1. 异常处理

1) 异常的分类

类别	原因	异步 / 同步	返回行为
中断	来自 I/O 设备的信号	异步	总是返回到下一条指令
陷阱	有意的异常	同步	总是返回到下一条指令
故障	潜在可恢复的错误	同步	可能返回到当前指令
终止	不可恢复的错误	同步	不会返回

a) 中断：来自处理器外部的 I/O 设备的信号。I/O 设备，例如网络适配器、磁盘控制器和定时器芯片

b) 陷阱：预先安排的事件（“埋地雷”），如单步跟踪、断点、系统调用（执行访管指令）等。是一种自愿中断。陷阱最重要的用途是在用户程序和内核之间提供一个像过程一样的接口，叫做系统调用。

   i. “syscall n”指令，会导致一个到异常处理程序的陷阱，这个处理程序解析参数，并调用适当内核程序。

   ii. 系统调用运行在内核模式中，内核模式允许系统调用执行特权指令，并访问定义在内核中的栈。

c) 故障：执行指令引起的异常事件，如溢出、非法指令、缺页（可恢复）、访问越权等。

   i. 处理程序要么重新执行引起故障的指令（已修复），要么终止（可能返回到当前指令）。

d) 终止：硬故障事件，此时机器将“终止”，调出中断服务程序来重启操作系统。例：非法指令，奇偶校验错误，机器检查。

   i. 中止当前程序

   ii. 非法内存引用：OS 发送 SIGSEGV 信号给用户进程（不尝试恢复），用户进程以“段错误”退出。

2. 进程

1) 上下文切换：OS 通过处理器调度让处理器轮流执行多个进程。实现不同进程中指令交替执行的机制称为进程的上下文切换。

a) 进程由常驻内存的操作系统代码块（称为内核）管理

   i. 内核不是一个单独的进程，而是作为现有进程的一部分运行。

b) 控制流通过上下文切换从一个进程传递到另一个进程。

c) 上下文就是内核重新启动一个被抢占的进程所需的状态。它由一些对象的值组成，这些对象包括通用目的寄存器、浮点寄存器、程序计数器、用户栈、状态寄存器、内核栈和各种内核数据结构，比如描述地址空间的页表、包含有关当前进程信息的进程表，以及包含进程已打开文件的信息的文件表。

2) 进程控制

a) 创建和终止进程

i. fork 函数：子进程返回 0，父进程返回子进程的 PID（fork 函数被调用一次，却返回两次）
    1. 子进程与父进程并发执行
    2. 不能预测父进程与子进程的执行顺序的
    3. 子进程得到与父进程虚拟地址空间相同的（但是独立的）一份副本
    4. 共享文件（stdout...）
    5. 用进程图刻画 fork--拓扑排序
ii. exit 函数：exit 函数以 status 退出状态来终止进程（另一种设置退出状态的方法是从主程序中返回一个整数值）。

b) 回收子进程
i. 如果父进程没有回收它的僵死子进程就终止了，内核安排 init 进程去回收它们（init 进程 PID 为 1，系统启动时创建，不会终止，是所有进程的祖先）。
ii. 僵死进程：由于子进程的结束和父进程的运行是一个异步过程，即父进程永远无法预测子进程到底什么时候结束，那么会不会因为父进程太忙来不及 wait 子进程，或者说不知道子进程什么时候结束，而丢失子进程结束时的状态信息呢？不会。因为 UNIX 提供了一种机制可以保证只要父进程想知道子进程结束时的状态信息，就可以得到。这种机制就是：在每个进程退出的时候，内核释放该进程所有的资源，包括打开的文件，占用的内存等，但是仍然为其保留一定的信息（包括进程号、退出状态等）。直到父进程 wait/waitpid 来取时才释放（或者其他一些情况）。
iii. wait 函数：挂起当前进程的执行直到它的一个子进程终止。返回已终止子进程的 pid。子进程完成结束的顺序是任意的（没有固定的顺序）
iv. waitpid 函数：挂起调用进程的执行，直到它的等待集合中的一个子进程终止。
    1. 参数 pid：设置等待集合的成员。 pid>0：该 pid 的子进程。pid=-1：所有子进程。
    2. 参数 statusp（NULL）：如果 statusp 参数是非空的，那么 waitpid 就会在 status 中放上关于导致返回的子进程的状态信息。
    3. 参数 options：修改默认行为。
    4. 错误条件：如果调用进程没有子进程，那么 waitpid 返回-1，并且设置 errno 为 ECHILD。如果 waitpid 函数被一个信号中断，那么它返回-1，并设置 errno 为 EINTR。

c) 让进程休眠
i. sleep 函数：sleep 函数将一个进程挂起一段指定的时间。如果请求的时间量已经到了，sleep 返回 0，否则返回还剩下的要休眠的秒数。
ii. pause 函数：该函数让调用函数休眠，直到该进程收到一个信号。

d) 加载并运行程序
i. execve 函数：详见书 p521 和 p585

1. 在当前进程中载入并运行程序
2. filename：可执行文件
3. argv：参数列表，惯例：argv[0]==filename
4. envp：环境变量列表
5. 覆盖当前进程的代码、数据、栈。保留：有相同的 PID，继承已打开的文件描述符和信号上下文。一次调用，0 次返回（如没有错误）

3. 信号

1) 发送信号

a) 进程组
   i. 每个进程只属于一个进程组
   ii. getpgrp（）返回当前进程的进程组 ID
   iii. setpgid（）改变自己或其他进程的进程组

b) 用/bin/kill 程序发送信号
   i. /bin/kill 程序可以向另外的进程或进程组发送任意的信号
   ii. 负的 PID 会导致信号被发送到进程组 PID 中的每个进程。

c) 输入 ctrl+c（ctrl+z）会导致内核发送一个 SIGINT（SIGTSTP）信号到前台进程组中的每个作业
   i. SIGINT-默认情况是终止前台作业
   ii. SIGTSTP-默认情况是停止（挂起）前台作业

2) 接收信号（当目的进程被内核强迫以某种方式对信号的发送做出反应时，它就接收了信号

a) 反应的方式：
   i. 忽略这个信号
   ii. 终止进程
   iii. 通过执行一个称为信号处理程序的用户层函数捕获这个信号（类似于响应异步中断而调用的硬件异常处理程序）

b) 内核检查（待处理信号）pnb = pending&~blocked
   i. 当内核把进程 p 从内核模式切换到用户模式时（例如，从系统调用返回或是完成了一次上下文切换），它会检查进程 p 的未被阻塞的待处理信号的集合（pending&~blocked）
   ii. pending 位向量中维护着待处理信号的集合。只要传送了一个类型为 k 的信号，内核就会设置 pending 中的第 k 位，而只要接收了一个类型为 k 的信号，内核就会清除 pending 中的第 k 位。
   iii. blocked 位向量中维护着被阻塞的信号的集合。

c) 信号的默认行为
   i. 进程可以通过使用 signal 函数修改和信号相关联的默认行为。唯一的例外是 SIGSTOP 和 SIGKILL，它们的默认行为是不能修改的。
   ii. signal 函数：可以修改和信号相关联的默认行为。
      1. 通过把处理程序的地址传递到 signal 函数从而改变默认行为，这

叫做设置信号处理程序。调用信号处理程序被称为捕获信号。执行信号处理程序被称为处理信号。

2. 信号处理程序是与主程序同时运行的独立逻辑流（不是进程）
3. 信号处理程序可以被其他信号处理程序中断
4. 当处理程序执行它的 return 语句时，控制（通常）传递回控制流中的进程被信号接收中断处的指令。

3) 编写信号处理程序（书 p534，这里注意 printf 和 write 函数）

4) 非本地跳转

a) setjmp 函数：在 env 缓冲区中保存当前调用环境，以供后面的 longjmp 使用，并返回 0。调用环境包括程序计数器、栈指针和通用目的寄存器。（只被调用一次，但返回多次）

b) longjmp 函数：从 env 缓冲区中恢复调用环境，然后触发一个从最近一次初始化 env 的 setjmp 调用的返回。然后 setjmp 返回，并带有非零的返回值 retval。（被调用一次，但从不返回）