CURSO DE ESPECIALIZAÇÃO EM AUTOMAÇÃO INDUSTRIAL

Fundamentos e Aplicações de Sistemas de Automação

Módulo 7:
Programação em Tempo Real

Just predictable and reliable
(Nossa definição)

Professor: Constantino Seixas Filho

sábado, 30 de outubro de 2004

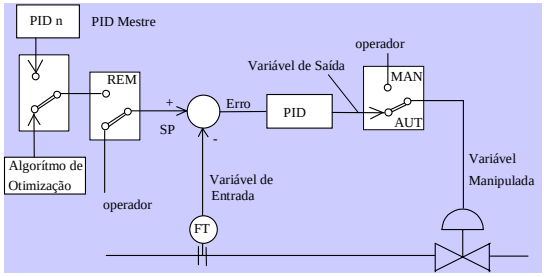
1

Programação em Tempo Real

- O nosso programa deve interagir com o ambiente e dar respostas adequadas a eventos que ocorrem e que não estão sincronizados com as operações internas do sistema

2

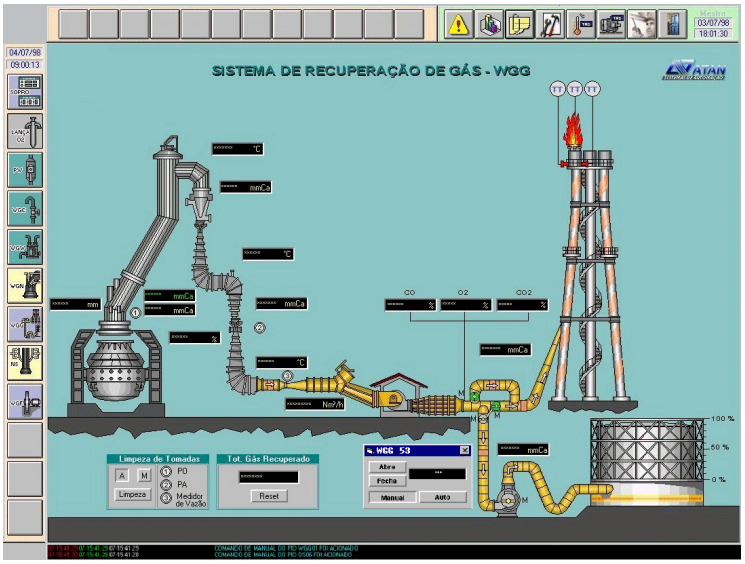
Exemplos – Sistema de controle de processos industriais



- Um sistema de controle de um processo amostra com uma freqüência definida as variáveis que desejamos controlar, por exemplo a temperatura de um forno. Objetivos:
  - Manter a variável de processo (PV) estável apesar das perturbações do processo.
  - Aumentar e diminuir a PV acompanhando mudanças no valor do Set-Point (SP) atuando sobre uma variável manipulada (MV).
- Como todos sabemos de nossa aulas de controle, a variável tempo entra na equação do algoritmo de controle. A velocidade da amostragem é função da constante de tempo do sistema.

3

Exemplo - Sistema supervisório



4

Sistema de controle incorporado de um míssil

- Um sistemas de mísseis Patriot deve localizar e perseguir um alvo móvel de alta velocidade, e destrui-lo, explodindo uma carga nas suas proximidades. O sistema de radares localiza e rastreia até 50 mísseis simultaneamente. O míssil recebe do sistema de radares a posição do alvo, determina a sua própria posição continuamente e atua nas aletas de posicionamento em tempo real para diminuir sua distância em relação ao alvo.

5

Vídeo game interativo

6

Definição de sistema de tempo real

A primeira definição foi extraída de [Bennet 88]:

"Dizemos que um computador opera em tempo real quando as ações realizadas por ele estão relacionadas com as escalas de tempo dos processos externos" .

Em geral existem alguns tempos a serem respeitados e existe um certo consenso na comunidade de automação sobre os valores a serem atingidos:

a) Tempo entre o comando de uma função da console de operação, por exemplo o acionamento do botão de liga de um equipamento, e a correspondente efetivação do comando no campo (1 segundo).

b) Tempo entre a ocorrência de um evento de processo e a sinalização de alarme na sala de operação (1 segundo).

c) Tempo entre a solicitação de um sinóptico e sua aparição na tela (1 segundo).

d) Tempo entre a modificação do status de um equipamento em uma console e sua atualização no vídeo do micro reserva (2 segundos).

Tempo entre o comando de um equipamento e a atualização de seu status na tela após retorno do campo (2 segundos).

7

Definição de sistema de tempo real

- "Sistemas de tempo real são sistemas computacionais em que podemos garantir que todas as funções programadas serão executadas em um intervalo máximo de tempo definido para cada função em toda e qualquer ocasião. Isto inclui as computações disparadas em resposta a eventos externos previstos no sistema"

8

Sistemas operacionais de tempo real

- **Os sistemas operacionais que garantem condições de tempo real para seus processos ou threads são chamados de Sistemas operacionais de tempo real ou RTOS (*Real time operating systems*)**
- **Sistema operacionais *hard real time*:**
  - **Sistemas hard real time são os que conseguem comprovar teoricamente a sua performance no pior caso**
- **Sistemas operacionais *soft real time*:**
  - **São aqueles cuja performance no pior caso foi observada, mas não provada**

9

Sistemas Síncronos e Assíncronos

Texec	Uma vez definido um Hardware, nós podemos assumir para cada thread uma duração que corresponda ao pior caso. Este tempo será chamado de Texec
T	Cada thread necessita ser escalonada e ter sua execução completada dentro de um tempo máximo estipulado. Por exemplo um algoritmo de controle PID para uma determinada malha será executado a cada 100 ms. A rotina, para verificar se todos os sensores estão operando normalmente, pode ser executada mais raramente, a cada 10 min e assim por diante. A este tempo chamaremos de período de escalonamento ou simplesmente T

Lembra-se do Tamagoshi ?

10

Sistemas síncronos ou *clock-driven*

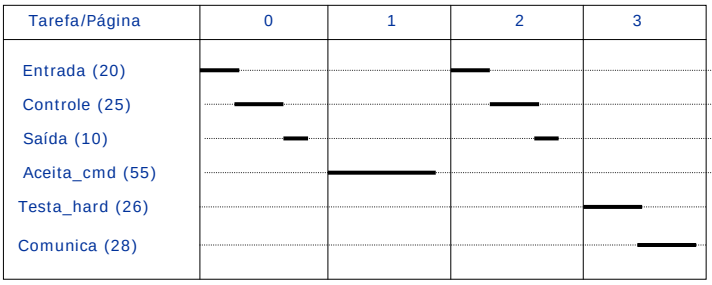
```
Escalonador_Síncrono
{    int pagina = 0;
     const int NumPaginas = 4;

     loop { Espera_Sinalização_de_Tempo();
            switch (pagina) {
            case 0: // Página 0
                        Leitura();   Controle(); Saída();
                        break;
            case 1: // Página 1
                        Aceita_cmd();
                        break;
            case 2: // Página 2
                        Entrada();  Controle(); Saída();
                        break;
            case 3: // Página 3
                        Testa_hard();  Comunica();
                        break;
            default: break;
            } //end_switch
            pagina = (pagina + 1) % NUM_PAGINAS;
     } // end_loop
} // end_process
```

Encontre exemplos no seu quotidiano !

11

Sistemas síncronos ou *clock-driven*
Diagrama Temporal



12

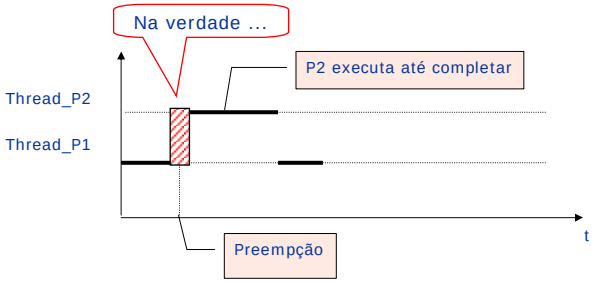
Sistema síncronos

Mecanismo adicional

```
switch (pagina)
{    case 0: // Página 0
            if (Lflag) Leitura();
            if (Cflag) Controle();
            if (Sflag) Saída();
            break;
```

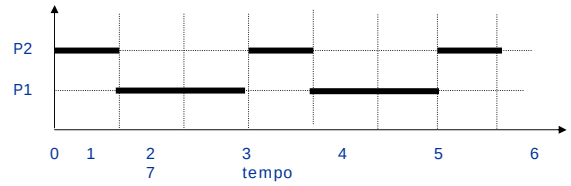

Sistemas assíncronos ou *event-driven*

- Para que as tarefas sejam escalonadas sem que haja nenhum planejamento inicial, como nos sistemas síncronos, e sem que tenhamos que medir a duração de cada tarefa, vamos precisar de dois conceitos auxiliares:
  - Programa privilegiado que haja como árbitro: escalonador preemptivo
  - Prioridade
- O chaveamento de contexto representa um *overhead* já que se traduz em trabalho inútil, sob o ponto de vista do que queremos realizar, e deve ser minimizado.

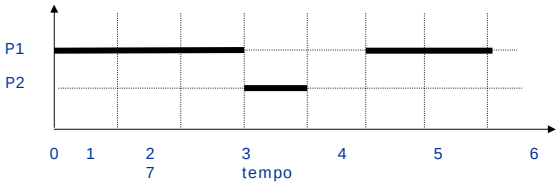


Algoritmos de escalonamento de tempo real

Processo	T (ms)	texec (ms)
P1	5	3
P2	3	1



- Diagrama temporal de alocação de tarefas para Pr_2 > Pr_1



- Diagrama temporal de alocação de tarefas para Pr_1 > Pr_2

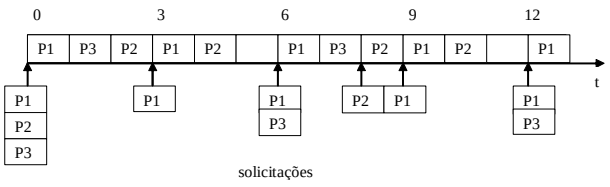
15

Escalonador RM (*Rate Monotonic*)

- Foi desenvolvido em 1973 por Liu e Layland. Aloca prioridades estaticamente na ordem inversa aos períodos de solicitação. Quanto mais rapidamente um processo precisa ser executado, maior a sua prioridade.
- RM é ótimo no sentido de que ele é sempre capaz de escalonar uma tarefa se um outro algoritmo estático o consegue.
- O teste de factibilidade para um conjunto de tarefas é dado por:

$$\sum_i \frac{t\_exec}{T} \leq n * (2^{\frac{1}{n}} - 1)$$

Processo	T (ms)	texec (ms)	Carga
P1	3	1	33.3%
P2	8	2	25%
P3	6	1	16.7%

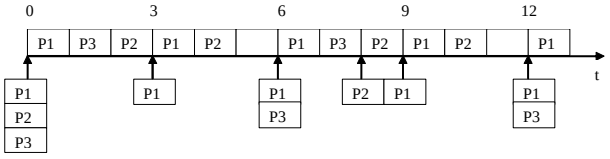


16

Escalonador EDF: (*Earliest Deadline First*)

- Escalona primeiro a tarefa cuja data limite irá se expirar primeiro
- O raciocínio por trás deste algoritmo é muito semelhante ao que praticamos em várias situações de nossa vida real: nós geralmente estudamos para a prova que está mais próxima, fazemos primeiro a proposta cuja data limite está mais próxima, e assim por diante
- Este algoritmo é dinâmico porque avalia a situação a cada instante para decidir que tarefa será escalonada primeiro não atribuindo prioridades estáticas às tarefas
- O teste de factibilidade para um dado conjunto de tarefas é dado por: $\sum_i \frac{t\_exec}{T} \leq 1$

Processo	T (ms)	texec (ms)	Carga
P1	3	1	33.3%
P2	8	2	25%
P3	6	1	16.7%



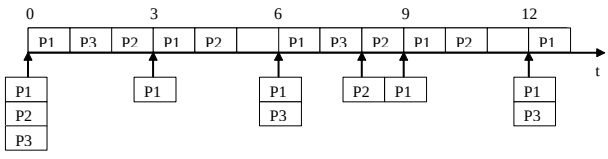
solicitações

17

Escalonador LLF (Least Laxity)

- O escalonador de margem de tempo mínima estuda a tarefa que oferece menor flexibilidade para ser escalonada. A margem de tempo corresponde à diferença entre: tempo para data limite e o tempo para completar a tarefa. Este tempo é computado dinamicamente. Se a margem é negativa então a tarefa já não conseguirá cumprir sua data limite e o escalonador falhou. A carga total deve ser menor que 1

Processo	T (ms)	texec (ms)	Carga
P1	3	1	33.3%
P2	8	2	25%
P3	6	1	16.7%



solicitações

18

Escalonador LLF (Least Laxity)

Tempo	Demanda	Deadline	Restante para terminar	Margem	Tarefa a ser escalonada
0	P1 P2 P3	3 8 6	1 2 1	2 6 5	P1
1	P2 P3	7 5	2 1	5 4	P3
2	P2	6	2	4	P2
3	P1 P2	3 5	1 1	2 4	P1
4	P2	4	1	3	P2
5	-	-	-	-	-
6	P1 P3	3 6	1 1	2 5	P1
7	P3	5	1	4	P3
8	P2	8	2	6	P2
9	P1 P2	3 7	1 1	2 6	P1
10	P2	6	1	5	P2
11	-	-	-	-	-

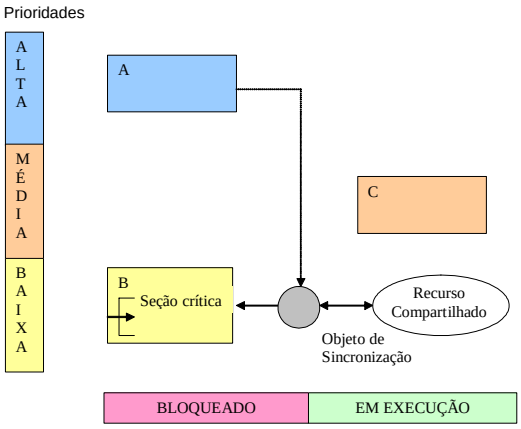
19

Escalonador MUF (*Maximum Urgency First*)

- É baseado num duplo critério. A criticidade de cada tarefa (alta ou baixa) é definida pelo programador para cada tarefa do sistema e constitui a parte estática da análise. Ela tem precedência sobre a parte dinâmica. Primeiro as tarefas são ordenadas em ordem crescente do seu período. As N primeiras tarefas que tenham o atributo crítico e cujo somatório da carga seja menor que 1 são selecionadas e forma o *task set*. A tarefa de alta criticidade com a menor margem é escalonada. As tarefas de baixa criticidade só são analisadas caso não haja mais tarefas de alta criticidade. O *overhead* como se nota é muito alto.

20

Inversão de prioridades



21

Como evitar que a inversão de prioridades ocorra ?

- A melhor solução é um mecanismo denominado de herança de prioridade. Ao se criar um objeto de sincronização, devemos definir um parâmetro adicional chamado de herança de prioridade. Com este parâmetro ativado, o processo que detém a posse de um objeto de sincronização, herda a prioridade do processo que fica bloqueado. Desta forma, B herdaria a prioridade de A e não seria preemptado por C.
- Infelizmente este recurso não existe no Windows NT, mas apenas no Windows CE.
- Segundo a documentação da Microsoft [Microsoft 95a], o WNT sai da situação de inversão aumentando randomicamente a prioridade das threads prontas para executar (B). Quando a prioridade de B se igualar à de C, B consegue sair da seção crítica. O Windows 95 detecta a situação A bloqueado por B e faz prioridade(B) = prioridade(A) até B sair da seção crítica. Isto corresponde a uma herança de prioridade automática.

22

O que aconteceu em Marte ?

From: Mike Jones <mbj@microsoft.com>
Sent: Sunday, December 07, 1997 6:47 PM
Subject: What really happened on Mars?

The Mars Pathfinder mission was widely proclaimed as "flawless" in the early days after its July 4th, 1997 landing on the Martian surface. Successes included its unconventional "landing" -- bouncing onto the Martian surface surrounded by airbags, deploying the Sojourner rover, and gathering and transmitting voluminous data back to Earth, including the panoramic pictures that were such a hit on the Web. But a few days into the mission, not long after Pathfinder started gathering meteorological data, the spacecraft began experiencing total system resets, each resulting in losses of data. The press reported these failures in terms such as "software glitches" and "the computer was trying to do too many things at once".

- A nave espacial norte americana Mars Pathfinder realizou um pouso em Marte no dia 4 de julho de 1997 liberando o jipe Sojouner para um passeio no solo marciano.
- Este robot deveria enviar um grande volume de dados e imagens para a Terra. Depois de alguns dias, a nave passou a sofrer *resets* espontâneos e aleatórios que causavam a total reinicialização do sistema. Os dados armazenados em RAM não eram perdidos, mas a atividade de aquisição só era recomeçada no dia seguinte.

23

O que aconteceu em Marte - Análise

- A nave possui uma CPU como o RTOS VxWorks da Wind River. Os demais componentes do computador, câmera e rádio se comunicam com a CPU via barramento VME e com o barramento 1553 que serve para conectar a parte móvel do módulo de aterragem.
- O sistema é constituído por diversas tarefas que executam ciclicamente com um período de 125 ms, de acordo com uma ordem pré estabelecida.
- Existem três tarefas principais em ordem crescente de prioridades:
  - ASI/MET que coleta dados dos acelerômetros, radar altimétrico e instrumentos meteorológicos do módulo de pouso.
  - bc_dist que distribui os dados coletados colocando-os em uma memória compartilhada.
  - bc_sched que prepara as transações para o próximo ciclo.
  - bc_sched deve executar sempre em seguida a bc_dist uma vez a cada ciclo.
  - Se bc_sched detecta que bc_dist não conseguiu terminar, reseta o computador (função watch dog).
  - A análise do log da seqüência de instruções, momentos antes da ocorrência da falha, revelou um caso típico de inversão de prioridades.
  - ASI/MET e bc_dist sincronizavam o acesso a uma lista de descritores compartilhada, através de um semáforo. Quando ASI/MET conquistava a posse do semáforo antes de bc_dist, algumas tarefas de média prioridade podiam ser escalonadas impedindo bc_dist de executar e assim fazendo com que a tarefa não cumprisse o seu limite de prazo. Quando bc_sched, de alta prioridade, entrava no final do ciclo e percebia que bc_dist não havia terminado, reinicializava o sistema.

24

O que aconteceu em Marte - Solução

- Detectado o problema, o parâmetro de herança de prioridade dos semáforos foram ativados e o novo código foi transmitido para a nave, que voltou a funcionar sem erros.
- A primeira pergunta que fazemos é por que este parâmetro não fora ativado como default ? A resposta é que programadores de tempo real estão sempre, na vida real, muito preocupados com o tempo de execução de suas tarefas e a ativação da herança de prioridades representava um pequeno aumento de overhead.

25

O que aconteceu em Marte – lições:

São muitas as lições que tiramos deste episódio:

1. Primeiro, que não devemos negligenciar a chance de ocorrência da inversão de prioridades, mesmo que isto nos pareça extremamente improvável.
2. Segundo, que devemos estar preparados para rastrear a nossa aplicação mesmo depois dela ser considerada depurada e em sua versão final.
3. Terceiro, que para sistemas críticos no tempo, temos que considerar a hipótese de ter de corrigir um software remotamente.

26

O tratamento de interrupções no WNT

IRQL	Função
31	Interrupção de erro de hardware (NMI)
30	Falha de alimentação
29	Comunicação entre processadores
28	Interrupção de Clock
27 12	Interrupção de Hardware IRQL 15 Interrupção de Hardware IRQL 0
11 4	Livre
3	Interrupção do depurador de software
2 1 0	Interrupções de SW para sincronização do sistema (priorização de trabalho em *device drivers* e componentes do executivo)

27

Device Drivers

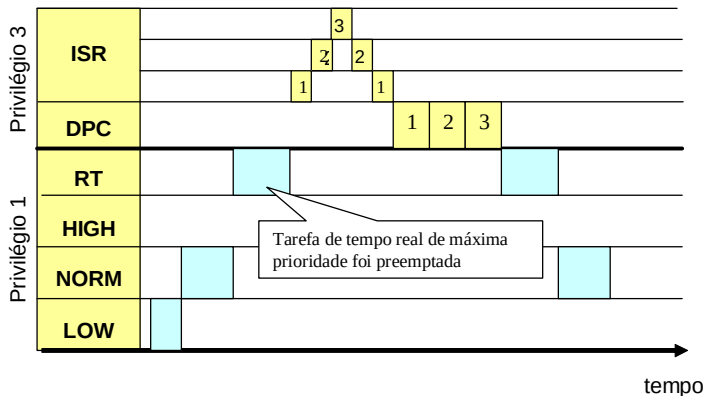
Um *device driver* no WNT possui quatro componentes:

Rotina de inicialização	Inicializa o hardware e define as estruturas de dados a serem usadas pelo driver
Rotina de serviço de interrupção (ISR)	Trata a interrupção do dispositivo controlado
Chamada adiada a procedimento (DPC – *Deferred Procedure Call*)	Realiza atividades menos críticas no tratamento da interrupção.
Thread do sistema	Alguns drivers completam suas atividades utilizando uma thread de menor nível de prioridade.

- Uma DPC não pode executar instruções de wait e deve ter toda a memória de que necessita alocada a priori na memória física, com paginação desabilitada
- Esta regra em geral é recomendada a toda tarefa de tempo real. Faltas de página trazem um *overhead* desaconselhável a qualquer aplicação *hard real time*

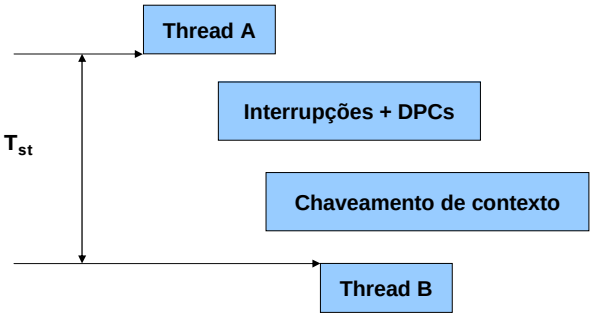
28

Atividade de rotinas de interrupção e DPCs



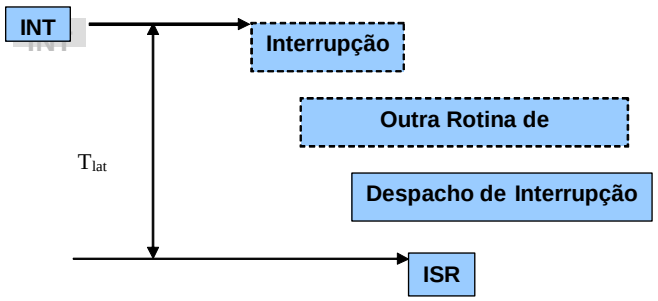
29

Análise de Desempenho
Latência de escalonamento / latência de troca de contexto *(Scheduling Latency / Thread Switch Latency)*



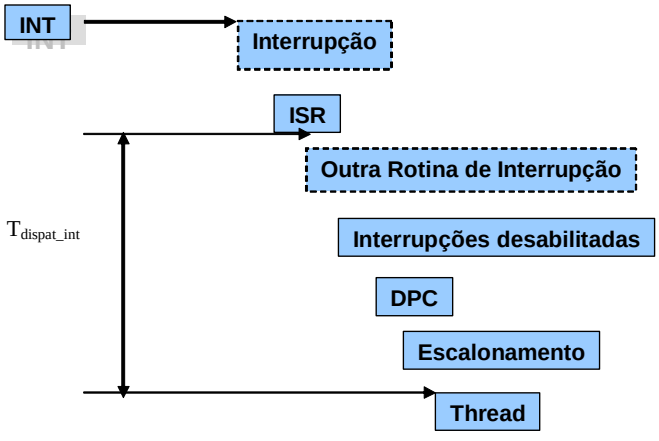
30

Análise de Desempenho
Tempo de latência de interrupção *(Interrupt latency)*



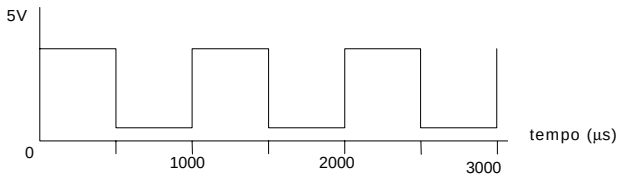
31

Tempo de latência de despacho de interrupção *(Interrupt Dispatch Latency)*



32

Tempo de resposta do kernel: atividades cíclicas



- Uma tarefa de tempo real de alta prioridade é colocada para ativar uma saída para o valor um durante um tempo definido, digamos 500µs ou na máxima velocidade possível. Depois deste tempo ela deve forçar a mesma saída para zero. O que se testa é a capacidade de uma dada tarefa de repetir um objetivo temporal especificado exercitando chamadas básicas do kernel (timer + I/O). As larguras de pulso são medidas para se verificar o desvio em ralação ao valor projetado. Quanto maior a variação (*jitter*), pior o determinismo do sistema. O sistema é submetido a carga externa de diversas naturezas para exame do seu comportamento. Interrupções, DPCs e tarefas de maior prioridade são as causas da variabilidade dos tempos de resposta.
- Este tipo de medida pode ser feito com um instrumento de medição externo, o que traz a vantagem de não utilizar nenhum recurso de software do sistema sob teste, o que perturbaria a medida.

33

Características de um RTOS

Sistemas Soft Real Time:

- SCADA
- MES
- PIMS
- LIMS

34

Características de um RTOS

1. Ser multihreaded e preemptivo.
2. Além de possuir threads, cada thread deve ter uma prioridade exclusiva. Quanto maior o número de prioridades disponíveis, melhor. É tarefa do projetista converter os requisitos de *deadline* das tarefas em prioridades das threads. Quando duas threads possuem a mesma prioridade, elas executam em regime de *time slicing*, prejudicando a condição de preemptividade.
3. Possuir mecanismos determinísticos de sincronização de threads
4. Estes mecanismos incluem soluções para o problema da exclusão mútua, gerenciamento de eventos, comunicação entre threads, etc.
5. Possuir um sistema de herança de prioridades para todos os seus objetos de sincronização.
6. O objetivo aqui é prevenir a inversão de prioridades.
7. Ter um comportamento temporal previsível, apresentando um limite superior conhecido: tempo de latência de interrupção, tempo de latência de escalonamento, etc. É fundamental saber quando um pedido, uma vez colocado, será efetivamente atendido.
8. Ser capaz de temporizar atividades assíncronas em relação ao clock da CPU na faixa de sub milisegundo e escalonar tarefas periódicas nesta base de tempo.

Algumas aplicações requerem o uso de temporizadores precisos, com boa repetibilidade de comportamento. Na ocorrência de uma interrupção de *timer,* uma ISR será escalonada, ou uma thread despertada para realizar alguma atividade crítica.

35

O Windows NT como RTOS

1. O Windows NT é multithreaded e preemptivo. As threads da classe de tempo real não podem ter sua prioridade modificadas pelo sistema operacional e seriam adequadas para uso em aplicações *hard real time*. A grande limitação é que existem apenas 5 níveis de prioridades para estas threads (7 se considerarmos os dois níveis extremos). Isto é muito pouco para aplicações reais. Também, como vimos, mesmo uma thread de prioridade 31 pode ser preemptada pelas rotinas de interrupção e DPCs o que prejudica a previsibilidade do seu comportamento.
2. O WNT possui mecanismos sofisticados de sincronização entre threads. Uma limitação é que as filas de espera dos objetos como Mutex são do tipo FIFO, portanto sem prioridade.
3. O Windows NT não implementa herança de prioridades.
4. O comportamento temporal do WNT é considerado bom para aplicações não críticas, mas apresenta grande variabilidade (desvio padrão).
5. O mecanismo de temporização denominado *timers multimídia* no WNT depende da implementação dos timers no HAL. A interrupção de clock pode ser gerada por um temporizador assíncrono externo ou por um RTC que fornece interrupções síncronas com o clock do micro. Depois de receber a interrupção, o HAL chama o kernel que decrementa os temporizadores pendentes e acorda os processos enfileirados, quando o tempo programado espira. Medidas efetuadas no range de 1ms [Jones & Regehr 98] demonstram uma grande variabilidade no valor obtido, que muitas vezes chega a ser o dobro do esperado (2ms). Os estudos realizados por Bruno Bittencourt [Bittencourt 00] com os sistema operacionais WNT e Windows 98, demonstraram que os demais processos de temporização, incluindo os *waitable timers,* apresentavam problemas semelhantes. Este trabalho utilizou o relógio interno do Pentium como referência. Um estudo mais conclusivo deveria utilizar um dispositivo temporizador externo, como por exemplo, um analisador de barramento.

36

Como melhorar a performance do WNT em aplicações críticas ?

Uma forma de fazer um uso mais otimizado do WNT seria observar os seguintes critérios:

- Executar as aplicações críticas como threads da classe real time.
- Forçar todo o código da aplicação a ficar residente na memória física para evitar a ocorrência de *page faults* (instrução *VirtualLock*)
- Manter as rotinas de interrupção (ISR) pequenas.
- Homologar todo o hardware e *device drives* utilizados para prevenir aplicações que desabilitem interrupções ou que executem muito tempo em nível privilegiado (ISRs e DPCs longas). Os maiores responsáveis por DPCs longas são: drives de disco, drives de rede e drives de vídeo.
- Fazer uso do cache. O WNT tenta executar uma thread no mesmo processador onde executou por último para tirar proveito do princípio da localidade de referência.

37

Extensores de Tempo Real

- O WNT não satisfaz as aplicações *hard real time*. Os programadores deste tipo de software tinham como únicas opções escolher um RTOS que melhor se adequasse aos seus projetos, usar uma API Win32 sobre um RTOS de mercado [Hildebrand 97a], ou fazer coexistir o NT com um RTOS, se possível sem modificá-lo. Entretanto a alternativa NT parecia muito atrativa.

Trata-se de uma commodity que oferece inúmeras vantagens:

1. Grande número de aplicativos existentes para esta plataforma.
2. Uso de uma API consagrada: Win32.
3. Grande aceitação por parte do mercado em geral.
4. Existência de uma suite de aplicativos que constituem o pacote Microsoft Office com grande sinergia em relação a aplicações de controle de processos. Hoje a maior parte dos usuários prefere gerar os seus próprios relatórios utilizando o Excel como ferramenta de formatação ao invés de geradores de relatórios dedicados.
5. Existência de um grande repertórios de drives para os principais periféricos do mercado: disco rígido, impressoras, monitores, etc. e para equipamentos industriais: CLPs, balanças, leitores de código de barras, etc.
6. Baixo custo.
7. Disponibilidade de um grande número de compiladores para as principais linguagens utilizadas.
8. Etc.

38

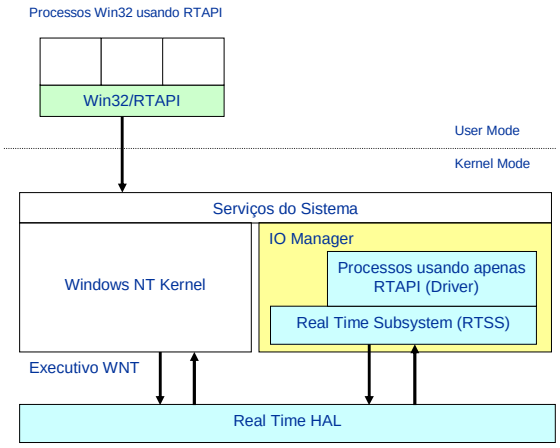
Extensores de Tempo Real
Pontos Negativos

1. Uso não previsto do HAL.
2. Problema de compatibilidade quando a Microsoft modifica o HAL.
3. Perda de confiabilidade, primeiro devido ao fato de se fazer o WNT funcionar fora de suas especificações de projeto, depois por inserir código que aumenta a complexidade do sistema como um todo. Os críticos dos extensores argumentam que dificilmente o número de homens hora despendidos no teste destas soluções é inferior ao que a Microsoft investiu.
4. Não portabilidade dos dois ambientes. O programador deve utilizar um conjunto diferente de instruções quando estiver codificando uma aplicação de tempo real. Isto também inclui o uso de um segundo sistema de desenvolvimento: compiladores, depuradores, etc.
5. Uso de uma biblioteca proprietária do lado do extensor, o que exige comprometimento com o fornecedor. O Windows NT é considerado um sistema aberto no sentido de que a documentação de apoio disponível a desenvolvedores é muito extensa.
6. Falta de reconhecimento e comprometimento formal da Microsoft com os fornecedores de extensores. A Microsoft não garante manter a compatibilidade e características de nenhum componente de seu software a fim de garantir a permanência destes produtos no mercado.
7. Falta de conexão entre a parte NT e o extensor. As aplicações de tempo real não podem fazer uso do COM (*Componet Object Model*), Active X, Direct X [Malina 97] e consequentemente dos drives OPC (*OLE for Process Control*) que tendem a se tornar padrões na indústria.

39

Extensores de Tempo Real
Principais Produtos

- **RTX 4.2 da VenturCom**
- **INTime 1.20 da Radisys**
- **Hyperkernel 4.3 da Imagination Systems**



40

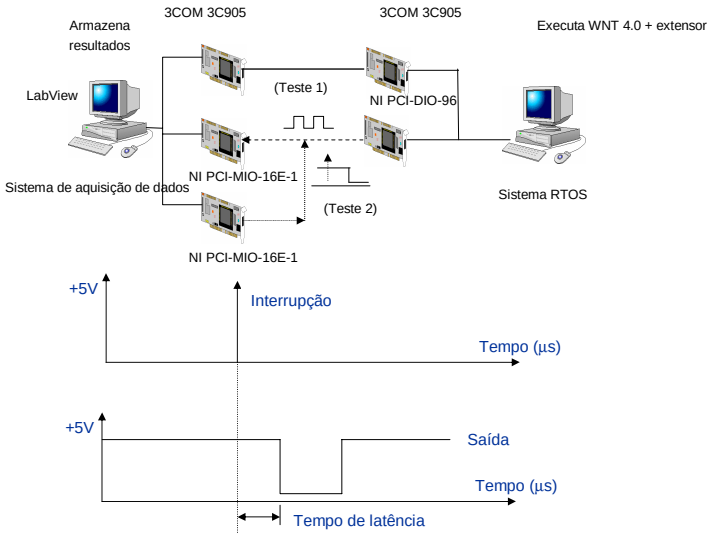
Desempenho dos extensores

	QNX 4.2[1]		WNT[2]		RTX[2]		INTime[2]		Hyperkernel[2]	
Indicador (µs)	T_{med}	T_{max}	T_{med}	T_{max}	T_{med}	T_{max}	T_{med}	T_{max}	T_{med}	T_{max}
Latência de interrupção	2.0	3.6	3.9	22.8	8.6	14.8	11.0	19	7.4	18.9
Latência de despacho de interrupção	2.1	3.2	10.8	29.0	5.3	10.5	25.9	39	6.3	18.4
Latência de escalonamento	1.2	8.0	5.5	12.7	3.8	9.3	4.5	24.2	2.0	4.8
Fonte	[DSEd 01]		[RTM 98b] (NT) e [DSEc 01] (Extensores)							

Plataforma: 1: Pentium 200Mhz
2: Pentium 200Mhz MMX

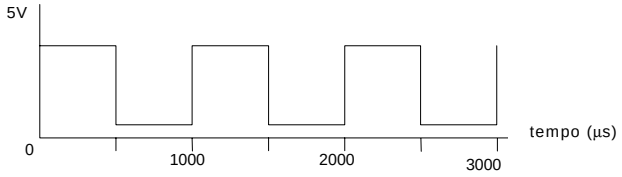
41

Relatórios da GM-Powertrain



42

Tempo de resposta do kernel: atividades cíclicas



- Uma tarefa de tempo real de alta prioridade é colocada para ativar uma saída para o valor um durante um tempo definido, digamos 500µs ou na máxima velocidade possível. Depois deste tempo ela deve forçar a mesma saída para zero. O que se testa é a capacidade de uma dada tarefa de repetir um objetivo temporal especificado exercitando chamadas básicas do kernel (timer + I/O). As larguras de pulso são medidas para se verificar o desvio em ralação ao valor projetado. Quanto maior a variação (*jitter*), pior o determinismo do sistema. O sistema é submetido a carga externa de diversas naturezas para exame do seu comportamento. Interrupções, DPCs e tarefas de maior prioridade são as causas da variabilidade dos tempos de resposta

- Este tipo de medida pode ser feito com um instrumento de medição externo, o que traz a vantagem de não utilizar nenhum recurso de software do sistema sob teste, o que perturbaria a medida.

43

Relatórios da GM-Powertrain

Dell XPS H266 128M RAM	NT Nativo	ISI HyperKernel	Radisys INTime	VenturCOM RTX
Média	999.95	499.48	499.48	499.48
Máximo	1711.10	505.70	521.45	508.10
Mínimo	238.05	493.70	486.10	490.75
Desvio padrão	45.72	0.78	1.30	0.38
Amostragens	500 000	500 000	500 000	500 000

Resultado ideal para largura do pulso: 500µs

Nematron PentiumPro 200 32MRAM	NT Nativo	ISI HyperKernel	Radisys INTime	VenturCOM RTX
Média	999.98	499.49	499.49	499.49
Máximo	3141.65	507.20	540.10	507.00
Mínimo	271.05	489.55	450.80	492.00
Desvio padrão	59.42	0.80	1.24	0.60
Amostragens	250 000	500 000	500 000	500 000

44

Windows CE – Características Básicas

- O Windows CE é um sistema operacional multithreded de 32 bits derivado do Windows 95 e NT.
- Mercado alvo: aplicações embedded e uso em computadores de pequeno porte mais especificamente o Pocket PC e computadores de bordo (Auto PC) que passarão a equipar nossos carros no futuro.
- A Microsoft define que o Windows CE será o seu sistema operacional para as aplicações de tempo real, inclusive as aplicações de telefonia e telecomunicações.
- A versão 3.0 veio a corrigir os maiores problemas da versão 2.1 e colocou o Windows CE no mesmo nível dos sistema operacionais mais especializados em aplicações embedded [Van Beneden 01].

45

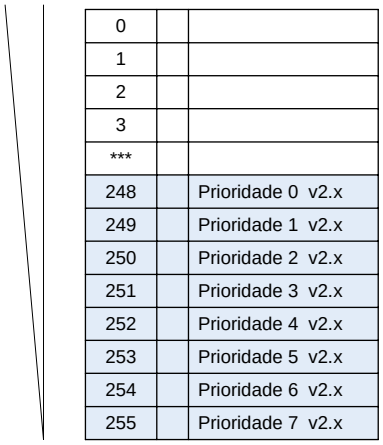
Características para aplicações de tempo real:

0		THREAD_PRIORITY_CRITICAL	Real time & Device Drives
1		THREAD_PRIORITY_HIGHEST	
2		THREAD_PRIORITY_ABOVE_NORMAL	Kernel
3		THREAD_PRIORITY_NORMAL	
4		THREAD_PRIORITY_BELOW_NORMAL	
5		THREAD_PRIORITY_LOWEST	Outras aplicações
6		THREAD_PRIORITY_ABOVE_IDLE	
7		THREAD_PRIORITY_IDLE	

Prioridades de threads no Windows CE versão 2.x

46

Prioridades de threads no Windows CE versão 3.0



0		
1		
2		
3		
***		
248		Prioridade 0 v2.x
249		Prioridade 1 v2.x
250		Prioridade 2 v2.x
251		Prioridade 3 v2.x
252		Prioridade 4 v2.x
253		Prioridade 5 v2.x
254		Prioridade 6 v2.x
255		Prioridade 7 v2.x

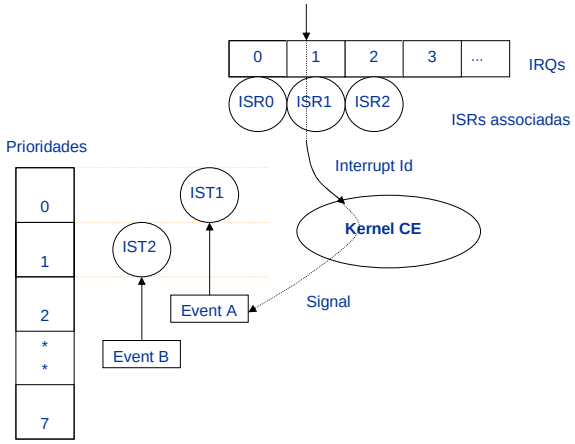
47

Características para tempo real

- O Windows CE 3.0 evita a inversão de prioridades através da implementação da herança de prioridades
- Em sua versão 3.0 o Windows CE apresenta quatro objetos de sincronização:
  - CriticalSections
  - Eventos nomeados e não nomeados
  - Mutexes
  - Semáforos contadores
- O valor do *tick* de 1 ms permite uma menor granularidade das operações de temporização (função Sleep, WaitForSingleObject, etc.)
- Ao contrário do Windows NT, no Windows CE a fila de espera por um objeto de sincronização é organizada como uma FIFO com prioridade. Assim não ocorre inversão de prioridades na espera de um recurso

48

Windows CE - Interrupções



49

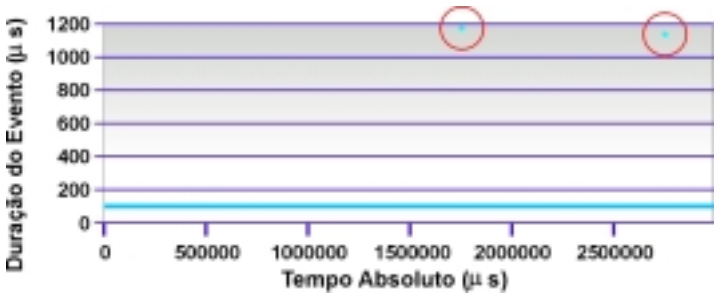
Windows CE - Interrupções

- O serviço das interrupções é baseado em Rotinas de Tratamento de interrupções (ISR) e em threads especiais denominadas IST (*Interrupt Service Threads*). O objetivo é manter as ISRs o menor possível para se obter tempos de latência mais baixos
- Cada IRQ (*Intrerupt Request Line*) está associada a uma ISR específica. Quando a interrupção ocorre, a ISR correspondente é acionada. Esta rotina executa em modo kernel e é muito breve, retornando ao kernel a identificação da interrupção (Interrupt Id)
- O kernel sinaliza um evento que está associado a uma IST. A IST deve estar esperando por este evento. Ela será então acordada para execução de acordo com sua prioridade
- Como as ISTs são associadas aos níveis máximos 0 ou 1 elas serão rapidamente escalonadas para execução. Assim que a IST é iniciada, o sistema de interrupções está habilitado a aceitar e tratar outro pedido de interrupção
- O Windows CE 3.0 já permite interrupções aninhadas, o que constituía uma das grandes limitações da versão anterior
- Na versão 2.1, cada ISR executava até completar e disparar a IST. Isto prejudicava o tempo de latência das interrupções de mais alto nível. O tempo de latência de interrupção da versão 3.0 já é comparável ao de outros RTOS

50

Desempenho do Windows CE
Tempo de latência de criação

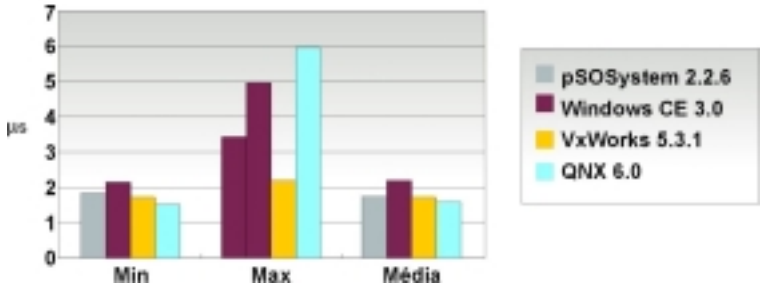
- Consta da criação repetida de uma thread por milhares de ciclos
- O tempo médio de criação foi de 100µs. O tempo máximo foi de 1µs, verificado em apenas 0.01 % dos casos (2 casos)



51

Desempenho do Windows CE
Tempo de latência de Interrupção

- O resultado para o tempo de latência foi compatível com o de outros RTOS do mercado

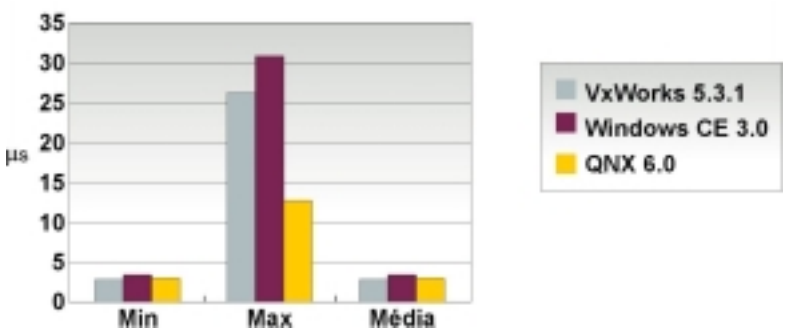


O que falta de informação neste gráfico segundo sua opinião ?

52

Tempo de latência de escalonamento

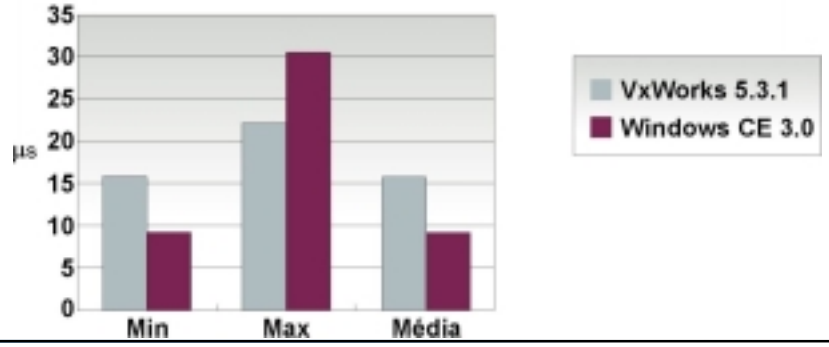
- Dez threads de igual prioridade pertencentes ao mesmo processo executam. Quando a thread se torna ativa, ela abandona o processador em prol da próxima thread da fila. O tempo de latência se mostrou independente do número de threads utilizado. O tempo máximo de 32µs foi observado quando uma thread se tornava ativa pela primeira vez em uma série.



53

Inversão de prioridade

- Foi medido o tempo desde o momento em que uma thread de alta prioridade solicitava um recurso compartilhado, utilizando um Mutex, até o momento em que a thread era escalonada para execução.
- Como o Windows CE 3.0 usa um mecanismo de herança de prioridade, isto envolve elevar a prioridade da thread de menor prioridade, esperar que ela termine de executar e chavear o contexto para a thread de maior prioridade. Os resultados foram satisfatórios



54

Performance do Windows CE versão 3.x

	QNX 6.1[1]		Windows CE 3.0[1]	
Indicador (µs)	T_{med}	T_{max}	T_{med}	T_{max}
Tempo de latência de interrupção	1.7	4.1	2.4	5.1
Tempo de latência de escalonamento	1.8	2.1	-	-
Tempo de troca de contexto	2.0	8.1	2.5	29.5
Fonte	[DSEb 01]			

Plataforma: [1]: Pentium 200Mhz MMX, placa mãe Chaintech

55

Polling x Interrupções

Sistemas Assíncronos:

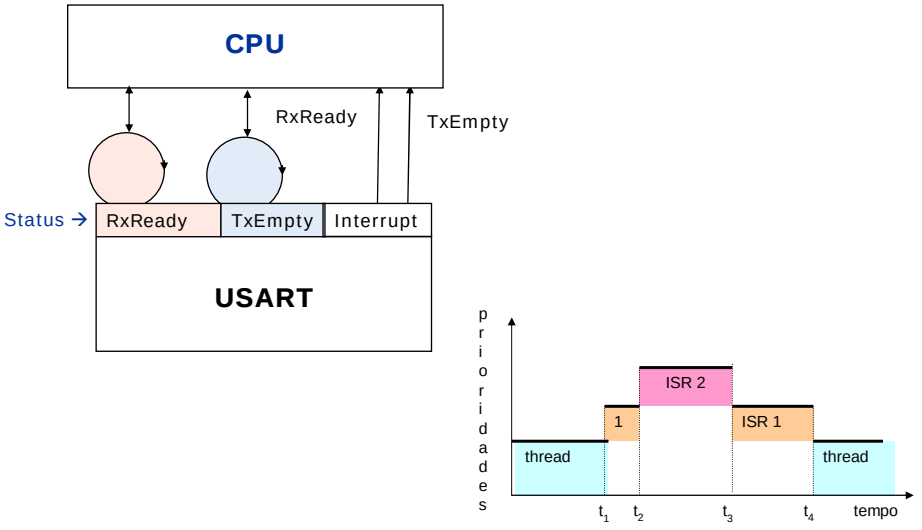
- A operação de I/O corresponde a uma thread assíncrona de alta prioridade.
- Quando o dispositivo está pronto para disponibilizar os dados, a tarefa é escalonada preemptivamente.
- A tarefa de I/O especial recebe nome de tratador de interrupções *(Interrupt Handler)* ou ISR (*Interrupt Service Routine*).
- A ISR, como já vimos, deve ser o mais breve possível.

Sistemas Síncronos (Polling):

- O *device driver* lê os registros de *status* do dispositivo de entrada periodicamente até detectar o evento esperado. O tempo de polling pode ser fixo, sendo ditado por uma interrupção de relógio, ou variável. Neste último caso a tarefa de amostragem é executada ciclicamente em sua velocidade máxima.
- O dispositivo externo deve reter o dado até a leitura pelo driver. Caso outro dado se torne disponível, antes que o anterior seja lido, o dado mais velho será destruído (erro de *overrun*), a menos que o dispositivo tenha alguma capacidade de armazenamento temporário (buffer).

56

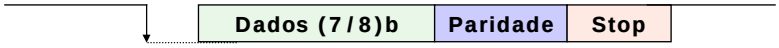
Polling x Interrupções Exemplo



57

Interrupções são sempre melhores que leitura cíclica (*polling*) ?

Comunicação serial:



Transmissão:

1. O *device driver* recebe mensagem solicitando a transmissão de um bloco de dados.
2. Os dados são copiados para um buffer interno do driver.
3. Os dados são pré processados. O driver pode inserir caracteres de controle de erro especiais como 2 ou 4 bytes de CRC (*Cyclic Redundant Code*) ou *check sum*. Alguns caracteres especiais como *Record Separator* (0x1E) do Unix podem ser expandidos para CR/LF de uso mais geral, etc.
4. O primeiro byte é carregado na UART.
5. O driver entra em espera e fica bloqueado. O S.O. carrega outra thread para executar.
6. LOOP:
   - 6.1. A UART transmite o dado e ativa o bit TxReady em uma palavra de status (flag), ou causa uma interrupção de TxReady.
   - 6.2. Ao receber a interrupção, a ISR do driver do dispositivo é ativada.
   - 6.3. A ISR verifica se existem mais dados para serem transmitidos e neste caso copia mais um byte no buffer da UART.

58

Comunicação Serial

Recepção:

1. A UART recebe um byte bit a bit e gera uma interrupção, ou sinaliza uma flag (RxReady)
2. A ISR recebe o byte e o coloca em um buffer interno do drive.

Toda vez que o usuário solicitar dados ao driver, os dados do buffer serão entregues. Estes dados também podem ser recebidos até algum caracter especial de final de bloco ser encontrado, ou a cada t ms.

59

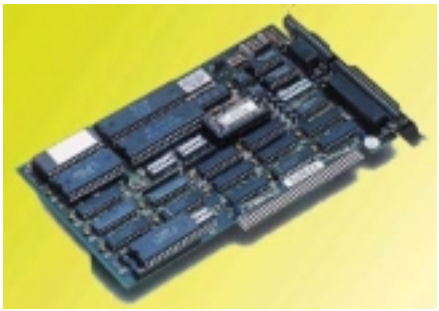
Análise de desempenho:

- Vamos assumir que a UART foi configurada para:
- 10 bits/caracter: 7 bits de dado, 1 bit de paridade, 1 stop bit e 1 start bit
- velocidade = 9600 bps = 960 caracteres/s
- Isto representa uma carga de 960 interrupções por segundo. Se a linha for full duplex, o computador irá receber 1920 interrupções por segundo !
- Esta carga representa um grande overhead para a CPU principal. Por outro lado se sabemos que a comunicação se dará todo o tempo em sua taxa máxima, seria mais eficiente se buscássemos os caracteres, por exemplo, uma vez a cada 50 ms. O número de interrupções cairia de 9600 para 20. Mas isto implica em ter que manipular 48 caracteres por transação.

60

Análise de desempenho:

- Para se obter um bom desempenho na transmissão serial é fundamental utilizar UARTS com buffers internos de vários caracteres
- Se a taxa de recepção for fixa e alta, polling será a melhor estratégia. Se os caracteres chegam esporadicamente, então a interrupção é o critério de escolha.



61

Exercícios

1) Marque os sistemas que na sua opinião são de tempo real:

( ) Marcapasso cardíaco de demanda (emite pulso só quando a condução do impulso elétrico da aurícula para o ventrículo falha).
( ) Sistema de reserva de passagens aéreas.
( ) Sistema de venda de livros pela Internet.
( ) Operação cardíaca a distância por controle remoto.
( ) Controlador de robot industrial numa linha de pintura.
( ) Videoconferência
( ) Sistema de controle do nível de açúcar no sangue humano.
( ) Comando remoto do robot Sojourner no solo de Marte.
( ) Sistema de monitoração de incêndio de um prédio.
( ) Sistema de monitoração de variáveis clínicas dos pacientes de um hospital.
( ) Sistema de controle de vôo de um aeroporto.
( ) Sistema de controle de abertura da pupila no ser humano.

62

Exercícios

2) De o valor verdade das seguintes afirmativas:

( ) Devemos concentrar todo o processamento do nosso *device driver* na rotina de interrupção (ISR).

( ) O kernel pode despachar três tipos de rotinas: ISRs, DPCs e threads do sistema e do usuário.

( ) Uma DPC de alta prioridade sempre preemptará uma DPC de baixa prioridade.

( ) Para que o WNT se comportasse como um RTOS seria desejável que todo o processamento da interrupção fosse feito num determinado nível de prioridade.

( ) O WNT permite interrupções aninhadas.

( ) DPCs têm prioridade mais alta que as threads do sistema

( ) Um *device driver* mal comportado, ligado a uma linha de interrupção de baixa prioridade, pode afetar o comportamento de um *device driver* ligado a uma IRQL de alta prioridade.

( ) ISRs e DPCs executam em modo privilegiado (ring 0) e portanto podem executar instruções privilegiadas tais como CLI (desabilita interrupções) ou STI (habilita interrupções).

( ) No Windows CE um processo pode ter uma thread na prioridade 0 e outra na prioridade 50.

63

Exercícios

3) Classifique as aplicações abaixo em *Hard Real Time* (H) ou *Soft Real Time* (S):

( ) Software de geração da folha de pagamentos

( ) Injeção eletrônica microprocessada

( ) Sistema de gerenciamento de Manutenção (MMS)

( ) Sistema SCADA

( ) Sistema de suspensão ativa de um carro de Fórmula I

( ) Sistema de gerenciamento de armazéns – WMS (Warehouse Management System)

( ) Sistema de arquivamento de dados históricos - PIMS (*Plant Information Management System)*

( ) Sistema de acompanhamento da manufatura – MES (*Manufacturing Execution System*)

( ) Sistema de controle baseado em SoftLogic

( ) Processamento de áudio e vídeo

64

Muito Obrigado

Perguntas?

Constantino Seixas Filho

constantino.seixas@task.com.br