|  |  |
| --- | --- |
| D:\imagens\design\logos\uern.png | Governo do Estado do Rio Grande do Norte  Secretaria de Estado da Educação, da Cultural e dos Desportos – SECD  UNIVERSIDADE DO ESTADO DO RIO GRANDE DO NORTE – UERN  Pró-Reitoria de Ensino e Graduação – PROEG  Ciências da Computação  **Sistemas de Tempo Real – Tópicos de aula** |

**1. Introdução aos sistemas de tempo real**

1.1 Aspectos gerais

* Proliferação do uso dos sistemas computacionais na sociedade ⭢ aparecimento de diversas aplicações com requisitos de tempo real
* Aplicações variam quanto a complexidade
  + Sistemas simples
    - Controladores inteligentes embutidos em aparelhos de utilidades domésticas (lavadoras, micro-ondas, aparelhos de DVD...)
  + Sistemas complexos
    - Sistemas militares de defesa
    - Sistemas de controle de plantas industriais
    - Controle de tráfego aéreo/ferroviário
* Algumas aplicações apresentam restrições de tempo mais rigorosas
  + Aplicações rigorosas
    - Monitoramento de pacientes em hospitais
    - Sistemas de supervisão e controle
    - Sistemas embarcados (robôs, aviões...)
  + Aplicações menos rigorosas
    - Videogame
    - Videoconferência
    - Aplicações multimídia
* Metodologias e ferramentas convencionais nem sempre garantem que os requisitos temporais sejam facilmente implementados
  + Os sistemas operacionais ou núcleos de tempo real que gerenciam interrupções e tarefas permitem então a programação de temporizadores que, na maioria dos casos, são ferramentas suficientes para a construção de STR
* As práticas de implementação atuais tem permitido atender de forma aceitável os requisitos da maioria dos STR
  + Técnicas e ferramentas convencionais apresentam limitações
* As exigências cada vez maiores sobre aspectos de segurança colocam em xeque as metodologias e ferramentas convencionais, sob pena de perdas sobre o aspecto financeiro, ambiental ou humano
* Nas aplicações mais críticas, são assumidas situações de operações extremas e pessimistas
  + Várias hipóteses são supostas
  + Consideram-se os piores cenários possíveis
  + Independente desses aspectos supostos, os STR devem atender todas as restrições temporais

1.2 Caracterização de um sistema de tempo real

* Dentre as diversas categorias de sistema, os sitemas de tempo real, de maneira geral, se encaixam na categoria de *Sistemas Reativos*
  + Sistemas reativos são aqueles em que reagem aos estímulos de entrada do ambiente, enviando respostas de maneira contínua
* **Definição:** Um *Sistema de Tempo Real* (STR) é um sistema computacional que deve reagir a estímulos oriundos do seu ambiente em prazos específicos
* O atendimento a esses prazos resulta em requisitos de natureza temporal sobre o comportamento desses sistemas
* Em cada reação o STR deve entregar um resultado correto dentro de um prazo específico, sob pena de ocorrer uma falha temporal
* O comportamento correto de um STR, portanto, não depende só da integridade dos dados obtidos (correção lógica ou *correctness*), mas também dos valores de tempo em que são produzidos (correção temporal ou *timeliness*)
* Uma reação que ocorra fora do prazo especificado pode se tornar sem utilidade ou até mesmo representar uma ameaça
* Um STR deve ser capaz de oferecer garantias de *correção temporal* para o fornecimento de todos os serviços que apresentem restrições temporais
  + Exemplo: Manuseio/Apresentação de vídeos em sistemas multimídia com áudio atrasado/adiantado com relação à imagem

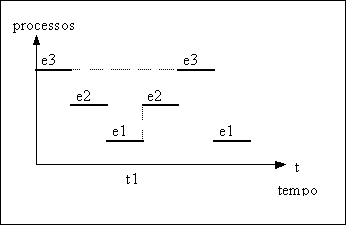
1.3 Classificação dos sistemas de tempo real

* Os STR podem ser classificados quanto ao aspecto de proteção (*safety*):
  + STR brandos (*Soft Real-Time Systems*): quando as consequências de uma falha devida ao tempo é da mesma ordem de grandeza dos benefícios do sistema
    - Exemplos: sistema de comutação telefônico, sistema de processamento bancário
    - **Também conhecidos como Sistemas Não-Críticos de Tempo Real (SNCTR)**
  + STR rígidos (*Hard Real-Time Systems*): quando as consequências de pelo menos uma falha temporal excedam em muito os benefícios normais do sistema
    - Exemplos: sistema de controle de tráfego aéreo/ferroviário, sistema de controle de planta nuclear
    - Nos casos em que há falhas em sistemas rígidos, observa-se que na maioria dos casos as falhas são catastróficas
    - **Também conhecidos como Sistemas Críticos de Tempo Real (SCTR)**
* Os SCTR podem ainda ser subdivididos em
  + SCTR seguros
    - Nesse tipo de sistema, um ou vários estados seguros podem ser alcançados na ocorrência de uma falha
    - Exemplo: parada obrigatória de trens no caso de falhas no sistema de sinalização ferroviária
  + SCTR operacionais
    - Na presença de falhas, somente parte do sistema continuará funcionando
    - Exemplo: queima do sensor do nível de combustível de um avião em pleno voo

**2. Processos**

2.1 Modelos de processos

* O modelo de processos conceitual vem sendo desenvolvido ao longo dos anos por projetistas de Sistemas Operacionais (SOs)
* Nesse modelo, todos os softwares que podem executar em um computador (inclusive, algumas vezes, o próprio sistema operacional) são organizados em **processos**
* Um processo é apenas um programa em execução acompanhado dos valores atuais do contador de programa, dos registradores e das variáveis
* A alternância entre esses processos faz com que o usuário tenha a ilusão de que os processos estão sendo executados de maneira paralela
* Dá-se a essa ilusão o nome de pseudo-paralelismo
* Dá-se o nome de **multiprogramação** ao mecanismo de trocas rápidas que realiza a alternância entre os processos
* Funcionamento do pseudo-paralelismo



* A alternância da CPU entre os processos faz com que a taxa na qual um processo realiza a sua computação não seja uniforme
  + Por esse motivo os processos não devem ser programados com hipóteses predefinidas sobre a temporização
* Quando um processo tem restrições críticas de tempo real, ou seja, quando eventos específicos devem ocorrer dentro de um intervalo de tempo prefixado (normalmente muito pequeno), deve-se tomar certos cuidados para que esses eventos ocorram
* Uma última observação a ser feita é que não se deve confundir processo com programa
  + A diferença entre os dois termos é sutil, mas crucial
  + Exemplo: Padeiro assando um bolo
    - A receita do bolo contém todos os passos necessários para que o bolo seja feito (ingredientes e modo de preparo). Obviamente, nessa analogia, a receita seria o programa
    - O processador em questão é o padeiro, que irá pegar os ingredientes (dados de entrada) e processá-los de acordo com a receita
    - O processo em si é a atividade realizada pelo padeiro de ler a receita, buscar os ingredientes e assar o bolo
    - Imagine então que a filha do padeiro entra na sala dizendo que foi picada por uma abelha. O padeiro vai anotar em que ponto da receita ele parou (salvar o estado atual do processo), buscar um livro de primeiros socorros (outro programa) e começar a seguir as instruções nele contidas
    - Observa-se nesse último caso que o processador alternou de um processo com menor prioridade (assar o bolo) para um processo com maior prioridade (fornecer cuidados médicos)
    - Quando o processo de maior prioridade encerrar, a CPU alterna novamente para o processo de assar o bolo, continuando do ponto em que parou
  + A ideia principal é que um processo constitui uma atividade. Ele possui um programa, entrada, saída e um estado. Um único processador pode ser compartilhado entre vários processos através de algum algoritmo de escalonamento

2.2 Criação de processos

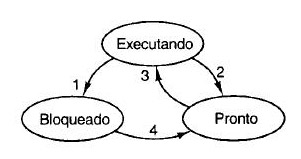
* Os SOs precisam assegurar de algum modo que todos os processos necessários à um propósito existam
* Em sistemas simples, tais como os fornos de micro-ondas, é possível criar todos os processos no momento da inicialização do sistema
* Já em sistemas complexos ou de propósito geral os SOs precisam ter mecanismos de criação e término de processos
* Há, basicamente, quatro possibilidades para criação de novos processos
  + Início do sistema
  + Execução de uma chamada de sistema de criação de processo a partir de um processo em execução
  + Requisição do usuário para criação de um novo processo
  + Início de um trabalho (*job*) em lote
* Tecnicamente, em todos esses casos, um novo processo é criado por um processo existente a partir da execução de uma chamada de sistema de criação de processos
* Nos sistemas Unix, há somente uma chamada para criar um novo processo, denominada fork
  + Depois da chamada os dois processos (pai e filho) possuem a mesma imagem de memória, as mesmas cadeias de caracteres no ambiente e os mesmos arquivos abertos
  + Normalmente o processo filho executa logo em seguida uma chamada da função execve (ou similar) para mudar a imagem de memória e executar um novo programa
* No Windows, uma única chamada de função (CreateProcess) trata tanto do processo de criação quanto da carga do programa correto no novo processo
  + Essa chamada possui 10 parâmetros, incluindo o programa a ser executado, os parâmetros da linha de comando, vários atributos de (in)segurança, bits de controle de arquivos abertos, informações de prioridade, especificações de janela, dentre outros
* Tanto no Unix quanto no Windows, após a criação de um processo, o pai e o filho possuem seus próprios espaços de endereçamento
  + Se um dos dois alterar uma palavra em seu espaço de endereçamento, essa mudança não será visível ao outro processo
  + No Unix, o espaço de endereçamento inicial do filho é uma cópia do espaço de endereçamento do pai, mas há dois espaços de endereçamento distintos
    - Nenhuma memória para escrita é compartilhada
    - Algumas implementações compartilham o código do programa entre os dois (já que não podem ser alterados)
  + É possível que os processos criados filhos compartilhem com seus pais alguns recursos, tais como os arquivos abertos
  + No Windows os espaços de endereçamento são diferentes desde o início

2.3 Término de processos

* Após terminar seu trabalho, um processo será encerrado em virtude de alguma das seguintes condições
  + Saída normal (voluntária)
  + Saída por erro (voluntária) – Normalmente há exibição de avisos
  + Erro fatal (involuntário)
  + Cancelamento por um outro processo (involuntário)
* Na maioria das vezes os processos terminam porque fizeram seu trabalho
* A terceira razão para o término de um processo é um erro causado pelo processo, muitas vezes por um erro de programação
  + Entre os vários exemplos estão a execução de uma instrução ilegal, referência a regiões de memória inexistentes ou a divisão por zero
  + Em alguns sistemas o programador pode fazer com que tais erros sejam tratados no próprio processo
* A quarta razão ocorre, por exemplo, quando a função kill é chamada por um processo nos sistemas Unix. No caso do Windows, a função chamada seria a TerminateProcess
  + Em ambos os casos, o processo que fizer a chamada deve ter a autorização necessária para fazê-la

2.5 Estados de processos

* Embora cada processo seja uma entidade independente, com seu próprio contador de programa e estado interno, muitas vezes os processos precisam interagir com outros
* Quando essa interação é impossibilitada por algum motivo, o processo em questão bloqueia e espera até que as informações possam ser lidas
  + Exemplo: $cat capitulo1 capitulo2 capitulo3 | grep tree
* Um processo bloqueia, obviamente, porque não pode prosseguir
  + Porque está esperando alguma entrada que ainda não está disponível
  + Porque o SO decidiu alocar a CPU para um outro processo, mesmo estando o processo bloqueado conceitualmente pronto e capaz de executar
  + Observe que essas duas situações são completamente diferentes
* Estados possíveis de um processo



* As transições possuem o seguinte significado
  1. Processo em execução bloqueia aguardando uma entrada
  2. O escalonador seleciona um outro processo para execução
  3. O escalonador seleciona esse processo para execução
  4. A entrada torna-se disponível
* Exemplo: Player de vídeo + Leitura de DVD

**3. Threads**

3.1 Modelo clássico

3.2 Implementação de threads no espaço do usuário

3.3 Implementação de threads no núcleo

3.4 Ativações do escalonador

**4. Comunicação entre processos**

4.1 Condições de corrida

4.2 Regiões críticas

4.3 Exclusão mútua e espera ociosa

4.4 Dormir e acordar

4.5 Semáforos

4.6 Mutex

4.7 Monitores

4.8 Troca de mensagens

4.9 Barreiras

**5. Escalonamento**

5.1 Introdução ao escalonamento

5.2 Escalonamento em sistemas em lote

5.2 Escalonamento em sistemas interativos

5.2 Escalonamento em sistemas de tempo real

5.3 Escalonamento de threads

6. Problemas clássicos

8. Modelagem de sistemas de tempo real por Redes de Petri

9. Desenvolvimento de sistemas de tempo real em Qt