

问题合计

课堂小测

1. 不同 ISA 的状态寄存器

• X86:

EFlags[32 位]或 RFLAGS[64 位]寄存器，包含进位标志(CF)、零标志(ZF)、警报标志(PF)、标志标志(TF)、符号标志(SF)、溢出标志(OF)、方向标志(DF)、中断允许标志(IF)等。

• ARM:

CPSR(Current Program Status Register)或 APSR(Application Program Status Register)以及 SPSR(Saved Program Status Register)，包含条件码(NEUT, Z, V, 中断屏蔽位(I), F, Thumb 模式位(T), 群模式位(M))等。

• RISC-V:

没有像 X86 或 ARM 那样的单一专用状态寄存器来存放所有条件标志。条件判断通常由比较指令直接将结果存入通用寄存器或连接于分支。但是，RISC-V 有一系列控制和状态寄存器(CSRs)，例如：

• mstatus, sstatus: 用于管理特权级、中断使能状态等。

• fcsr: 用于浮点状态。

传统意义上的“状态”(如进位、零标志)的处理方式与其 ISA 不同。

• SVC(Supervisor Call, 常用名 SWI - Software Interrupt)

• MIPS:

int sys80(传递方式):

• SYSCALL.

• RISC-V:

• ECALL (Environment Call)

2. 不同 ISA 对应的访管指令是什么？

• X86:

INT(0x80):

CPU 通过中断描述符表(IDT-Interrupt Descriptor Table)找到索引表，从中读取内存地址或对应的中断服务例程(Interrupt Handler)。

• RISC-V:

约定遵循标准的 RISC-V ABI (Application Binary Interface)。

• 系统调用号：通过寄存器 a7 传递。

• 参数：前两个参数在 a0~a2 传递，a5 传递。

• 返回值：通过寄存器 a0 返回。对于 64 位更高的返回值，可能还需要使用 a1。

• ARM:

SVC(Supervisor Call, 常用名 SWI - Software Interrupt)

• MIPS:

int sys80(传递方式):

• SYSCALL.

• RISC-V:

• ECALL (Environment Call)

3. RISC-V 架构中用于发起系统调用的指令是什么？该指令的执行过程有哪些重要步骤？

• 指令：ECALL (Environment Call)

• 执行过程重要步骤：

1. 触发陷入(Trap):

ECALL 指令发一个环境调用异常，异常来源是当前执行的特权级别(用户态 U-mode、监管态 S-mode 或机器 M-mode)。

2. 切换特权级：

处理器通常会切换到更高的特权级(例如，从 U-mode 切换到 S-mode 或 M-mode，具体取决于系统配置)。

3. 保存 PC：

当前程序计数器(PC)的值被保存到相应的异常程序计数器寄存器中：

■ M-mode 保留在 sepc

■ U-mode 保留在 rvcpc(如果配置了用户态陷入下文，主要是通用寄存器(General-Purpose Registers, GPRs)，可用来保存浮点寄存器等。这些信息通常保存

在中断处理中，另一个在中断处理

- 保存状态：
当前的一些状态信息(如先前的特权级、中断使能状态)被保存到相应状态寄存器(`inststatus, sstatus, usstatus`)中。
- 更新状态：
状态寄存器被更新(例如，中断可能被禁用，特权级被改变)。
- 设置 PC：
在 X86 上作为中断处理的一部分需要更多的上下文。
- 通用机制：
操作系统的代码通常会设置一个中断向量表(Interrupt Vector Table)或一个特定的全局跳转表，指向一个统一的入口点或部分的全局跳转表，指向一个统一的入口点或部分的全局跳转表，指向一个统一的入口点或部分的全局跳转表，指向一个统一的入口点或部分的全局跳转表。

6. 不同 ISA 上怎样找到系统调用入口程序？

• X86:

EFlags[32 位]或 RFLAGS[64 位]寄存器，包含进位标志(CF)、零标志(ZF)、警报标志(PF)、标志标志(TF)、符号标志(SF)、溢出标志(OF)、方向标志(DF)、中断允许标志(IF)等。

• ARM:

CPSR(Current Program Status Register)或 APSR(Application Program Status Register)以及 SPSCR(Saved Program Status Register)，包含条件码(NEUT, Z, V, 中断屏蔽位(I), F, Thumb 模式位(T), 群模式位(M))等。

• RISC-V:

没有像 X86 或 ARM 那样的单一专用状态寄存器来存放所有条件标志。条件判断通常由比较指令直接将结果存入通用寄存器或连接于分支。但是，RISC-V 有一系列控制和状态寄存器(CSRs)，例如：

• mstatus, sstatus: 用于管理特权级、中断使能状态等。

• fcsr: 用于浮点状态。

传统意义上的“状态”(如进位、零标志)的处理方式与其 ISA 不同。

7. 不同的 ISA，当发生中断/异常/系统调用时，上下文是怎么保存的？

• 下文保存通常分为两部分：硬件自动保存和软件(操作系统)处理程序，保存。

• 系统调用号：在 ea0, ea1, ea2, ea3, ea4, ea5, ea6, ea7, ebb, ebs, es, es1, es2, es3, es4, es5, es6, es7, ebs1, ebs2, ebs3, ebs4, ebs5, ebs6, ebs7, ebs8, ebs9, ebs10, ebs11, ebs12, ebs13, ebs14, ebs15, ebs16, ebs17, ebs18, ebs19, ebs20, ebs21, ebs22, ebs23, ebs24, ebs25, ebs26, ebs27, ebs28, ebs29, ebs30, ebs31, ebs32, ebs33, ebs34, ebs35, ebs36, ebs37, ebs38, ebs39, ebs40, ebs41, ebs42, ebs43, ebs44, ebs45, ebs46, ebs47, ebs48, ebs49, ebs50, ebs51, ebs52, ebs53, ebs54, ebs55, ebs56, ebs57, ebs58, ebs59, ebs60, ebs61, ebs62, ebs63, ebs64, ebs65, ebs66, ebs67, ebs68, ebs69, ebs70, ebs71, ebs72, ebs73, ebs74, ebs75, ebs76, ebs77, ebs78, ebs79, ebs80, ebs81, ebs82, ebs83, ebs84, ebs85, ebs86, ebs87, ebs88, ebs89, ebs90, ebs91, ebs92, ebs93, ebs94, ebs95, ebs96, ebs97, ebs98, ebs99, ebs9, ebs10, ebs11, ebs12, ebs13, ebs14, ebs15, ebs16, ebs17, ebs18, ebs19, ebs100, ebs101, ebs102, ebs103, ebs104, ebs105, ebs106, ebs107, ebs108, ebs109, ebs110, ebs111, ebs112, ebs113, ebs114, ebs115, ebs116, ebs117, ebs118, ebs119, ebs1100, ebs1101, ebs1102, ebs1103, ebs1104, ebs1105, ebs1106, ebs1107, ebs1108, ebs1109, ebs11010, ebs11011, ebs11012, ebs11013, ebs11014, ebs11015, ebs11016, ebs11017, ebs11018, ebs11019, ebs110100, ebs110101, ebs110102, ebs110103, ebs110104, ebs110105, ebs110106, ebs110107, ebs110108, ebs110109, ebs110110, ebs110111, ebs110112, ebs110113, ebs110114, ebs110115, ebs110116, ebs110117, ebs110118, ebs110119, ebs1101100, ebs1101101, ebs1101102, ebs1101103, ebs1101104, ebs1101105, ebs1101106, ebs1101107, ebs1101108, ebs1101109, ebs1101110, ebs1101111, ebs1101112, ebs1101113, ebs1101114, ebs1101115, ebs1101116, ebs1101117, ebs1101118, ebs1101119, ebs11011100, ebs11011101, ebs11011102, ebs11011103, ebs11011104, ebs11011105, ebs11011106, ebs11011107, ebs11011108, ebs11011109, ebs11011110, ebs11011111, ebs11011112, ebs11011113, ebs11011114, ebs11011115, ebs11011116, ebs11011117, ebs11011118, ebs11011119, ebs110111100, ebs110111101, ebs110111102, ebs110111103, ebs110111104, ebs110111105, ebs110111106, ebs110111107, ebs110111108, ebs110111109, ebs110111110, ebs110111111, ebs110111112, ebs110111113, ebs110111114, ebs110111115, ebs110111116, ebs110111117, ebs110111118, ebs110111119, ebs1101111100, ebs1101111101, ebs1101111102, ebs1101111103, ebs1101111104, ebs1101111105, ebs1101111106, ebs1101111107, ebs1101111108, ebs1101111109, ebs1101111110, ebs1101111111, ebs1101111112, ebs1101111113, ebs1101111114, ebs1101111115, ebs1101111116, ebs1101111117, ebs1101111118, ebs1101111119, ebs11011111100, ebs11011111101, ebs11011111102, ebs11011111103, ebs11011111104, ebs11011111105, ebs11011111106, ebs11011111107, ebs11011111108, ebs11011111109, ebs11011111110, ebs11011111111, ebs11011111112, ebs11011111113, ebs11011111114, ebs11011111115, ebs11011111116, ebs11011111117, ebs11011111118, ebs11011111119, ebs110111111100, ebs110111111101, ebs110111111102, ebs110111111103, ebs110111111104, ebs110111111105, ebs110111111106, ebs110111111107, ebs110111111108, ebs110111111109, ebs110111111110, ebs110111111111, ebs110111111112, ebs110111111113, ebs110111111114, ebs110111111115, ebs110111111116, ebs110111111117, ebs110111111118, ebs110111111119, ebs1101111111100, ebs1101111111101, ebs1101111111102, ebs1101111111103, ebs1101111111104, ebs1101111111105, ebs1101111111106, ebs1101111111107, ebs1101111111108, ebs1101111111109, ebs1101111111110, ebs1101111111111, ebs1101111111112, ebs1101111111113, ebs1101111111114, ebs1101111111115, ebs1101111111116, ebs1101111111117, ebs1101111111118, ebs1101111111119, ebs11011111111100, ebs11011111111101, ebs11011111111102, ebs11011111111103, ebs11011111111104, ebs11011111111105, ebs11011111111106, ebs11011111111107, ebs11011111111108, ebs11011111111109, ebs11011111111110, ebs11011111111111, ebs11011111111112, ebs11011111111113, ebs11011111111114, ebs11011111111115, ebs11011111111116, ebs11011111111117, ebs11011111111118, ebs11011111111119, ebs110111111111100, ebs110111111111101, ebs110111111111102, ebs110111111111103, ebs110111111111104, ebs110111111111105, ebs110111111111106, ebs110111111111107, ebs110111111111108, ebs110111111111109, ebs110111111111110, ebs110111111111111, ebs110111111111112, ebs110111111111113, ebs110111111111114, ebs110111111111115, ebs110111111111116, ebs110111111111117, ebs110111111111118, ebs110111111111119, ebs1101111111111100, ebs1101111111111101, ebs1101111111111102, ebs1101111111111103, ebs1101111111111104, ebs1101111111111105, ebs1101111111111106, ebs1101111111111107, ebs1101111111111108, ebs1101111111111109, ebs1101111111111110, ebs1101111111111111, ebs1101111111111112, ebs1101111111111113, ebs1101111111111114, ebs1101111111111115, ebs1101111111111116, ebs1101111111111117, ebs1101111111111118, ebs1101111111111119, ebs11011111111111100, ebs11011111111111101, ebs11011111111111102, ebs11011111111111103, ebs11011111111111104, ebs11011111111111105, ebs11011111111111106, ebs11011111111111107, ebs11011111111111108, ebs11011111111111109, ebs11011111111111110, ebs11011111111111111, ebs11011111111111112, ebs11011111111111113, ebs11011111111111114, ebs11011111111111115, ebs11011111111111116, ebs11011111111111117, ebs11011111111111118, ebs11011111111111119, ebs110111111111111100, ebs110111111111111101, ebs110111111111111102, ebs110111111111111103, ebs110111111111111104, ebs110111111111111105, ebs110111111111111106, ebs110111111111111107, ebs110111111111111108, ebs110111111111111109, ebs110111111111111110, ebs110111111111111111, ebs110111111111111112, ebs110111111111111113, ebs110111111111111114, ebs110111111111111115, ebs110111111111111116, ebs110111111111111117, ebs110111111111111118, ebs110111111111111119, ebs1101111111111111100, ebs1101111111111111101, ebs1101111111111111102, ebs1101111111111111103, ebs1101111111111111104, ebs1101111111111111105, ebs1101111111111111106, ebs1101111111111111107, ebs1101111111111111108, ebs1101111111111111109, ebs1101111111111111110, ebs1101111111111111111, ebs1101111111111111112, ebs1101111111111111113, ebs1101111111111111114, ebs1101111111111111115, ebs1101111111111111116, ebs1101111111111111117, ebs1101111111111111118, ebs1101111111111111119, ebs11011111111111111100, ebs11011111111111111101, ebs11011111111111111102, ebs11011111111111111103, ebs11011111111111111104, ebs11011111111111111105, ebs11011111111111111106, ebs11011111111111111107, ebs11011111111111111108, ebs11011111111111111109, ebs11011111111111111110, ebs11011111111111111111, ebs11011111111111111112, ebs11011111111111111113, ebs11011111111111111114, ebs11011111111111111115, ebs11011111111111111116, ebs11011111111111111117, ebs11011111111111111118, ebs11011111111111111119, ebs110111111111111111100, ebs110111111111111111101, ebs110111111111111111102, ebs110111111111111111103, ebs110111111111111111104, ebs110111111111111111105, ebs110111111111111111106, ebs110111111111111111107, ebs110111111111111111108, ebs110111111111111111109, ebs110111111111111111110, ebs110111111111111111111, ebs110111111111111111112, ebs110111111111111111113, ebs110111111111111111114, ebs110111111111111111115, ebs110111111111111111116, ebs110111111111111111117, ebs110111111111111111118, ebs110111111111111111119, ebs1101111111111111111100, ebs1101111111111111111101, ebs1101111111111111111102, ebs1101111111111111111103, ebs1101111111111111111104, ebs1101111111111111111105, ebs1101111111111111111106, ebs1101111111111111111107, ebs1101111111111111111108, ebs1101111111111111111109, ebs1101111111111111111110, ebs1101111111111111111111, ebs1101111111111111111112, ebs1101111111111111111113, ebs1101111111111111111114, ebs1101111111111111111115, ebs1101111111111111111116, ebs1101111111111111111117, ebs1101111111111111111118, ebs1101111111111111111119, ebs11011111111111111111100, ebs11011111111111111111101, ebs11011111111111111111102, ebs11011111111111111111103, ebs11011111111111111111104, ebs11011111111111111111105, ebs11011111111111111111106, ebs11011111111111111111107, ebs11011111111111111111108, ebs11011111111111111111109, ebs11011111111111111111110, ebs11011111111111111111111, ebs11011111111111111111112, ebs11011111111111111111113, ebs11011111111111111111114, ebs11011111111111111111115, ebs11011111111111111111116, ebs11011111111111111111117, ebs11011111111111111111118, ebs11011111111111111111119, ebs110111111111111111111100, ebs110111111111111111111101, ebs110111111111111111111102, ebs110111111111111111111103, ebs110111111111111111111104, ebs110111111111111111111105, ebs110111111111111111111106, ebs110111111111111111111107, ebs110111111111111111111108, ebs110111111111111111111109, ebs110111111111111111111110, ebs110111111111111111111111, ebs110111111111111111111112, ebs110111111111111111111113, ebs110111111111111111111114, ebs110111111111111111111115, ebs110111111111111111111116, ebs110111111111111111111117, ebs110111111111111111111118, ebs110111111111111111111119, ebs1101111111111111111111100, ebs1101111111111111111111101, ebs1101111111111111111111102, ebs1101111111111111111111103, ebs1101111111111111111111104, ebs1101111111111111111111105, ebs1101111111111111111111106, ebs1101111111111111111111107, ebs1101111111111111111111108, ebs110111111

- 中), 各自独立的内存块确保它们不会相互干扰。2. 上下文需要: 每个进程内部状态执行上一个文本(如局部变量、防止溢出)需要独立保证。3. 存储空间中处理: 进程处理一个系统期时可能产生冲突, 需要单独的空间来处理冲突内容, 增强系统稳定性。
- `se (unit64)`: 定义一个文本(如局部变量、防止溢出)需要独立保证。4. 安全性: 独立内存块防止进程对其他进程的操作, 提高系统的安全性。
- 上下文切换:
- `transframe [struct context *]`: 保存用户态到内核态切换时的寄存器状态, 包含用户程序计数器(`pc`)、栈指针(`sp`)和其他通用寄存器, 当系统调用或中断发生时用于保存用户态上下文。
 - `context [struct file *]NOFILE]`: 进程打开文件表, 每个元素指向一个打开的文件结构。
 - `cmd [struct inode *]`: 当前工作目录的inode指针, 用于相对路径解析。
- 文件系统相关:
- `ofile [struct file *]NOFILE]`: 进程打开文件, 将当前线程的寄存器值(包括程序计数器、栈指针等)保存到线程控制块(TCB)或线程中。
 - `mem_usage`: 通过缓存淘汰和交换技术, 可以使用大于物理内存的地址空间, 提高物理内存利用率。
 - `Shared Memory Management`: 程序员和编译器负责的是连续的虚拟地址空间, 操作系统负责的是物理内存的分配。
1. 进程隔离(Processor Isolation): 每个进程拥有独立的虚拟地址空间, 防止相互干扰, 提高安全性。
2. 内存管理(Simplified Memory Management): 程序员和编译器负责的是连续的虚拟地址空间, 操作系统负责的是物理内存的分配。
3. 效率的内存使用(Efficient Memory Usage): 通过缓存淘汰和交换技术, 可以使用大于物理内存的地址空间, 提高物理内存利用率。
4. 共享内存(Shared Memory Model): 不同进程可以方便地将同一物理内存映射到各自空间, 实现共享库和进程间通信。
5. 灵活性(Flexibility): 程序可以在物理内存的任何位置加载和运行, 无需修改代码。
13. 进程运行时 PC 指向的地址是物理内存地址, 虚拟地址? 采用这一地址方式的好处是什么?
- 答: 进程运行时 PC (Program Counter) 指向的是虚拟地址 (Virtual Address)。
- 好处:
- 保证前线程上一下文, 将当前线程的寄存器值(包括程序计数器、栈指针等)保存到线程控制块(TCB)或线程中。
 - 选择下一个线程: 调度器根据调度算法选择下一个要运行的线程。
 - 修复指针上一下文: 从目标线程的 TCB 或线程中加载其保存的寄存器值, 包括恢复其栈指针 SP 为该线程的值。
 - 切换线程流: 通知爱程序员 PC 的值, CPU 开始执行目标线程的指令。
- 注意事项: 与进程切换不同, 线程切换不需要切换页表(因为同一进程共享地址空间), 因此线程切换更快。线程切换主要发生在以下步驟完成:
- `switch()`: 将当前 CPU 寄存器状态保存到当前进程的 `context` 结构中。
 - `do { ... } while (true)`: 当前工作目录的 inode 指针, 保存的寄存器包括 `pc`, `sp`, `sr`=`11` (`RIS-C1`) 架中的调用者保存寄存器。
2. 加载进程上下文:
- 从新进程的 `context` 结构中恢复寄存器状态到 CPU。
 - 这是第 CPU 继续执行新进程之前的指令流。
3. 页面切換:
- 通过修改 RISC-V 的 `setp` 寄存器, 将当前页表切入。
 - 改变了虚拟地址到物理地址的映射关系。
4. 内存切换:
- 每个进程有自己的内核线程(stack), 切换进程时会切换内存。
 - 这确保了每个进程在内核态建立的执行环境进程切换通常发生在以下情况:
- 时时中断触发时间片用尽。
 - 进程主动调用 `sleep()` 等待某事件。
 - 任务执行结束执行。
12. 进程的有效代码在地址空间中是从 0x0 开始吗? 为什么?
- 答: 不一定。虽然逻辑上代码段通常被链接器放置在虚拟地址空间的地址区域, 但并不总是从 0x0 开始。
- 原因 1: 地址偏移引用检测 (Null Pointer Dereference Detection): 操作系统通常会将虚拟
- 栈帧内变量的指针就会变成空指针, 解引用它会致未定义行为。
- 为什么需要自己的栈:
- 独立的执行流, 管理各自的函数调用、返回地址和执行状态。
 - 局部变量存储: 每个线程需要独立空间存放自己的函数局部变量。
 - 函数调用参数和返回值: 独立处理函数调用时的参数传递和返回值。
 - 地址空间划分: 线程 1 通过线程 2 打开一个函数, 该函数有一个全局变量 `int local_var = 10;`。线程 2 通过某种通常是精简设计的方式将 `local_var` 这个地址交给线程 1, 例如, 通过修改 `A[1], B[1], C[1], ..., A[10], B[1], C[1]`(共 30 步)全局变量。
- 危险情况: 在线程 1 访问时, 线程 2 可能已经将该变量回写, 通过线程 2 读取后, 通过线程 1 写入。这样空间可能已被释放或被后续的函数调用覆盖。`\{local_var\}` 指向的内容。
- 高级情况: 在线程 1 访问时, 线程 2 仍在可能正在运行。如果线程 2 已经将该变量回写, 通过线程 1 读取后, 通过线程 2 写入。这样空间可能已被释放或被后续的函数调用覆盖。
- 低级情况: 在线程 1 访问时, 线程 2 仍在可能正在运行。如果线程 2 已经将该变量回写, 通过线程 1 读取后, 通过线程 2 写入。这样空间可能已被释放或被后续的函数调用覆盖。
- 平均周转时间 = $(T + 20 + 30) / 3 = 60 / 3 = 20 \text{ ms}$
- 平均周转时间 = $(T + S) / 3 = 60 / 3 = 20 \text{ ms}$
- CPU 利用率 = $T / (T + S)$
- (b) $Q > T$
- (c) $S < Q - T$
- (d) $Q = S$
- (e) $Q > T$
- (f) $Q = S$
- (g) $Q < T$
- (h) $Q = S$
- (i) $Q < T$
- (j) $Q = S$
- (k) $Q < T$
- (l) $Q = S$
- (m) $Q < T$
- (n) $Q = S$
- (o) $Q < T$
- (p) $Q = S$
- (q) $Q < T$
- (r) $Q = S$
- (s) $Q < T$
- (t) $Q = S$
- (u) $Q < T$
- (v) $Q = S$
- (w) $Q < T$
- (x) $Q = S$
- (y) $Q < T$
- (z) $Q = S$
- (aa) $Q < T$
- (bb) $Q = S$
- (cc) $Q < T$
- (dd) $Q = S$
- (ee) $Q < T$
- (ff) $Q = S$
- (gg) $Q < T$
- (hh) $Q = S$
- (ii) $Q < T$
- (jj) $Q = S$
- (kk) $Q < T$
- (ll) $Q = S$
- (mm) $Q < T$
- (nn) $Q = S$
- (oo) $Q < T$
- (pp) $Q = S$
- (qq) $Q < T$
- (rr) $Q = S$
- (ss) $Q < T$
- (tt) $Q = S$
- (uu) $Q < T$
- (vv) $Q = S$
- (ww) $Q < T$
- (xx) $Q = S$
- (yy) $Q < T$
- (zz) $Q = S$
- (aa) $Q < T$
- (bb) $Q = S$
- (cc) $Q < T$
- (dd) $Q = S$
- (ee) $Q < T$
- (ff) $Q = S$
- (gg) $Q < T$
- (hh) $Q = S$
- (ii) $Q < T$
- (jj) $Q = S$
- (kk) $Q < T$
- (ll) $Q = S$
- (mm) $Q < T$
- (nn) $Q = S$
- (oo) $Q < T$
- (pp) $Q = S$
- (qq) $Q < T$
- (rr) $Q = S$
- (ss) $Q < T$
- (tt) $Q = S$
- (uu) $Q < T$
- (vv) $Q = S$
- (ww) $Q < T$
- (xx) $Q = S$
- (yy) $Q < T$
- (zz) $Q = S$
- (aa) $Q < T$
- (bb) $Q = S$
- (cc) $Q < T$
- (dd) $Q = S$
- (ee) $Q < T$
- (ff) $Q = S$
- (gg) $Q < T$
- (hh) $Q = S$
- (ii) $Q < T$
- (jj) $Q = S$
- (kk) $Q < T$
- (ll) $Q = S$
- (mm) $Q < T$
- (nn) $Q = S$
- (oo) $Q < T$
- (pp) $Q = S$
- (qq) $Q < T$
- (rr) $Q = S$
- (ss) $Q < T$
- (tt) $Q = S$
- (uu) $Q < T$
- (vv) $Q = S$
- (ww) $Q < T$
- (xx) $Q = S$
- (yy) $Q < T$
- (zz) $Q = S$
- (aa) $Q < T$
- (bb) $Q = S$
- (cc) $Q < T$
- (dd) $Q = S$
- (ee) $Q < T$
- (ff) $Q = S$
- (gg) $Q < T$
- (hh) $Q = S$
- (ii) $Q < T$
- (jj) $Q = S$
- (kk) $Q < T$
- (ll) $Q = S$
- (mm) $Q < T$
- (nn) $Q = S$
- (oo) $Q < T$
- (pp) $Q = S$
- (qq) $Q < T$
- (rr) $Q = S$
- (ss) $Q < T$
- (tt) $Q = S$
- (uu) $Q < T$
- (vv) $Q = S$
- (ww) $Q < T$
- (xx) $Q = S$
- (yy) $Q < T$
- (zz) $Q = S$
- (aa) $Q < T$
- (bb) $Q = S$
- (cc) $Q < T$
- (dd) $Q = S$
- (ee) $Q < T$
- (ff) $Q = S$
- (gg) $Q < T$
- (hh) $Q = S$
- (ii) $Q < T$
- (jj) $Q = S$
- (kk) $Q < T$
- (ll) $Q = S$
- (mm) $Q < T$
- (nn) $Q = S$
- (oo) $Q < T$
- (pp) $Q = S$
- (qq) $Q < T$
- (rr) $Q = S$
- (ss) $Q < T$
- (tt) $Q = S$
- (uu) $Q < T$
- (vv) $Q = S$
- (ww) $Q < T$
- (xx) $Q = S$
- (yy) $Q < T$
- (zz) $Q = S$
- (aa) $Q < T$
- (bb) $Q = S$
- (cc) $Q < T$
- (dd) $Q = S$
- (ee) $Q < T$
- (ff) $Q = S$
- (gg) $Q < T$
- (hh) $Q = S$
- (ii) $Q < T$
- (jj) $Q = S$
- (kk) $Q < T$
- (ll) $Q = S$
- (mm) $Q < T$
- (nn) $Q = S$
- (oo) $Q < T$
- (pp) $Q = S$
- (qq) $Q < T$
- (rr) $Q = S$
- (ss) $Q < T$
- (tt) $Q = S$
- (uu) $Q < T$
- (vv) $Q = S$
- (ww) $Q < T$
- (xx) $Q = S$
- (yy) $Q < T$
- (zz) $Q = S$
- (aa) $Q < T$
- (bb) $Q = S$
- (cc) $Q < T$
- (dd) $Q = S$
- (ee) $Q < T$
- (ff) $Q = S$
- (gg) $Q < T$
- (hh) $Q = S$
- (ii) $Q < T$
- (jj) $Q = S$
- (kk) $Q < T$
- (ll) $Q = S$
- (mm) $Q < T$
- (nn) $Q = S$
- (oo) $Q < T$
- (pp) $Q = S$
- (qq) $Q < T$
- (rr) $Q = S$
- (ss) $Q < T$
- (tt) $Q = S$
- (uu) $Q < T$
- (vv) $Q = S$
- (ww) $Q < T$
- (xx) $Q = S$
- (yy) $Q < T$
- (zz) $Q = S$
- (aa) $Q < T$
- (bb) $Q = S$
- (cc) $Q < T$
- (dd) $Q = S$
- (ee) $Q < T$
- (ff) $Q = S$
- (gg) $Q < T$
- (hh) $Q = S$
- (ii) $Q < T$
- (jj) $Q = S$
- (kk) $Q < T$
- (ll) $Q = S$
- (mm) $Q < T$
- (nn) $Q = S$
- (oo) $Q < T$
- (pp) $Q = S$
- (qq) $Q < T$
- (rr) $Q = S$
- (ss) $Q < T$
- (tt) $Q = S$
- (uu) $Q < T$
- (vv) $Q = S$
- (ww) $Q < T$
- (xx) $Q = S$
- (yy) $Q < T$
- (zz) $Q = S$
- (aa) $Q < T$
- (bb) $Q = S$
- (cc) $Q < T$
- (dd) $Q = S$
- (ee) $Q < T$
- (ff) $Q = S$
- (gg) $Q < T$
- (hh) $Q = S$
- (ii) $Q < T$
- (jj) $Q = S$
- (kk) $Q < T$
- (ll) $Q = S$
- (mm) $Q < T$
- (nn) $Q = S$
- (oo) $Q < T$
- (pp) $Q = S$
- (qq) $Q < T$
- (rr) $Q = S$
- (ss) $Q < T$
- (tt) $Q = S$
- (uu) $Q < T$
- (vv) $Q = S$
- (ww) $Q < T$
- (xx) $Q = S$
- (yy) $Q < T$
- (zz) $Q = S$
- (aa) $Q < T$
- (bb) $Q = S$
- (cc) $Q < T$
- (dd) $Q = S$
- (ee) $Q < T$
- (ff) $Q = S$
- (gg) $Q < T$
- (hh) $Q = S$
- (ii) $Q < T$
- (jj) $Q = S$
- (kk) $Q < T$
- (ll) $Q = S$
- (mm) $Q < T$
- (nn) $Q = S$
- (oo) $Q < T$
- (pp) $Q = S$
- (qq) $Q < T$
- (rr) $Q = S$
- (ss) $Q < T$
- (tt) $Q = S$
- (uu) $Q < T$
- (vv) $Q = S$
- (ww) $Q < T$
- (xx) $Q = S$
- (yy) $Q < T$
- (zz) $Q = S$
- (aa) $Q < T$
- (bb) $Q = S$
- (cc) $Q < T$
- (dd) $Q = S$
- (ee) $Q < T$
- (ff) $Q = S$
- (gg) $Q < T$
- (hh) $Q = S$
- (ii) $Q < T$
- (jj) $Q = S$
- (kk) $Q < T$
- (ll) $Q = S$
- (mm) $Q < T$
- (nn) $Q = S$
- (oo) $Q < T$
- (pp) $Q = S$
- (qq) $Q < T$
- (rr) $Q = S$
- (ss) $Q < T$
- (tt) $Q = S$
- (uu) $Q < T$
- (vv) $Q = S$
- (ww) $Q < T$
- (xx) $Q = S$
- (yy) $Q < T$
- (zz) $Q = S$
- (aa) $Q < T$
- (bb) $Q = S$
- (cc) $Q < T$
- (dd) $Q = S$
- (ee) $Q < T$
- (ff) $Q = S$
- (gg) $Q < T$
- (hh) $Q = S$
- (ii) $Q < T$
- (jj) $Q = S$
- (kk) $Q < T$
- (ll) $Q = S$
- (mm) $Q < T$
- (nn) $Q = S$
- (oo) $Q < T$
- (pp) $Q = S$
- (qq) $Q < T$
- (rr) $Q = S$
- (ss) $Q < T$
- (tt) $Q = S$
- (uu) $Q < T$
- (vv) $Q = S$
- (ww) $Q < T$
- (xx) $Q = S$
- (yy) $Q < T$
- (zz) $Q = S$
- (aa) $Q < T$
- (bb) $Q = S$
- (cc) $Q < T$
- (dd) $Q = S$
- (ee) $Q < T$
- (ff) $Q = S$
- (gg) $Q < T$
- (hh) $Q = S$
- (ii) $Q < T$
- (jj) $Q = S$
- (kk) $Q < T$
- (ll) $Q = S$
- (mm) $Q < T$
- (nn) $Q = S$
- (oo) $Q < T$
- (pp) $Q = S$
- (qq) $Q < T$
- (rr) $Q = S$
- (ss) $Q < T$
- (tt) $Q = S$
- (uu) $Q < T$
- (vv) $Q = S$
- (ww) $Q < T$
- (xx) $Q = S$
- (yy) $Q < T$
- (zz) $Q = S$
- (aa) $Q < T$
- (bb) $Q = S$
- (cc) $Q < T$
- (dd) $Q = S$
- (ee) $Q < T$
- (ff) $Q = S$
- (gg) $Q < T$
- (hh) $Q = S$
- (ii) $Q < T$
- (jj) $Q = S$
- (kk) $Q < T$
- (ll) $Q = S$
- (mm) $Q < T$
- (nn) $Q = S$
- (oo) $Q < T$
- (pp) $Q = S$
- (qq) $Q < T$
- (rr) $Q = S$
- (ss) $Q < T$
- (tt) $Q = S$
- (uu) $Q < T$
- (vv) $Q = S$
- (ww) $Q < T$
- (xx) $Q = S$
- (yy) $Q < T$
- (zz) $Q = S$
- (aa) $Q < T$
- (bb) $Q = S$
- (cc) $Q < T$
- (dd) $Q = S$
- (ee) $Q < T$
- (ff) $Q = S$
- (gg) $Q < T$
- (hh) $Q = S$
- (ii) $Q < T$
- (jj) $Q = S$
- (kk) $Q < T$
- (ll) $Q = S$
- (mm) $Q < T$
- (nn) $Q = S$
- (oo) $Q < T$
- (pp) $Q = S$
- (qq) $Q < T$
- (rr) $Q = S$
- (ss) $Q < T$
- (tt) $Q = S$
- (uu) $Q < T$
- (vv) $Q = S$
- (ww) $Q < T$
- (xx) $Q = S$
- (yy) $Q < T$
- (zz) $Q = S$
- (aa) $Q < T$
- (bb) $Q = S$
- (cc) $Q < T$
- (dd) $Q = S$
- (ee) $Q < T$
- (ff) $Q = S$
- (gg) $Q < T$
- (hh) $Q = S$
- (ii) $Q < T$
- (jj) $Q = S$
- (kk) $Q < T$
- (ll) $Q = S$
- (mm) $Q < T$
- (nn) $Q = S$
- (oo) $Q < T$
- (pp) $Q = S$
- (qq) $Q < T$
- (rr) $Q = S$
- (ss) $Q < T$
- (tt) $Q = S$
- (uu) $Q < T$
- (vv) $Q = S$
- (ww) $Q < T$
- (xx) $Q = S$
- (yy) $Q < T$
- (zz) $Q = S$
- (aa) $Q < T$
- (bb) $Q = S$
- (cc) $Q < T$
- (dd) $Q = S$
- (ee) $Q < T$
- (ff) $Q = S$
- (gg) $Q < T$
- (hh) $Q = S$
- (ii) $Q < T$
- (jj) $Q = S$
- (kk) $Q < T$
- (ll) $Q = S$
- (mm) $Q < T$
- (nn) $Q = S$
- (oo) $Q < T$
- (pp) $Q = S$
- (qq) $Q < T$
- (rr) $Q = S$
- (ss) $Q < T$
- (tt) $Q = S$
- (uu) $Q < T$
- (vv) $Q = S$
- (ww) $Q < T$
- (xx) $Q = S$
- (yy) $Q < T$
- (zz) $Q = S$
- (aa) $Q < T$
- (bb) $Q = S$
- (cc) $Q < T$
- (dd) $Q = S$
- (ee) $Q < T$
- (ff) $Q = S$
- (gg) $Q < T$
- (hh) $Q = S$
- (ii) $Q < T$
- (jj) $Q = S$
- (kk) $Q < T$
- (ll) $Q = S$
- (mm) $Q < T$
- (nn) $Q = S$
- (oo) $Q < T$
- (pp) $Q = S$
- (qq) $Q < T$
- (rr) $Q = S$
- (ss) $Q < T$
- (tt) $Q = S$
- (uu) $Q < T$
- (vv) $Q = S$
- (ww) $Q < T$
- (xx) $Q = S$
- (yy) $Q < T$
- (zz) $Q = S$
- (aa) $Q < T$
- (bb) $Q = S$
- (cc) $Q < T$
- (dd) $Q = S$
- (ee) $Q < T$
- (ff) $Q = S$
- (gg) $Q < T$
- (hh) $Q = S$
- (ii) $Q < T$
- (jj) $Q = S$
- (kk) $Q < T$
- (ll) $Q = S$
- (mm) $Q < T$
-

- 非抢占式调度算法:
 1. FCFS (First-Come, First-Served): 按到达顺序执行, 进程一且获得 CPU 就运行到完或主动放弃。
 2. SJF (Shortest Job First): 选择预期执行时间最短的进程运行, 但一旦开始执行就不会被抢占。

26. 公平共享调度 (Fair-share scheduling), 保证调度 (Guaranteed Scheduling) 和彩票调度 (Lottery scheduling): 学习并总结这几种调度算法 (原理, 缺点, 策略, 适用场景)

• 公平共享调度 (Fair-share Scheduling):

1. 原理: 根据用户或用户组分配 CPU 时间, 确保每个用户获得预定比例的系统资源, 而不是只关注单个进程。即使一个用户有多个进程, 也只能获得其应得的资源份额。
2. 策略:
 1. 每个用户 (或用户组) 分配一个权重, 表示其实获得系统资源的比例。
 2. 系统跟踪每个用户的实际使用的资源量, 并调整调度策略以实现平衡实际使用与目标分配之间的差距。
 3. 用户实际获得的 CPU 时间与其权重和活跃进程数成正比。

• 优点:

1. 实现简单, 方策快速 (只需随机数生成和简单计算)。
2. 支持不同优先级 (通过不同彩票数量)。
3. 具有良好的公平性, 长期运行中, 可以获得均匀的 CPU 时间与彩票数成正比)。
4. 可以拟多种调度算法。

• 缺点:

1. 短期公平性无法保证 (随机性可能导致短期期内资源分配不均)。
2. 不适合实时系统 (无法提供确定性保障)。
3. 通过牺牲响应时间换取公平性。
4. 适用于单机环境。

• 应用:

1. 在单机上阅读 X11 源代码, 列举出 2 个引发调度的原因? 给出代码的具体文件和行数。
2. 中断处理时间片用尽:

1. 文件: kernel/trap.c 第 75-77 行

```
// give up the CPU if this is a timer interrupt.
if (which_dev == 2)
    yield();
```

此处是 usertrap 函数中, 当检测到类型为 2 的设备中断 (即定时中断时, 调用 yield() 函数让出 CPU)。

2. 中断处理的完整流程:

1. 例: 通过 `w_stimeout(c, time0 + 100000)` 设置一次中断, 约 0.1 秒后触发

2. 中断处理: CPU 跳转到 kernel/proc_S 中先设置的中断处理入口点。

3. 保存寄存器: kernel/proc_S 保存所有调用者保存寄存器 caller-saved registers 到内核栈。

4. 调用 C 语言处理函数: 跳转到 trap.c 中的 kernel/trap() 或 inserttrap() 函数处理中断。

5. 判断中断类型: 调用 `tevintr()` 函数确定是否为定时中断 (返回值为 2)

6. 定时中断处理: 在 trap.c 中的 devintr() 函数内部, 如果检测到时钟中断 [causes 寄存器值为 0x8000000000000000], 会调用 `clockintr()` 函数。

7. 计时器增加: `clockintr()` 函数增加系统 tick 计数, 数值等待时钟的进程, 并设置下一次时钟中断。

8. 主动让出 CPU: 回到中断处理主流程后, 如果确认是时钟中断 (which_dev == 2), 调用 `yield()` 函数。

9. 进程主动调用 sleep 等事件:

• 文件: kernel/proc.c 第 557-559 行

```
// Go to sleep.
p->chan = SLEEPING;
p->state = SLEEPING;
schedule();
```

10. 休眠: 会调用睡眠函数, 如商业服务器、资源预留系统。

• 优点:

1. 提供可预测的性能保证。
2. 确保资源分配的公平性。
3. 防止进程被“饿死”。

• 缺点:

1. 需维护详细的进程历史记录 (CPU 使用时间)。
2. 不考虑进程优先级的实时性需求。

• 应用:

1. 提供服务质量保证的环境, 如商业服务器、资源预留系统。

• 彩票调度 (Lottery Scheduling):

• 原理: 一种概率性、随机化的调度机制, 通过进程分配“彩票”来决定其优先级, 定期抽签决定“抽中”下一个运行的进程。

• 策略:

1. 每个进程分配一定的彩票 (彩票反向位数加权)。
2. 调度器随机抽取一张彩票。
3. 将该彩票的进程提升下一个 CPU 时期。

• 特性:

1. 彩票转让 (Ticket Transfer): 进程可临时将彩票转让给其他进程。

• 优点:

1. 提供公平性。
2. 支持抢占式调度。
3. 现代操作系统的典型: 引入了文件系统、命令行解释器、在线帮助等现代操作系统的前身。
4. MULTICS 的前身: 为后续的 MULTICS 系统奠定了基础, 而 MULTICS 又影响了 UNIK 的设计

这是保证时间关键任务 (尤其是硬实时任务) 能及时响应的基础保障。

2. 彩票通胀 (Ticket Inflation): 在适当情况下, 进程可能长期占有更多的系统资源。

3. 彩票数量: 不同用户组可以有自己的“货币体系”。

• 优点:

1. 实现简单, 方策快速 (只需随机数生成和简单计算)。
2. 支持不同优先级 (通过不同彩票数量)。
3. 具有良好的公平性, 长期运行中, 可以获得均匀的 CPU 时间与彩票数成正比)。
4. 可以拟多种调度算法。

• 缺点:

1. 短期公平性无法保证 (随机性可能导致短期期内资源分配不均)。
2. 不适合实时系统 (无法提供确定性保障)。
3. 通过牺牲响应时间换取公平性。
4. 适用于单机环境。

• 应用:

1. 在单机上阅读 X11 源代码, 列举出 2 个引发调度的原因? 给出代码的具体文件和行数。
2. 中断处理时间片用尽:

1. 文件: kernel/trap.c 第 75-77 行

```
// give up the CPU if this is a timer interrupt.
if (which_dev == 2)
    yield();
```

此处是 usertrap 函数中, 当检测到类型为 2 的设备中断 (即定时中断时, 调用 yield() 函数让出 CPU)。

2. 中断处理的完整流程:

1. 例: 通过 `w_stimeout(c, time0 + 100000)` 设置一次中断, 约 0.1 秒后触发

2. 中断处理: CPU 跳转到 kernel/proc_S 中先设置的中断处理入口点。

3. 保存寄存器: kernel/proc_S 保存所有调用者保存寄存器 caller-saved registers 到内核栈。

4. 调用 C 语言处理函数: 跳转到 trap.c 中的 kernel/trap() 或 inserttrap() 函数处理中断。

5. 判断中断类型: 调用 `tevintr()` 函数确定是否为定时中断 (返回值为 2)

6. 定时中断处理: 在 trap.c 中的 devintr() 函数内部, 如果检测到时钟中断 [causes 寄存器值为 0x8000000000000000], 会调用 `clockintr()` 函数。

7. 计时器增加: `clockintr()` 函数增加系统 tick 计数, 数值等待时钟的进程, 并设置下一次时钟中断。

8. 主动让出 CPU: 回到中断处理主流程后, 如果确认是时钟中断 (which_dev == 2), 调用 `yield()` 函数。

9. 进程主动调用 sleep 等事件:

• 文件: kernel/proc.c 第 557-559 行

```
// Go to sleep.
p->chan = SLEEPING;
p->state = SLEEPING;
schedule();
```

10. 休眠: 会调用睡眠函数, 如商业服务器、资源预留系统。

• 优点:

1. 提供可预测的性能保证。
2. 确保资源分配的公平性。
3. 防止进程被“饿死”。

• 缺点:

1. 需维护详细的进程历史记录 (CPU 使用时间)。
2. 不考虑进程优先级的实时性需求。

• 应用:

1. 提供服务质量保证的环境, 如商业服务器、资源预留系统。

• 彩票调度 (Lottery Scheduling):

• 原理: 一种概率性、随机化的调度机制, 通过进程分配“彩票”来决定其优先级, 定期抽签决定“抽中”下一个运行的进程。

• 策略:

1. 每个进程分配一定的彩票 (彩票反向位数加权)。
2. 调度器随机抽取一张彩票。
3. 将该彩票的进程提升下一个 CPU 时期。

• 特性:

1. 彩票转让 (Ticket Transfer): 进程可临时将彩票转让给其他进程。

• 优点:

1. 提供公平性。
2. 支持抢占式调度。
3. 现代操作系统的典型: 引入了文件系统、命令行解释器、在线帮助等现代操作系统的前身。
4. MULTICS 的前身: 为后续的 MULTICS 系统奠定了基础, 而 MULTICS 又影响了 UNIK 的设计

• 2. 彩票通胀 (Ticket Inflation): 在适当情况下, 进程可能长期占有更多的系统资源。

3. 彩票数量: 不同用户组可以有自己的“货币体系”。

• 优点:

1. 实现简单, 方策快速 (只需随机数生成和简单计算)。
2. 支持不同优先级 (通过不同彩票数量)。
3. 具有良好的公平性, 长期运行中, 可以获得均匀的 CPU 时间与彩票数成正比)。
4. 可以拟多种调度算法。

• 缺点:

1. 短期公平性无法保证 (随机性可能导致短期期内资源分配不均)。
2. 不适合实时系统 (无法提供确定性保障)。
3. 通过牺牲响应时间换取公平性。
4. 适用于单机环境。

• 应用:

1. 在单机上阅读 X11 源代码, 列举出 2 个引发调度的原因? 给出代码的具体文件和行数。
2. 中断处理时间片用尽:

1. 文件: kernel/trap.c 第 75-77 行

```
// give up the CPU if this is a timer interrupt.
if (which_dev == 2)
    yield();
```

此处是 usertrap 函数中, 当检测到类型为 2 的设备中断 (即定时中断时, 调用 yield() 函数让出 CPU)。

2. 中断处理的完整流程:

1. 例: 通过 `w_stimeout(c, time0 + 100000)` 设置一次中断, 约 0.1 秒后触发

2. 中断处理: CPU 跳转到 kernel/proc_S 中先设置的中断处理入口点。

3. 保存寄存器: kernel/proc_S 保存所有调用者保存寄存器 caller-saved registers 到内核栈。

4. 调用 C 语言处理函数: 跳转到 trap.c 中的 kernel/trap() 或 inserttrap() 函数处理中断。

5. 判断中断类型: 调用 `tevintr()` 函数确定是否为定时中断 (返回值为 2)

6. 定时中断处理: 在 trap.c 中的 devintr() 函数内部, 如果检测到时钟中断 [causes 寄存器值为 0x8000000000000000], 会调用 `clockintr()` 函数。

7. 计时器增加: `clockintr()` 函数增加系统 tick 计数, 数值等待时钟的进程, 并设置下一次时钟中断。

8. 主动让出 CPU: 回到中断处理主流程后, 如果确认是时钟中断 (which_dev == 2), 调用 `yield()` 函数。

9. 进程主动调用 sleep 等事件:

• 文件: kernel/proc.c 第 557-559 行

```
// Go to sleep.
p->chan = SLEEPING;
p->state = SLEEPING;
schedule();
```

10. 休眠: 会调用睡眠函数, 如商业服务器、资源预留系统。

• 优点:

1. 提供可预测的性能保证。
2. 确保资源分配的公平性。
3. 防止进程被“饿死”。

• 缺点:

1. 需维护详细的进程历史记录 (CPU 使用时间)。
2. 不考虑进程优先级的实时性需求。

• 应用:

1. 提供服务质量保证的环境, 如商业服务器、资源预留系统。

• 彩票调度 (Lottery Scheduling):

• 原理: 一种概率性、随机化的调度机制, 通过进程分配“彩票”来决定其优先级, 定期抽签决定“抽中”下一个运行的进程。

• 策略:

1. 每个进程分配一定的彩票 (彩票反向位数加权)。
2. 调度器随机抽取一张彩票。
3. 将该彩票的进程提升下一个 CPU 时期。

• 特性:

1. 彩票转让 (Ticket Transfer): 进程可临时将彩票转让给其他进程。

• 优点:

1. 提供公平性。
2. 支持抢占式调度。
3. 现代操作系统的典型: 引入了文件系统、命令行解释器、在线帮助等现代操作系统的前身。
4. MULTICS 的前身: 为后续的 MULTICS 系统奠定了基础, 而 MULTICS 又影响了 UNIK 的设计

• 2. 彩票通胀 (Ticket Inflation): 在适当情况下, 进程可能长期占有更多的系统资源。

3. 彩票数量: 不同用户组可以有自己的“货币体系”。

• 优点:

1. 实现简单, 方策快速 (只需随机数生成和简单计算)。
2. 支持不同优先级 (通过不同彩票数量)。
3. 具有良好的公平性, 长期运行中, 可以获得均匀的 CPU 时间与彩票数成正比)。
4. 可以拟多种调度算法。

• 缺点:

1. 短期公平性无法保证 (随机性可能导致短期期内资源分配不均)。
2. 不适合实时系统 (无法提供确定性保障)。
3. 通过牺牲响应时间换取公平性。
4. 适用于单机环境。

• 应用:

1. 在单机上阅读 X11 源代码, 列举出 2 个引发调度的原因? 给出代码的具体文件和行数。
2. 中断处理时间片用尽:

1. 文件: kernel/trap.c 第 75-77 行

```
// give up the CPU if this is a timer interrupt.
if (which_dev == 2)
    yield();
```

此处是 usertrap 函数中, 当检测到类型为 2 的设备中断 (即定时中断时, 调用 yield() 函数让出 CPU)。

2. 中断处理的完整流程:

1. 例: 通过 `w_stimeout(c, time0 + 100000)` 设置一次中断, 约 0.1 秒后触发

2. 中断处理: CPU 跳转到 kernel/proc_S 中先设置的中断处理入口点。

3. 保存寄存器: kernel/proc_S 保存所有调用者保存寄存器 caller-saved registers 到内核栈。

4. 调用 C 语言处理函数: 跳转到 trap.c 中的 kernel/trap() 或 inserttrap() 函数处理中断。

5. 判断中断类型: 调用 `tevintr()` 函数确定是否为定时中断 (返回值为 2)

6. 定时中断处理: 在 trap.c 中的 devintr() 函数内部, 如果检测到时钟中断 [causes 寄存器值为 0x8000000000000000], 会调用 `clockintr()` 函数。

7. 计时器增加: `clockintr()` 函数增加系统 tick 计数, 数值等待时钟的进程, 并设置下一次时钟中断。

8. 主动让出 CPU: 回到中断处理主流程后, 如果确认是时钟中断 (which_dev == 2), 调用 `yield()` 函数。

9. 进程主动调用 sleep 等事件:

• 文件: kernel/proc.c 第 557-559 行

```
// Go to sleep.
p->chan = SLEEPING;
p->state = SLEEPING;
schedule();
```

10. 休眠: 会调用睡眠函数, 如商业服务器、资源预留系统。

• 优点:

1. 提供可预测的性能保证。
2. 确保资源分配的公平性。
3. 防止进程被“饿死”。

• 缺点:

1. 需维护详细的进程历史记录 (CPU 使用时间)。
2. 不考虑进程优先级的实时性需求。

• 应用:

1. 提供服务质量保证的环境, 如商业服务器、资源预留系统。

• 彩票调度 (Lottery Scheduling):

• 原理: 一种概率性、随机化的调度机制, 通过进程分配“彩票”来决定其优先级, 定期抽签决定“抽中”下一个运行的进程。

• 策略:

1. 每个进程分配一定的彩票 (彩票反向位数加权)。
2. 调度器随机抽取一张彩票。
3. 将该彩票的进程提升下一个 CPU 时期。

• 特性:

1. 彩票转让 (Ticket Transfer): 进程可临时将彩票转让给其他进程。

• 优点:

1. 提供公平性。
2. 支持抢占式调度。
3. 现代操作系统的典型: 引入了文件系统、命令行解释器、在线帮助等现代操作系统的前身。
4. MULTICS 的前身: 为后续的 MULTICS 系统奠定了基础, 而 MULTICS 又影响了 UNIK 的设计

• 2. 彩票通胀 (Ticket Inflation): 在适当情况下, 进程可能长期占有更多的系统资源。

3. 彩票数量: 不同用户组可以有自己的“货币体系”。

• 优点:

1. 实现简单, 方策快速 (只需随机数生成和简单计算)。
2. 支持不同优先级 (通过不同彩票数量)。
3. 具有良好的公平性, 长期运行中, 可以获得均匀的 CPU 时间与彩票数成正比)。
4. 可以拟多种调度算法。

• 缺点:

1. 短期公平性无法保证 (随机性可能导致短期期内资源分配不均)。
2. 不适合实时系统 (无法提供确定性保障)。
3. 通过牺牲响应时间换取公平性。
4. 适用于单机环境。

• 应用:

1. 在单机上阅读 X11 源代码, 列举出 2 个引发调度的原因? 给出代码的具体文件和行数。
2. 中断处理时间片用尽:

1. 文件: kernel/trap.c 第 75-77 行

```
// give up the CPU if this is a timer interrupt.
if (which_dev == 2)
    yield();
```

此处是 usertrap 函数中, 当检测到类型为 2 的设备中断 (即定时中断时, 调用 yield() 函数让出 CPU)。

2. 中断处理的完整流程:

1. 例: 通过 `w_stimeout(c, time0 + 100000)` 设置一次中断, 约 0.1 秒后触发

2. 中断处理: CPU 跳转到 kernel/proc_S 中先设置的中断处理入口点。

3. 保存寄存器: kernel/proc_S 保存所有调用者保存寄存器 caller-saved registers 到内核栈。

4. 调用 C 语言处理函数: 跳转到 trap.c 中的 kernel/trap() 或 inserttrap() 函数处理中断。

5. 判断中断类型: 调用 `tevintr()` 函数确定是否为定时中断 (返回值为 2)

6. 定时中断处理: 在 trap.c 中的 devintr() 函数内部, 如果检测到时钟中断 [causes 寄存器值为 0x8000000000000000], 会调用 `clockintr()` 函数。

7. 计时器增加: `clockintr()` 函数增加系统 tick 计数, 数值等待时钟的进程, 并设置下一次时钟中断。

8. 主动让出 CPU: 回到中断处理主流程后, 如果确认是时钟中断 (which_dev == 2), 调用 `yield()` 函数。

9. 进程主动调用 sleep 等事件:

• 文件: kernel/proc.c 第 557-559 行

```
// Go to sleep.
p->chan = SLEEPING;
p->state = SLEEPING;
schedule();
```

10. 休眠: 会调用睡眠函数, 如商业服务器、资源预留系统。

• 优点:

1. 提供可预测的性能保证。
2. 确保资源分配的公平性。
3. 防止进程被“饿死”。

• 缺点:

1. 需维护详细的进程历史记录 (CPU 使用时间)。
2. 不考虑进程优先级的实时性需求。

• 应用:

1. 提供服务质量保证的环境, 如商业服务器、资源预留系统。

• 彩票调度 (Lottery Scheduling):

• 原理: 一种概率性、随机化的调度机制, 通过进程分配“彩票”来决定其优先级, 定期抽签决定“抽中”下一个运行的进程。

• 策略:

1. 每个进程分配一定的彩票 (彩票反向位数加权)。
2. 调度器随机抽取一张彩票。
3. 将该彩票的进程提升下一个 CPU 时期。

• 特性:

1. 彩票转让 (Ticket Transfer): 进程可临时将彩票转让给其他进程。

• 优点:

1. 提供公平性。
2. 支持抢占式调度。</

This image is a study guide for the Linux kernel exam, organized into several main sections:

- 基础知识**: Includes basic concepts like memory hierarchy, page tables, and virtual memory.
- 进程与线程**: Details the Linux process structure, scheduling, and thread management.
- 文件系统**: Focuses on the ext4 file system, inode management, and file operations.
- 设备驱动**: Discusses character devices, block devices, and the DMA API.
- 中断与时钟**: Explains interrupt handling, timer drivers, and the tickless kernel.
- 内核链接与构建**: Provides instructions on how to build and debug the kernel.
- 内核模块**: Covers module loading, symbol resolution, and module signatures.
- 内核锁**: Describes various locking mechanisms used in the kernel.
- 内存管理**: A large section detailing the memory subsystem, including the page allocator, slab allocator, and memory protection features like KASLR and ASLR.
- 文件I/O与磁盘子系统**: Details the disk I/O stack, including the bio layer, submit API, and various queueing disciplines.
- 网络子系统**: Covers network protocols, device drivers, and the TCP/IP stack.
- 安全与审计**: Discusses SELinux, AppArmor, and auditd.
- 电源管理**: Details power consumption and battery life optimization.
- 其他主题**: Includes a section on the SystemTap debugger and a summary of the Linux kernel's evolution and future directions.

The guide is highly technical, containing many code snippets, tables, and diagrams to illustrate complex concepts. It is intended for students preparing for the Linux kernel developer certification exam.

因此，在这个版本的XV6中，`vncpy`不设置COW标志。因为执行的命令是完全的内存复制（代码、数据、堆栈等）。

52. Linux中执行fork()时是怎么设置COW的？

- 释放旧地址空间的头部**：`fork()`时，内核为子进程创建父进程的页表的副本。这时，这句话：‘调用VMA虚地址写入显存时，映射页面映射，减少COW头部引用计数（即引用计数零则释放物理页面，增加物理页面引用计数）’指定的可执行文件（如ELF文件头、分析文件头、确定程序的代码段、数据段、BSS段的大小、位置以及程序的入口点）。
- 共享物理页面**：关键在于，内核并不立即复制物理内存。相反，父子进程的页表项（PTE）最初都指向相同的物理页面。
- 标记为只读**：为激发COW，内核将父子进程中指向这些共享物理页的PTE都标记为只读（清除Write权限）。相关子进程的内存区域描述符（vm_area_struct）来指示这些区域是COW区域。
- 解锁页锁**：当进程共享或进程写入某共享页的只读页面时，会触发一个页错误（Page Fault）异常。
- 更新页目录**：内部页错误处理器会看到这是一个对COW页的写操作。
- 全局与局部**：内核将父进程分配给新物理的页面，并将原始共用页面的内容复制到这个新的物理页面。如果引用计数为1，那么另一个进程对应的PTE也可以被标记为可写，因为它现在是最新的拥有者。
- 更新页表**：内核重新写入进程的PTE，使其指向这个新的物理页面，并将PTE标记为可写。
- 引用计数**：引用计数，内核会减少原物理页面的引用计数。如果引用计数为1，那么另一个进程对应的PTE也可被标记为可写，因为它是唯一的一个拥有者。
- 全局与局部**：内核重新写入进程的PTE，使其指向这个新的物理页面。如果设置，在`exec()`执行后，将从新写入地址。
- 可执行文件格式**（如ELF头）：内核需要解析来理解如何加载和运行程序。
- 旧线程的VMA和PTE**：在跨线程地空间内，内核检查这些结构，特别是PTE中的写权限。如果失败，则将从新写入地址。
- 线程数据库实现与应用**：通过`vncpy`函数，将线程的物理页表、指针、COW状态以及物理引用计数，通过线程数据库写入线程的线程页表。这样可以加快线程间的数据共享。
- 线程数据库实现与应用**：通过`vncpy`函数，将线程的物理页表、指针、COW状态以及物理引用计数，通过线程数据库写入线程的线程页表。这样可以加快线程间的数据共享。

53. XV6中`vncpy()`的作用？`vncpy`函数中如何设置COW标志的？

• `vncpy()`的作用：该函数用于在`fork()`系统调用时，复制父进程的用户地址空间到子进程。它通过父进程（old页表，指针大小`sz`）的用户空间查找对应的页表页。1. 查找对应的页表页（PTE）。2. 取得该页面的物理地址（pa）和权限标志（flags）。

3. 分配一个新的物理页表（mem = `kalloc()`）。4. 将父进程物理页面（pa）的内容复制到新分配的物理页表（mem）中。5. 使用`mappte()`将这个新的、包含副本数据的物理页表（mem）映射到子进程的页表（new）中相应的物理地址，并继承父进程页面的权限标志（flags）。

简而言之，`vncpy`为子进程创建了父进程内存的一个完整物理副本。

• **如何设置COW标志**：在XV6实现中，`vncpy`没有实现时复制（Copy-on-Write, COW）机制。它没有父子进程共享物理页面。PTE（写）没有写权限标志。它没有使用或设置任何特定的标志来表示页面是COW状态。

3. **从后备页面链表指向进程的工作集（由“软”缺页异常触发）：**

- 事件：分配零页给页面。当进程发生需求零页异常（Demand Zero Page Fault）时，内核会从其工作集中移除尚未被重映射的页面。最近从其工作集中移除的页面会重新映射到物理页面上，满足时，操作系统会从全零的页面中取出一个页面分配给进程。并将其加入进程的工作集。
- 2. **如果设置COW标志**：在XV6实现中，`vncpy`没有实现时复制（Copy-on-Write, COW）机制。它没有父子进程PTE指向下相同的物理页并清除写权限位。而不是调用这个`vncpy()`。

54. 若`fork()`之后调用`exec()`替换地址空间内空闲时，会发生什么事件？处理该事件时要检查哪些标志？

• 事件：`exec()`系统调用会加载一个新的程序到当前进程的内存空间，完全替换掉`fork()`之后继承的`main()`。

2. 全部 32 个 RISC-V 整数寄存器, 完整的用户模式

```

void usertrapret(void) {
    ...
    // 为下一次用户空间 trap 配置 trapframe
    p->trapframe->kernel_sp = r_sstack(); // kernel pageable
    p->trapframe->kernel_tp = p->ksstack + PGSIZE; // p-堆栈 s-线程切换
    p->trapframe->kernel_trap = (uint64)usertrap;
    p->trapframe->kernel_hartid = r_tp0(); // hartid for interrupt
    // set up the registers that trampoline_S's switch will use to handle traps
    // set S Previous Privilege mode to User.
    x |= ~STATUS_SPP; // clear spp to 0 for user mode
    w_status(x);
    // set S Exception Program Counter to the saved user context
    // set S Exception Program Counter to the saved user context
    // save user to scratch, a0
    userret:
    # 交换 a0 和 scratch
    eswr a0, scratch
    # 保存用户寄存器到 TRAPFRAME
    sd ra, 40(a0)
    # ... (从 trampoline_S 中的 userret
    sd sp, 48(a0)
    sd sp, 56(a0)
    sd tp, 61(a0)
    sd t0, 72(a0)
    # ... (保存所有寄存器)
    # 复原到新的进程上下文操作过程:
    # - 从目标进程的 ra, sp-s0-11 到前进程的 context
    # - 从目标进程的 context, 结构加载 ra, sp-s0-11
    # - ra 的值将决定返回地址, 这使得执行它可以切换到新进程 ra, sp
    # 交换到用户页表
    # 交换所有用户寄存器
    id ra, 40(a0)
    id sp, 48(a0)
    # ... (从 trampoline_S 中的 userret
    id t1, 16(a0)
    # 交换到 usertrap()
    # 交换内核页表地址:
    id t1, 16(a0)
    eswr satp, t1
    sfence, via zero, zero
    zero
    # 跳转到 usertrap()
    jr t1
}

```

另,x64-riscv 中用户态内核态 Trap 的寄存器保存机制在 x64-riscv 中, 进程上下文切换只需要保存较少寄存器, 而用户模式下的 trap 只需要保存少于或等于更全的寄存器管理。下面详细解释这一机制:

当进程从用户态到内核态发生 trap 时, 需要在 trapframe 中保存更多的寄存器。

Trapframe 结构

1. 交换 a0 与 scratch (包含 trapframe 地址):

- 因为需要访问 trapframe, 但又不能丢失 a0 寄存器的值, scratch 在进入用户态之前设置为指向 trapframe 的指针, 通过交换以同时保留 a0 的值并获取 trapframe 地址。

2. 将所有用户寄存器保存到 trapframe:

- 将所有用户寄存器保存到 trapframe。

3. 交换 a0 与 scratch (包含 trapframe 地址):

- 需要切换到自己的内核线程, 通过调用 trapframe 中的 sret 指令正确地将程序的下一条指令写入 scratch 地址。

4. 切换到 trapframe 中的 userret:

- 将用户寄存器回写入 scratch 地址。

5. 跳转到 trapframe:

- 完成中断处理并返回。

这个代码段:

从用户模式到内核模式发生 trap 时, x64 需要在 trapframe 结构中保存完整的处理器状态:

```

struct trapframe {
    /* 0 */ uint64 kernel_sp; // 内核页表
    /* 1 */ uint64 kernel_tp; // 线程切换
    /* 2 */ uint64 kernel_trap; // usertrap() 函数地址
    /* 3 */ uint64 kernel_hartid; // 保存的内核 tp 寄存器值
    /* 4 */ uint64 ra; // 保存的用户程序计数器
    /* 5 */ uint64 sp; // 保存的用户页表
    /* 6 */ uint64 a0; // 保存的用户空间内存
    /* 7 */ uint64 s0; // 保存的用户空间内存
    /* 8 */ uint64 s1; // 保存的用户空间内存
    /* 9 */ uint64 s2; // 保存的用户空间内存
    /* 10 */ uint64 s3; // 保存的用户空间内存
    /* 11 */ uint64 s4; // 保存的用户空间内存
    /* 12 */ uint64 s5; // 保存的用户空间内存
    /* 13 */ uint64 s6; // 保存的用户空间内存
    /* 14 */ uint64 ad; // 保存的用户空间内存
    /* 15 */ uint64 at; // 保存的用户空间内存
    /* 16 */ uint64 af; // 保存的用户空间内存
    /* 17 */ uint64 ag; // 保存的用户空间内存
    /* 18 */ uint64 ah; // 保存的用户空间内存
    /* 19 */ uint64 al; // 保存的用户空间内存
    /* 20 */ uint64 ar; // 保存的用户空间内存
    /* 21 */ uint64 as; // 保存的用户空间内存
    /* 22 */ uint64 at; // 保存的用户空间内存
    /* 23 */ uint64 ar; // 保存的用户空间内存
    /* 24 */ uint64 al; // 保存的用户空间内存
    /* 25 */ uint64 as; // 保存的用户空间内存
    /* 26 */ uint64 at; // 保存的用户空间内存
    /* 27 */ uint64 ar; // 保存的用户空间内存
    /* 28 */ uint64 al; // 保存的用户空间内存
    /* 29 */ uint64 as; // 保存的用户空间内存
    /* 30 */ uint64 at; // 保存的用户空间内存
    /* 31 */ uint64 ar; // 保存的用户空间内存
}

```

当从用户模式到内核模式发生 trap 时, x64 需要在 trapframe 结构中保存完整的处理器状态:

```

struct trapframe {
    /* 0 */ uint64 kernel_sp; // kernel pageable
    /* 1 */ uint64 kernel_tp; // p-堆栈 + PGSIZE
    /* 2 */ uint64 kernel_trap = (uint64)usertrap;
    /* 3 */ uint64 kernel_hartid = r_tp0(); // hartid for interrupt
    // set up the registers that trampoline_S's switch will use to handle traps
    // set S Previous Privilege mode to User.
    x |= ~STATUS_SPP; // clear spp to 0 for user mode
    w_status(x);
    // set S Exception Program Counter to the saved user context
    // set S Exception Program Counter to the saved user context
    // save user to scratch, a0
    userret:
    # 交换 a0 和 scratch
    eswr a0, scratch
    # 保存用户寄存器到 TRAPFRAME
    sd ra, 40(a0)
    # ... (从 trampoline_S 中的 userret
    sd sp, 48(a0)
    sd sp, 56(a0)
    sd tp, 61(a0)
    sd t0, 72(a0)
    # ... (保存所有寄存器)
    # 复原到新的进程上下文操作过程:
    # - 从目标进程的 ra, sp-s0-11 到前进程的 context
    # - 从目标进程的 context, 结构加载 ra, sp-s0-11
    # - ra 的值将决定返回地址, 这使得执行它可以切换到新进程 ra, sp
    # 交换到用户页表
    # 交换所有用户寄存器
    id ra, 40(a0)
    id sp, 48(a0)
    # ... (从 trampoline_S 中的 userret
    id t1, 16(a0)
    # 交换到 usertrap()
    # 交换内核页表地址:
    id t1, 16(a0)
    eswr satp, t1
    sfence, via zero, zero
    zero
    # 跳转到 usertrap()
    jr t1
}

```

2. 全部 32 个 RISC-V 整数寄存器, 完整的用户模式

```

CPU 状态
void usertrapret(void) {
    ...
    // 为下一次用户空间 trap 配置 trapframe
    p->trapframe->kernel_sp = r_sstack(); // kernel pageable
    p->trapframe->kernel_tp = p->ksstack + PGSIZE; // p-堆栈 s-线程切换
    p->trapframe->kernel_trap = (uint64)usertrap;
    p->trapframe->kernel_hartid = r_tp0(); // hartid for interrupt
    // set up the registers that trampoline_S's switch will use to handle traps
    // set S Previous Privilege mode to User.
    x |= ~STATUS_SPP; // clear spp to 0 for user mode
    w_status(x);
    // set S Exception Program Counter to the saved user context
    // set S Exception Program Counter to the saved user context
    // save user to scratch, a0
    userret:
    # 交换 a0 和 scratch
    eswr a0, scratch
    # 保存用户寄存器到 TRAPFRAME
    sd ra, 40(a0)
    # ... (从 trampoline_S 中的 userret
    sd sp, 48(a0)
    sd sp, 56(a0)
    sd tp, 61(a0)
    sd t0, 72(a0)
    # ... (保存所有寄存器)
    # 复原到新的进程上下文操作过程:
    # - 从目标进程的 ra, sp-s0-11 到前进程的 context
    # - 从目标进程的 context, 结构加载 ra, sp-s0-11
    # - ra 的值将决定返回地址, 这使得执行它可以切换到新进程 ra, sp
    # 交换到用户页表
    # 交换所有用户寄存器
    id ra, 40(a0)
    id sp, 48(a0)
    # ... (从 trampoline_S 中的 userret
    id t1, 16(a0)
    # 交换到 usertrap()
    # 交换内核页表地址:
    id t1, 16(a0)
    eswr satp, t1
    sfence, via zero, zero
    zero
    # 跳转到 usertrap()
    jr t1
}

```

在 switch_S 中实现了上下文切换, 保存和恢复这些寄存器, 请简要说明以下寄存器的作用: sstack, sepc, seause, sstatus, sstatus_a0, scratch, a0

3. 返回用户空间 (usertrapret 和 userret)

4. 其他情况包含:

- 内核数据结构 (前 5 个字节): trap 处理程序需要的信息
- 关于核的信息

方面 用户-内核 Trap 改变 S-U 或 保持在 S 模式

线程切换

上文切换

4. status (Supervisor Status Register)

- 作用: 控制和反映处理器的当前状态
- 包含各状态位, 如:
- SIE (Supervisor Interrupt Enable), 控制是否开启 S-mode 中断
- SPIE (Supervisor Previous Privilege), 记录 trap 前的特权模式
- SPIE (Supervisor Previous Privilege), 记录 trap 前的特权模式
- 4. 其他情况包含 Q 表示不是中断

2. uartintr 函数流程:

```

// UART 中断处理函数
void uartintr(void) {
    // 在存放输入字符串处处理
    while(1) {
        int c = uart_getc(); // 尝试从 UART 读取一个字节
        if(c == -1) { // 没有字符可读
            break;
        }
        consoleintr(c); // 将字符传送给控制台处理器函数
    }
    uartstart();
}

```

• 作用: 处理 UART 通用异步收发器设备产生的中断

• 主要用于处理串行通信的数据接收中断

• 流程:

1. 检查 UART 技术看寄存器判断是否有数据可读
2. 如果有数据, 从 UART 数据寄存器读取数据
3. 将读取的数据加入到 console buffer 中
4. 如果读到回车符, 喇叭等特殊输入的进程将环处理导致没有更多数据

1. 中断处理函数流程:

a) devintr 函数流程:

```

1. devintr 函数流程:
1. devintr:
    // 检查并处理设备中断
    // 返回并中断忙状态, 2 表示中断, 1 表示其他中断, 0 表示没有中断
    // 问题: 阅读 uart.c 和 trap.c 中有关中断的代码, 简要总结 devintr 和 uartintr 两个函数内部流程, 及 devintr 内调用 uartintr 的判断条件。
    // 2. 如果有数据, 从 UART 数据寄存器读取数据
    // 回答:
    void usertrapret(void) {
        ...
        // 作用: 保存 trap 发生时的程序数据值
        // 在 trap 发生时, 处理器会自动将当前 PC 保存到 sstatus 中
        // 在 trap 处理完后, 通过 sret 指令返回时, 处理器会将正确的值加载到 PC 中
        // 代码参考:
        void usertrapret(void) {
            ...
            // set up trapframe values that userret will need when
            // the process next traps into the kernel.
            // the process next traps into the kernel.
            p->trapframe->kernel_sp = p->sstack + PGSIZE; // kernel pageable
            p->trapframe->kernel_tp = p->ksstack + PGSIZE; // p-堆栈 + PGSIZE
            p->trapframe->kernel_trap = (uint64)usertrap;
            p->trapframe->kernel_hartid = r_tp0(); // hartid for interrupt
            // set up trapframe in user mode
            // set up the registers that trampoline_S's sret will use, 略
            // get to user space.
            // set S Previous Privilege mode to User.
            unsigned long x = _sstatus();
            x &= ~STATUS_SPP; // clean spp to 0 for user mode
            x |= SSTATUS_SIE; // enable interrupts in user mode
            w_status(x);
            // set S Exception Program Counter to the saved user pc.
            w_spc(r_sstack->enc);
            // ...
        }
    }

```

b) devintr 函数流程:

```

2. devintr:
    // 检查是否为软中断 (最高位为 1, 中断码为 1)
    // 如果是 CPU 时钟引起的软件中断...
    if(epirr() == 0) {
        clockintr(); // 处理时钟中断
    }
    // 清除软件中断处理位
    return 2; // 不是中断
}

```

c) devintr 函数:

```

3. devintr:
    // 检查是否为软中断 (最高位为 1, 中断码为 1)
    // 如果是 CPU 时钟引起的软件中断...
    if(epirr() == 0) {
        clockintr(); // 处理时钟中断
    }
    // 清除软件中断处理位
    return 2; // 不是中断
}

```

d) devintr 函数:

```

4. devintr:
    // 检查是否为软中断 (最高位为 1, 中断码为 1)
    // 如果是 CPU 时钟引起的软件中断...
    if(epirr() == 0) {
        clockintr(); // 处理时钟中断
    }
    // 清除软件中断处理位
    return 2; // 不是中断
}

```

e) 中断处理函数流程 (interrupt)

回答:

1. devintr 函数流程:

1. devintr:

```

1. devintr:
    // 检查并处理设备中断
    // 返回并中断忙状态, 2 表示中断, 1 表示其他中断, 0 表示没有中断
    // 问题: 阅读 uart.c 和 trap.c 中有关中断的代码, 简要总结 devintr 和 uartintr 的判断条件。
    // 2. 如果有数据, 从 UART 数据寄存器读取数据
    // 回答:
    void usertrapret(void) {
        ...
        // 作用: 保存 trap 发生时的程序数据值
        // 在 trap 发生时, 处理器会自动将当前 PC 保存到 sstatus 中
        // 在 trap 处理完后, 通过 sret 指令返回时, 处理器会将正确的值加载到 PC 中
        // 代码参考:
        void usertrapret(void) {
            ...
            // set up trapframe values that userret will need when
            // the process next traps into the kernel.
            // the process next traps into the kernel.
            p->trapframe->kernel_sp = p->sstack + PGSIZE; // kernel pageable
            p->trapframe->kernel_tp = p->ksstack + PGSIZE; // p-堆栈 + PGSIZE
            p->trapframe->kernel_trap = (uint64)usertrap;
            p->trapframe->kernel_hartid = r_tp0(); // hartid for interrupt
            // set up trapframe in user mode
            // set up the registers that trampoline_S's sret will use, 略
            // get to user space.
            // set S Previous Privilege mode to User.
            unsigned long x = _sstatus();
            x &= ~STATUS_SPP; // clean spp to 0 for user mode
            x |= SSTATUS_SIE; // enable interrupts in user mode
            w_status(x);
            // set S Exception Program Counter to the saved user pc.
            w_spc(r_sstack->enc);
            // ...
        }
    }

```

2. devintr:

3. devintr:

4. devintr:

5. devintr:

6. devintr:

7. devintr:

8. devintr:

9. devintr:

10. devintr:

11. devintr:

12. devintr:

13. devintr:

14. devintr:

15. devintr:

16. devintr:

17. devintr:

18. devintr:

19. devintr:

20. devintr:

21. devintr:

22. devintr:

23. devintr:

24. devintr:

25. devintr:

26. devintr:

27. devintr:

28. devintr:

29. devintr:

30. devintr:

31. devintr:

32. devintr:

33. devintr:

34. devintr:

35. devintr:

36. devintr:

37. devintr:

38. devintr:

39. devintr:

40. devintr:

41. devintr:

42. devintr:

43. devintr:

44. devintr:

45. devintr:

46. devintr:

47. devintr:

48. devintr:

49. devintr:

50. devintr:

51. devintr:

52. devintr:

53. devintr:

54. devintr:

55. devintr:

56. devintr:

57. devintr:

58. devintr:

59. devintr:

60. devintr:

61. devintr:

62. devintr:

63. devintr:

64. devintr:

65. devintr:

66. devintr:

67. devintr:

68. devintr:

69. devintr:

70. devintr:

71. devintr:

72. devintr:

73. devintr:

74. devintr:

75. devintr:

76. devintr:

77. devintr:

78. devintr:

79. devintr:

80. devintr:

81. devintr:

82. devintr:

83. devintr:

84. devintr:

85. devintr:

86. devintr:

87. devintr:

88. devintr:

89. devintr:

90. devintr:

91. devintr:

92. devintr:

93. devintr:

94. devintr:

95. devintr:

96. devintr:

97. devintr:

98. devintr:

99. devintr:

100. devintr:

101. devintr:

102. devintr:

103. devintr:

104. devintr:

105. devintr:

106. devintr:

107. devintr:

108. devintr:

109. devintr:

110. devintr:

111. devintr:

112. devintr:

113. devintr:

114. devintr:

115. devintr:

116. devintr:

117. devintr:

118. devintr:

119. devintr:

120. devintr:

121. devintr:

122. devintr:

123. devintr:

124. devintr:

125. devintr:

126. devintr:

127. devintr:

128. devintr:

129. devintr:

130. devintr:

131. devintr:

132. devintr:

133. devintr:

134. devintr:

135. devintr:

136. devintr:

137. devintr:

138. devintr:

139. devintr:

140. devintr:

141. devintr:

142. devintr:

143. devintr:

144. devintr:

145. devintr:

146. devintr:

147. devintr:

148. devintr:

149. devintr:

150. devintr:

151. devintr:

152. devintr:

153. devintr:

154. devintr:

155. devintr:

156. devintr:

157. devintr:

158. devintr:

159. devintr:

160. devintr:

161. devintr:

162. devintr:

163. devintr:

164. devintr:

165. devintr:

166. devintr:

167. devintr:

168. devintr:

169. devintr:

170. devintr:

171. devintr:

172. devintr:

173. devintr:

174. devintr:

175. devintr:

176. devintr:

177. devintr:

178. devintr:

179. devintr:

180. devintr:

181. devintr:

182. devintr:

183. devintr:

184. devintr:

185. devintr:

186. devintr:

187. devintr:

188. devintr:

189. devintr:

190. devintr:

191. devintr:

192. devintr:

193. devintr:

194. devintr:

195. devintr:

196. devintr:

197. devintr:

198. devintr:

199. devintr:

200. devintr:

201. devintr:

202. devintr:

203. devintr:

204. devintr:

205. devintr:

206. devintr:

207. devintr:

208. devintr:

209. devintr:

210. devintr:

211. devintr:

212. devintr:

213. devintr:

214. devintr:

215. devintr:

216. devintr:

217. devintr:

218. devintr:

219. devintr:

220. devintr:

221. devintr:

222. devintr:

223. devintr:

224. devintr:

225. devintr:

226. devintr:

227. devintr:

228. devintr:

229. devintr:

230. devintr:

231. devintr:

232. devintr:

233. devintr:

234. devintr:

235. devintr:

236. devintr:

237. devintr:

238. devintr:

239. devintr:

240. devintr:

241. devintr:

242. devintr:

243. devintr:

244. devintr:

245. devintr:

246. devintr:

247. devintr:

248. devintr:

249. devintr:

250. devintr:

251. devintr:

252. devintr:

253. devintr:

254. devintr:

255. devintr:

256. devintr:

257. devintr:

258. devintr:

259. devintr:

260. devintr:

261. devintr:

262. devintr:

263. devintr:

264. devintr:

265. devintr:

266. devintr:

267. devintr:

268. devintr:

269. devintr:

270. devintr:

271. devintr:

272. devintr:

273. devintr:

274. devintr:

275. devintr:

276. devintr:

277. devintr:

278. devintr:

279. devintr:

280. devintr:

281. devintr:

282. devintr:

283. devintr:

284. devintr:

285. devintr:

286. devintr:

287. devintr:

288. devintr:

289. devintr:

290. devintr:

291. devintr:

292. devintr:

293. devintr:

294. devintr:

295. devintr:

296. devintr:

297. devintr:

298. devintr:

299. devintr:

300. devintr:

301. devintr:

302. devintr:

303. devintr:

304. devintr:

305. devintr:

306. devintr:

307. devintr:

308. devintr:

309. devintr:

310. devintr:

311. devintr:

312. devintr:

313. devintr:

314. devintr:

315. devintr:

316. devintr:

317. devintr:

318. devintr:

319. devintr:

320. devintr:

321. devintr:

322. devintr:

323. devintr:

324. devintr:

325. devintr:

326. devintr:

327. devintr:

328. devintr:

329. devintr:

330. devintr:

331. devintr:

332. devintr:

333. devintr:

334. devintr:

335. devintr:

336. devintr:

337. devintr:

338. devintr:

339. devintr:

340. devintr:

341. devintr:

342. devintr:

343. devintr:

344. devintr:

345. devintr:

346. devintr:

347. devintr:

348. devintr:

349. devintr:

350. devintr:

351. devintr:

352. devintr:

353. devintr:

354. devintr:

355. devintr:

356. devintr:

357. devintr:

358. devintr:

359. devintr:

360. devintr:

361. devintr:

362. devintr:

363. devintr:

364. devintr:

365. devintr:

366. devintr:

367. devintr:

368. devintr:

369. devintr:

370. devintr:

371. devintr:

372. devintr:

373. devintr:

374. devintr:

375. devintr:

376. devintr:

377. devintr:

378. devintr:

379. devintr:

380. devintr:

381. devintr:

382. devintr:

383. devintr:

384. devintr:

385. devintr:

386. devintr:

387. devintr:

388. devintr:

389. devintr:

390. devintr:

391. devintr:

392. devintr:

393. devintr:

394. devintr:

395. devintr:

396. devintr:

397. devintr:

398. devintr:

399. devintr:

400. devintr:

401. devintr:

402. devintr:

403. devintr:

404. devintr:

405. devintr:

406. devintr:

407. devintr:

408. devintr:

409. devintr:

410. devintr:

411. devintr:

412. devintr:

413. devintr:

414. devintr:

415. devintr:

416. devintr:

417. devintr:

418. devintr:

419. devintr:

420. devintr:

421. devintr:

422. devintr:

423. devintr:

424. devintr:

425. devintr:

426. devintr:

427. devintr:

428. devintr:

429. devintr:

430. devintr:

431. devintr:

432. devintr:

433. devintr:

434. devintr:

435. devintr:

436. devintr:

437. devintr:

438. devintr:

439. devintr:

440. devintr:

441. devintr:

442. devintr:

443. devintr:

444. devintr:

445. devintr:

446. devintr:

447. devintr:

448. devintr:

449. devintr:

450. devintr:

451. devintr:

452. devintr:

453. devintr:

454. devintr:

455. devintr:

456. devintr:

457. devintr:

458. devintr:

459. devintr:

460. devintr:

461. devintr:

462. devintr:

463. devintr:

464. devintr:

465. devintr:

466. devintr:

467. devintr:

468. devintr:

469. devintr:

470. devintr:

471. devintr:

472. devintr:

473. devintr:

474. devintr:

475. devintr:

476. devintr:

477. devintr:

478. devintr:

479. devintr:

480. devintr:

481. devintr:

482. devintr:

483. devintr:

484. devintr:

485. devintr:

486. devintr:

487. devintr:

488. devintr:

489. devintr:

490. devintr:

491. devintr:

492. devintr:

493. devintr:

494. devintr:

495. devintr:

496. devintr:

497. devintr:

498. devintr:

499. devintr:

500. devintr:

501. devintr:

502. devintr:

503. devintr:

504. devintr:

505. devintr:

506. devintr:

507. devintr:

508. devintr:

509. devintr:

510. devintr:

511. devintr:

512. devintr:

513. devintr:

514. devintr:

515. devintr:

516. devintr:

517. devintr:

518. devintr:

519. devintr:

520. devintr:

521. devintr:

522. devintr:

523. devintr:

524. devintr:

525. devintr:

526. devintr:

527. devintr:

528. devintr:

529. devintr:

530. devintr:

531. devintr:

532. devintr:

533. devintr:

534. devintr:

535. devintr:

536. devintr:

537. devintr:

538. devintr:

539. devintr:

540. devintr:

541. devintr:

542. devintr:

543. devintr:

544. devintr:

545. devintr:

546. devintr:

547. devintr:

548. devintr:

549. devintr:

550. devintr:

551. devintr:

552. devintr:

553. devintr:

554. devintr:

555. devintr:

556. devintr:

557. devintr:

558. devintr:

559. devintr:

560. devintr:

561. devintr:

562. devintr:

563. devintr:

564. devintr:

565. devintr:

566. devintr:

567. devintr:

568. devintr:

569. devintr:

570. devintr:

571. devintr:

572. devintr:

573. devintr:

574. devintr:

575. devintr:

576. devintr:

577. devintr:

578. devintr:

579. devintr:

580. devintr:

581. devintr:

582. devintr:

583. devintr:

584. devintr:

585. devintr:

586. devintr:

587. devintr:

588. devintr:

589. devintr:

590. devintr:

591. devintr:

592. devintr:

593. devintr:

594. devintr:

595. devintr:

596. devintr:

597. devintr:

SJF (Shortest Job First) 最短作业优先:

- 特点: 可以是非抢占式或抢占式。
STCF/STTN: 选择预计下一个 CPU 执行时间最短的进程并运行。
- 特点: 在抢占式情况下, 如果所有进程同时到达, 平均等待时间和平均周转时间最长。
- 缺点:
 - 难以准确预测下一个 CPU 执行时间。
 - 可能最长作业饥饿 (Starvation), 即长作业一直得不到运行机会。
 - 抢占上书版本中, 如果一个长作业已经开始执行, 却后来来了更短的工作, 也必须等长作业完成或阻塞。
- STC (Shortest Time-to-Completion First) / SRTN (Shortest Remaining Time Next) 最短剩余时间优先:**
- 特点: 抢占式的 SJF。调度器选择剩余执行时间最短的进程。如果一个新到达的进程比当前正在执行进程的剩余时间要短, 则抢占当前进程。
- 优点:
 - 提供一定的平均周转时间, 相比抢占式 SJF, 对后来到达的短作业更快。
 - 同样存在被长执行时间的困难。
 - 抢占与下文讲的会带来额外的开销。
- RR (Round Robin) 时间片轮转:**
- 特点: 地址式。将所有就绪进程放入FCFS队列。调度器选择一个进程, 分配一个固定的时间片(Quantum), 期间内单机运行。若进程未完成或阻塞, 则被抢占并移到就绪队列末尾。
- 优点:
 - 公平性高, 每个进程都能获得运行机会。
 - 响应时间较短, 适合分时系统和交互系统。实现相对简单。
- 缺点:
 - 性能对时间片长度敏感, 时间片太长, 上下文切换大; 时间片太短, 则退化为FCFS。
 - 平均周转时间常常比 SJF/STCF 差。平均周转时间常比 FCFS 快。

总结:

- | 算法名 | 核心特点 | 优点 | 缺点 |
|-------------|----------------------|--|--------------------------------|
| FIFO / FCFS | 按到达顺序执行, 并发分配和检查代码 | 简单, 公平, 能保证 | 时间最长, 周转时间最长, 阻塞时间长, 会长时间占用CPU |
| SJF | 按优先级抢占, 并发分配, 内存带宽保证 | 公平, 能降低平均周转时间, 提高性能, 长作业可能轮流执行, 短作业可同时到达 | 时间最长, 周转时间最长, 阻塞时间长, 会长时间占用CPU |
| STCF / SRTN | 按优先级抢占, 并发分配, 复制... | 最佳平均, 能以预期执行时间, 对短作业, 下文切换开销小, 可抢占响应快 | 时间最长, 周转时间最长, 阻塞时间长, 会长时间占用CPU |

构件体。xv6 没有实现内核线程。

2. 进程优先: xv6 的调度策略没有实现进程优先级。

3. 挑选顺序: xv6 采用简单的轮流(Round-Robin)方式, 按固定顺序给全局进程表 proc 数组, 选择第一个状态为 RUNNABLE 的进程运行。

4. 可能的优化思路:

- 引入优先级(Priority Scheduling):
 - 在 struct proc 中增加一个 priority 字段。
 - 优先级进队列, 便不再等待另一个 RUNNABLE 进程, 而是选择最高优先级的 RUNNABLE 进程。
 - 优先级进队列的创建, 根据进程的等待时间或 CPU 使用情况调整, 倾向于提高等待时间或 IO 设备进程的优先级。这可以部分模拟 SJF/SRTCF 的效果, 改善平均周转时间。
 - 优先级进队列的创建, 也可以是动态调整的, 例如, 根据进程的等待时间或 CPU 使用情况, 动态调整优先级, 平衡负载, 减少多核环境下全局进程表的竞争。
- 引入优先级(Multi-level Feedback Queue, MLFQ):
 - 维护多个运行队列, 每个队列对应一个优先级。
 - 先进先出, 是最简单的实现。
 - 避免某些 CPU 过载而其他 CPU 空闲。
 - 近似 SJF/SRTCF, 尝试模拟运行时间, 例如基于历史运行时间, 先忙则先执行。(实现复杂且预测不一定准确)。

详细分析:

- 1. 调度单位:** xv6 的调度是基于进程(Process)的, 内核管理的基本单位是 struct proc 结构体。(只在 kernel/proc.c)。xv6 没有实现内存管理的函数, 每个进程拥有自己独立的线程空间。内核线程, 上下文等。上下文切换(switch)保存和恢复的是整个进程的上下文(struct context)。
- 2. 进程优先:** xv6 的调度策略没有实现进程优先级。所有进程都用相同的调度策略。
- 3. 挑选顺序:** xv6 的调度器 (scheduler) 通过 Round-Robin 方式。它按固定的全局进程表 proc 数组 (forip = proc; p &proc[NPROC]; p += 1) 找到第一个状态为 RUNNABLE 的进程时, 就选择该进程运行。

设计好处:

- 隔离性(Isolation): 通过 fork() 创建独立的子进程来隔离调用环境, 即使子进程修改父进程的全局变量, 它也不会影响父进程 Shell 的稳定性。
- 代理与简写(Signature): 它只需要知道每个外部命令的具体实现。它只需要通过统一的 fork() 和 exec() 接口来创建和加载程序, exec() 代替直接操作的程序都可以被 Shell 行政。
- 资源管理(Resource Management): fork() 复制文件描述符, 以便子进程能够使用它们。在子进程 exec() 之前分配资源, 之后释放资源, 从而保证了父子进程共享同一套文件描述符。内核内部的 READER 和 PINGER 处理。wait() 机制保证父子进程可以收回子进程结束时遗留的资源, 防止资源泄漏。
- 并发性(Concurrency): fork() 后父子进程并发执行。Shell 可以并行地运行命令, 通过 fork() 等待子进程返回命令, 或通过 wait() 来继续接收命令。命令, 通过 sh.c 中 truncat 的 BACK 类型实现, 提高了交互效率。

4. 等待进程结束(wait):

- 1. Shell 读取并解析命令:** Shell 进程 (sh) 首先通过 getcmd() 读取用户输入的命令, 然后调用 parsecmd() 解析命令, 将其构造成为脚本表示 (如 exec, pipe 等)。
- 2. 创建子进程(fork):** 对于大多数命令除了像 cd 这样的内置命令, Shell 进程执行完毕, 以便回收子进程资源并获取其退出状态。Shell 则调用 wait() 系统调用 (对应 kernel/proc.c 中的 wait 函数)。

- fork() 会创建一个新的子进程, 这个子进程几乎是父进程 (Shell) 的一个副本。它复制父进程内的存空间 (umalloc)、文件描述符 (fdopen)、当前工作目录 (dup) 等。
- 关键区别在于 (对父进程返回子进程的 PID, 对子进程返回 0)。这使得代码可以区分父子进程。释放子进程后的资源 (如 pipe 结构体, 内核线程等, 通过 freepool), 并返回子进程的 PID。
- wait() 会检测调用进程是否处在 ZOMBIE 状态的子进程。ZOMBIE 状态表示子进程已经 exit() 但其父进程尚未调用 wait()。
- 如果找到子进程, 对父进程返回前台命令, 并通过 freepool, 并返回子进程的 PID, 并返回子进程的退出状态, x86 会将子进程转入 SLEEPING 状态 (通过 sleep(), &wait_lock)。直到某个子进程调用 exit()。

子进程调用 exit() 时, 会将其状态设置为 ZOMBIE, 保存退出状态, 并唤醒可能正在 ZOMBIE 状态的父进程 (wakeip(p->parent))。

// Return -1 if this process has no children.

// ...省略中断开启 ...

int found = 0;

for (p = proc; p < &proc[NPROC]; p++) {

 acquire(&p->lock);

 if (p->state == RUNNABLE) {

 // Switch to chosen process.

 p->state = RUNNING;

 c->proc = p;

 switch(&c->context, &p->context);

 } // Process is done running for now,

 c->proc = 0;

 found = 1;

 if (p->state == ZOMBIE) {

 // 我们前面几个问题的解答(scheduler)。

 // 本题中管理的基本单位是 struct proc

1. 调度单位: xv6 的调度是基于进程 (Process) 的。

内核中管理调度的基本单位是 struct proc

2. 找到僵尸进程

3. 通知 SJF/STCF:

尝试预测进程运行时间 (例如基于历史运行时间, 偏先调度值作业, 实现复杂且预测不一定准确)。