WIND RIVER

Education Services

异常，中断和定时器

Agenda

异常, 中断和定时器

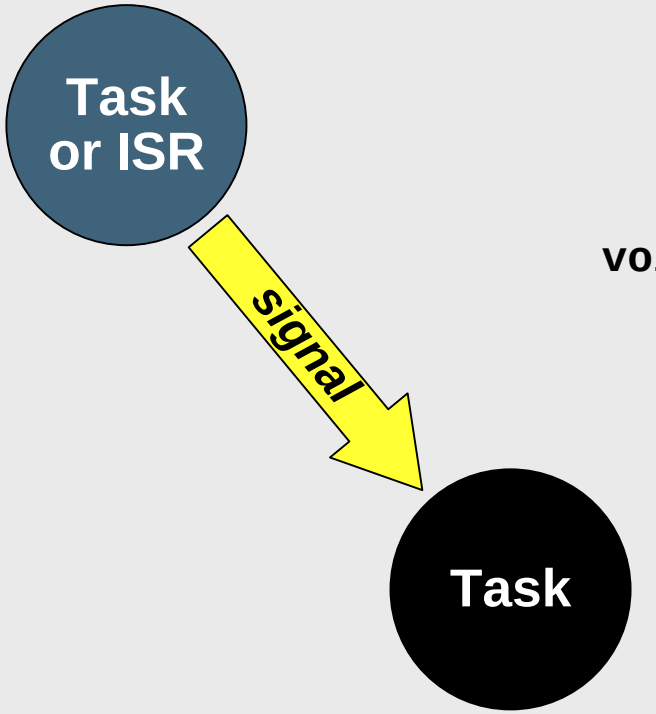
- 异常处理和信号
- 安装信号处理程序来处理异常
- 中断服务程序基础
- 中断处理的例子
- 中断服务程序设计指导
- 定时器和系统时钟
- 看门狗定时器
- 查询
- 辅助时钟

Exception Handling Overview

- *exception* 是计划外的事件
  - 例如: 非法指令, 数据或指令错误, 浮点或整数溢出, 除以0等
  - CPU产生内部中断
  - 强制改变PC指针到一个预定地址来处理exception
- BSP 在启动阶段安装异常处理程序
  - VxWorks 通过发送信号(signal)跟用户任务通信
  - 用户安装 signal 服务程序然后运行
  - 如果不安装 signal 服务程序, 执行系统的缺省操作
    - 典型情况下任务被中止
    - 打印异常信息

Signals

```
kill(tid,signo);
```




```
void normalCode ()
     {
     …
     /* incoming
      * signal */
     …
     }
```

```
void handler ()
     {
     …
     /* deal with
      * signal */
     …
     }
```

VxWorks Signals

- 对于没有安装服务程序的signal, 都将被忽略
- 可以安装处理程序处理 *SIGKILL* 信号(否则将忽略)
- 没有 *SIGPIPE*, 或 *SIGURG* 的信号处理函数可用
- 如果执行 taskDelay( ) 的任务被信号中断, 将设置 errno = *EINTR* 并返回 *ERROR*

Important Caveats

- 对于通常的任务间通信不建议使用signal, 因为:
  - 当发生优先级继承时, 处理 signal 的任务优先级不是期望的优先级
  - 扰乱了预期的执行顺序
  - 如果 signal 的处理函数和接收 signal 的任务调用了相同的函数, 将引发重入问题
  - 收到 signal 的任务将即时终止, 如果持有互斥信号量将引起共享资源状态不一致
- sigLib (以及 POSIX) 支持 SystemV 和 BSD 类型的 signal 接口
  - 不要混合使用

Agenda

异常, 中断和定时器

- 异常处理和信号
- 安装信号处理程序来处理异常
- 中断服务程序基础
- 中断处理的例子
- 中断服务程序设计指导
- 定时器和系统时钟
- 看门狗定时器
- 查询
- 辅助时钟

Registering a Signal Handler

signal (signo, handler)

signo　Signal number

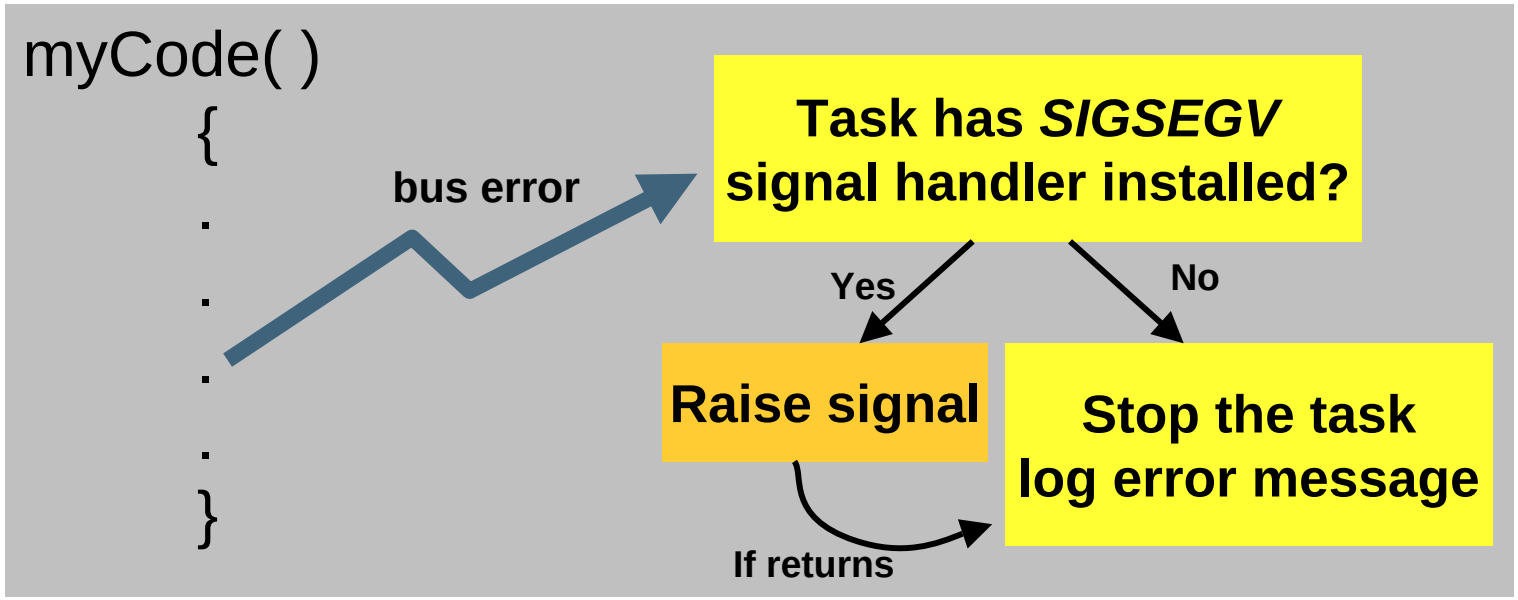
handler　signal 服务程序 (如果忽略这个signal, 则使用*SIG_IGN*)

- 返回前面注册的signal 服务程序或者 *SIG_ERR*
- Signal 服务程序应该按照下面的模式声明

```
void sigHandler (int sig); /* Or */
void sigHandler (int sig, int code, struct sigcontext * p);
        /* (See notes below) */
```

Signals and Exceptions

- 硬件 exceptions 包括总线错误(bus error), 地址错误(address error), 除0错误(divide by zero), 浮点溢出(floating point overflow), 等
- 系统缺省定义了32个 signal 与 exception 对应(例如 *SIGBUS* 对应于PowerPC的总线错误; *SIGFPE* 对应于浮点不可用)



Exception Signal Handler

- Exception 典型的 signal 服务程序
  - ***taskExit( )*** – 终止任务
  - ***taskRestart( )*** – 重启任务
  - ***longjmp( )*** – 恢复到 setjmp( ) 存储的位置执行
  - ***exit()*** – 结束进程
  - 直接返回
    - 出异常的任务处于挂起状态

Agenda

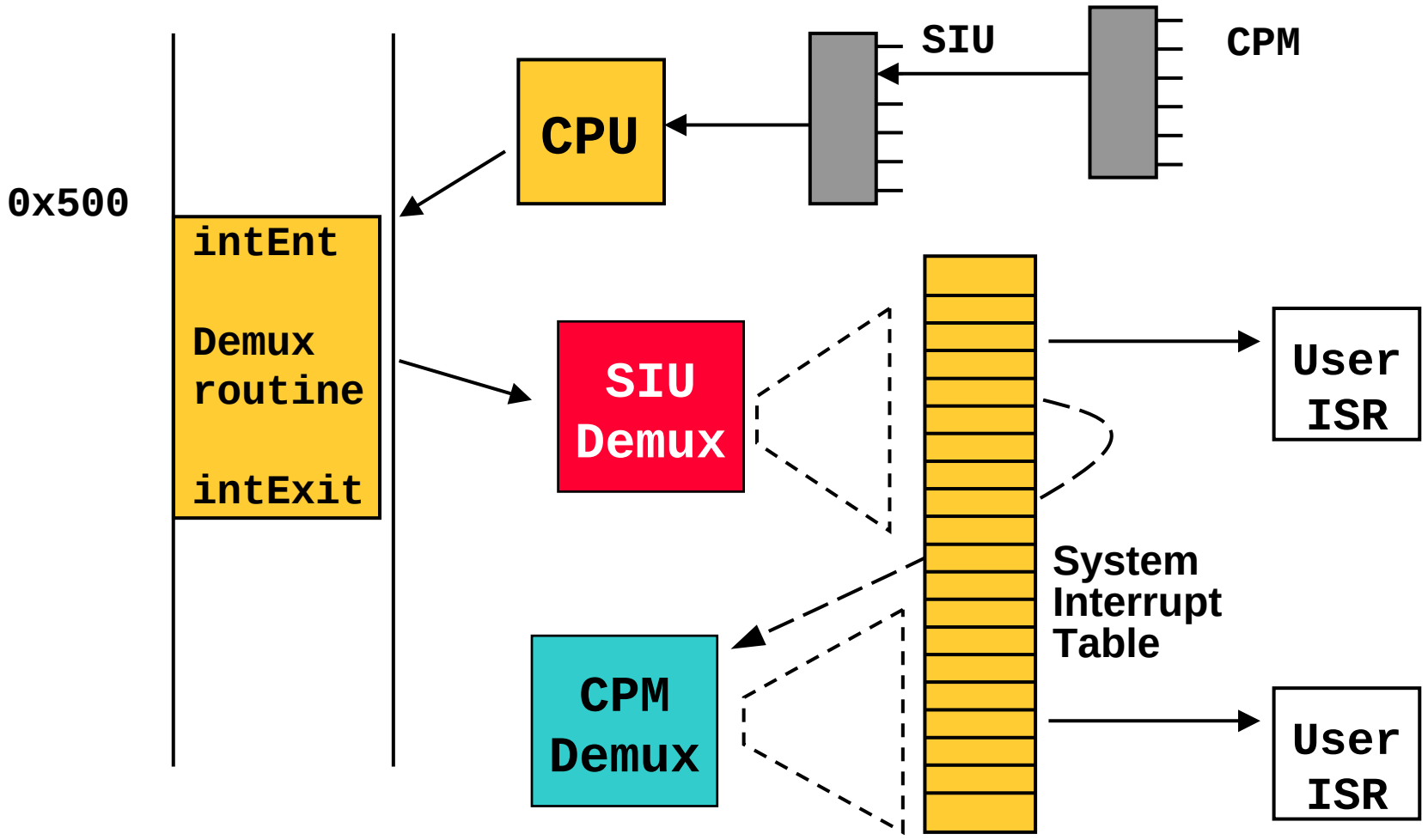
异常, 中断和定时器

- 异常处理和信号
- 安装信号处理程序来处理异常
- 中断服务程序基础
- 中断处理的例子
- 中断服务程序设计指导
- 定时器和系统时钟
- 看门狗定时器
- 查询
- 辅助时钟

Hardware Device Interrupts

- 中断用于通知cpu有事件发生
- 用户可以自定义中断服务程序, 当中断发生后系统会自动执行
- 中断服务程序
  - 中断服务程序不是任务
  - 中断服务程序不能阻塞(pend)
  - 中断服务程序与任务通信只能通过信号量, 队列或“volatile”类型的全局变量等
  - 中断服务程序应该设计得尽量简单
- 在cpu内, 定时器是最普通的中断源

Interrupts(PPC)



Interrupts API

- **intConnect(vector, pISR, pArg)**
  - 为中断向量注册中断服务程序
- **intDisconnect(vector, pISR, pArg)**
  - 删除指定中断向量上的中断服务程序
- **intEnable(vector)**
  - 使能中断
- **intDisable(vector)**
  - 禁止中断

Device Drivers

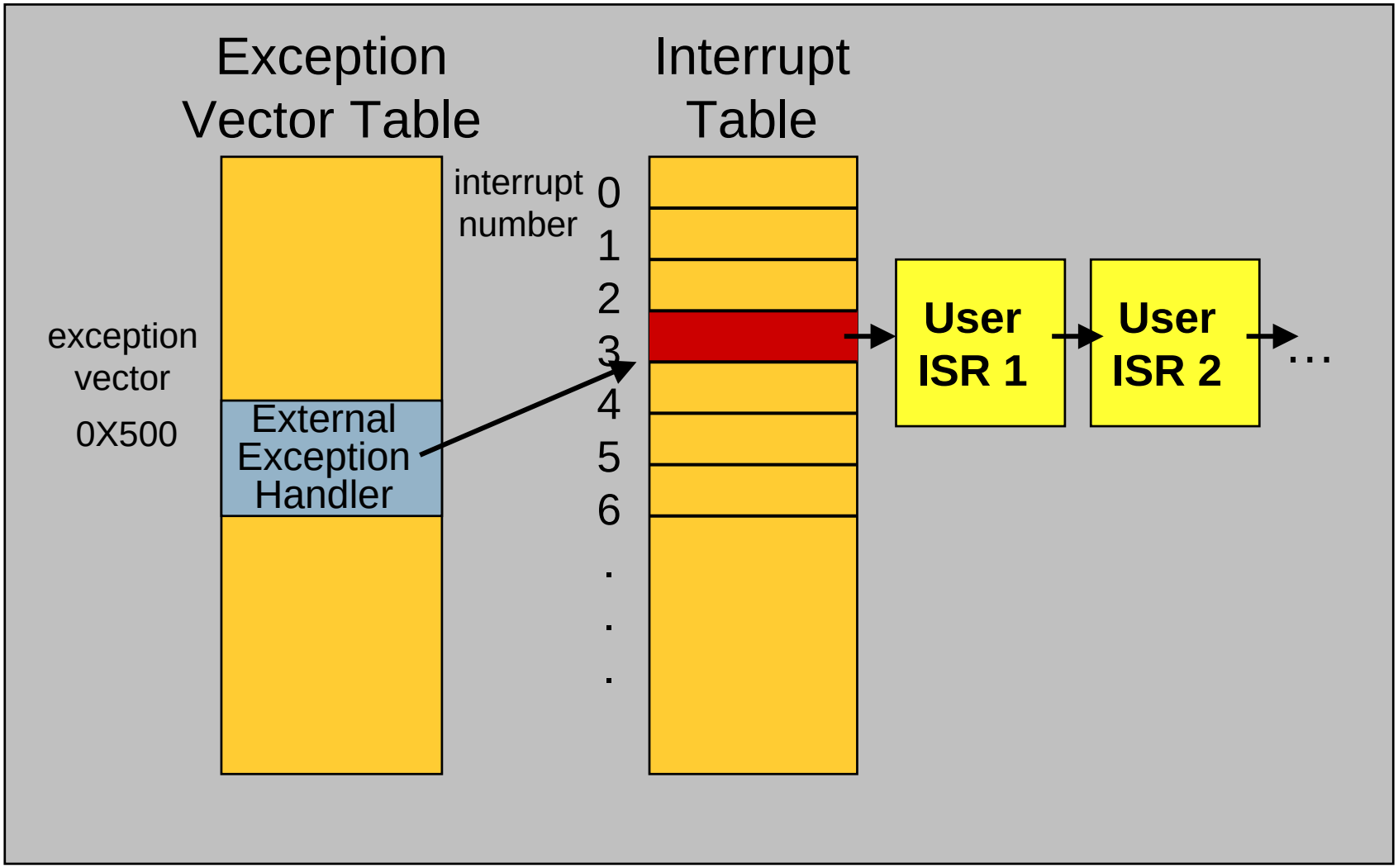
- 相关内容可以参考:
  - *VxWorks Device Driver Developer's Guide*
  - *VxWorks BSP Developer's Guide*
  - *VxWorks Kernel Programmer's Guide*
  - *Wind River Workbench Users Guide*
  - *General Purpose Platform, VxWorks Edition, Board Support Package Workshop*

Agenda

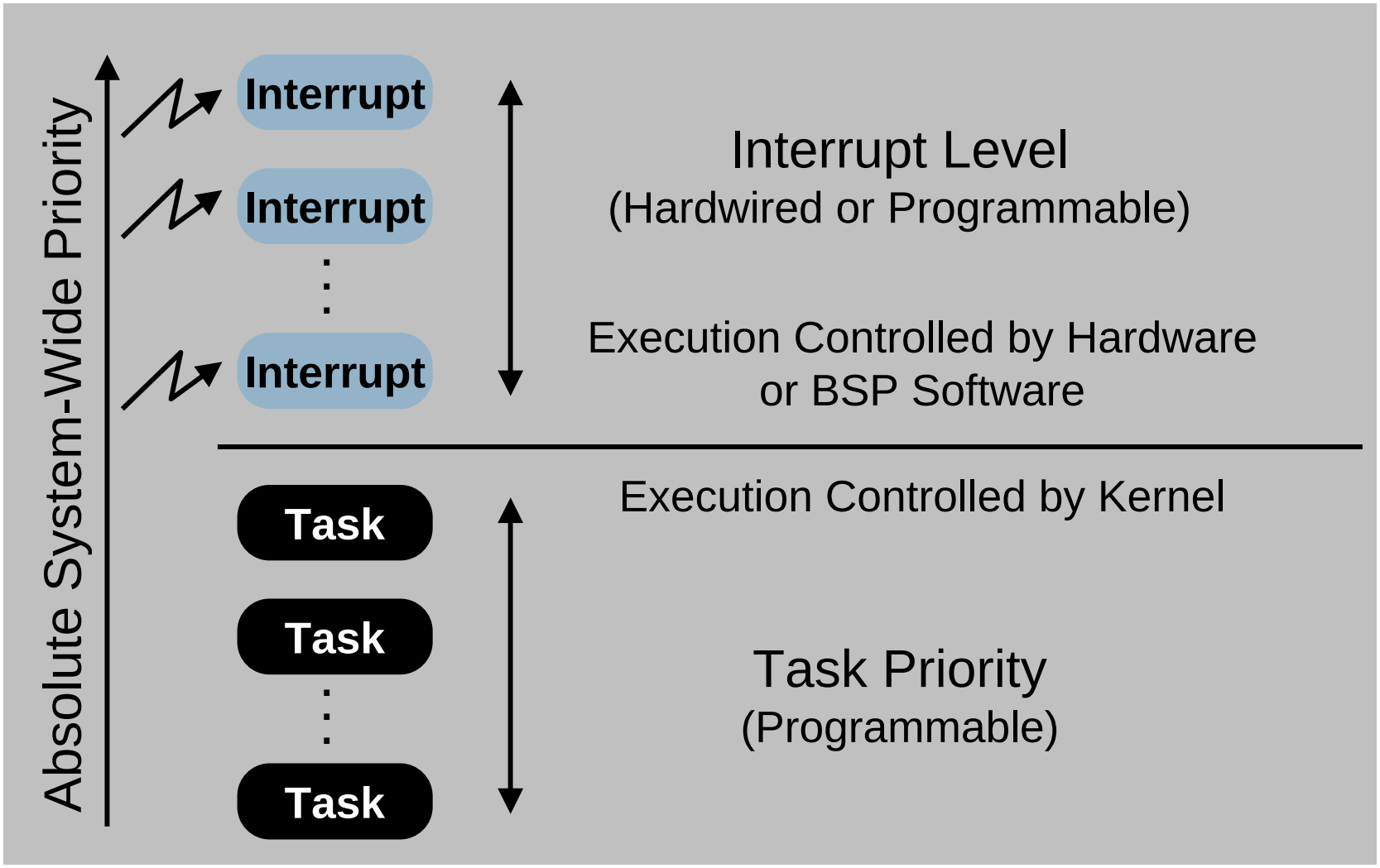
异常, 中断和定时器

- 异常处理和信号
- 安装信号处理程序来处理异常
- 中断服务程序基础
- 中断处理的例子
- 中断服务程序设计指导
- 定时器和系统时钟
- 看门狗定时器
- 查询
- 辅助时钟

Interrupt Handling Example (PowerPC)

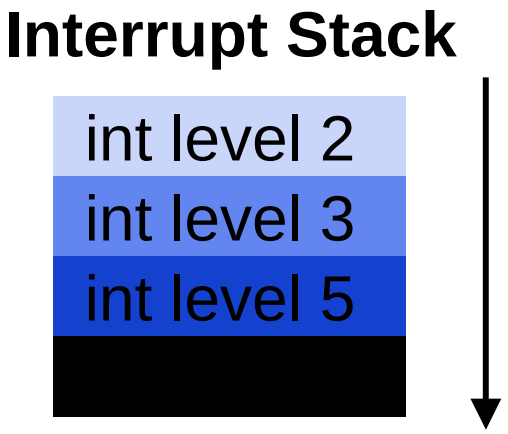


Interrupts and Priorities



Interrupt Stack

- 许多cpu架构都使用单一的中断专用栈 (只有一个)
- 中断栈在系统启动阶段分配
- 中断栈的大小有宏 *ISR_STACK_SIZE* 来定义; 定义在通用头文件 configAll.h 中, 如果需要修改, 建议在 config.h 中先 #undef,再#define新值
- 必须足够大, 能够满足最坏的嵌套情况



Agenda

异常, 中断和定时器

- 异常处理和信号
- 安装信号处理程序来处理异常
- 中断服务程序基础
- 中断处理的例子
- 中断服务程序设计指导
- 定时器和系统时钟
- 看门狗定时器
- 查询
- 辅助时钟

ISR Restrictions

- 所有就绪的中断服务程序都执行完, 任务才能执行, 无论多高的任务优先级, 都不能凌驾于中断之上
- 部分函数在中断服务程序中不允许调用
  - 不允许带有阻塞操作的拿函数
  - 不能调用 *semTake( )*
  - 不能调用 *malloc( )* (可能使用semTake)
  - 不能调用IO系统函数 *(*no *printf( )*) (可能使用信号量)
  - 对于使用外部浮点协处理器(FP co-processor)的cpu, 不能使用浮点指令
    - 没有对这部分寄存器执行压栈出栈操作
- *VxWorks Kernel Programmer's Guide* 中包含了一份可以在中断中调用的列表

ISR Guidelines

- 保持中断服务程序尽量短, 否则:
  - 将对同优先级或低优先级中断造成延迟
  - 延迟所有的任务执行
  - 难以调试
- 避免在中断中使用浮点操作
  - 速度太慢
  - 中断服务程序必须显式的调用 fppSave( ) 和 fppRestore( )
- 尽最大可能把工作交给相关任务去做, 尤其是:
  - 持续时间长的工作
  - 重要程度不高的工作

Typical ISR

- 读写IO寄存器
- 或通过下列方式与任务通信
  - 写内存
  - 以non-block方式写消息队列
  - 释放二进制信号量或计数信号量
  - 任务和中断共享的变量通常增加 “volatile” 类型

Debugging ISRs

- 调用 logMsg 把调试信息打印在串口

  logMsg (“foo = %d\n”, foo, 0, 0, 0, 0, 0);
- 发送请求给 tLogTask 做打印
  - 格式化字符串带有6个参数
  - 参数必须是4字节的类型(long, int, unsigned long)
- Or,
  - 系统级调试
    - WDB agent
    - Emulator

Agenda

异常，中断和定时器

- 异常处理和信号
- 安装信号处理程序来处理异常
- 中断服务程序基础
- 中断处理的例子
- 中断服务程序设计指导
- 定时器和系统时钟
- 看门狗定时器
- 查询
- 辅助时钟

Timers

- 定时器使用户定义的中断服务程序, 将在定期执行, 用于:
  - 论询(polling)硬件
  - 检查系统错误(例如热插拔)
  - 中止不合时宜的操作
- VxWorks 提供了两个定时通用接口
  - 系统时钟 (System clock)(必选)
  - 辅助时钟 (Auxiliary clock)(可选)

System Clock

- 系统时钟中断服务程序
  - 增加 tick 计数 (使用 ***tickGet( )*** 测试)
  - 更新任务的 delay 时间和信号量的 timeout 时间
  - 检查时间片轮转调度 (round-robin rescheduling)
  - 这些操作都可能引起重新调度
- 缺省时钟频率是 60hz
  - **int *sysClkRateGet( )*** – 返回每秒的 tick 数
  - ***void sysClkRateSet (freq)*** – 设置主时钟频率 (每秒的ticks)
    - 如果超出范围返回 -1
    - 应该只在系统初始化阶段调用一次

Agenda

Watchdog Timers

- Exception Handling and Signals
- Installing Signals to Handle Exceptions
- Interrupt Service Routine Basics
- Interrupt Handling Example
- ISR Guidelines
- Timing and the System Clock
- **Watchdog Timers**
- Polling
- The Auxiliary Clock

Watchdog Timers

- Kernel space only
  - Use POSIX timers in user space (not covered here)
- User interface to system clock
- Allows a C routine to execute after a specified time delay
- Upon expiration of delay, connected routine runs
  - As part of system clock ISR
  - Subject to ISR restrictions
    - No *printf(), malloc(),* and so on

Creating Watchdog Timers

- To create a watchdog timer

  WDOG_ID wdCreate ()

  Returns watchdog id, or *NULL* on error

- To start (or restart) a watchdog timer

  STATUS wdStart (wdId, delay, pRoutine, parameter)

  wdId Watchdog id, returned from wdCreate()

  delay Number of ticks to delay

  pRoutine Routine to call when delay has expired

  parameter Argument to pass to routine (int)

Using Watchdogs

- For periodic code execution

```
wdId = wdCreate();
wdStart (wdId, DELAY_PERIOD, myWdIsr, 0);

void myWdIsr(int param)
        {
        doit (param);
        wdStart (wdId, DELAY_PERIOD,myWdIsr, param);
        }
```

- *doit( )* routine might
  - Poll some hardware device
  - Unblock some task
  - Check if system errors are present

Detecting Missed Deadlines

- To detect a missed deadline

```
WDOG_ID wdId;

void foo(void)
        {
        wdId = wdCreate( );
        /* Must finish each cycle in under 10 seconds */
        FOREVER
                {
                wdStart (wdId, DELAY_10_SEC, fooISR, 0);
                fooDoWork( );
                }
        }

void fooISR (int param)
        {
        /* Handle missed deadline */
        ...
        }
```

Stopping Watchdogs

- To cancel a previously started watchdog

  STATUS wdCancel (wdId)

- To free watchdog timer resources (and cancel previous start)

  STATUS wdDelete (wdId)

Agenda

Polling

- Exception Handling and Signals
- Installing Signals to Handle Exceptions
- Interrupt Service Routine Basics
- Interrupt Handling Example
- ISR Guidelines
- Timing and the System Clock
- Watchdog Timers
- **Polling**
- The Auxiliary Clock

Polling Issues

- Could poll at task time or interrupt time
  - Interrupt time polling is more reliable, but
  - Task time polling has a smaller impact on the rest of the system
- Polling at interrupt time typically done using watchdogs
- To poll at task time, there are two options
  - ***taskDelay( )*** – efficient, but imprecise since subject to “drift”
  - ***wdStart( )*** **+** ***semGive( )*** – much more robust since synchronized with system clock

Polling Caveats

- Code accurate only if system clock rate is multiple of 15hz

```
void myWdISR ( )
        {
        wdStart (myWdId, sysClkRateGet()/15, myWdISR, 0);
        pollMyDevice();
        }
```

- Do not set the system clock rate too high
  - Overhead in each clock tick
- Use auxiliary clock to poll
  - At high speeds, or
  - Irregular intervals

Agenda

The Auxiliary Clock

- Exception Handling and Signals
- Installing Signals to Handle Exceptions
- Interrupt Service Routine Basics
- Interrupt Handling Example
- ISR Guidelines
- Timing and the System Clock
- Watchdog Timers
- Polling
- **The Auxiliary Clock**

Auxiliary Clock

- For high speed polling
- Precludes using shell routine ***spy()***, which also uses auxiliary clock
- Some routines to manipulate auxiliary clock

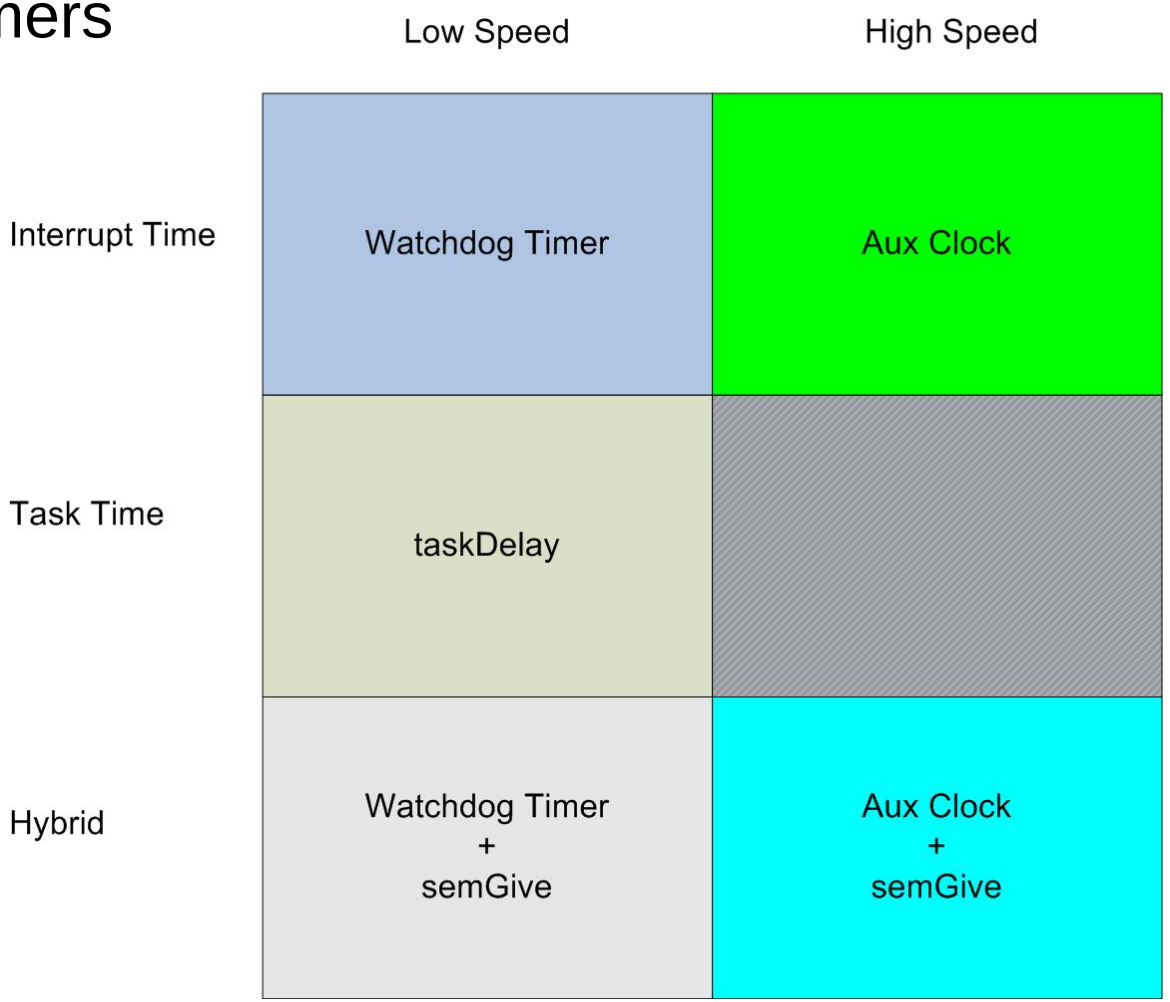
sysAuxClkConnect( )	Connect ISR to aux clock
sysAuxClkRateSet( )	Set aux clock rate
sysAuxClkEnable( )	Start aux clock
sysAuxClkDisable( )	Stop aux clock

Summary

- Using signals for exception handling
  - ***signal( )***
  - ***exit( )***
  - ***taskRestart( )***
  - ***longjmp( )***
- Interrupt Service Routines have a limited context
  - **No** Blocking
  - **No** I/O system calls

Summary (Continued)

Polling with timers



Questions

1. What is an exception?
2. How do VxWorks exception handlers communicate with user tasks?
3. If an exception signal handler returns, the offending task will be stopped. (true/false)
4. What are a common source of interrupts?
5. Interrupts cannot preempt the highest priority task. (true/false)
6. An ISR can call semTake( ) if it needs to do something fast. (true/false)
7. What will an exception occurring during interrupt time generate?
8. A watchdog timer can run in the context of any task, as long as interrupts are serviced. (true/false)

Review

In this chapter you learned to:

- Install signals to handle exceptions
- Register a signal handler
- Handle interrupts
- Debug ISRs
- Use watchdog timers to execute user-defined routines at periodic intervals or after specified time delays
- Change the system clock rate
- Poll at both task, and interrupt, time
- Use the auxiliary clock for high speed polling