CS302 Assignment 3

12011702 张镇涛

Q1

(1)

"Three Easy Pieces" 指的是 **virtualization, Concurrency, Persistence**

Virtualization

书上原话：

> The primary way the OS does this is through a general technique that we call virtualization. That is, the OS takes a physical resource (such as the processor, or memory, or a disk) and transforms it into a more general, powerful, and easy-to-use virtual form of itself. Thus, we sometimes refer to the operating system as a virtual machine.

虚拟化(Virtualization)是指操作系统将硬件的物理资源进行转换，转换成一个更加通用，强大和便捷的虚拟形式，从而实现底层硬件与顶层应用之间的隔离，对资源的访问提供统一的接口。

Concurrency

书上原话：

> Another main theme of this book is concurrency. We use this conceptual term to refer to a host of problems that arise, and must be addressed, when working on many things at once (i.e., concurrently) in the same program. The problems of concurrency arose first within the operating system itself; as you can see in the examples above on virtualization, the OS is juggling many things at once, first running one process, then another, and so forth. As it turns out, doing so leads to some deep and interesting problems.

并发性(Concurrency)是指操作系统同时处理多个进程的运行，但是在某一时刻只能运行一个进程，在下一时刻运行另一个，以此在多个进程间进行切换。

Persistence

书上原话：

> The third major theme of the course is persistence. In system memory, data can be easily lost, as devices such as DRAM store values in a volatile manner; when power goes away or the system crashes, any data in memory is lost. Thus, we need hardware and software to be able to store data persistently; such storage is thus critical to any system as users care a great deal about their data.

持久性(Persistence)是指需要硬件和软件对于数据持久化保存的能力，防止数据因为断电等原因在内存中丢失。

(2)

Virtualization: Chapter 3, 5, 9-10
Concurrency: Chapter 4, 6-8
Persistence: Chapter 11-15

Q2

书上原话：

> If the decision is made to switch, the OS then executes a low-level piece of code which we refer to as a context switch. A context switch is conceptually simple: all the OS has to do is save a few register values for the currently-executing process (onto its kernel stack, for example) and restore a few for the soon-to-be-executing process (from its kernel stack). By doing so, the OS thus ensures that when the return-from-trap instruction is finally executed, instead of returning to the process that was running, the system resumes execution of another process.

因此上下文切换的主要步骤：

1. 将当前执行进程的上下文（CPU寄存器, PC）保存到内核的栈上
2. 从内核栈上恢复将要切换的进程上下文
3. 当从trap返回时跳转到PC寄存器指向的新地址执行新的进程

Q3

(1)

system call mechanism:

`fork()` 的系统调用将会创建一个与父进程几乎完全一样的新子进程。

当`fork()`系统调用的时候，操作系统需要进入内核态。此时内核会进行新的子进程的创建，当创建结束时，会重新回到用户态。

PCB:

PCB是进程控制块，当`fork()`调用的时候，父进程会将PCB信息拷贝给子进程, 子进程的PCB仅更新PID, 指向父节点的指针等信息。

address space:

通常，内核为子进程创建一个新的地址空间，这个地址空间是父进程地址空间的拷贝，但是它们是分离且独立的。子进程会从和父进程相同的地方开始执行，但是他们的地址空间是分离的。

实际上，更准确来说，Linux 的fork()采用Copy on Write(写时拷贝), 这是一种可以推迟甚至避免拷贝数据的技术。内核此时并不复制整个进程的地址空间，而是让父子进程共享同一个地址空间。只用在需要写入的时候才会复制地址空间，从而使各个进程拥有各自的地址空间。

CPU scheduler:

在`fork()`创建了子进程后，内核将进程加到ready队列中，CPU调度器它会决定下一步哪个进程先运行。

context switch:

当`fork()`系统调用时，需要内核进行上下文切换，此时需要保存和加载进程上下文数据 (在对应PCB中)，然后跳转到新的程序计数器继续执行。

return values of the system call:

如调用`fork()`的是父进程，则返回的值生成的子进程的PID；对于子进程，则返回值为0。

如返回结果是-1，则说明生成子进程失败。

(2)

what happens when the kernel handles the exit() system call

`exit()`主要的流程是：

1. 关闭该进程打开的文件列表
2. 内核释放所有该进程被分配的内存
3. 释放该进程相关的用户空间的内存（包括program code以及分配到的内存）

但是进程的PID仍然保留在内核的进程表中。

discussion on the zombie state

僵尸态是指进程所依附的资源已经释放，但是其PID仍然没有被回收。

内核将子进程的`SIGCHLD`通知父进程，等待父进程处理。

how it is related to the wait() system call

与父进程wait()有无有关：

子进程在exit()后内核会通知父进程`SIGCHLD`信号。

如果父进程**在wait()状态**：

内核为父进程注册一个信号处理流程。

处理流程：首先接收并移除`SIGCHLD`信号，其次在内核空间中移除子进程，从而解除“僵尸态“。

最后内核取消注册信号处理流程，并且将移除的子进程的PID作为wait()返回值。

如果父进程**没有处于wait()状态**：

子进程通知的`SIGCHLD`无效，父进程不会处理。子进程将保持“僵尸态”。

Q4

Three methods of transferring the control of the CPU from a user process to OS kernel

1. 系统调用
2. 异常
3. 中断

比较:

系统调用：

程序主动向操作系统的请求，比如像exit(), fork()等，这将会从用户态转变为内核态，在内核处理完之后，再重新返回到用户态。

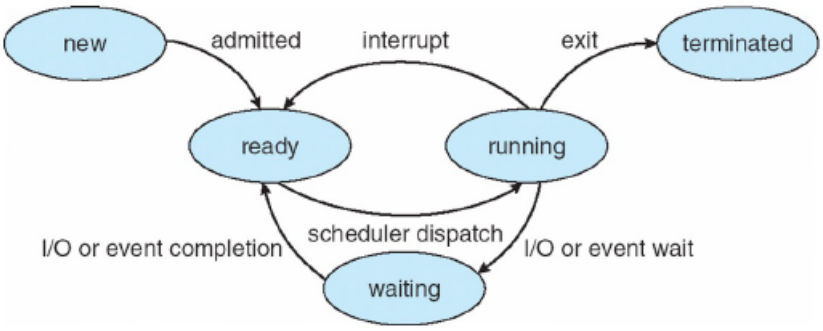
异常：

由程序内部预料之外的行为，比如说非法的地址访问，除数为0等产生，当异常产生时系统切换到内核态后由对应handler处理这个异常。

中断：

由外部设备引起，比如说时钟中断，或者是I/O中断，其独立于用户进程，当中断产生时系统切换到内核态后由对应handler处理这个中断。

Q5



State	Description
new	进程PCB已经被创建，但是没有分配资源
ready	进程已经被分配资源，但是没有在CPU上运行，正在队列中等待调度
running	进程正在CPU上运行
waiting	进程由于I/O或某些事件，不能继续执行，会释放CPU占用并进入阻塞状态
terminated	进程结束或异常终止

reasons for process state transitions：

new -> ready: 操作系统为进程分配资源进入队列等待调度器调度

ready->running: 进程根据CPU调度成为下一个被执行的进程

running->waiting: 进程请求I/O或其他事件，需要进入waiting状态

waiting->ready: 进程结束等待I/O或者其他事件完成，进入ready队列等待调度运行

running->ready: 当前进程被调度，进入ready队列重新等待再次执行

running->terminated: 进程完成它的任务并退出