作业三：进程与线程

毛浩宇，PB16061023

计算机在工作过程中需要不停地执行程序。本次作业总结程序运行的过程，也就是进程与线程的概念。

1. 什么是进程？进程与程序的关系是什么？

程序是一个可以被计算机执行的文件。它是静态的，并经常与若干动态链接库关联。而进程是一个正在被执行的文件，它是动态的，并且已被载入内存，在内存中拥有属于自己的数据。没有进程，计算机不能执行任何东西。

2. 操作系统掌握进程的哪些信息？

进程的执行必然需要经过操作系统的管理。每一个进程在操作系统内部用进程控制块（PCB）来表示。PCB 中储存进程编号（PID）、进程的状态（比如等待、就绪或执行中）、程序计数器（下一条指令的地址）、寄存器信息、内存管理信息、I/O 状态信息等。PCB 在用双向链表来存储。

3. 操作系统是如何创建进程的？创建进程之后进程是如何执行的？

在知道以上概念之后，我们可以开始探讨操作系统如何创建一个进程了。

所有的进程都由一个父进程创建并管理。所以很明显，进程间的关系可以用树来表示。树的根节点，也就是第一个进程，是 Init 进程。它直接或间接地创建了所有进程。有些操作系统在创建进程时，还可以设置一些选项，比如资源共享，执行选项，和地址分配选项等。

在 Unix 系统中，创建进程主要有 fork 和 exec 两种方式。

当我们执行 fork 系统调用时，进程将会被一分为二。fork 操作会把除内核中存储的进程信息之外的内容复制给子进程。因此 fork()之后的代码段将分别被两个进程执行两次。如果处理器是单核的，操作系统将先执行父进程后执行子进程。父进程与子进程的 fork 函数的返回值有差异。对父进程，调用将返回子进程的 ID，对子进程，调用将返回 0。一般用这个返回值来确定哪些操作由父进程执行，哪些操作由子进程来执行。

exec 系统调用可以运行一个与本进程无关的程序。exec()的执行流程如下：先找到要执行的程序文件，然后清空原有进程的所有常量、局部变量、全局变量，清空所有代码与动态分配存储，复位寄存器。这表现为原有进程被覆盖或替换了，因此原进程的代码将不会被继续执行，但进程编号等信息不变。

4. 我们如何使用系统调用来严格控制进程运行的次序？

我们可以观察到在 fork 函数执行之后先执行父进程后执行子进程。如果我们想要调整代码执行的次序，比如在子进程结束运行之后继续执行父进程，或者让父进程在某处暂停，等待子进程的返回值，那该怎么办呢？

这时我们需要用到 wait 函数。如果在需要等待子进程的位置加入一个 wait 函数，那么父进程在运行到该位置时将被挂起，当一个子进程结束后再被唤起。如果子进程在父进程执行到 wait 函数之前就已结束，那么 wait 函数就不会挂起父进程，子进程将会被彻底的清除掉。

wait 函数不能等待特定的子进程。为了解决这个问题，我们有 waitpid 这个函数。这个函数可以让我们等待特定 PID 的子进程。

5. 如何结束一个进程？

由于内核的空间比较紧张（PID，进程数量具有上限），因此子进程在执行完毕之后需要结束以便为内核腾出空间。当子进程运行 exit 函数之后，子进程在内核中存储的几乎全部信息和所有进程资源都会被清除，并通过 wait 函数向父进程发送信号，通知子进程已被结束。然后父进程会把子进程从进程表中删除。如果不删除，就会变成僵尸进程，没有代码运行，

还占用内核空间。由此可见，子进程的结束是部分依赖父进程的。如果在子进程结束之前父进程就已经结束了呢？在 Unix 系统下，Init 进程会接管子进程。Init 函数会周期性地调用 wait 函数，以清除系统中的僵尸进程。

6. 什么是线程？我们为什么要使用线程？

很多情况下，一个程序的功能比较复杂，需要分解为多个子任务；一个服务器需要同时为多个客户端提供服务，然而为此单独开辟一个进程所消耗的资源过大。这时就需要一个轻量级的进程——线程。

7. 线程具有哪些特点？有哪些分类？

线程之间的代码是共享的，线程之间共享所有全局变量区和动态变量区，可以相互访问内存，只有局部变量是独立的。这样做的好处是可以多任务同时处理，响应速度快，内存占用少，避免了不同进程之间相互切换造成的资源浪费。但是这样做也有其挑战：分配与平衡任务、数据分核处理需要恰到好处的控制。线程分为异步线程与同步线程。异步线程指父线程在子线程释放之后继续执行，这种情况下数据共享很少；同步线程指父进程等待所有子线程结束之后才继续运行。一般这种情况下数据共享很多，存在依赖关系，类似于 fork。

8. 那么操作系统是如何调度线程的呢？

操作系统调度线程有三种方式：多对一，一对一以及多对多。

在一对一调度方式中，多个线程在内核中被视为一个进程。这样做便于内核管理，但是如果一个线程被阻塞，其余的线程也不能工作。这降低了系统的处理效率。在一对一的调度方式中，每一个线程在内核中是独立的。这使操作系统可以独立的处理每一个线程，虽然增加资源消耗，但是具有很高的并发性，提高了工作效率。而多对多的调度方式可以描述为一个多个线程到多个内和结构的映射。它也具备很高的并发性，但是管理复杂，很少使用。

9. 线程的隐式实现

一个程序可能包含成百上千的线程。如果所有的线程都要显式地实现，那么程序的准确性往往难以保证。这时需要一种简单易行的进程实现方式——进程的隐式实现。

创建线程也是需要开销的。然而操作系统一般来说同时会存在数量很多的线程，一个一个地当场创建很耗时间，这时该如何创建线程呢？一种常用的方式是操作系统提供一个线程池。一个线程池已经提前创建好若干个线程。如果用户需要使用线程，就从线程池中取出线程使用，如果线程结束，就将其放回线程池中等待下一次使用。如果线程池中没有空余的线程，就只能等待某一线程结束之后才能创建新的线程。隐式实现还有另一种实现方式是 OpenMP，这是一套编译处理方案，可以让程序员容易地实现并行运算。

10. 线程在使用过程中还有哪些其他问题需要强调？

首先讨论创建新进程时如何处理线程的问题。在使用 fork 系统调用时，操作系统会复制所有线程；在使用 exec 系统调用时，操作系统会把所有的线程都清除并覆盖掉。

然后讨论信号处理问题。如果一个进程想给线程发送信号，那么哪一个线程会接受信息？一般来说操作系统有不同系统调用实现不同的功能，使信息能够准确地接收。

还有线程的局部变量存储问题。一般来说局部变量只能在函数内部可见。如果需要跨函数的存储共享，需要一些特殊手段，比如 TLS。

最后是线程的终止问题。线程的终止有两种方式：延迟终止与异步终止。异步终止是指线程在收到终止信号之后立即结束。这种方案的问题在于因为线程涉及资源共享，突然终止很容易造成管理混乱。延迟终止是指先检查要终止的线程是否满足结束条件才进行终止。这种方式比较安全。

11. 总结

操作系统要高效地完成任务，必须要合理地管理好进程与线程。进程与线程的实现与调度会对操作系统的高效运转产生巨大影响。