

Sistemas Encaixados

Tema 3: Sistemas Operativos

Oscar García Lorenzo

Escola Politécnica Superior de Enxeñaría

Sistemas Operativos

1 Introducción e Conceptos Básicos

- Conceptos básicos dos SO
- Estructura do Sistema Operativo

2 Interrupcóns

3 Procesos, Fíos e Tarefas

- Procesos, Fíos e Tarefas
- Planificación
- Teoría de Programación en Tempo Real

4 Comunicación entre Procesos/Tarefas

Índice

1 Introducción e Conceptos Básicos

- Conceptos básicos dos SO
- Estructura do Sistema Operativo

2 Interrupcóns

3 Procesos, Fíos e Tarefas

- Procesos, Fíos e Tarefas
- Planificación
- Teoría de Programación en Tempo Real

4 Comunicación entre Procesos/Tarefas

Introducción

- Tan só cun programa principal un sistema encaixado ou computacional non pode funcionar
- Necesita polo menos outro programa diferente que se encargue de iniciar o sistema:
 - Inicializar a memoria, as variables
 - Cargar o programa principal
 - Iniciar a execución do programa principal
- Ademáis hai que responder á comunicación con periféricos ou certas accións do procesador dunha forma determinada
 - Necesitase código extra
- Canto máis complexo sexa o sistema computacional máis complexo será ese código
- A fronteira onde ese código se converte nun Sistema Operativo (SO) é difusa

Firmware

Firmware - Sistemas Encaixados

É unha clase específica de software que ofrece control e funcionalidades de baixo nivel para un hardware determinado

- Defínese máis por onde está instalado que polo que fai
 - Pode está instalado en memoria ROM (non se pode modificar), EPROM, EEPROM ou Flash (pódese modificar)
 - Nos sistemas encaixados pode chegar a ser todo o SO
 - Nos periféricos ou SE usase para facilitar a interconexión e ofrecer interfaces
 - Pode chegar a conter todo o software necesario para o funcionamento do SE

Firmware

Firmware - BIOS

É unha clase específica de software que ofrece control e funcionalidades de baixo nivel para un hardware determinado

- Nas computadoras complexas pode ser a BIOS (Basic Input/Output System)
 - Era a interface dos IBM, converteuse nun estándar de facto
 - Comproba que o hardware funcione
 - Ofrece certa abstracción do sistema (teclado, almacenamento de datos ...)
 - Así os programas non se teñen que preocupar das pequenas variacións do hardware
 - Primeiro programa en executarse ao arrincar
 - Sobre ela cárgase o SO
 - Nunha computadora pode haber moitos más firmwares
 - Os propios dos discos duros, tarxetas gráficas, etc.

Bootloader

Bootloader

É unha clase específica de software que inicia (*boots*) unha computadora ou SE

- O nome ven de *bootstrap loader*: referencia á frase "levantase tirando dos propios cordóns"
 - O bootloader está gardado en memoria ROM, EPROM, EEPROM ou Flash
 - Esta memorias soen ser más lentas que a RAM
 - Os programas ou SO están gardados en memoria non volátil (disco duro, por exemplo), hai que copialos á memoria RAM (de instruccións), a non ser que se poidan executar directamente desde esa memoria non volátil (ROM, EPROM)

Bootloader

Bootloader

É unha clase específica de software que inicia (*boots*) unha computadora ou SE

- A memoria (pila, montón, variables) hai que inicializala, xa que a RAM bórrase ao apagar
- O bootloader máis sinxelo carga a memoria, variables e programa na RAM e despois salta á instrucción indicada por MAIN

Bootloader

Bootloader - SE

- Un bootloader pode non ser necesario en SE
- O programa principal pódese escribir directamente en memoria non volátil, xunto co código que inicializa a memoria
 - JTAG (Joint Test Action Group), norma IEEE 1149.1, Standard Test Access Port and Boundary-Scan Architecture
 - Permite escribir directamente á memoria dun SE
 - Requiere hardware específico
- Un bootloader simplifica a programación
 - Pódese usar a conexión USB
 - O bootloader trae código para que funcione e para copiar datos á memoria
 - Pódense escribir programas más estándar (dif. hardware)
 - Facilita actualizacóns
 - En Arduino hai un bootloader chamado **Optiboot**

Bootloader

Bootloader - Computadoras

- En computadoras é necesario para cargar o SO
- De feito úsanse varios en cadea (*multiboot*)
- Primeira etapa:
 - En tempos antigos estaban limitados en tamaño, podían facer poucas cousas
 - Por se estandar quedaron así
 - Exemplo: A BIOS, carga un bootloader de segunda etapa
- Segunda etapa:
 - Máis complexos, capaces de cargar o SO
 - Exemplo: GRUB (o que aparece nas aulas de informática para escollecer Linux ou Windows)
 - Se se escolle Windows carga outro bootloader, o Windows Boot Manager/Loader (BOOTMGR)

Librarias

Libraria

Recursos software para reutilizar e compartir entre diferentes programas

- Implementa comportamentos coñecidos aos que se accede por unha interface bien definida
- O nome ven de *Library*, que realmente tradúcese por biblioteca
- Enlazado (*Linking*)
 - Ao compilar códigos distintos este se referenciañ entre eles por **símbolos**
 - Para poder usar códigos referenciados por **símbolos** hai que enlazarlos

Librarias

Librarias Estáticas/Compartidas

- **Librarias Estáticas**

- O enlazado realizase na compilación ou creación do executable (enlazado estático)
- O código necesario da libraría cópiase directamente no executable
 - Maior tamaño de código
 - Varios programas cunha soa libraría teñen a súa copia

- **Librarias Compartidas**

- O enlazado realizase durante a execución (enlazado dinámico)
- O código necesario da libraría vai a parte
 - Menor tamaño de código
 - Varios programas cunha soa libraría usan o mesmo código

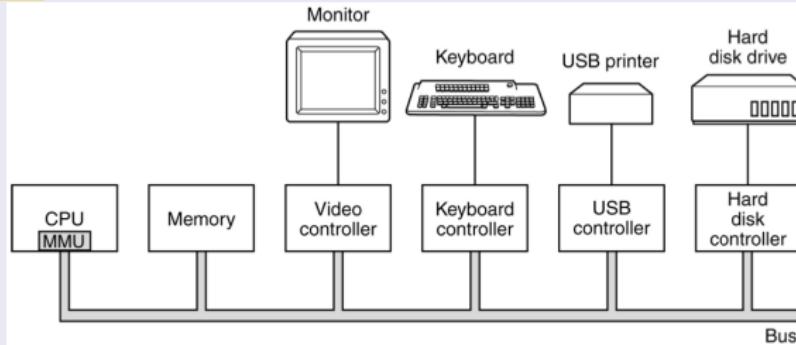
- Os SE soen ter estáticas, o seu código cópiase xunto co programa, as compartidas requieren dun SO

Sistema Operativo

Sistema Operativo

Código do sistema que manexa o hardware, os recursos software e ofrece servizos comúns aos programas

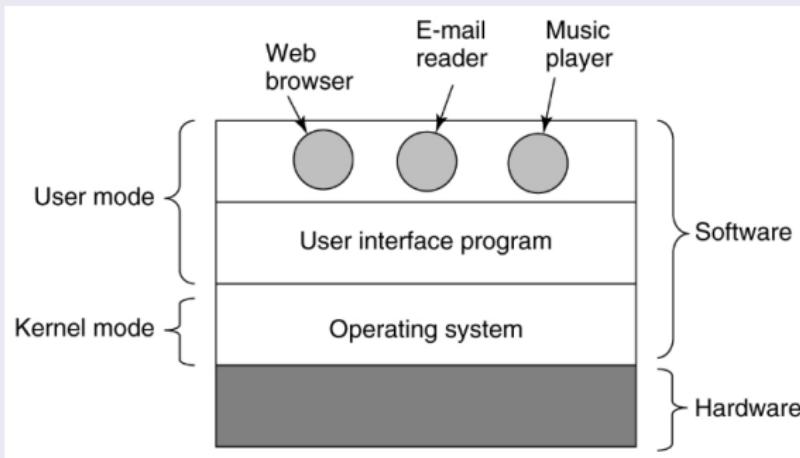
- Canto más complexo o sistema más difícil a administración e utilización óptima
- Un SO administra os recursos e ofrece un modelo más sinxelo



Sistema Operativo

Sistema Operativo

- O usuario interactúa co shell ou GUI
- Permite a execución doutros programas



Sistema Operativo

O SO realiza dúas funcións básicas

- Proporciona aos programadores/programas un conxunto abstracto de recursos
 - Os programas de aplicacións interactúan co SO
 - Os usuarios finais interaccionan coa interface de usuario (shell ou GUI)
- Administra os recursos hardware
- Asignación ordenada de recursos a programas
- Resolver conflictos entre programas ou usuarios
- Compartición de recursos: tempo (p.e. cpu) e espazo (p.e. memoria)

Sistema Operativo

Modos de execución: núcleo e usuario

- Modo núcleo: Acceso a todo o hardware e recursos
- Modo usuario: Non se permite E/S e xestión de memoria
- Chamada ao sistema: o programa usuario accede aos recursos do sistema (TRAP)
 - Instrucción SYSCALL en x86
 - Instrucción SWI en Arm EABI
 - Tamén as pode xerar o software ou hardware

Sistema Operativo

Bibliografía

Andrew S. Tanenbaum

Sistemas operativos modernos

(3a edición)

Editorial Prentice-Hall, 2009

Sistema Operativo*

Hai diferentes tipos de SO:

- Mainframe: procesamento de moitos traballos e E/S, servidores web de alto rendemento. UNIX
- Servidores: servizos a varios usuarios e compartición de recursos hardware e software. Linux, Windows, Solaris
- Multiprocesadores: varias CPUs. Windows, Linux
- Computadores personais: bo soporte a un so usuario. Windows, Linux, Mac OS
- PDA: telefonía, fotografía digital, etc. Windows mobile, Android, iOS
- Encaixados: aplicacións pechadas. QNX, VxWorks
- Tempo real: tempo como parámetro chave, sistemas de control. FreeRTOS

Sistema Operativo

- Esta materia non trata sobre SO
- É un campo moi, moi amplio
- Nos veremos as características más importantes para os SE
- E/S: próximo tema, salvo interrupcóns
- Multiprogramación:
 - Distintos periféricos poden controlarse con diferentes programas
 - Os diferentes programas deben executarse simultáneamente (ou parecelo)
 - Moi difícil de facer sen SO

Sistema Operativo

SO de servidores, multiprocesadores, PC

Son os coñecidos Linux, Windows, MacOs ..., veremos Linux

- Ofrecen multiprogramación: Procesos (*Process*) e Fíos (*Threads*)
 - Reparten o uso do procesador de maneira xusta entre os programas e usuarios
- Espazo de direccións
- Arquivos, Entrada/Saída, Protección, Liña de comandos ...
- POSIX (Portable Operating System Interface, X ven de UNIX)

Sistema Operativo

SO en Tempo real

Veremos o FreeRTOS

- Ofrece multiprogramación: Tarefas (*Task*)
 - Reparten o uso do procesador de maneira que se cumplen condicións estrictas de tempo
 - As tarefas teñen prioridade
- Espazo de direccións, máis sinxelo que Linux, sen memoria virtual
- Entrada/Saída, Protección?
- Certo soporte para POSIX
- Existe unha versión que funciona coma libraría Arduino

Multiprogramación

Proceso (Linux)

Programa en execución

- Toda a información necesaria para a súa execución
- Espazo de direccións: programa, datos e pila
- Lista de direccións desde 0 a un valor máximo
- Registros (PC, SP ...)
- Lista de arquivos, alarmas ...

Execución de procesos simultáneamente (Multiprogramación)

- Suspensión/reanudación de procesos: Gardar toda a información sobre o proceso
- Táboa de procesos

Multiprogramación

Proceso (Linux)

- Creación/terminación de procesos
 - Procesos fillos
 - Cooperación e sincronización
 - Comunicación entre procesos
 - Sinais
- Identificación de usuario (UID) e grupo (GID)
 - Superusuario: poder especial, pode saltarse as regras de protección

Multiprogramación

Fío/Thread (Linux)

Máis sinxelo que un proceso, pero tamén se executa

- Dependen dun proceso
- Espazo de direccións compartido
- Rexistros e outra información propia
- Esenciais en multinúcleo

Multiprogramación

Espazo de direccións (Linux)

Programas executándose en memoria principal

- Varios programas
- Evitar interferencias

Memoria Virtual

- Espazo de direccións en memoria principal e disco
- Desacopla o espazo de direccións do proceso da memoria principal
 - Os procesos pensan que teñen a memoria enteira
 - O SO traduce entre direccións virtuais e direccións físicas
- Administrada polo sistema operativo

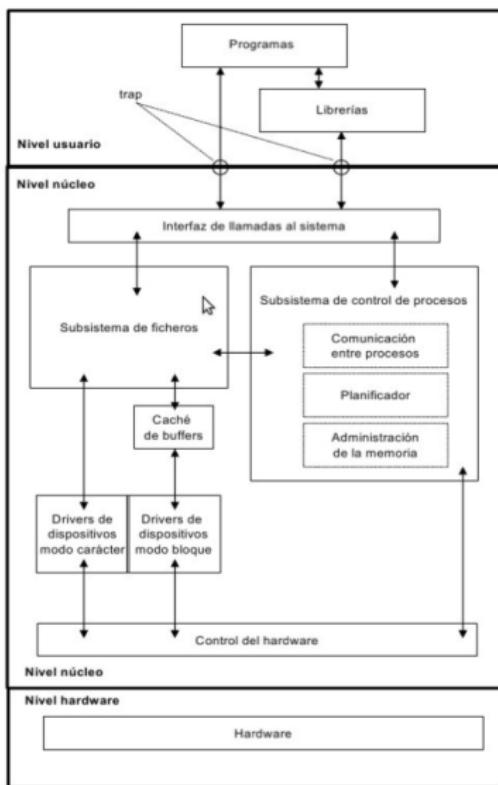
Multiprogramación

Tarefas (FreeRTOS)

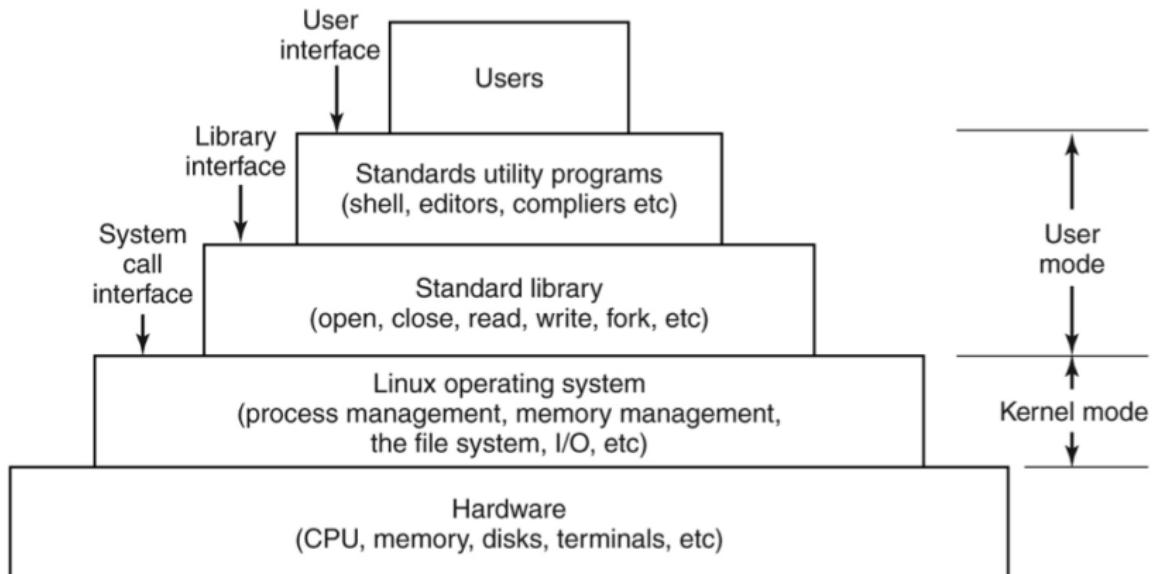
Similar a Procesos POSIX

- Na suspensión/reanudación gardan a súa información na súa propia pila
 - Cada tarefa ten unha pila propia onde garda os seus datos
- Prioridade: importante para o funcionamento en tempo real
- Implementanse coma se fosen funcións en C
- Pensados para correr indefinidamente (terminación explícita)
- Non poden devolver valores

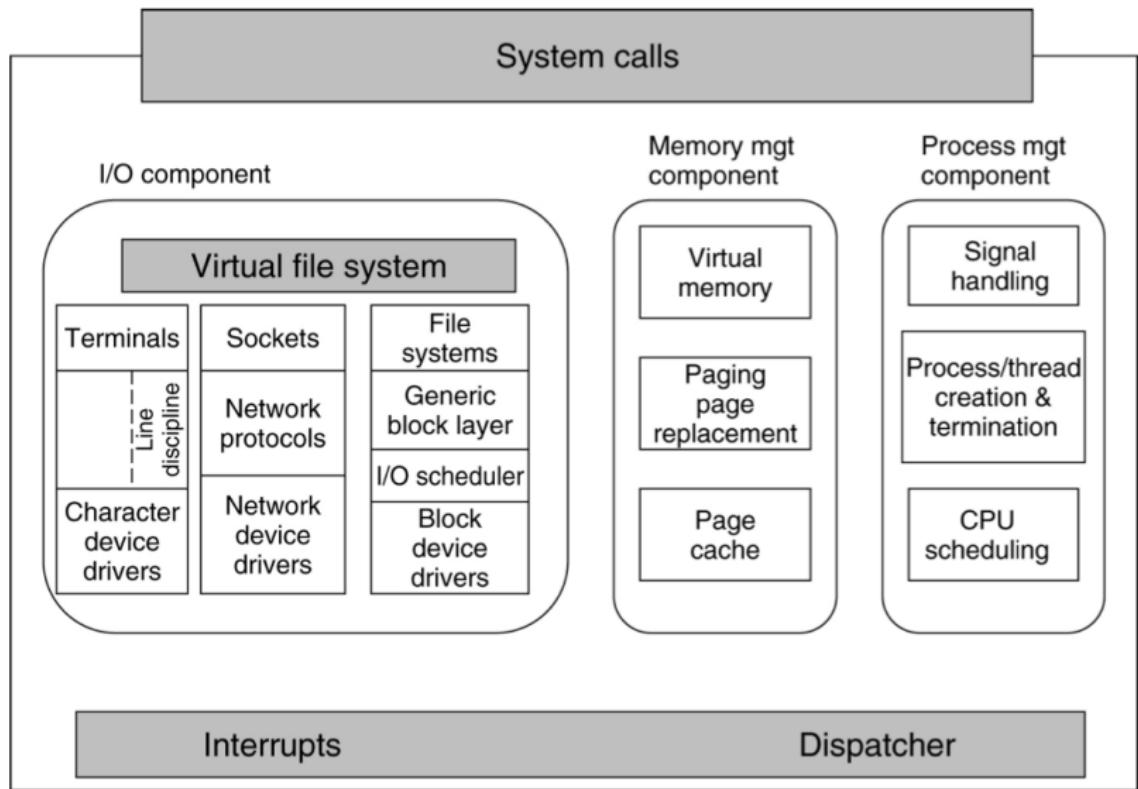
Estructura de Unix*



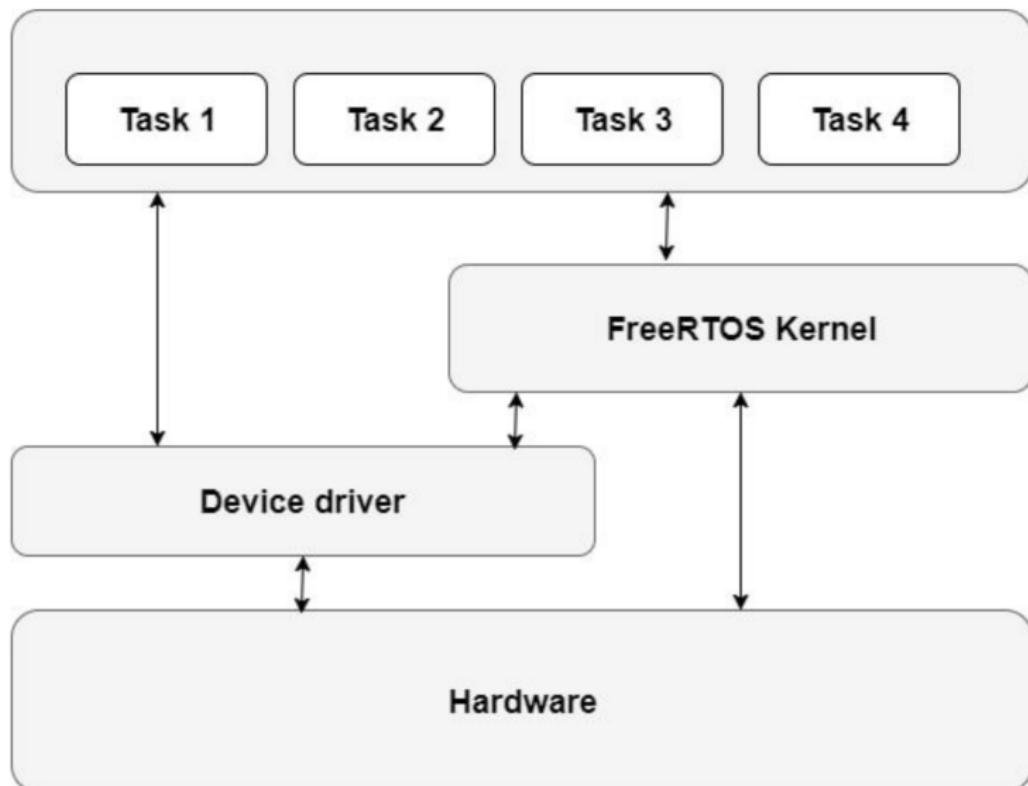
Niveis de Linux*



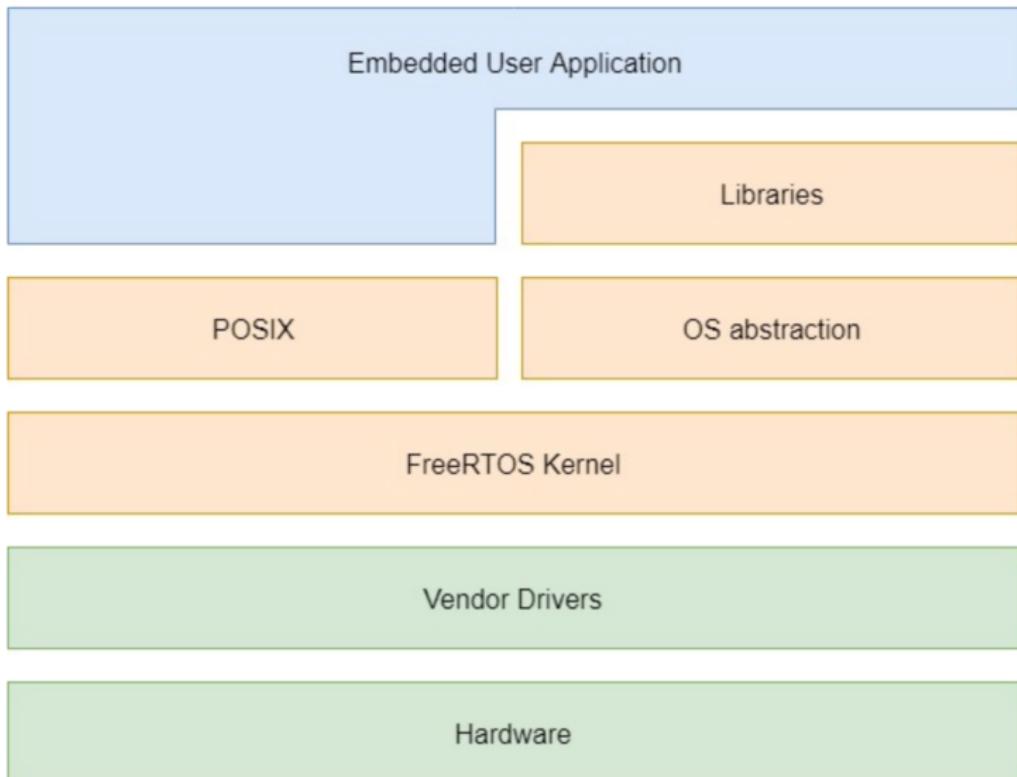
Núcleo de Linux*



Arquitectura de FreeRTOS*



Arquitectura de FreeRTOS + POSIX*



Índice

1 Introducción e Conceptos Básicos

- Conceptos básicos dos SO
- Estructura do Sistema Operativo

2 Interrupciones

3 Procesos, Fíos e Tarefas

- Procesos, Fíos e Tarefas
- Planificación
- Teoría de Programación en Tempo Real

4 Comunicación entre Procesos/Tarefas

Interrupciones

Interrupciones

- Forzan ao procesador a pausar a súa actividade
- Executan un código chamado *rutina de servizo de interrupción* (interrupt service routine (ISR))
- Requeren unha combinación de hardware e software
- A ISR responde a un evento hardware ou software
 - A un *Interrupt Request* IRQ
- Cando a ISR acaba continúase a execución que se pausara
- Úsanse para responder a eventos hardware internos ou externos:
 - Púlsase un botón nun periférico
 - Recíbese unha mensaxe nun porto
 - Acceso a memoria restrinxida
 - Instrucción inválida

Interrupciones

Interrupciones: Usos

- Permiten ao procesador facer multitasking (varias tarefas á vez)
 - Preventiva (preferente): Unha nova tarefa máis urgente toma o control e para a tarefa antiga, esta continúa tras acabar a nova
 - Non-preventiva (sen preferencia): Unha nova tarefa non pode tomar o control do procesador ata que acabe a actual
 - Aquí pódese usar o reloxo do sistema para ir dando tempo equitativamente a cada tarefa
- Permiten ao procesador responder a interacciones humanas ou eventos urgentes
 - Unha alternativa e o "polling": preguntar periodicamente que está a pasar (comprobar todos os perifericos)
 - Para usar interrupciones requírese soporte hardware

Interrupciones

Números de Interrupción (Arm Cortex-M)

- Para identificar as interrupción úsase un número do -15 ao 240
- Definidos polos fabricantes, non se poden cambiar
- Interrupciones de sistema:
 - Os primeiros 15, negativos
 - Definidos por Arm
 - CMSIS (Cortex Microcontroller Software Interface Standard)
- Interrupciones de periféricos:
 - Os outros 240, empezando en 0
 - Definidos polo fabricante da placa
- Non se teñen que usar todos os números
- Cando se procesa o número gárdase no program status register (PSR) (Non se usa complemento 2, súmase 15)

Número de interrupción para STM32L4

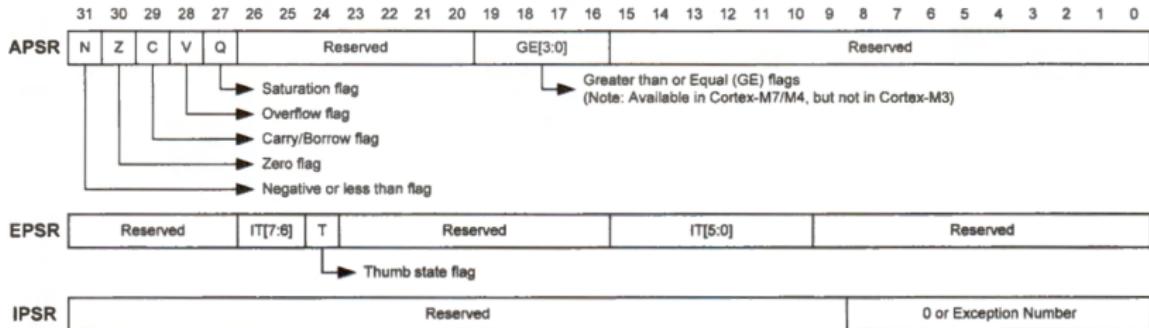
Cortex-M4 Processor
Exceptions Numbers

-14	Non-maskable interrupt
-13	Hard fault
-12	Memory management
-11	Bus fault
-10	Usage fault
-5	Supervisor call (SVCall)
-4	Debug monitor
-2	PendSV
-1	SysTick

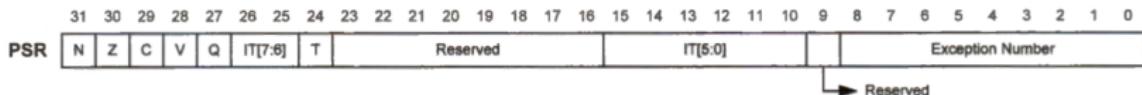
STM32L4 specific
Interrupt Numbers

0	VWDG	16	DMA1_CH6	32	I2C1_ER	48	FMC	64	COMP	80	RNG
1	PVD	17	DMA1_CH7	33	I2C2_EV	49	SDMMC1	65	LPTIM1	81	FPU
2	TAMPER_STAMP	18	ADC1_ADC2	34	I2C2_ER	50	TIM5	66	LPTIM2		
3	RTC_WKUP	19	CAN1_TX	35	SPI1	51	SPI3	67	OTG_FS		
4	FLASH	20	CAN1_RX0	36	SPI2	52	UART4	68	DMA2_Channel6		
5	RCC	21	CAN1_RX1	37	USART1	53	UART5	69	DMA2_Channel7		
6	EXTI0	22	CAN1_SCE	38	USART2	54	TIM6_DAC	70	LPUART1		
7	EXTI1	23	EXTI9_5	39	USART3	55	TIM7	71	QUADSPI		
8	EXTI2	24	TIM1_BRK	40	EXTI15_10	56	DMA2_Channel1	72	I2C3_EV		
9	EXTI3	25	TIM1_UP	41	RTC_Alarm	57	DMA2_Channel2	73	I2C3_ER		
10	EXTI4	26	TIM1_TRG	42	DFSDM3	58	DMA2_Channel3	74	SAI1		
11	DMA1_CH1	27	TIM1_CC	43	TIM8_BRK	59	DMA2_Channel4	75	SAI2		
12	DMA1_CH2	28	TIM2	44	TIM8_UP	60	DMA2_Channel5	76	SWPMI		
13	DMA1_CH3	29	TIM3	45	TIM8_TRG	61	DFSDM0	77	TSC		
14	DMA1_CH4	30	TIM4	46	TIM8_CC	62	DFSDM1	78	LCD		
15	DMA1_CH5	31	I2C1_EV	47	ADC3	63	DFSDM2	79			

Número de interrupción para STM32L4



$$\text{PSR} = \text{APSР} + \text{EPSР} + \text{IPSR}$$



Rutinas de interrupción

ISR/Interrupt Handler

- As ISR pódense chamar tamén **interrupt handler**
- É unha rutina que se chama cada vez que haxa unha interrupción
 - Chámaa o hardware
- Soen ter unha versión por defecto no código de inicio (startup code) do sistema
 - Pode ser un bucle infinito
- Se se define unha rutina nova co mesmo nome úsarase esa
- Non devolven ningún valor porque as chama o hardware
- Non requieren parámetros (salvo SVC_HANDLER)

Rutinas de interrupción

Reset Handler

- Unha ISR que carga o programa principal cando se encende ou resetea o procesador
- Inicializa a memoria, as variables
- Salta a MAIN

Vector de interrupciones

Vector de interrupciones

- Hai una ISR para cada interrupción
- Gárdase nunha táboa a dirección de inicio de cada ISR
- O índice da taboa é o número de interrupción + 15
- Gárdase na memoria a partir de 0x00000004
- Cando se inicia o procesador:
 - O valor na dirección 0x00000000 úsase para inicializar SP
 - O valor na dirección 0x00000004 úsase para inicializar PC:
 - chamar ao RESET_HANDLER (interrupción -15)

Vector de interrupciones

Vector de interrupciones

- Os Cortex-M teñen un *nested vectored interrupt controller* (NVIC) que permite definir prioridades para as interrupciones:
 - Cando hai unha interrupción comproba a súa prioridade
 - Se está executando un handler de máis prioridade ten que esperar
 - O RESET_HANDLER ten prioridade -3, que é a máis alta
 - Un número maior indica menor prioridade
- As primeiras 15 interrupciones implican un comportamento extraño do sistema, son as excepcións de sistema (system exception)
- O resto implica a periféricos, son as interrupciones externas (external interrupts)
- A táboa de vector de interrupción pódese mover para arrincar desde distintas zonas de memoria

Chamada a ISR

Chamada a ISR: Preservación do entorno

- Cando se chama a unha ISR esta ten que preservar o entorno
- O programa que se executa non pode saber que se chama a unha subrutina
- A ISR garda os rexistros R1-R3, R12, LR, PC, e PSR na pila (interrupt stacking)
- Ela mesma os recupera ao terminar (interrupt unstacking)
- Tamén limpa os bits do NVIC
- Como non se sabe cando ocorrerá unha interrupción hai que gardar PC
 - A instrucción BX LR non funciona igual que nunha subrutina, en LR está unha dirección que chama a outra subrutina para que realice o unstacking da pila adecuada (main, MSP, ou process, PSP)

Prioridade interrupciones



lite.augmented

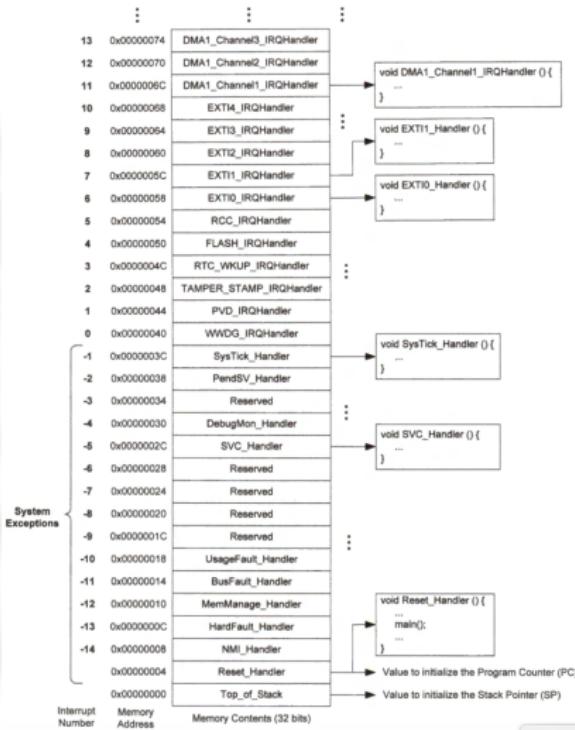
1

Interrupción encadeada (tail-chained)

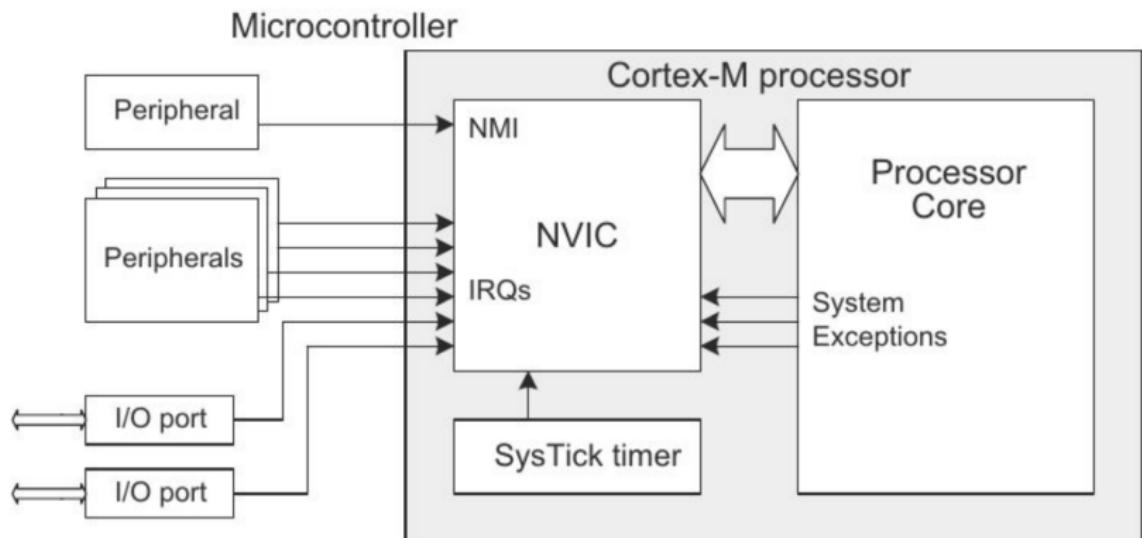


Imaxes desde STM training resources

Número de interrupción para STM32L4*



NVIC: nested vectored interrupt controller



NVIC

Nested Vectored Interrupt Controller (NVIC)

Permite configurar as interrupciones

- Habilitar e deshabilitar interrupciones individualmente
- Definir a súa prioridade individualmente
 - Depende do fabricante, 4 ou 8 bits, non entraremos en detalles
- Marcar ou desmarcar o bit de manexo dunha interrupción
- Cada interrupción ten 6 bits de control:
 - Enable, Disable, Pending, Un-pending, Active, Software trigger
 - Gárdanse en rexistros ISER, ICER, ISPR, ICPR, IABR, STIR
- Os Cortex-M teñen 8 rexistros de cada, (ISER0-ISER7), así $32 \times 8 = 255$ interrupciones
 - Mapeados a memoria para poder acceder a eles

NVIC

- Para habilitar unha interrupción gárdase 1 no seu *Enable Bit*, escribir un 0 non implica nada
- Para deshabilitar unha interrupción gárdase 1 no seu *Disable Bit*, escribir un 0 non implica nada
- Se ocorre unha interrupción, escribese un 1 no *Pending Bit* se non se pode procesar agora a interrupción, e se procesará despois
- Escribir un 1 no *Un-pending Bit* quita a interrupción da lista de pendentes, xa non se procesará
- Se ocorre unha interrupción, escribese un 1 no *Active Bit* se si se pode procesar agora a interrupción, e se pon a 1 ao chamar ao handler
- Escribir un 1 no *Software Trigger Bit* implica procesar agora esa interrupción, a maioría de interrupcóns non permiten facer isto

Deshabilitar interrupción globalmente

- Pódense habilitar e deshabilitar interrupcóns globalmente
- Úsanse máscaras nos rexistros especiais PRIMASK, FAULTMASK e BASEPRI
- PRIMASK = 1 deshabilita as interrupcóns configurables (non as execpcóns de sistema, nin as non enmascarables, NMI, definidas polo fabricante)
- FAULTMASK = 1 deshabilita as interrupcóns configurables (non as non enmascarables, NMI, definidas polo fabricante)
- BASEPRI maior de 0 deshabilita as interrupcóns cun valor de prioridade más alta (menor prioridade), dise que as enmascara ("masked", o resto están "unmasked")

Interrupciones externas EXTI

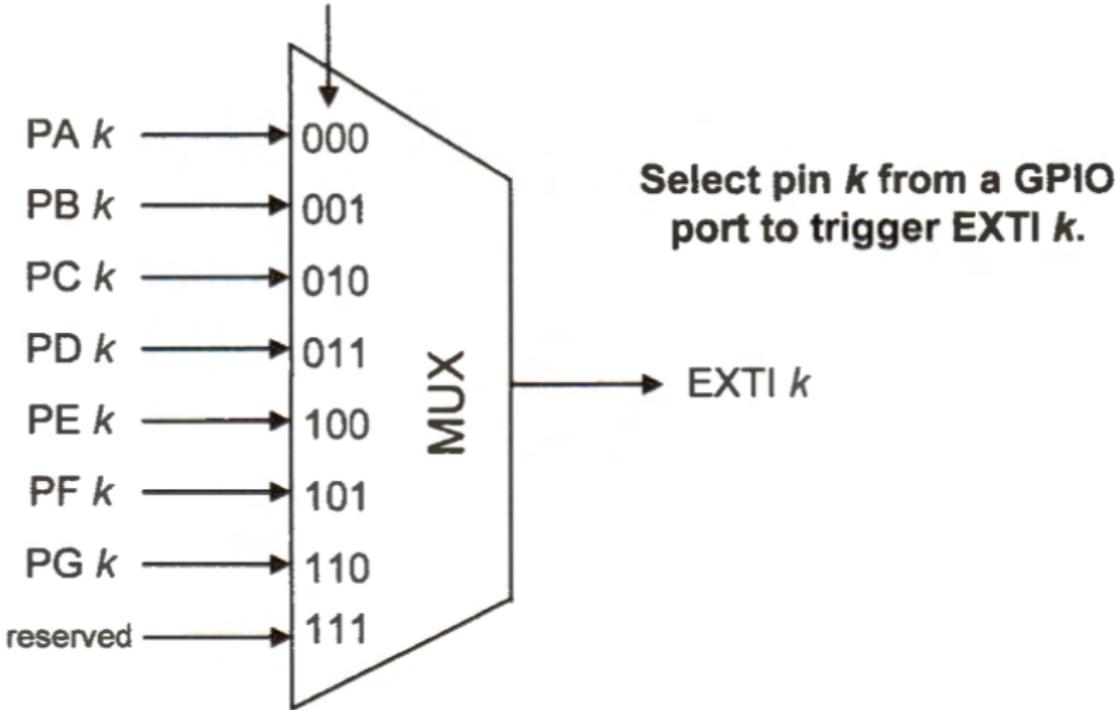
- Vimos que hai 16 interrupciones marcadas como EXTI (External Interrupt)
- Hai un controlador de interrupciones externas que comproba a voltaxe nos pins e lanza a interrupción no NVIC
- Hai máis pins que números de EXTI
 - Os pins do mesmo número conectanse á mesma EXTI
 - P.e: se temos unha interrupción programada no pin 3 do porto GPIO_A non podemos programar unha no porto 3 da GPIO_B, xa que ambos usan EXTI3
- Para usar estas interrupciones hai que programar tamén o controlador de interrupciones externas:
 - Configurar o pins, seleccionar o borde que se detectará (subida, baixada, ambos)
 - Activar (*enable*) a interrupción tanto no controlador externo como non NVIC
 - Escribir o ISR EXTIK_IRQHandler() correspondente

Interrupcions externas EXTI

- As interrupcions de EXTI0 a EXTI4 teñen o seu propio número, pódense programar todas á vez
- As interrupcions EXTI9-5 e EXTI15-10 comparten dous números
 - Pódense programar varios pins GPIO para que as lancen (p.e. 7 e 5)
 - Despois facer polling na ISR para saber onde foi

Interrupciones externas EXTI

EXTI_k bits in
SYSCFG_EXTICR1,2,3,4 registers



Índice

1 Introducción e Conceptos Básicos

- Conceptos básicos dos SO
- Estructura do Sistema Operativo

2 Interrupcóns

3 Procesos, Fíos e Tarefas

- Procesos, Fíos e Tarefas
- Planificación
- Teoría de Programación en Tempo Real

4 Comunicación entre Procesos/Tarefas

Procesos (POSIX/Linux)

Que é un proceso?

- Proceso: O concepto máis importante dos SO
 - Abstracción dun programa en execución
- Execución concurrente aén cunha soa CPU
 - En cualquier computador estanse a executar moitos procesos simultáneamente
 - Varias CPUs virtuais
 - A CPU commuta dun proceso a outro
 - Apariencia de pseudoparalelismo

Procesos

O modelo de proceso

- Software en execución = procesos secuenciais
 - Código, datos, pila, rexistros, . . .
 - **Multiprogramación**: conmutación entre procesos
 - Proceso hardware
 - Varios PC (contador de programa) lóxicos, so un PC físico
 - Planificación de procesos: selección do proceso que debe executarse

Procesos

Cal é a diferenza entre un proceso e un programa?

- **Programa:** secuencia de instruccíons e estruturas de datos necesarias para a execución. Almacenase nun ficheiro executable (código máquina)
- **Proceso:** programa en execución. É un programa cargado que inclúa:
 - código,
 - datos,
 - pilas,
 - espazo de direccións
 - sinais,
 - ...

Xestión de Procesos

Creación de procesos

- Situacóns:
 - Arranque do sistema (procesos de 1o e 2o plano)
 - Desde outro proceso (chamadas ao sistema)
 - Solicitud dun usuario (comandos) (caso particular do anterior)
- Tipos de procesos:
 - De usuario
 - Do núcleo ou kernel
 - Demonios

Xestión de Procesos

Chamada ao sistema *fork*

- Crea un proceso fillo idéntico ao proceso pai (memoria, variables, arquivos, ...)
- Saída
 - Pai: O identificador de proceso (**pid**) do proceso fillo
 - Fillo: 0
- Depois, cun simple `if()` pódese executar un código diferente
- Cambiar a imaxe do proceso: `exec()`
 - Carga as rexións de código pila e datos do novo programa

Uso de fork()

```
main() {  
    int par, x=0;  
    if ((par=fork())== -1) {  
        printf("erro na execución do fork");  
        exit(0); }  
    else if par==0 { /* proceso filho  
        x=x+2;  
        printf("\nproceso filho, x= %d\n", x);  
    } else /* proceso pai  
        printf("\nproceso proceso pai, x= %d\n", x);  
    printf("Finalizar\n"); }
```

Terminación de Procesos

Como de termina un proceso?

- Saída normal (voluntaria)
 - Termina o seu traballo, exit (enteiro)
- Saída por erro (voluntaria)
 - Descubrese un erro, exit (enteiro)
- Erro fatal (involuntaria)
 - Erro do programa (división por cero, acceso non permitido a memoria, ...)
- Eliminado por outro proceso (involuntaria)
 - Chamada ao sistema kill

Terminación de Procesos

Chamada ao sistema *exit*

- *exit* (condición)
- condición: enteiro que se devolve ao proceso pai
- O proceso pai pode examinar condición
- Non produce saída
- O proceso desconectase da árbore de procesos
- *exit* implícitos: ao finalizar *main*

Esperar á terminación dun proceso

Sincronización coa terminación dun proceso fillo

```
resultado = wait (pid)
```

- Devolve o pid do fillo ou -1 se erro (ou non ten fillos)
- Variable de retorno en ret
- Alternativa: resultado=waitpid(pid, &ret, opt);
- Permite especificar o pid do fillo (-1 se calquera) polo que está esperando

Uso de wait()

```
main() {  
    int pid, estado;  
    if (fork() == 0) {  
        printf ("\nMensaje 1\n");  
        sleep(4);  
        exit(3);  
    } else {  
        pid=wait(&estado);  
        printf ("\nFinalizar");  
    }  
}
```

Xerarquía de procesos

Xerarquía de procesos

- Un proceso, os seus fillos e os seus descendentes forman un grupo
 - Enviar sinais ao grupo
 - Tratalas individualmente
- Proceso inicial: proceso 1 ou INIT
 - Crea un `fork()` para cada terminal, esperan a que alguén inicie sesión
 - Ao iniciar sesión iniciase SHELL e a partires desa todos os demás (comandos, programas de usuario, aplicacións)
- En Windows non hai xerarquía:
 - O pai recibe un TOKEN que usa para xestionar o fillo
 - Pode pasar o TOKEN a outros procesos (invalida xerarquía)

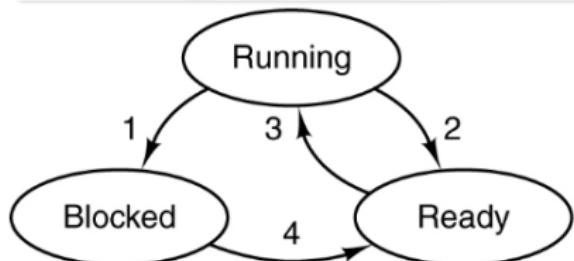
Estados dun Proceso

Un proceso pode estar en moitos estados, os 3 más importantes:

- *Correndo* (Running)
 - O Proceso está a usar o procesador
 - Se so hai un procesador/núcleo so pode haber unha tarefa correndo á vez
- *Listo* (Ready)
 - O proceso está preparado para usar o procesador, pero non pode pasar a *Correndo* porque outra proceso está *Correndo*
- *Bloqueado* (Blocked)
 - O Proceso está esperado por un evento externo ou temporal, e non pode pasar a *Correndo*

Estados dun Proceso

- Os Procesos son independentes, pero poden ter interacción
 - *cat capitulo1 capitulo2 capitulo3 | grep arbol*
- *grep* pode bloquearse se está listo para executarse antes de que termine *cat*
- Pode ser detido polo SO (cede a CPU a outro proceso)

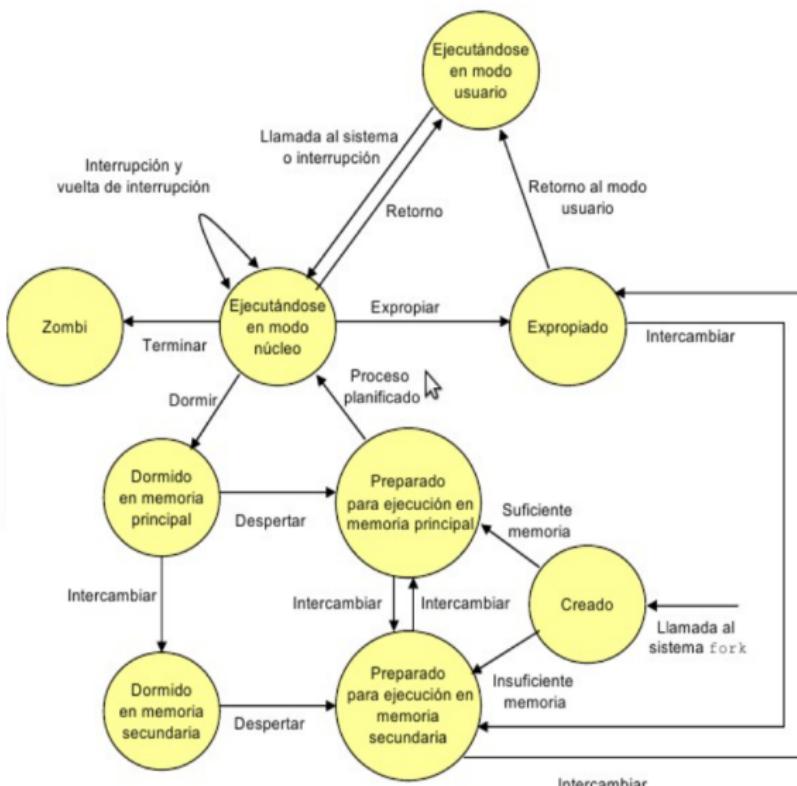


1. Process blocks for input
2. Scheduler picks another process
3. Scheduler picks this process
4. Input becomes available

Estados dun Proceso: Modelo de Procesos

- Non todos os procesos son iguais
 - Programas de usuario
 - Parte do SO: peticóns do sistema de arquivos, acceso a disco, ...
- Exemplo: Nunha interrupción de disco
 - Detense o proceso actual que pasa ao estado *listo*
 - Ejecuta o manexador asociado á interrupción de disco (ISR) (o proceso destinatario estaba *bloqueado* esperando pola interrupción)
 - Cando termina o manexador (leeuse o disco), o proceso destinatario bloqueado desbloquease e pasa a *listo*
 - Planificase o seguinte uso da CPU entre os procesos en estado *listo*

Más estados dun Proceso*



Fíos

Fío (Thread)

- Tradicionalmente cada proceso tiña un so fío (secuencia de instruccóns)
- Hoxe en día poden ter varios
 - No mesmo espazo de direccóns
 - Con execución pseudoparalela (planificables)
 - Son como procesos independentes, salvo que comparten espazo de direccóns (*proceso dentro dun proceso*)
- A comutación entre fíos e de 10 a 100 veces máis rápida que entre procesos

Fíos

Porque se utilizan?

- Aplicacóns que requieren varias actividades simultáneas que colaboran entre si
 - Algunha pódese bloquear, pero as outras poden continuar
- Diferenza cos procesos: os fíos comparten o espazo de direccóns
 - Os fíos comparten os datos
 - Isto non ocorre entre os procesos: moitas aplicacóns non se poden organizar como procesos independentes
- O fío é un proceso lixeiro
 - Máis fáciles (rápidos) de crear, destruir, . . .
- Útiles cando hai cálculo e E/S (ou varias CPUs)
 - Se todos queren acceder á CPU non se mellora o rendemento

Fíos

Os fíos non son tan independentes coma os procesos

- Comparten espazo de direccións, archivos, procesos fillos, alarmas, sinais, ...
- Son propiedade do mesmo usuario
- **Cada fío ten a súa propia pila**

Per process items

Address space
Global variables
Open files
Child processes
Pending alarms
Signals and signal handlers
Accounting information

Per thread items

Program counter
Registers
Stack
State

Fíos

Creación de fíos

- Un fío pode estar en varios estados
 - En ejecución, bloqueado, listo ou terminado
- O proceso comenza cun único fío, pero pode crear máis fíos
 - `pthread_create()` (Linux/POSIX)
 - Especifica o procedemento que executa o novo fío
 - Cada fío ten un identificador (TID)
- Chamadas ao sistema para
 - Terminar (`pthread_exit()`) (Linux/POSIX)
 - Esperar á terminación doutro fío (`pthread_join()`)
 - Entregar a CPU voluntariamente (`pthread_yield()`)
 - ...

Fíos en POSIX

```

#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>

#define NUMBER_OF_THREADS 10

void *print_hello_world(void *tid)
{
    /* This function prints the thread's identifier and then exits. */
    printf("Hello World. Greetings from thread %d0, tid);
    pthread_exit(NULL);
}

int main(int argc, char *argv[])
{
    /* The main program creates 10 threads and then exits. */
    pthread_t threads[NUMBER_OF_THREADS];
    int status, i;

    for(i=0; i < NUMBER_OF_THREADS; i++) {
        printf("Main here. Creating thread %d0, i);
        status = pthread_create(&threads[i], NULL, print_hello_world, (void *)i);

        if (status != 0) {
            printf("Oops. pthread_create returned error code %d0, status);
            exit(-1);
        }
    }
    exit(NULL);
}

```

Thread call	Description
Pthread_create	Create a new thread
Pthread_exit	Terminate the calling thread
Pthread_join	Wait for a specific thread to exit
Pthread_yield	Release the CPU to let another thread run
Pthread_attr_init	Create and initialize a thread's attribute structure
Pthread_attr_destroy	Remove a thread's attribute structure

- Más de 60 llamadas a funciones
- Hilo: identificador, registros y atributos
 - Atributos: tamaño de pila, parámetros de planificación, ...

Fíos

Fíos en Linux

- SO multifío: Os fíos xestionanse no espazo do núcleo (kernel)
- Disolve a distinción entre procesos e fíos

`pid=clone(función,pila_ptr,bandeiras_compart,arg);`

- O fío crease no proceso actual ou nun novo proceso
- O fío comeza a súa execución en `funcion(arg)`
 - Pila privada: `SP=pila_ptr`
 - `bandeiras_compart` especifica o nivel de compartición

Flag	Meaning when set	Meaning when cleared
<code>CLONE_VM</code>	Create a new thread	Create a new process
<code>CLONE_FS</code>	Share umask, root, and working dirs	Do not share them
<code>CLONE_FILES</code>	Share the file descriptors	Copy the file descriptors
<code>CLONE_SIGHAND</code>	Share the signal handler table	Copy the table
<code>CLONE_PID</code>	New thread gets old PID	New thread gets own PID
<code>CLONE_PARENT</code>	New thread has same parent as caller	New thread's parent is caller

Tarefa/Task (FreeRTOS)

Tarefa/Task

É un concepto similar a fío POSIX, pero funcionan dunha maneira algo distinta

- Na suspensión/reanudación gardan a súa información na súa propia pila
 - Cada tarefa ten unha pila propia onde garda os seus datos
- Prioridade: importante para o funcionamento en tempo real
- Implementanse coma se fosen funcións en C
- Pensados para correr indefinidamente (terminación explícita)
- Non poden devolver valores
- Non teñen xeranquía (non hai pai/fillo)

Implementación de Task

Implementación de Task

- Deben devolver `void`
- Usan un bucle infinito
 - Recoméndase que durman se non teñen que facer
- Deben de terminar explicitamente
- Non poden ter `return`
- Implementase unha función e despois se inicia

Implementación dunha Tarefa

```
void vATaskFunction( void *pvParameters )
{
    for( ;; )
    {
        -- Task application code here. --
    }
    vTaskDelete( NULL );
}
```

Implementación e execución dunha Tarefa

```
/* Task to be created. */
void vTaskCode( void * pvParameters )
{
    /* The parameter value is expected to be 1 as 1 is passed in the
    pvParameters value in the call to xTaskCreate() below.
    configASSERT( ( ( uint32_t ) pvParameters ) == 1 );

    for( ; ; )
    {
        /* Task code goes here. */
    }
}

/* Function that creates a task. */
void vOtherFunction( void )
{
BaseType_t xReturned;
TaskHandle_t xHandle = NULL;

    /* Create the task, storing the handle. */
    xReturned = xTaskCreate(
                    vTaskCode,          /* Function that implements the task. */
                    "NAME",            /* Text name for the task. */
                    STACK_SIZE,         /* Stack size in words, not bytes. */
                    ( void * ) 1,        /* Parameter passed into the task. */
                    tskIDLE_PRIORITY, /* Priority at which the task is created. */
                    &xHandle );        /* Used to pass out the created task's handle. */

    if( xReturned == pdPASS )
    {
        /* The task was created. Use the task's handle to delete the task. */
        vTaskDelete( xHandle );
    }
}
```

Estados dunha Tarefa

Unha tarefa pode estar en 4 estados diferentes:

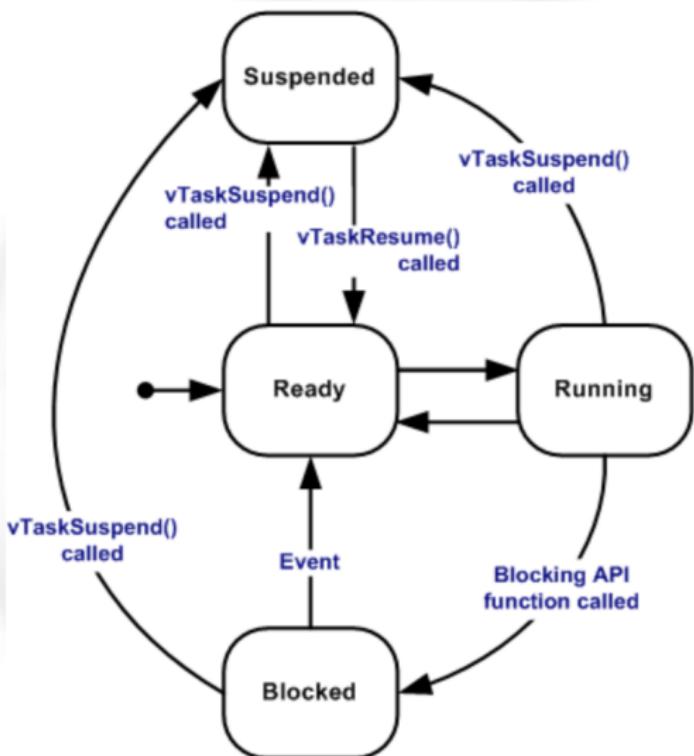
- *Correndo* (Running)
 - A Tarefa está a usar o procesador
 - Se so hai un procesador/núcleo so pode haber unha tarefa correndo á vez
- *Lista* (Ready)
 - A Tarefa está preparada para usar o procesador, pero non pode pasar a *Correndo* porque outra tarefa de máis prioridade está *Correndo*

Estados dunha Tarefa

Unha tarefa pode estar en 4 estados diferentes:

- *Bloqueada* (Blocked)
 - A Tarefa está esperado por un evento externo ou temporal, e non pode pasar a *Correndo*
 - Exemplo `vTaskDelay()` bloquea a tarefa por un periodo de tempo
 - Normalmente teñen un *timeout*, un tempo tras o cual se desbloquean
- *Suspendida* (Suspended)
 - A Tarefa non pode pasar a *Correndo*
 - Non ten *timeout*
 - So se pode entrar explicitamente con `vTaskSuspend()` e `xTaskResume()`

Transición entre estados dunha Tarefa



Prioridade dunha Tarefa

Unha tarefa ten que ter unha prioridade

- Van desde 0 a (`configMAX_PRIORITIES - 1`), definida en `FreeRTOSConfig.h`
- O máximo soe ser 32, non se recomenda un número moi grande
- O 0 é a mínima prioridade
- A tarefa en *correndo* sempre será a tarefa coa prioridade máis alta
- Sempre hai unha tarefa con prioridade 0: a *Idle Task* (tafeña inactiva)
 - Encargase de liberar a memoria das tarefas eliminadas
 - So se executa se non hai nada que facer
 - Pode modificarse para incluir funcionalidades, pero hai limitacións porque nunca pode estar *bloqueada*
 - Soe usarse para poñer o microcontrolador en modo de baixo consumo

Planificación de Procesos

Que é a planificación de procesos?

- Varios procesos e fíos compiten pola CPU
 - Procesos en estado *listo*
- Que se executa a continuación?
 - Planificador de procesos
 - Algoritmo de planificación
- Sistemas diferentes, planificación diferente
 - PC: poucos procesos activos, a planificación é pouco importante (limitados pola E/S, aínda que cada vez hai máis procesos que necesitan moita CPU)
 - Servidores: moitos procesos que requieren CPU, a planificación é crítica
 - Sistemas Encaixados: planificación crítica polo tempo real e a E/S
- Débese asegurar un uso eficiente da CPU
- Planificador=Calendarizador (*Scheduler*)

Planificación de Procesos

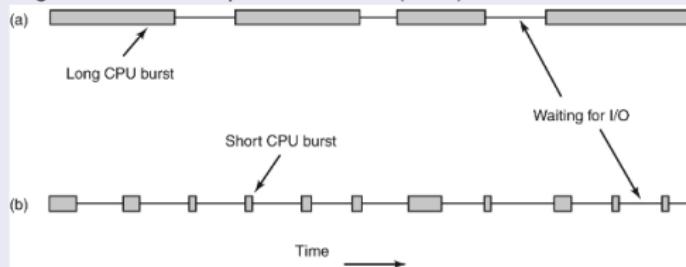
Comutación de procesos (context switch ou cambio de contexto)

- **A conmutación de procesos é cara**
 - Paso de modo usuario a modo núcleo
 - Guardar o estado do proceso (incluíndo rexistros) na táboa de procesos, o mapa de memoria
 - Seleccionar un novo proceso mediante o algoritmo de planificación
 - Cargar o novo proceso (rexistros, mapa de memoria, ...)Pódese quitar información do proceso na cache e na memoria principal
- Se as conmutacións de procesos son frecuentes, poden chegar a consumir unha fracción considerable do tempo de CPU

Planificación de Procesos

Tipos de procesos e planificación

- Os procesos alternan ráfagas de cálculo con peticóns de E/S (disco)

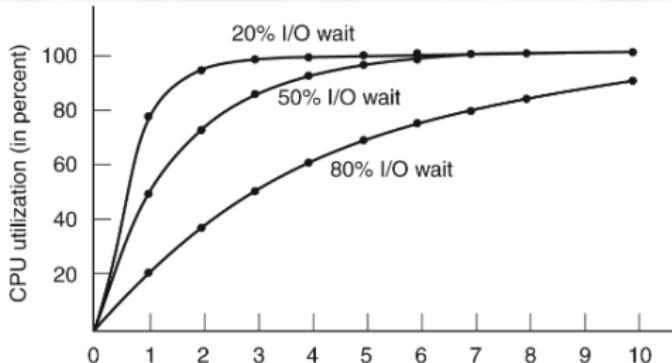


- Procesos limitados a cálculo:** Ráfagas de CPU longas e esperas infrecuentes por operacóns de E/S
- Procesos limitados a E/S:** Ráfagas de CPU curtas e esperas frecuentes por E/S
- Ao aumentar a velocidade da CPU os procesos tenden a ser limitados por E/S
- Factor chave: **duración da ráfaga de CPU**

Planificación de Procesos*

Utilización da CPU

- Estratexia: Se un proceso limitado a E/S quere executarse, debe obter rápidamente a CPU
- Obxectivo: Desperdiciar pouco tempo de CPU (p.e. < 10 %)
 - Se os procesos usan o 80 % do seu tempo esperando en operáns E/S, debe de haber 10 procesos en memoria
 - Se usan o 20 %, é suficiente con 2 procesos en memoria



Planificación de Procesos

Cando planificar un proceso?

- Na creación dun novo proceso: executase o proceso pai ou o fillo?
- Un proceso termina: Que proceso listo se executa?
 - Se non hai ningún proceso listo, executase o **proceso inactivo**
- Un proceso bloquease: Tense en conta a razón do bloqueo?
- Unha interrupción de E/S: Debese executar o proceso que estaba esperando pola E/S?
- Interrupción de reloxo: Cambiase de proceso?
 - Algoritmo **Non apropiativo** (Non-preemptive): seleccionase un proceso e se executa ata que se bloquea
 - Algoritmo **Apropiativo** (Preemptive): seleccionase un proceso e se executa durante un tempo máximo fixado (*quantum de tempo*)

Algoritmos de planificación*

All systems

- Fairness - giving each process a fair share of the CPU
- Policy enforcement - seeing that stated policy is carried out
- Balance - keeping all parts of the system busy

Batch systems

- Throughput - maximize jobs per hour
- Turnaround time - minimize time between submission and termination
- CPU utilization - keep the CPU busy all the time

Interactive systems

- Response time - respond to requests quickly
- Proportionality - meet users' expectations

Real-time systems

- Meeting deadlines - avoid losing data
- Predictability - avoid quality degradation in multimedia systems

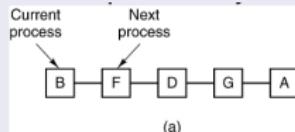
Tipos de algoritmos

- Distintos sistemas, algoritmos de planificación diferentes
- Procesamiento por lotes:** non hai procesos que esperen para obter unha resposta rápida
 - Non apropiativos ou apropiativos con longos períodos
 - Reduce a conmutación de procesos
- Interactivo:** evitar que un proceso acapare a CPU
 - Apropiativos
- De **tempo real:** condicionadas polas exixencias do sistema

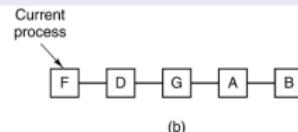
Planificación en Sistemas Interactivos

Planificación por turno circular (Round-Robin)

- A cada proceso se lle asigna un *quantum*
- Conmutase a outro proceso (cambio de contexto) se:
 - Terminase o quantum
 - O proceso actual se bloquea
- Lista de procesos ejecutables



(a)



(b)

Planificación en Sistemas Interactivos

Determinación da duración do *quantum*

- O cambio de contexto require tempo
- Parte do tempo de CPU desperdiçiase por sobrecarga administrativa
- *Quantum* demasiado curto:
 - Demasiados cambios de contexto
- *Quantum* maior:
 - Tempo de espera longos cando hai moitos procesos
 - Moitos procesos bloqueanse antes de terminar o seu *quantum*
 - *Mala resposta ás peticións interactivas curtas*
- En PC: *Quantum* de 20 ms a 50 ms

Planificación en Sistemas Interactivos

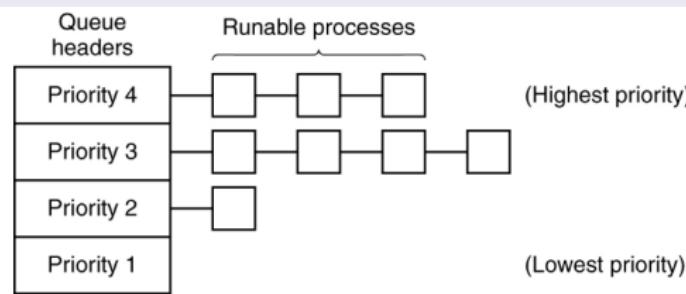
Planificación por prioridade

- Cada proceso ten unha prioridade
- Executase o proceso listo coa prioridade máis alta
- Evitar que procesos con alta prioridade copen a CPU
 - En cada interrupción do reloxo reducese a prioridade do proceso en execución e se outro proceso ten prioridade máis alta, conmutase de proceso
 - Un *quantum* a cada proceso. Terminado o *quantum* elixese o proceso de prioridade máis alta
- Asignación dinámica de prioridades
 - Os procesos limitados a E/S deben recibir a CPU inmediatamente cando a necesiten
 - Asignar prioridade = $1/f$, onde f é a fracción do último *quantum* utilizada polo proceso, da bo servizo aos procesos limitados a E/S

Planificación por prioridade

Planificación por turno circular (Round-Robin)

- Agrupación de procesos en clases de prioridade, con planificación de turno circular en cada clase
 - Se hai procesos de prioridade 4, seleccionanse por turno circular dentro de esa clase
 - Se a clase de prioridade 4 está baleira, ejecutanse os da clase de prioridade 3
 - As clases deben axustarse dinámicamente



Planificación en Sistemas Interactivos*

Outros algoritmos

- O proceso más curto a continuación
 - Estimación do tempo de execución en base a execucións anteriores: $T = a \times T_0 + (1 - a) \times T_1$, T_0 tempo estimado, T_1 tempo da última execución, $0 \leq a \leq 1$
- Planificación garantida
 - Con n usuarios (ou procesos) cada usuario (ou proceso) recibe $1/n$ de tempo de CPU
- Planificación por sorteo
 - Danse aos procesos boletos e a planificación faise por sorteo
 - Os procesos reciben diferente número de boletos
- Planificación por partes equitativas
 - Asignaselle unha fracción da CPU a cada usuario, que pode depender do número de procesos que teña

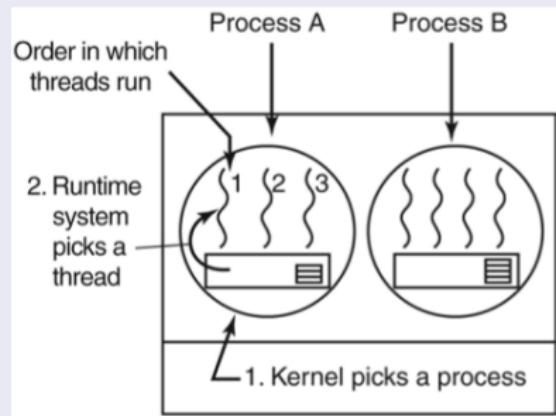
Planificación de Fíos

Dous niveis de paralelismo, procesos e fíos

- Fíos a nivel de usuario
 - O SO planifica procesos
 - Cada proceso ten un planificador de fíos local
 - Planificadores específicos para unha aplicación
- Fíos a nivel de núcleo
 - O núcleo selecciona un fío sen importar a que proceso pertence (aínda que pode telo en conta)
 - Comutar entre fíos de procesos diferentes es máis custoso que entre fíos do mesmo proceso
 - Pódese ter en conta na planificación
 - Se un fío se bloquea, non se bloquea todo o proceso

Planificación de Fíos

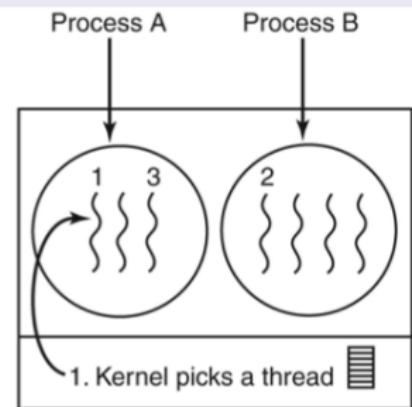
Exemplo: quantum de 50 ms, fíos con ráfagas de 5 ms



Possible: A1, A2, A3, A1, A2, A3
 Not possible: A1, B1, A2, B2, A3, B3

(a)

Fíos a nivel de usuario



Possible: A1, A2, A3, A1, A2, A3
 Also possible: A1, B1, A2, B2, A3, B3

(b)

Fíos a nivel de núcleo

Planificación de Tarefas (FreeRTOS)

Planificación de tarefas en FreeRTOS

Por defecto FreeRTOS usa unha planificación apropiativa de prioridade fixa con turno circular con reparto de tempo de tarefas da mesma prioridade

Planificación de Tarefas (FreeRTOS)

Planificación de tarefas en FreeRTOS

- **Prioridade fixa (fixed-priority):**
 - O calendarizador nunca cambia a prioridade dunha tarefa (salvo para aumentala temporalmente en certos casos, herencia de prioridade)
- **Apropiativa (preemptive):**
 - Sempre corre a tarefa con máis prioridade
 - Se unha tarefa de máis prioridade está *bloqueada*, pero pasa a *lista* (por interrupción, p.e.) execútase inmediatamente e bota fora á actual
- **Turno circular (round robin):**
 - As tarefas coa mesma prioridade toman turnos entrando á *correndo*
- **Reparto de tempo (time-sliced):**
 - As tarefas execútanse cun *quantum*, despois poden cambiar a tarefas da mesma prioridade

Planificación de Tarefas (FreeRTOS)

Consecuencias do planificador

- No programa o calendarizador (scheduler) iniciase despois de declarar e crear todas as tarefas
- Para dúas tarefas de diferente prioridade:
 - A de prioridade alta pode executarse sempre
 - A de baixa prioridade pode que nunca se execute (*task starvation*, morre de fame)
 - As tarefas de alta prioridade teñen que entrar *bloqueada* ou *suspendida* por si mesmas
 - Non se pode usar espera activa, hai que usar eventos

Planificación de Tarefas (FreeRTOS)

Consecuencias do planificador

- **A Apropiación pode desactivarse**
 - Unha tarefa so sae de execución (*correndo*) se ela o pide
 - TASKYIELD() ou entra en *bloqueada* ou *suspendida*
 - Tamén se hai unha interrupción
- **O reparto de tempo (time-slicing) pode configurarse ou desactivarse**
 - O *quantum* pode ser infinito
 - Nunca se cambia entre tarefas da mesma prioridade

Exemplo de Planificación (FreeRTOS)

Exemplo de Planificación (FreeRTOS)

- Exemplo oficial
- Sistema cun teclado e unha pantalla LCD
- Tarefa *Key*: Manexar as pulsacións de tecras
 - Debería procesarse en menos de 100 ms para que o usuario non note retardo e saiba que pulsou funcionou a pulsación
- Tarefa *Control*: Función de control
 - Depende dunha entrada dixital
 - Debe ser mostreada e filtrada, despois realizar a operación de control
 - Cada 2 ms
 - Para a correcta operación do filtro a temporalidade debe de ser non maior de 0.5 ms

Exemplo de Planificación (FreeRTOS)

Tarefa KEY

```
void vKeyHandlerTask( void *pvParameters )
{
    // Procesar as pulsacións de tecras é
    // algo continuo
    // así que o implementamos cun bucle infinito
    for( ;; )
    {
        [Suspender á espera dunha pulsación]
        [Procesar a pulsación]
    }
}
```

Exemplo de Planificación (FreeRTOS)

Tarefa CONTROL

```
void vControlTask( void *pvParameters )
{
    for( ;; )
    {
        [Suspender esperado 2ms desde o último
         Mostrear a entrada]
        [Filtrar a entrada]
        [Realizar o algoritmo de control]
        [Saida do resultado]
    }
}
```

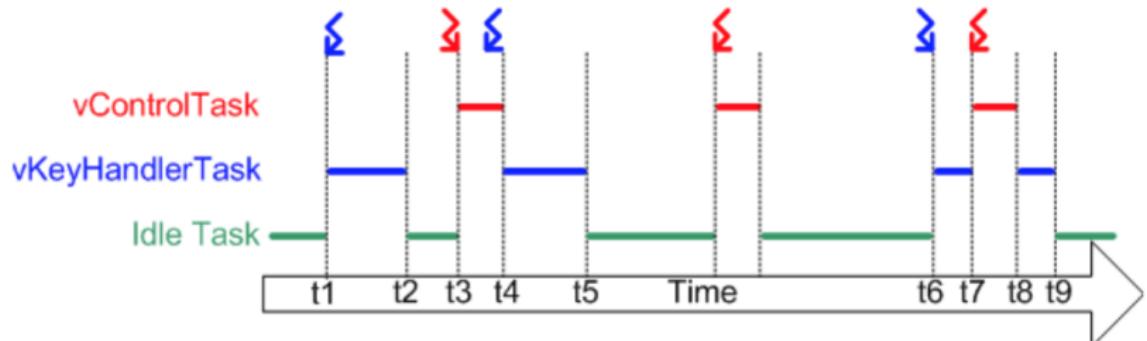
Exemplo de Planificación (FreeRTOS)

Exemplo de Planificación (FreeRTOS)

- A Tarefa *Control* ten maior prioridade
 - Ten un tempo límite más estricto que o manexador de tecrado
 - As consecuencias de superar o tempo límite son peores

Exemplo de Planificación (FreeRTOS)

Exemplo de Planificación (FreeRTOS)



↓ = Key Press Event

↑ = Timer Event

Exemplo de Planificación (FreeRTOS)

Exemplo de Planificación (FreeRTOS)

- Ao principio ningún pode executarse. *Key* espera á pulsación da tecra, *Control* ao tempo adecuado para procesar. Executase o Idle Task
- En t1 púlsase unha tecra, *Key* execútase porque ten máis prioridade que o Idle Task()
- En t2 *Key* acabou e suspendese, retómase o Idle Task
- En t3 un evento de reloxo indica que hai que realizar o ciclo de control. *Control* ten a max prioridade, execútase
- Entre t3 e t4 púlsase unha tecra pero, como *Key* ten menos prioridade que *Control*, séguese executando *Control*
- En t4 *Control* acabou de procesar, así que se suspende. *Key* pode agora executarse porque é a tarefa con máis prioridade

Exemplo de Planificación (FreeRTOS)

Exemplo de Planificación (FreeRTOS)

- En t5 *Key* acaba de manexar a pulsación e se suspende. Non hai tarefas así que se executa Idle Task
- Entre t5 e t6 ocorre un evento de tempo, execútase *Control* pero non se pulsan tecras
- En t6 púlsase unha tecra e se executa *Key*
- En t7 ocorre un evento de tempo. *Key* aínda non acabou de procesar a pulsación, pero como *Control* ten maior prioridade *Key* é suspendido polo SO e execútase *Control*
- En t8 *Control* remata co ciclo, é entón reanúdase *Key*. *Key* non se da conta de que foi suspendido e reanudade polo SO e funcion correctamente

Planificación de Tarefas (FreeRTOS)

Multiprocesador: Asymmetric multiprocessing (AMP)

- O Sistema Encaixado pode ter varios procesadores/núcleos
- Nese caso pode haber varias instancias de FreeRTOS, unha en cada procesador
- Poden ter diferente arquitectura
- Cada procesador/núcleo ten o seu calendarizador
- Deben compartir algo de memoria para poder comunicacarse as instancias
 - Sincronización manual

Planificación de Tarefas (FreeRTOS)

Multiprocesador: Symmetric multiprocessing (SMP)

- O Sistema Encaixado pode ter varios núcleos
- Teñen a mesma arquitectura e comparte a memoria
- Pode haber unha soa instancia de FreeRTOS, mesmo calendarizador
 - Hai varias tarefas en *correndo*
 - Pode haber tarefas de distintas prioridades, xa non corre so a máis prioritaria
- Pode configurarse para correr so tarefas da mesma prioridade
 - Así evitas problemas con códigos escritos pensando na execución AMP
- As tarefas poden configurarse para correr so nuns núcleos determinados
 - Así podes controlar mellor as prioridades, pero máis complexo que o anterior

Clasificación xeral dos Sistemas en Tempo Real (STR)

Tempo Real Duro

O tempo de resposta debe garantirse a toda costa

- Unha resposta tardía pode ter consecuencias fatais
- Exemplos:
 - Aplicación de guía e control de misiles
 - Aplicacións de sistemas médicos

Tempo Real Suave

O tempo de resposta debe garantirse a toda costa

- Unha resposta tardía non produce graves danos, pero si un deterioro do funcionamento global
- Exemplos:
 - Aplicacións multimedia
 - Aplicacións para banca

Como se logra o comportamento en Tempo Real?

Como se logra o comportamento en Tempo Real?

- Divídese o programa en varios procesos
 - Comportamento predecible e coñecido de antemán
 - Tempos de vida curtos, a sua execución finaliza en moito menos dun segundo
- Cando se detecta un evento externo, o planificador é responsable de planificar os procesos respetando o cumplimento dos tempos límite
 - Eventos (Tarefas) **periódicos** o **aperiódicos**

Cando é planificable un Sistema Tempo Real?

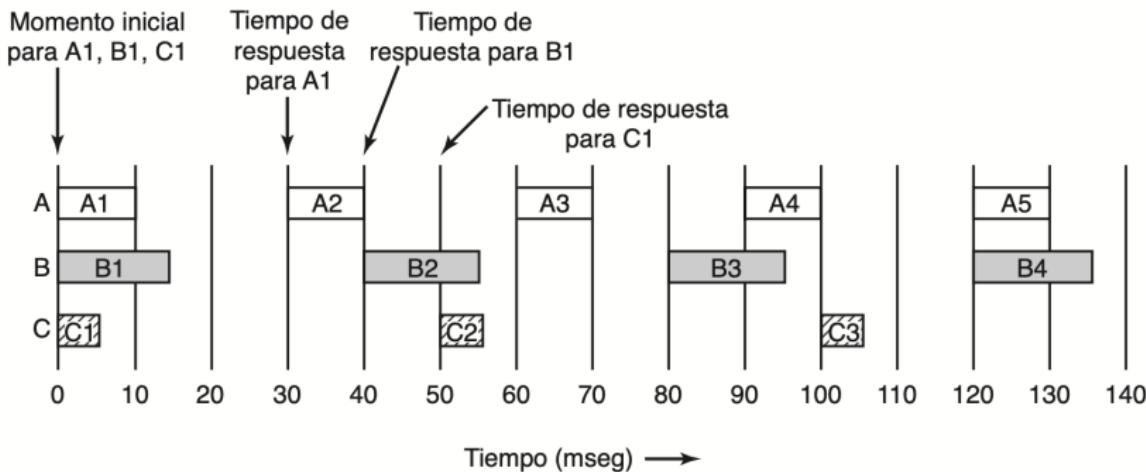
Cando é planificable un STR?

No caso de ter que atender n fluxos de eventos **periódicos**, o STR será planificable se $U \leq 1$:

$$\sum_{i=1}^m \frac{C_i}{P_i} \leq 1$$

- n : número de eventos
- U : factor de utilización
- C_i : tempo máximo de execución do evento i
- P_i : período do evento i
- C_i/P_i : fracción de CPU que está utilizando o evento i

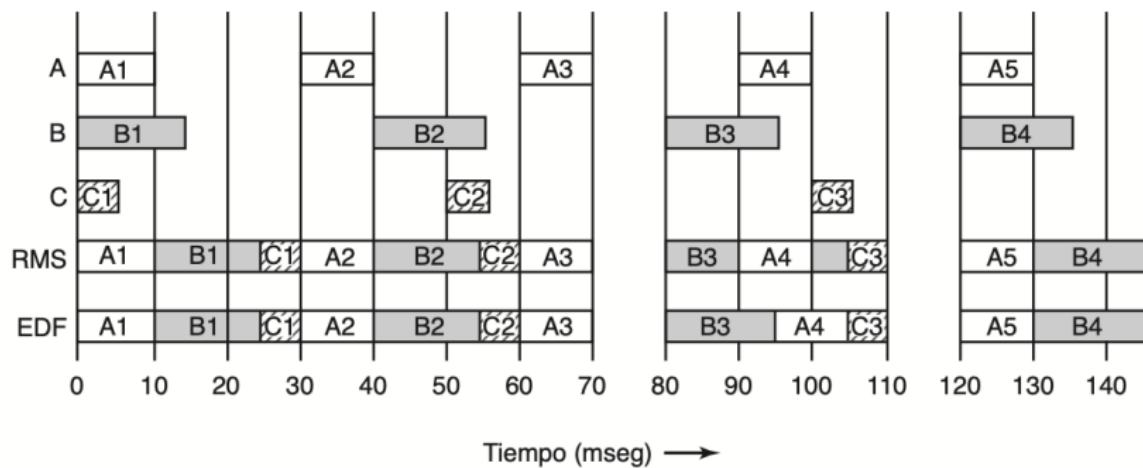
Cando é planificable un Sistema Tempo Real?



- $C_A = 10ms, 30 \text{veces/s} \rightarrow P_A = 1000/30 \approx 30ms$ (NTSC)
- $C_B = 15ms, 25 \text{veces/s} \rightarrow P_B = 1000/25 = 40ms$ (PAL)
- $C_C = 5ms, 20 \text{veces/s} \rightarrow P_C = 1000/20 = 50ms$ (NTSC/PAL reducido)
- $U = 0,808$
- Ignorase o tempo de cambio de contexto

Algoritmo RMS

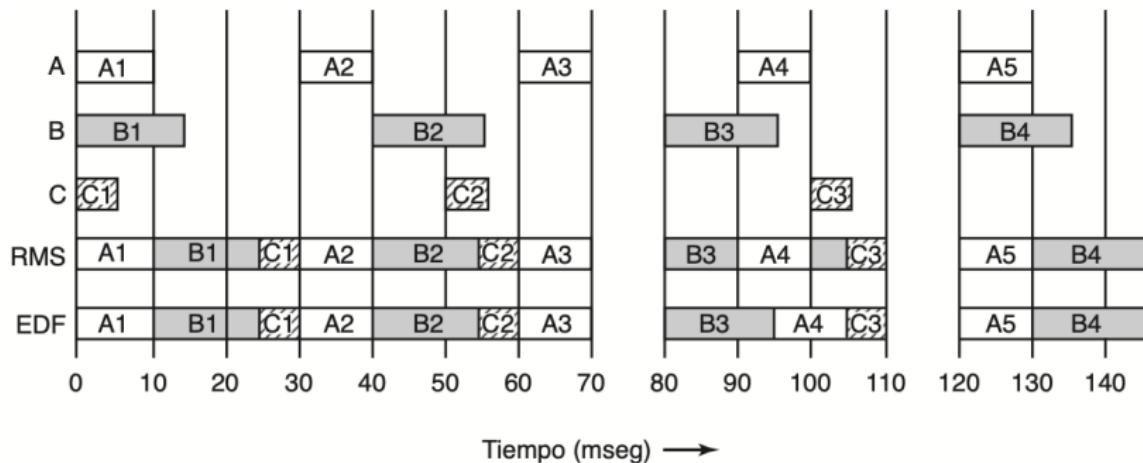
Algoritmo de programación monotónica en frecuencia (Rate Monotonic Scheduling)



- Sólo para procesos periódicos e independientes
- Necesita el mismo tiempo de CPU en cada ráfaga
- Prioridad → frecuencia de ocurrencia
- $Pri_A = 33, Pri_B = 25, Pri_C = 20$

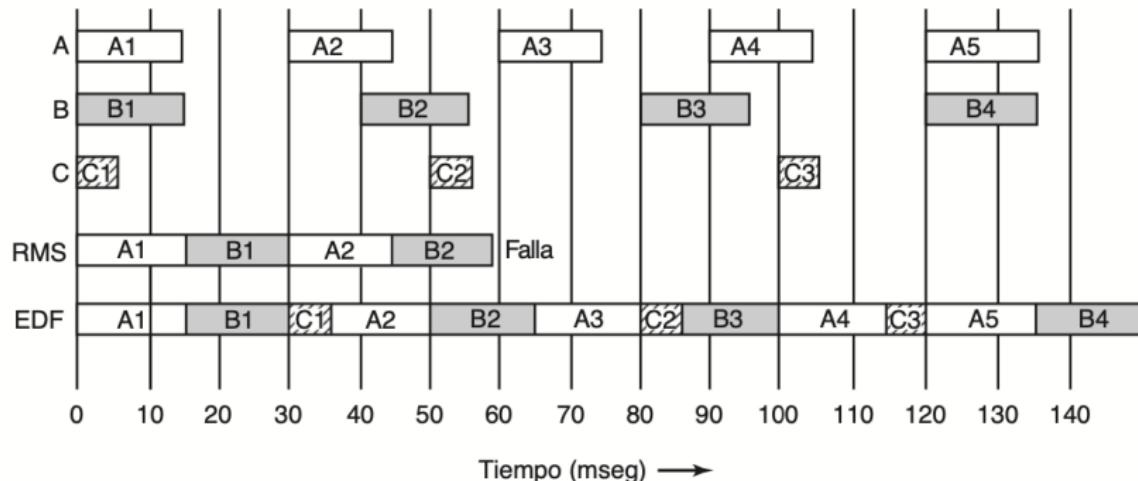
Algoritmo EDF

Algoritmo de menor tempo de resposta (Earliest Deadline First)



- Para procesos periódicos e aperiódicos
- Tiempo de CPU diferente en cada ráfaga
- O que acabe antes primeiro (*Deadline*)
- En $t=90$ A está listo, pero acabaría en 120, igual que B, así que sigue B porque non hai cambio contexto

Outro exemplo



- Aumentamos $C_A = 15ms$
- $U = 0,975$
- En RMS C pásase do seu tempo de resposta (*deadline*)
- En EDF funciona, pero é un algoritmo máis complexo
- En EDF hai un oco cada 200ms, ese 0,025
- Por estatística RMS require dunha U moito menor que 1

Algo máis

- Por estatística RMS require dunha U moito menor que 1

$$\sum_{i=1}^m \frac{C_i}{P_i} \leq 1$$

- Tende a $\ln 2 = 0,693\dots$, con 3 é 0,780

- Recordade: FreeRTOS usa un algoritmo apropiativo de prioridade fixa, o máis sinxelo
- FreeRTOS non permite definir *deadlines* (algo similar con `vTaskDelayUntil()`)
- Para usar estos hai outras librarías que modifican o freeRTOS, pero non parecen moi populares

Índice

1 Introducción e Conceptos Básicos

- Conceptos básicos dos SO
- Estructura do Sistema Operativo

2 Interrupcóns

3 Procesos, Fíos e Tarefas

- Procesos, Fíos e Tarefas
- Planificación
- Teoría de Programación en Tempo Real

4 Comunicación entre Procesos/Tarefas

Referencias

Imaxes desde:

Y. Zhu. "Embedded Systems with ARM Cortex-M Microcontrollers in Assembly Language and C". Third Edition. E-Man Press LLC. 2017. ISBN-13: 978-0982692660

Andrew S. Tanenbaum "Sistemas operativos modernos"(3a edición). Editorial Prentice-Hall, 2009