链接过程中，带 static 属性的全局变量属于**局部符号**

简述 C 编译过程对非寄存器实现的 int 全局变量与非静态 int 局部变量处理的区别。包括存储区域、赋初值、生命周期、指令中寻址方式等

什么是共享库（动态连接库）？简述动态链接的实现方法

答：共享库（动态连接库）是一个.so 的目标模块（elf 文件），在运行或加载时，由动态链接器程序加载到任意的内存地址，并和一个和内存中的程序（如当前可执行目标文件）动态完全链接为一个可执行程序。使用它可节省内存与硬盘空间，方便软件的更新升级。如标准 C 库 libc.so 。

加载时动态链接：应用程序第一次加载和运行时，通过 ld-linux.so 动态链接器重定位动态库的代码和数据到某个内存段，再重定位当前应用程序中对共享库定义的符号的引用，然后将控制传递给应用程序（此后共享库位置固定了并不变）。

运行时动态链接：在程序执行过程中，通过 dlopen/dlsym 函数加载和连接共享库，实现符号重定位，通过 dlclose 卸载动态库。

什么是静态库？使用静态库的优点是什么？

8. 异常和操作系统

改变控制流的两种机制：1.跳转和分支 2. 调用和返回，只能对**程序状态**变化做出反应

异常控制流发生在计算机系统的**所有层次：**

1. **低层机制**(硬件层)：**异常**（操作系统和硬件共同实现）
   硬件检测到的事件会触发控制转移到异常处理程序
2. **高层机制：**
   进程上下文切换 （ 通过操作系统和硬件定时器实现 ）
   信号 （ 操作系统实现，进程级的异常处理 ）
   非本地跳转 (进程内的异常控制流（用户级）C 运行库实现、setjmp() longjmp())

异常处理：每种类型的事件有一个唯一的异常号，异常号是到异常表条目的索引，异常表条目中存放着异常处理程序的地址，异常表起始地址存放在异常表基址寄存器中。

1. 异常种类（低层机制）：

1. **异步发生(外部产生)：**

   1） 中断：

   返回行为：总返回到下一条指令

   触发原因：**I/O 设备的信号**（中断是在当前指令结束后才会触发，因此返回到下一条指令见怪不怪）

2. **同步发生（内部产生）：**

1） 陷阱：

返回行为：总返回到下一条指令

触发原因：**系统调用 syscall：** read()、fork()、execve()、exit()【有意的异常】

系统调用参数**都是**通过寄存器传递而不是栈，**%rax 传递系统调用号，%rdi 传递第一个参数(**%rdi,%rsi, %rdx, %r10, %r8, %r9**)，系统调用返回返回值在%rax，负数返回值表明发生了错误**

2） 故障：

返回行为：可能返回到当前指令，并重新执行当前指令；可能终止(abort)

触发原因：

1.缺页(可恢复)

2.浮点异常(除法错误【除以 0】——不可恢复终止程序)

3.非法内存引用(引用了未定义的虚拟存储区域或写入只读文本段——发送“段错误”SIGSEGV 并且不可恢复终止程序)

3） 终止：

返回行为：不会返回

触发原因：

1.非法指令

2.奇偶校验错误

3.机器检查【不可恢复的错误】

2.进程上下文切换和系统调用

进程的祖先：init 进程，pid 为 1

进程提供给应用程序两个关键抽象：1. 逻辑控制流 2. 私有地址空间

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

进程从**用户模式变为内核模式的唯一方法**是**通过中断、故障、陷入系统调用这样的异常**。

控制流通过**上下文切换**从一个进程传递到另一个进程

操作系统实现交错执行的机制称为**上下文切换，**操作系统运行所需的所有状态信息称为**上下文**

系统级上下文和**用户级上下文的地址空间**共同构成了进程整个存储器映像（P42 pdF）

上下文内容包括: 1.通用目的寄存器 2.浮点寄存器 3.用户栈 4.状态寄存器 5.内核栈 6.内核数据结构(mm_struct 包括虚拟内存一级页表指针 pgd、程序计数器、PID、可执行目标文件的名字、指向用户栈的指针)

引发上下文切换的事件：**系统调用、中断（或进程捕获一个信号时）**

程序员角度认为进程总处于 1. 运行、2. 停止、3. 终止 状态之一

Linux 系统级函数遇到错误时，通常返回-1 并设置全局整数变量 **errno** 标示出错原因

创建进程 Fork 函数：

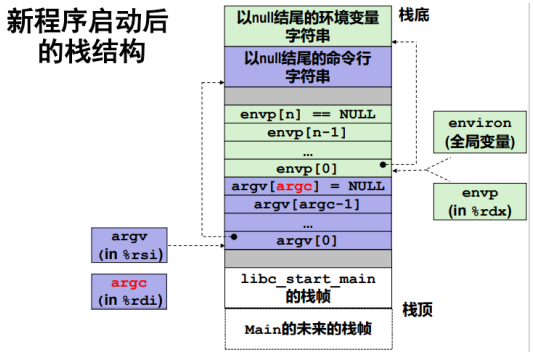
1 次调用、2 次返回（子进程返回 **0**，父进程返回**子进程 PID**）

新创建的子进程得到和父进程虚拟地址空间相同的一份**副本**（**只是内容相同但完全独立**）

子进程获得与父进程任何打开文件描述符相同的副本(**共享文件**)
最大区别：子进程有不同于父进程的 **PID**

进程退出 exit 函数：
1 次调用、0 次返回

加载和运行程序 Execve 函数： int execve(char *filename, char *argv[], char *envp[])
一次调用，0 次返回（如没有错误）【函数除调用一次**从不返回**，除非有错误】
argv：参数列表（**%rsi**）【argv[0]==filename（**filename 为文件名所在的地址!!!**）】
argv 里存的都是地址，最后 argv[argc]=NULL 在内存中为 00 00 00 00 00 00 00 00
null 结尾的命令行字符串不是以小端方式存储的，且 00 表示没有规定任何值。
envp：环境变量列表（存放在**%rdx**）
argc： 参数数量 (存放在**%rdi**)



程序使用 execve()加载时替换和覆盖当前进程的代码段、数据段、堆和栈【保留 PID，继承已打开的文件描述符和信号上下文】

3.信号

shell 是一个交互型应用级程序，代表用户运行其他程序，shell 执行一系列的**读/求值**步骤
shell 需要用信号回收后台子进程。
Linux 常考信号类型：

名称	相应事件	默认行为
SIGINT(SIGTSTP)	Linux 键盘的中断: Ctrl+c (Ctrl+z)	终止(停止或挂起)
SIGKILL	杀死程序(该信号不能被捕获不能被忽略)	终止
SIGFPE	除 0（浮点异常）	终止
SIGSEGV	无效的内存引用（段故障）	终止
SIGALRM	来自 alarm 函数的定时器信号	终止
SIGCHLD	一个子进程停止或者终止	忽略
SIGCONT	如果进程停止则继续该进程	忽略

发送信号：

内核发送信号给**目的进程**【方式：更新目的进程上下文中某个状态】

发送信号的原因：

1. 内核检测到一个系统事件（如：除 0【SIGFPE】、子进程终止【SIGCHLD】）
2. 进程调用了 **kill** 系统调用，**显式地**请求内核发送一个信号到目的进程（**一个进程可以发送信号给它自己**）

发送信号方法：

1. 采用/bin/kill 程序向另外的进程或进程组发送任意信号：
   每个进程只属于一个进程组，负的 PID 会导致信号被发送到进程组 PID 中的每个进程
2. 用键盘发送信号：ctrl+c 终止（信号 SIGINT） ctrl+z 挂起或暂停(SIGTSTP)
3. 用 kill / alarm 函数发送信号

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

反应的方式：

1. 忽略这个信号
2. 终止进程
3. 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（类似中断时调用异常处理程序），返回时：返回到下一条指令

一个待处理信号最多只能被接收一次，一个进程可以选择**阻塞**接收某种信号

阻塞的信号仍可以被发送，但不会被接收，直到进程取消对该信号的阻塞

每个信号类型都有一个**预定义默认行为**：

1. 进程终止
2. 进程停止（挂起）直到被 SIGCONT 信号重启
3. 进程忽略该信号

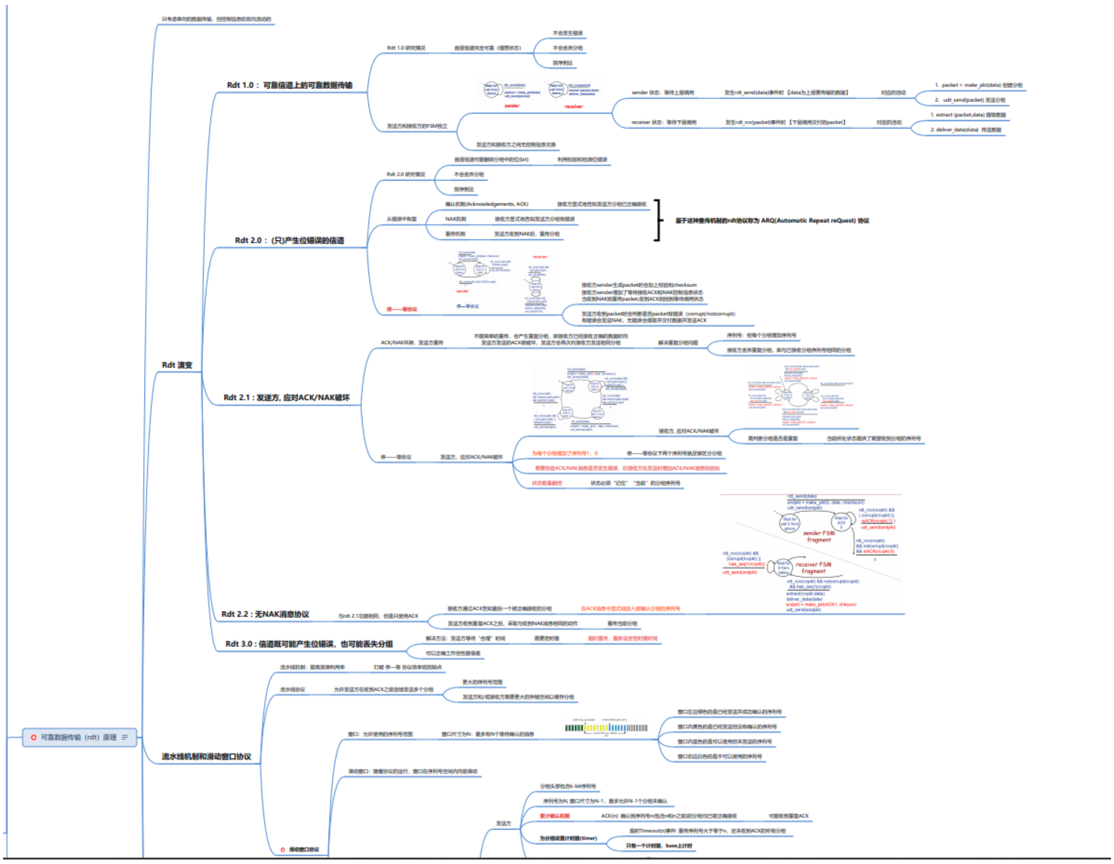
进程可以使用 signal (int signum, handler_t *handler)函数修改和信号相关联的默认行为

信号处理程序（handler 地址处的函数）可以被其他信号处理程序中断

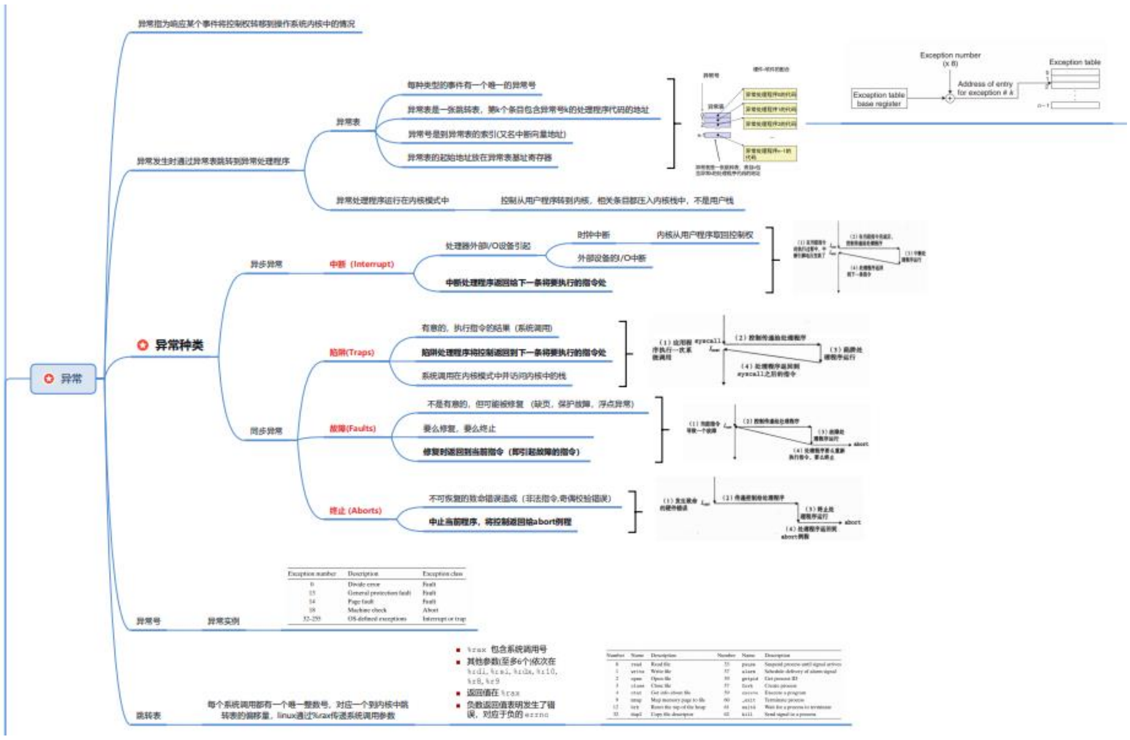
Linux 阻塞信号机制：隐式阻塞机制（一个 SIGINT 信号处理程序不能被另一个 SIGINT 信号中断）、显示阻塞和解除阻塞机制

编写信号处理程序的原则

1. 安全的信号处理：
   G0：处理程序尽可能简单
   G1：在处理程序中只调用异步信号安全的函数
   G2：保存和恢复 errno
   G3：阻塞所有信号保护对共享全局数据结构的访问
   G4：用 volatile 声明全局变量
   G5：用 sig_atomic_t 声明标志
2. 正确的信号处理：
   未处理的信号是不排队的
3. 编写可移植的信号处理
4. 用同步流以避免并发错误
5. 消除竞争的正确 Shell 程序
6. 显式地等待信号



可靠数据传输（rdt）原理
Rdt 演变
Rdt 1.0：可靠信道上的可靠数据传输
Rdt 2.0：（只）产生位错误的信道
Rdt 2.1：发送方，应对ACK/NAK破坏
Rdt 2.2：无NAK消息协议
Rdt 3.0：信道既可能产生位错误，也可能丢失分组
流水线机制和滑动窗口协议
滑动窗口协议



异常
异常指为响应某个事件将控制权转移到操作系统内核中的情况
异常发生时通过异常表跳转到异常处理程序
异常表
每种类型的事件有一个唯一的异常号
异常表是一张跳转表，第k个条目包含异常号k的处理程序代码的地址
异常号是到异常表的索引(又名中断向量地址)
异常表的起始地址放在异常表基址寄存器
异常处理程序运行在内核模式中
控制从用户程序转到内核，相关条目都压入内核栈中，不是用户栈
Exception number (x 8)
Exception table base register
Address of entry for exception # k
Exception table
异常种类
异步异常
中断（Interrupt）
处理器外部I/O设备引起
时钟中断
内核从用户程序取回控制权
外部设备的I/O中断
中断处理程序返回给下一条将要执行的指令处
同步异常
陷阱(Traps)
有意的，执行指令的结果（系统调用）
陷阱处理程序将控制返回到下一条将要执行的指令处
系统调用在内核模式中并访问内核中的栈
故障(Faults)
不是有意的，但可能被修复（缺页，保护故障，浮点异常）
要么修复，要么终止
修复时返回到当前指令（即引起故障的指令）
终止 (Aborts)
不可恢复的致命错误造成（非法指令,奇偶校验错误）
中止当前程序，将控制返回给abort例程
异常号
异常实例
Exception number
Description
Exception class
0
Divide error
Fault
13
General protection fault
Fault
14
Page fault
Fault
18
Machine check
Abort
32-255
OS-defined exceptions
Interrupt or trap
跳转表
每个系统调用都有一个唯一整数号，对应一个到内核中跳转表的偏移量，linux通过%rax传递系统调用参数
%rax 包含系统调用号
其他参数(至多6个)依次在 %rdi, %rsi, %rdx, %r10, %r8, %r9
返回值在 %rax
负数返回值表明发生了错误，对应于负的 errno