操作，提高内存利用率和系统性能。

这两种方法结合使用，可以有效优化需求分页，提高操作系统的整体性能。

What is Copy-on-Write? **Copy-on-Write** (COW) allows both parent and child processes to initially *share* the same pages in memory

What is modify(dirty) bit? What’s its usage?

* **Copy-on-Write** (COW) allows both parent and child processes to initially *share* the same pages in memory

What’s the stage of page replacement?

想要的page—free frame—换，更新页表/框架表

What is Belady’s Anomaly? 增加frame数目造成更多的page fault

What are the algorithms for page replacement? FIFO, LRU, Page buffering (free frames)

What are the counting algorithms?

**LFU Algorithm**: replaces page with smallest count

**MFU Algorithm**: based on the argument that the page with the smallest count was probably just brought in and has yet to be used

What are the allocation schemes? Fixed固定分配，留一些自由池, priority优先级

What are the difference between global and local allocation?

What is thrashing? Why does thrashing occur? **Thrashing** ≡ a process is busy swapping pages in and out Σ size of locality > total memory size

Why does demand paging work? **Locality model**

What are **memory-mapped files**? Memory-mapped file I/O allows file I/O to be treated as routine memory access by **mapping** a disk block to a page in memory

What is buddy system? Allocates memory from fixed-size segment consisting of physically-contiguous pages

* Advantage – quickly coalesce unused chunks into larger chunk
* Disadvantage - fragmentation

What is TLB reach? TLB Reach - The amount of memory accessible from the TLB

What is I/O Interlock? **I/O Interlock** – Pages must sometimes be locked into memory