1. [20 pts] Read Chapter 2 of “Three Easy Pieces” (https://pages.cs.wisc.edu/~remzi/OSTEP/intro.pdf). Answer the following questions:
(1) What are the “three easy pieces” of operating systems? Explain each of them with your own words.
(2) How do these “three easy pieces” map to the chapters in the “dinosaur book”?
A:(1)它们分别是虚拟化(virtualization)、并发(concurrency)和持久性(persistence)。虚拟化(virtualization)指的是操作系统将一些物理资源(physical resources)（如处理器、内存、磁盘等）转化为更通用且更强大的虚拟化资源。例如通过虚拟化实现多个程序同时运行(共享 CPU)，通过虚拟化实现多个程序同时访问指令和数据(共享内存)，通过虚拟化等实现多个程序访问设备(共享磁盘)。并发(concurrency)指的是一系列由于并发同时处理多件事情时产生的问题或者简单来说是同时干很多事情。例如当两个进程同时修改同一块内存区域的值时，由于读取、修改、存储三条指令并非原子指令，所以可能由于指令间出现进程切换或其他原因，而导致出现与实验预期不符的结果，这就是由并发所产生的问题。持久性(persistence)指的是操作系统有时需要持久的保存一些数据，以免在断电或系统崩溃时丢失数据。如通过文件系统管理在磁盘上长期保存的数据，包括读取、写入等。
(3) 虚拟化(virtualization)最初在 Chapter1 的 1.7 进行了引入介绍，而后在 Chapter10 介绍了虚拟化中的内存虚拟化，在 Chapter18 介绍了虚拟机，并在 18.6 介绍了操作系统组件和虚拟化的联系。并发(concurrency)在 Chapter4 的线程与并发(Threads & Concurrency)中进行了介绍解释。最后，持久化(persistence)在“dinosaur book”中相关的部分大致有 Chapter11 的 Mass-Storage Structure 相关内容以及 Chapter9 的 Main memory 相关内容，还有 Chapter13、14、15 的 File system 相关内容。

2.[20 pts] Read Chapter 6 of “Three Easy Pieces” (https://pages.cs.wisc.edu/~remzi/OSTEP/cpu-mechanisms.pdf) and explain what happens during context switch in detail?
A：context switch 简单来讲是为当前正在执行的进程保存一些相关寄存器的值，然后为即将执行的进程恢复一些相关寄存器的值。详细来讲，当进行 context switch 时，操作系统会执行底层汇编代码，首先保存通用寄存器(general purpose register)，程序计数器(PC)，当前运行程序的内核栈指针(kernel stack point)，然后恢复寄存器和程序计数器的值，将内核栈指针修改为指向下个需要执行的进程的内核栈。然后执行 return-from-trap，接下来刚刚设置的下个需要执行的进程开始执行。至此，context switch 过程结束。

3. [20 pts] Read slides “L03 Processes I” and “L04 Processes II” and answer the following questions:
(1) Explain what happens when the kernel handles the fork() system call (hint: your answer should include the system call mechanism, PCB, address space, CPU scheduler, context switch, return values of the system call).
(2) Explain what happens when the kernel handles the exit() system call (hint: your answer should include discussion on the zombie state and how it is related to the wait() system call).
A：(1)fork()的机制用于创建新的进程，并将父进程的相关内容复制给子进程，且通过父进程 fork()返回值为子进程 PID、子进程 fork()返回值为 0 来区分父子进程，此外如果返回值为 -1 代表创建失败。在 fork()过程中，子进程复制父进程的 PCB，并更新相关内容（如 PID，fork()返回值等），而父进程 User space 的内容如物理内存复制有两种情况，一种是将相关内容全部复制到子进程的新的物理内存地址（较为低效）；另一种是子进程指向与父进程的同一块物理内存空间，如在代码、全局变量等内容上父进程与子进程共享一块内存区域，子

进程可以共享读取内容，在父或子进程需要修改写入时再将内容复制一份到子进程的内存空间（较为高效）。其中第二种复制方式被称之为 copy-on-write。子进程创建完毕之后，会被添加至 running list，被 CPU scheduler 所调度。而进程调度就会涉及到上下文切换(context switch)，值得注意的是不能确定接下来就是子进程开始运行，比如当父进程结束后，按照 running list 会执行列表中的下一个进程（不一定是刚刚创建的子进程），也有可能父进程执行 wait()系统调用将自己挂起直至子进程运行结束再继续执行父进程。因此，上下文切换所涉及的进程要视具体情况而定。
(2)当执行 exit()时，kernel 会释放所有该进程所申请的内存，并且关闭所有该进程打开的文件，然后该进程对应的 user space memory 的所有内容会被释放（包括程序代码和其申请的内存），此时该进程的 PID 还是保留着，该进程的状态被设置为 zombie，最后 kernel 会发送一个 SIGCHLD 信号给该进程的父进程用于告知其子进程的结束，而该 zombie 进程会保留在 task list 内等待其父进程来处理，至此 exit()结束。下面讨论 exit()与 wait()的关系，在默认情况下父进程会无视子进程结束的信号，但倘若父进程曾执行 wait()将自己挂起，则该父进程将会被 kernel 唤醒来给其子进程“收尸”（释放 kernel space 的内容，如将它从进程表和任务列表中剔除），之后父进程将再次被设置为默认不响应 SIGCHLD。这是 exit()发生在 wait()之后的情况，而当子进程先 exit()后，父进程再执行 wait()时，父进程会识别到 kernel 已经给它发送过 SIGCHLD 信号了，于是父进程会立刻开始给对应的子进程“收尸”，“收尸”的操作与上面所讲的相同。

4. [20 pts] What are the three methods of transferring the control of the CPU from a user process to OS kernel? Compare them in detail.
A：三种使内核从用户进程处得到 CPU 控制权（即从用户态切换到内核态）的方式分别是：该进程进行系统调用，该进程产生异常，中断产生。第一种方式相当于用户进程主动转交 CPU 控制权给内核，因为用户进程需要借用内核的权限去做一些只有内核态能做的事情，如从硬盘读取数据，从键盘获取输入等。第二种方式是当用户进程在执行的过程中产生了异常事件，如由于缺少某虚拟地址到物理地址的映射而产生的缺页异常，此时 CPU 的控制权被转交至内核，由相关内核程序处理异常。第三种方式是在用户进程执行过程中出现了需要内核干涉处理的中断事件，如外围设备在完成用户请求操作后会发送中断信号给内核，此时 CPU 控制权转交至内核，由内核处理完中断后再跳转回原先触发中断的用户进程指令。值得一提的是，第一种方式的系统调用实现的机理是使用了操作系统为用户设计的一个特殊中断，进而由内核识别该特殊中断进行相关系统调用，从而实现由用户态切换到内核态。

5. [20 pts] Describe the life cycle of a process (hint: explain the reasons for process state transitions).
A：首先一个进程新创建后处于 new 状态，在获取除 CPU 相关资源后，会被允许进入 ready 状态等待 CPU scheduler 的调度，当被调度后被分配到 CPU 资源就会进入 running 状态。而当进程处于 running 状态时，接下来该进程可能过渡到三种状态：1.若进程触发中断，则又会回到 ready 状态；2.若进程需要进行 I/O 或事件等待，则会进入 waiting 状态；3.若进程执行完毕后退出（exit），则会进入 terminated 状态。当进程从 running 状态进入 waiting 状态后，若 I/O 或事件等待结束，则会重新回到 ready 状态，等待 CPU scheduler 调度。