Métodos de Sincronização

Eduardo Ferreira dos Santos

Ciência da Computação
Centro Universitário de Brasília – UniCEUB

Maio, 2017

Sumário

1. Sistemas multiprogramáveis
2. Mecanismos de sincronização
3. Modelos de sistema de tempo real

1 Sistemas multiprogramáveis

2 Mecanismos de sincronização

3 Modelos de sistema de tempo real

Programação concorrente

- Na programação concorrente existe mais de uma tarefa sendo executada ao mesmo tempo. Ex.: Fatorial
- No caso de múltiplas tarefas é necessário haver comunicação entre elas.

Memória compartilhada As tarefas compartilham área de memória;

Troca de mensagens Sinais trocados entre processos.

Exclusão mútua [Tanenbaum and Machado Filho, 1995]

- Solução: impedir que mais de um processo acesse o dado ao mesmo tempo.
- Deve ser executada somente quando um dos processos estiver acessando o recurso compartilhado;
- A parte do código onde o acesso ao recurso é feito é chamada de região crítica.

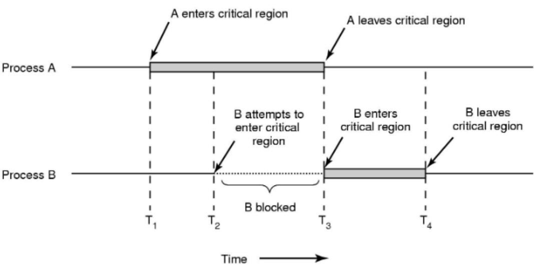

Figura 1.1: Região crítica

Semáforo Arduíno

```
/*
 * Example to demonstrate thread definition, semaphores, and thread sleep.
 */
#include <NilRTOS.h>

// The LED is attached to pin 13 on Arduino.
const uint8_t LED_PIN = 13;

// Declare a semaphore with an inital counter value of zero.
SEMAPHORE_DECL(sem, 0);
//------------------------------------------------------------------------------
/*
 * Thread 1, turn the LED off when signalled by thread 2.
 */
// Declare a stack with 128 bytes beyond context switch and interrupt needs.
NIL_WORKING_AREA(waThread1, 128);

// Declare the thread function for thread 1.
NIL_THREAD(Thread1, arg) {
  while (TRUE) {

    // Wait for signal from thread 2.
    nilSemWait(&sem);

    // Turn LED off.
    digitalWrite(LED_PIN, LOW);
  }
}
```

Semáforo Arduíno II

```
//------------------------------------------------------------------------------
/*
 * Thread 2, turn the LED on and signal thread 1 to turn the LED off.
 */
// Declare a stack with 128 bytes beyond context switch and interrupt needs.
NIL_WORKING_AREA(waThread2, 128);

// Declare the thread function for thread 2.
NIL_THREAD(Thread2, arg) {

  pinMode(LED_PIN, OUTPUT);

  while (TRUE) {
    // Turn LED on.
    digitalWrite(LED_PIN, HIGH);

    // Sleep for 200 milliseconds.
    nilThdSleepMilliseconds(200);

    // Signal thread 1 to turn LED off.
    nilSemSignal(&sem);

    // Sleep for 200 milliseconds.
    nilThdSleepMilliseconds(200);
  }
}
```

Semáforo Arduíno III

```
//------------------------------------------------------------------------------
/*
 * Threads static table, one entry per thread.  A thread's priority is
 * determined by its position in the table with highest priority first.
 *
 * These threads start with a null argument.  A thread's name may also
 * be null to save RAM since the name is currently not used.
 */
NIL_THREADS_TABLE_BEGIN()
NIL_THREADS_TABLE_ENTRY("thread1", Thread1, NULL, waThread1, sizeof(waThread1))
NIL_THREADS_TABLE_ENTRY("thread2", Thread2, NULL, waThread2, sizeof(waThread2))
NIL_THREADS_TABLE_END()
//------------------------------------------------------------------------------
void setup() {
  // Start Nil RTOS.
  nilSysBegin();
}
//------------------------------------------------------------------------------
// Loop is the idle thread.  The idle thread must not invoke any
// kernel primitive able to change its state to not runnable.
void loop() {
  // Not used.
}
```

UniCEUB Centro Universitário de Brasília

Troca de mensagens

- Somente entre dois processos;
- Os processos precisam ter a execução sincronizada.

Figura 1.2: Troca de mensagens [Chagas, 2016]

Observações

- Qualquer programa concorrente pode ser implementado com qualquer uma das duas técnicas:
  1. Memória compartilhada;
  2. Troca de mensagens.
- No quesito tempo, em geral o mecanismo de memória compartilhada é mais eficiente [FARINES and MELO, 2000];
- Programar a sincronização e comunicação das tarefas garante a corretude lógica, mas não garante a corretude temoral;
- A corretude temporal depende de:
  - Escalonamento das tarefas;
  - Capacidade do hardware de cumprir os requisitos temporais.

1 Sistemas multiprogramáveis

2 Mecanismos de sincronização

3 Modelos de sistema de tempo real

Tratadores, sensores e atuadores

- Para garantir o atendimento das restrições temporais normalmente é necessário algum tipo de componente especial;
- Utilização de tratadores de dispositivos (*device drivers*);
- Sistemas embarcados (*Embedded Systems*): integração entre a aplicação e os periféricos utilizados;
- Tratadores de interrupção: altera o mecanismo de interrupção para garantir o atendimento das restrições temporais;
- Isolar um estímulo externo que vai iniciar uma tarefa: sensores;
- Dado um estímulo ou um comando, o atuador executa algum tipo de tarefa, principalmente interagindo com o meio. Ex.: Portas e comportas.

Temporizadores

- Definição: mecanismo de contagem do tempo.
- Sistemas de tempo real quase sempre precisam realizar tarefas que envolvem a leitura do tempo;
- Mecanismo implantado no hardware que gera uma interrupção com base no tempo: temporizador de hardware;
- Quando o temporizador de hardware gera uma interrupção o STR precisa atualizar os temporizadores lógicos;

Granularidade

- Definição: intervalo mínimo de tempo medido pelo temporizador. Ex.: A cada 100ms, a cada 1s, etc.
- O que acontece com o STR se a granularidade for de 100ms e uma tarefa precisa ser executada a cada 1250 ms?
- Qualquer temporização realizada pelo STR será sempre uma aproximação que depende da granularidade do sistema;
- Quais são os fatores que afetam a contagem de tempo pelo temporizador?

Métricas de desempenho

- Como saber se um sistema operacional é mais ou menos apropriado para aplicações de tempo real?
  - Tempo de chaveamento entre duas tarefas;
  - Latência até o início do tratador de interrupção;
    - Tempo para salvar o contexto atual;
    - Tempo de desvio para o tratador de interrupção.
  - Tempo de resposta das chamadas de sistema (SYSCALL).
- O desempenho de um sistema de tempo real está ligado ao atendimento das restrições temporais e não à velocidade de execução;
- Onde inserir o mecanismo de tratamento das restrições de tempo?
- Importância do teste de escalonabilidade.

Troca de contexto

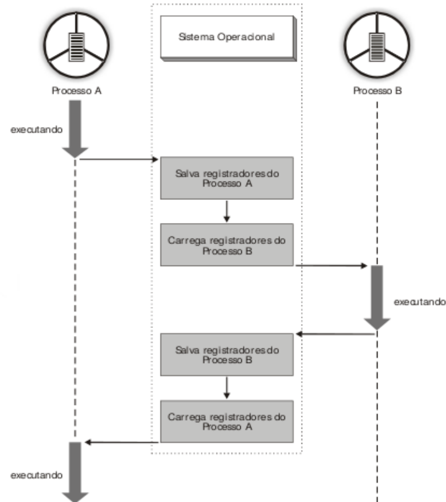

Figura 2.1: Troca de contexto [Galvin et al., 2013]

Tratamento de interrupções

(A)
interrupção
latência
k-des
hw
trat
tempo

(B)
interrupção
latência
k-des
hw
k-des
trat
tempo

k-hab = Kernel com interrupções habilitadas
k-des = Kernel com interrupções desabilitadas
hw = Tempo que o hardware leva para ativar o tratador
trat = Código que trata a interrupção do hardware

(C)
interrupção
latência
k-hab
hw
k-hab
trat
tempo

Figura 2.2: Tratamento de interrupções no kernel [FARINES and MELO, 2000]

Múltiplos tratadores de interrupção

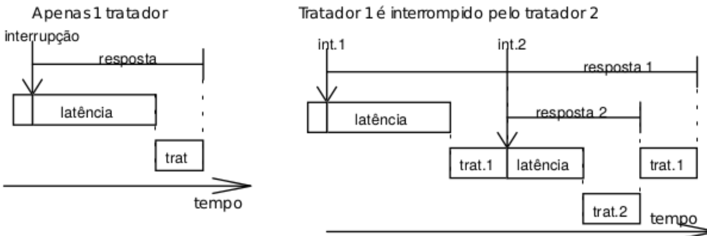

Figura 2.3: Tempo de resposta de um tratador simples [FARINES and MELO, 2000]

Exemplo de uma SYSCALL

Lembre-se: o processador só pode rodar um programa de cada vez!

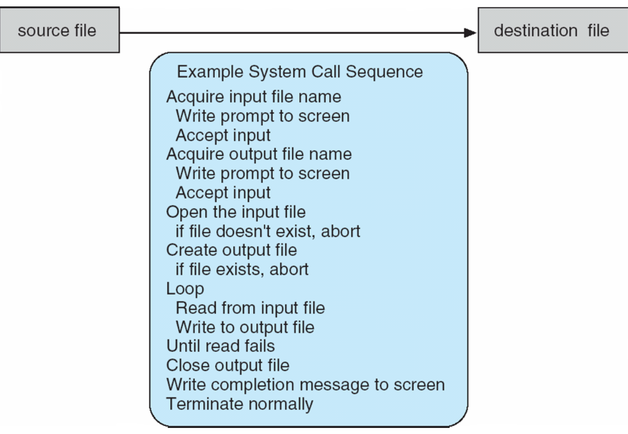

Figura 2.4: Copiando o conteúdo para um arquivo [Galvin et al., 2013]

UniCEUB Centro Universitário de Brasília

Execução da SYSCALL

Address
0xFFFFFFFF

User space

Return to caller
Trap to the kernel
5 Put code for read in register
4
10
Increment SP 11
Call read
3 Push fd
2 Push &buffer
1 Push nbytes

Library procedure read

User program calling read

6
9

Kernel space (Operating system)

Dispatch 7 8 Sys call handler

0

Figura 2.5: Fluxo de execução da chamada read [Tanenbaum and Machado Filho, 1995]

Execução da SYSCALL (passos)

1. Armazena os bytes;
2. Carrega no buffer;
3. Gera o descritor de arquivo (fd);
4. Chama a rotina da biblioteca (call);
5. Executa a instrução TRAP. Nesse momento a chamada é "promovida" ao modo kernel;
6. Passa a instrução para um endereço específico do kernel;
7. Ativa o endereço específico para a chamada (registradores);
8. Rotina de tratamento das chamadas de sistema;
9. Retorna para a instrução TRAP. Podem também bloquear o programa que a chamou;
10. Retorna ao programa do usuário;
11. Limpa a pilha.

Avaliação de um STR [FARINES and MELO, 2000]

- É possível desativar todos aqueles mecanismos que tornam o comportamento temporal menos previsível, sendo memória virtual o exemplo típico?
- Os tratadores de dispositivo atendem as requisições conforme as prioridades da aplicação ou simplesmente pela ordem de chegada?
- A mesma questão pode ser feita com respeito aos módulos do sistema responsáveis pela alocação de memória e pelo sistema de arquivos.
- O kernel do sistema pode ser interrompido a qualquer momento para a execução de um tratador de interrupção?
- Uma thread executando código do kernel pode ser *preemptada* por outra thread de prioridade mais alta, quando esta outra deseja executar código da aplicação?
- Uma thread executando código do kernel pode ser *preemptada* por outra thread de prioridade mais alta, quando esta outra deseja fazer uma chamada de sistema?

Avaliação de um STR (cont.) [FARINES and MELO, 2000]

- Qual o tempo necessário para chavear o contexto entre duas threads da mesma tarefa?
- Qual o tempo necessário para chavear o contexto entre duas threads de tarefas diferentes?
- Qual a latência até o início da execução de um tratador de interrupções?
- Qual o tempo de execução de cada chamada de sistema?
- Qual o maior intervalo de tempo contínuo no qual as interrupções permanecem desabilitadas?
- No caso do sistema permitir várias threads executarem simultaneamente código do kernel, qual o pior caso de bloqueio associado com as estruturas de dados?

UniCEUB
Centro Universitário de Brasília

1. Sistemas multiprogramáveis

2. Mecanismos de sincronização

3. Modelos de sistema de tempo real

Modelo geral

- Normalmente os sistemas de tempo real são reativos e baseados na abordagem estímulo-resposta;
- Processos separados para cada tipo de sensor e atuador.

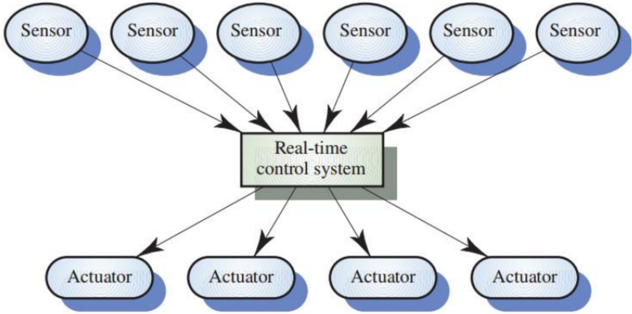

Figura 3.1: Modelo geral de programação para sistemas de tempo real [Chagas, 2016]

Notações [Chagas, 2016]

- Desenvolver um projeto é basicamente realizar a transformações entre diferentes notações em diferentes estágios.
- Nos níveis mais altos, as transformações não são bem definidas.

  Informal Linguagem corrente.

  Estruturada Representação gráfica, Componentes bem definidos, Interconexões bem definidas.

  Formal Base matemática e Descrições precisas.

UniCEUB
Centro Universitário de Brasília

Principais atividades do projeto

1. Seleção da plataforma (hardware e SO)
2. Identificar os estímulos (sensores) /respostas (atuadores).
3. Analisar restrições temporais (timing) para cada estímulo (restrições de tempo).
4. Alocar estímulo e processamento a processos concorrentes.
5. Projeto de processos (concorrentes), de acordo com a arquitetura do sistema.

Principais atividades do projeto (cont.)

6. Projetar os algoritmos para fazer o processamento necessário

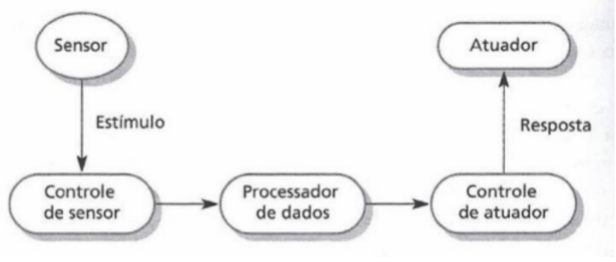

Figura 3.2: Algoritmo de sensores [Chagas, 2016]

7. Projeto de dados
8. Teste de escalabilidade

Considerações

- Coordenação de processos (semáforos, regiões críticas);
- Pode ser difícil realizar análises para avaliar se as restrições temporais serão atendidas;
- Linguagens OO podem não ser eficientes para implementar um STR;
- Os processos podem executar com diferentes velocidades.

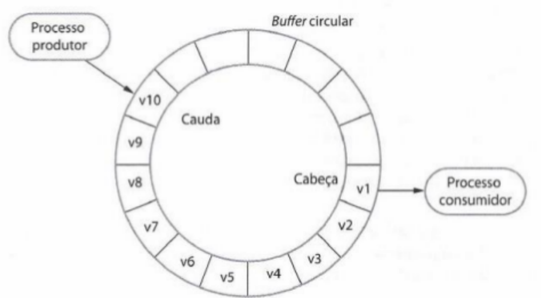

Figura 3.3: Exclusão mútua em *buffer* circular [Chagas, 2016]

Chagas, F. (2016).
Notas de aula do Prof. Fernando Chagas.

FARINES, J. M. and MELO, R. (2000).
Sistemas de Tempo Real, volume 1.
IME-USP.

Galvin, P. B., Gagne, G., and Silberschatz, A. (2013).
Operating system concepts.
John Wiley & Sons, Inc.

Tanenbaum, A. S. and Machado Filho, N. (1995).
Sistemas operacionais modernos, volume 3.
Prentice-Hall.

OBRIGADO!!!
PERGUNTAS???