1.简述现代计算机系统需要存储层级的原因

现代计算机系统需要存储层级的原因是为了提高系统的性能和效率。存储层级是指计算机系统中的多个层次化的存储器组件，按照速度和容量来划分，从高速、小容量的存储器到低速、大容量的存储器。

以下是现代计算机系统需要存储层级的几个主要原因：

层级化的速度和容量：不同层级的存储器具有不同的速度和容量特性。高速缓存位于 CPU 内部，速度非常快，但容量较小；主内存速度较慢，但容量较大；辅助存储器（如硬盘或固态硬盘）速度更慢，但容量更大。通过在不同层级上使用不同类型的存储器，可以在速度和容量之间取得平衡。

空间局部性和时间局部性：在计算机程序中，存在空间局部性和时间局部性的特点。空间局部性指的是程序倾向于访问相邻的存储位置，时间局部性指的是程序倾向于多次访问同一存储位置。存储层级的设计可以利用这些局部性特点，将频繁访问的数据存储在更高层级的存储器中，以便更快地访问，从而提高系统的性能。

访问时间的差异：不同层级的存储器访问时间存在差异。高速缓存的访问时间比主内存快，而主内存的访问时间比辅助存储器快。通过将经常使用的数据存储在更接近 CPU 的高速缓存中，可以减少对较慢存储器的访问次数，从而提高系统的响应速度。

成本和功耗考虑：高速、大容量的存储器比低速、小容量的存储器更昂贵。通过在存储层级中使用不同成本和功耗的存储器，可以在保证性能的前提下降低系统成本和功耗。

综上所述，存储层级的设计是为了兼顾计算机系统的性能、容量、成本和功耗等方面的需求，以提供高效的存储访问和数据管理。

2.在页式虚拟存储中，过大或过小的页分别会引起什么问题？

在页式虚拟存储中，过大或过小的页会引起不同的问题：

过大的页：

内部碎片：如果页的大小超过了实际需要的内存大小，会导致内部碎片的产生。内部碎片是指一个页中有一部分空间被浪费，没有被有效利用。这会浪费存储器资源，降低内存的利用率。

页面置换的开销：当页的大小较大时，每次进行页面置换时，需要调入或调出更多的数据，这会增加页面置换的开销和延迟。

过小的页：

外部碎片：如果页的大小过小，会导致外部碎片的产生。外部碎片是指多个小空闲区域的累积，无法满足大块内存需求。这可能导致存储器浪费，无法有效地分配和管理内存。

页表大小：页表是用于映射虚拟地址到物理地址的数据结构，如果页的大小太小，将会导致页表的大小变得很大，占用更多的内存空间和访问时间。

因此，在页式虚拟存储中，选择合适大小的页是很重要的。适当的页大小可以减少内部碎片或外部碎片的产生，并提高内存的利用效率和系统性能。

3.

（1）查阅 RISC-V 规范，简述页表条目中位 7 至位 0 各具有什么功能？

根据 RISC-V 规范，页表条目是用于页表管理的数据结构，在其中位 7 至位 0 具有以下功能：

位 0："V"（Valid）位

当该位为 1 时，表示页表条目有效，指向一个有效的物理页框。

当该位为 0 时，表示页表条目无效，可能是未分配的或无效的页。

位 1："R"（Read）位

当该位为 1 时，表示该页可以被读取。
当该位为 0 时，表示该页不可被读取。
位 2："W"（Write）位
当该位为 1 时，表示该页可以被写入。
当该位为 0 时，表示该页不可被写入。
位 3："X"（Execute）位
当该位为 1 时，表示该页可以被执行。
当该位为 0 时，表示该页不可被执行。
位 4："U"（User）位
当该位为 1 时，表示该页可以被用户态访问。
当该位为 0 时，表示该页仅限内核态访问。
位 5："G"（Global）位
当该位为 1 时，表示该页被标记为全局页，即该页在不同的地址空间之间共享。
当该位为 0 时，表示该页是本地页，仅在当前地址空间内有效。
位 6："A"（Accessed）位
当该位为 1 时，表示该页已被访问过。
当该位为 0 时，表示该页尚未被访问。
位 7："D"（Dirty）位
当该位为 1 时，表示该页已被写入过。
当该位为 0 时，表示该页尚未被写入。
这些位在页表条目中的设置和解释可用于控制和管理虚拟内存系统，以确保正确的访问权限、保护内存数据，并支持其他高级特性，如页面共享和惰性页面分配等。具体使用方法和含义可能会受到操作系统和硬件的实现方式的影响。
（2）如果用户能自由修改自己的页表，会出现什么问题？
如果用户能自由修改自己的页表，将会引发严重的安全和稳定性问题。下面是几个可能的问题：
安全漏洞：通过修改页表，用户可以将某个本不应该被访问的内存区域映射到可访问的地址空间，从而绕过安全机制。这可能导致越权访问、恶意软件的执行以及系统的不稳定性。
内存损坏：用户对页表的随意修改可能导致内存的错误访问或者破坏，例如将一个页面映射到多个地址或者将不相干的页映射到同一个地址。这种行为会破坏内存的一致性，导致系统崩溃、数据损坏和程序错误。
资源浪费：用户对页表的随意修改可能导致内存资源的浪费。例如，用户可以将大量的虚拟内存映射到实际物理内存上，占用过多的系统资源，导致其他进程无法获得足够的内存空间。
无法预测的行为：由于用户可以自由修改页表，系统无法对其行为进行有效的控制和管理。这导致了无法预测的系统行为，使得系统变得不可靠和不稳定。
为了解决这些问题，现代操作系统采用了虚拟内存机制，通过在内核态下控制页表的修改，将对页表的访问和修改限制在特权级别较高的操作系统内部。这样可以确保页表的一致性、安全性和稳定性，并为不同的用户和进程提供合适的虚拟内存空间，实现资源隔离和保护。
（3）在 RISCV 的虚拟内存管理中，一个 X/W/R 全部为 0 的有效页表条目有什么含义？
在 RISC-V 的虚拟内存管理中，一个有效页表条目的 X（Execute）、W（Write）和 R（Read）位全部为 0 时，具有以下含义：
禁止执行：X 位为 0 表示该页不可被执行，即禁止在该页上执行指令。这种设置可以用于保护敏感代码或数据，防止它们被误执行。

禁止写入：W 位为 0 表示该页不可被写入，即禁止对该页进行写操作。这样的设置可以用于保护只读数据，防止其被修改。
禁止读取：R 位为 0 表示该页不可被读取，即禁止从该页中读取数据。这种设置可以用于保护机密数据，确保只有特定的权限或特权级别才能读取该页的内容。
当一个有效页表条目的 X/W/R 位全部为 0 时，通常表示该页是一个权限受限的页，具有只读、只执行或不可访问的特性。这种设置可以提供更细粒度的内存访问权限控制，确保对内存的合法访问，并增加系统的安全性。
需要注意的是，具体的虚拟内存管理方案和操作系统实现可能会有所不同，因此，对于这样的页表条目含义的准确解释应结合特定的实现和上下文来理解。建议在具体应用中参考相关的文档和规范以获得更准确的信息。

4.
（1）在页表条目已经存在 X/W/R 位的情况下，PMP 控制寄存器中的 X/W/R 位有什么作用？
在 RISC-V 架构中，PMP 控制寄存器用于实现物理内存的保护和访问控制。对于一个页表条目已经存在 X/W/R 位的情况，PMP 控制寄存器中的 X/W/R 位可以进一步限制对该物理内存区域的访问权限。
具体来说，PMP 控制寄存器中的 X/W/R 位用于指定对应物理内存区域的执行、写入和读取权限。当某个 PMP 控制寄存器的 X/W/R 位设置为 1 时，表示对应的物理内存区域可以进行对应的操作；而当对应的位设置为 0 时，表示对应的物理内存区域不允许进行对应的操作。
PMP 控制寄存器的设置可以用于实现更细粒度的内存保护和访问控制。它可以与页表条目的 X/W/R 位相结合，提供更灵活的权限管理。例如，即使页表条目中的 X/W/R 位允许对某个页进行读取和写入操作，但通过 PMP 控制寄存器将对应的物理内存区域的写入权限设置为 0，就可以进一步禁止对该区域的写入操作。
需要注意的是，具体的 PMP 控制寄存器的功能和使用方式会根据不同的 RISC-V 实现和操作系统而有所不同。因此，在具体的应用中，建议参考相关的文档和规范，以了解特定系统中 PMP 控制寄存器的详细作用和配置方法。
（2）PMP 寄存器中的 L 位和 A 位有什么作用？
在 RISC-V 架构中，PMP 控制寄存器中的 L 位（Locked）和 A 位（Access）具有以下作用：
L 位（Locked）：当 L 位为 1 时，表示对应的 PMP 控制寄存器条目被锁定，不可被修改。这样可以确保关键的内存保护设置不会被非授权的操作修改。锁定后的 PMP 控制寄存器条目只能通过特权级别较高的操作或者硬件复位来解锁和修改。
A 位（Access）：A 位用于记录对应的 PMP 控制寄存器条目所映射的物理内存区域是否被访问过。当内存访问发生时，硬件会根据对应的 PMP 控制寄存器条目的 A 位进行设置。这可以用于监控和记录对物理内存区域的访问情况，以实现访问追踪或者性能分析。
PMP 控制寄存器的 L 位和 A 位是可选的，并不是所有的 RISC-V 实现都必须支持这些功能。具体的 PMP 控制寄存器功能和位字段定义可能会有所不同，因此，对于这些位的详细含义和作用，需要参考具体的 RISC-V 实现、文档和规范。
需要注意的是，PMP 控制寄存器的配置和使用涉及特权级别的操作和系统级别的设置，因此需要在合适的权限和上下文中进行，并且需要遵循相关的规范和指导。

5.
（1）如果页大小为 4KB，每个页表条目使用 8 字节空间，内存系统按字节寻址，则使用完整的 64 位虚拟地址时，一个单级页表系统需要多大的空间用于存储页表?

如果页大小为 4KB，每个页表条目使用 8 字节空间，并且内存系统按字节寻址，则一个单级页表系统需要多大的空间用于存储页表可以通过计算来确定。
首先，计算每个页表条目所对应的页框数：
每个页表条目占用 8 字节空间，而每个页框大小为 4KB（即 2^12 字节）。因此，每个页表条目对应的页框数为：2^12 / 8 = 2^9 = 512。
接下来，计算需要多少个页表条目来映射所有的虚拟页表：
由于每个页表条目对应一个页框，而一个单级页表系统需要映射的虚拟页表数等于物理内存的页框数，因此需要的页表条目数为：2^64 / 2^12 = 2^52。
最后，计算页表所需的总空间：
每个页表条目占用 8 字节空间，需要的页表条目数为 2^52 个，因此，页表所需的总空间为：8 * 2^52 = 2^55 字节。
以字节为单位，一个单级页表系统需要 2^55 字节的空间来存储页表。
（2）实际上，多数真实系统仅限制使用 64 位系统的一部分位作为有效的访存空间，例如 Sv48 即仅使用 48 位的虚拟地址空间，则保持其他假设不变时，一个单级页表系统存储页表所需要的空间被降低到多少？
如果使用 48 位的虚拟地址空间（如 Sv48），则需要重新计算单级页表系统存储页表所需的空间。
根据假设，使用 48 位的虚拟地址空间，则只有 48 位用于寻址，而不是完整的 64 位。
首先，计算每个页表条目所对应的页框数，方法与之前相同：
每个页表条目占用 8 字节空间，每个页框大小为 4KB（2^12 字节）。因此，每个页表条目对应的页框数为：2^12 / 8 = 2^9 = 512。
接下来，计算需要多少个页表条目来映射所有的虚拟页表：
使用 48 位的虚拟地址空间，需要映射的虚拟页表数等于物理内存的页框数，即：2^48 / 2^12 = 2^36。
最后，计算页表所需的总空间：
每个页表条目占用 8 字节空间，需要的页表条目数为 2^36 个，因此，页表所需的总空间为：8 * 2^36 = 2^39 字节。
以字节为单位，一个单级页表系统需要 2^39 字节的空间来存储页表，这是在使用 48 位虚拟地址空间的情况下。
（3）多级页表为什么可以降低虚拟内存系统的实际页表存储开销？
多级页表可以降低虚拟内存系统的实际页表存储开销，主要有以下几个原因：
分层结构：多级页表采用了分层的结构，将整个虚拟地址空间划分为多个较小的区域。每个层级的页表仅负责映射其对应的区域，而不是整个地址空间。这样可以大大减小每个页表的大小和存储开销。
延迟分配：多级页表允许按需分配页表条目。当发生页面访问时，只有当前所需的页表条目才会被分配和加载到内存中。其他层级的页表条目可以延迟分配，只在需要时才进行加载。这种延迟分配的机制减少了实际内存中存储页表所需的空间。
稀疏性：由于虚拟地址空间中的大部分区域可能是未分配或未使用的，多级页表充分利用了页表的稀疏性。只有被映射的虚拟页才需要对应的页表条目。未映射的虚拟页不需要占用实际内存中的页表空间，从而减少了存储开销。
页表共享：多级页表可以实现页表的共享，即多个虚拟地址空间共享同一级的页表。这在多个进程或线程之间共享相同的代码或只读数据时非常有用。通过页表共享，可以节省存储开销，并提高内存利用率。
总体而言，多级页表通过分层结构、延迟分配、稀疏性和页表共享等机制，有效地降低了虚

拟内存系统的实际页表存储开销。这使得虚拟内存系统能够管理大型的虚拟地址空间，提供灵活的内存管理和保护，并在不浪费大量内存资源的情况下，满足各个进程的需求。