自学章节回顾操作系统中异常控制流、文件系统、I/O 操作与系统调用，请在自学后回答以下问题：

1. 当系统发生缺页中断时，

①CPU 和操作系统分别做了哪些具体操作？

CPU:

1.CPU 在执行指令时，尝试访问一个未加载到内存中的页面（页表中对应的页表项未设置为有效），会触发缺页中断。

2.CPU 暂停当前执行的指令，保存当前的执行状态（如程序计数器和处理器状态寄存器），然后跳转到操作系统的中断处理程序入口。

操作系统：

1.保存上下文，操作系统保存当前进程的上下文，以便中断处理结束后能恢复。
2.检查页表，操作系统查阅页表，确定触发缺页中断的虚拟地址所属的页表项。
3.确定页面位置，操作系统检查页面是否在硬盘的某个位置（如交换区或文件系统）。如果页面不存在（即非法内存访问），则可能会引发一个段错误（Segmentation Fault）或其他异常处理。
4.分配物理内存，如果页面在硬盘上，操作系统需要在物理内存中分配一个页框。如果没有可用的页框，操作系统可能需要通过页置换算法（如 LRU、FIFO）选择一个页框，并将其内容写回硬盘。
5.加载页面，将缺失的页面从硬盘读取到分配的物理内存页框中。
6.更新页表，更新页表项，将新的物理页框地址写入页表，并设置该页表项为有效。
7.恢复上下文操作系统恢复进程的上下文，将 CPU 的控制返回给引发缺页中断的进程。
8.继续执行，CPU 继续执行中断前的指令，由于页表已更新，缺页中断不再发生，指令能够正常执行。

②如果此时有外部键盘按键事件，系统如何响应，具体的处理过程是什么？

当缺页中断处理过程中有外部键盘按键事件发生时，键盘中断信号会发出，触发**键盘中断**处理程序

1.键盘中断触发，键盘中断触发键盘按键事件通过键盘控制器发送中断信号给CPU。

2.保存上下文，CPU 接收到键盘中断信号后，暂时停止当前的中断处理（缺页中断处理），保存当前的中断处理状态，包括寄存器和程序计数器。

3.执行键盘中断处理程序

CPU 跳转到键盘中断处理程序，操作系统开始处理键盘中断。

键盘中断处理程序读取键盘缓冲区，获取按键信息。

键盘中断处理程序将按键信息存储在系统缓冲区或队列中，供操作系统或应用程序稍后处理。

4.恢复上下文

键盘中断处理程序完成后，操作系统恢复之前保存的上下文，回到缺页中断处理程序的执行状态。

5.继续缺页中断处理

操作系统继续处理缺页中断，完成缺页中断的所有步骤。

缺页中断处理完成后，操作系统恢复被缺页中断打断的进程的上下文。

6.处理键盘事件

一旦缺页中断处理完成并且系统回到正常运行状态，操作系统或相关应用程序可以从系统缓冲区或队列中读取按键信息并进行相应处理。

2. 某应用程序涉及大量读写操作，如果将其从 Linux 系统移植到目标平台，例如可信执行环境（Trust Execution Environment, TEE)，需要进行哪些改进、修正？如果参考 POSIX Specification, 可做哪些调整或设计？

TEE 提供了一个安全的运行环境，可以确保应用程序在受到攻击的情况下依然能够保护其数据和代码的完整性和机密性。

可在安全性、系统接口、文件和存储管理、并发处理、性能优化、调试和测试等方面考虑。

1. 安全性和权限管理

数据加密：确保所有存储的数据在传输和存储过程中都使用强加密算法进行加密。

访问控制：严格控制应用程序对资源（文件、设备等）的访问权限，仅允许必要的权限。

密钥管理：安全管理密钥，包括生成、存储和销毁密钥的安全策略。

2. 系统调用和接口

POSIX 兼容性：确保应用程序的系统调用兼容 POSIX 规范。许多 TEE 环境提供了有限的 POSIX 接口，确保读写操作和文件管理操作能够在这些接口下正常工作。

系统调用封装：将系统调用封装在抽象层中，以便在不同平台之间切换时能够更容易地适配不同的系统接口。

3. 文件和存储管理

安全存储：利用 TEE 提供的安全存储接口进行文件读写操作，确保数据在存储时得到保护。

文件系统兼容性：检查目标平台的文件系统兼容性，调整应用程序以适应不同的文件系统结构和特性。

4. 并发和多线程处理

线程管理：确保应用程序使用 POSIX 线程（pthread）或其他目标平台支持的线程管理接口。

同步机制：利用目标平台提供的同步机制（如互斥锁、信号量）进行线程间同步，避免竞争条件。

5. 性能优化

I/O 性能调优：根据目标平台的 I/O 特性进行性能优化，如调整缓冲区大小、使用异步 I/O 操作等。

资源利用优化：最大限度地利用目标平台的硬件特性，提高读写操作的效率。

6. 调试和测试

日志和调试工具：利用目标平台提供的日志和调试工具，对应用程序进行调试和性能分析。

测试用例调整：根据目标平台的特性，调整和增加测试用例，确保应用程序在新平台上稳定运行。

参考 POSIX Specification,可做的调整：

文件操作：

使用 POSIX 标准的文件操作函数（如 open, read, write, close）进行文件读写。

处理不同文件系统的路径和文件名差异。

进程和线程管理：

使用 POSIX 线程（pthread）进行线程创建和管理。

使用 POSIX 信号和互斥锁进行线程同步。

同步和互斥：

使用 POSIX 互斥锁（pthread_mutex_t）和条件变量（pthread_cond_t）进行线程间的同步。

使用 POSIX 信号量（sem_t）进行进程间或线程间的同步。

内存管理：

使用 POSIX 标准的内存管理函数（如 malloc, free）进行内存分配和释放。

确保内存访问在 TEE 环境中的安全性和完整性。

错误处理：

使用 POSIX 标准的错误处理机制（如 errno）进行错误检测和处理。

确保错误处理代码在目标平台上能正确工作。

输入/输出：

使用 POSIX 标准的 I/O 操作函数（如 printf, scanf）进行输入输出操作。

处理目标平台可能存在的输入输出差异。