Técnicas de Programação em C em Sistemas Embarcados

Daniel Quadros

dqsoft.blogspot.com

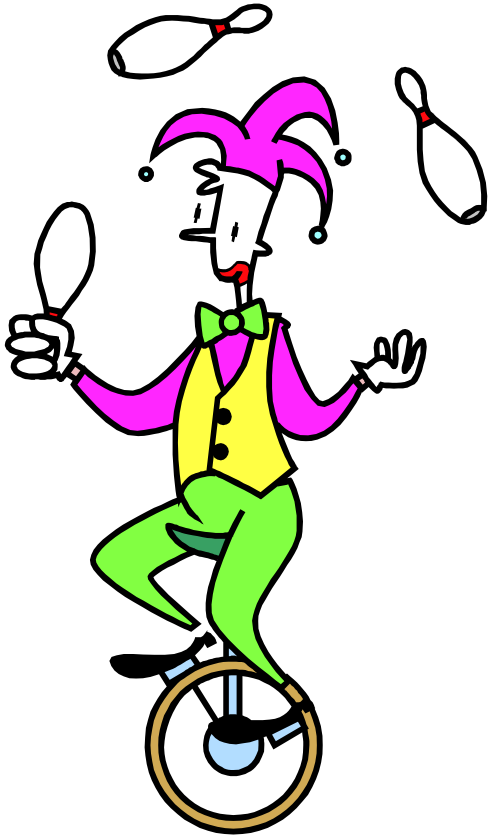
O Palestrante

- Engenheiro Eletrônico 81
- Scopus Tecnologia (82-86)
- Humana Informática (86-93)
- Seal Eletrônica (93-02)
- Tamid Tecnologia (02-)

A Palestra

- O Problema da Simultaneidade
- Loop Infinito de Polling
- Máquinas de Estado
- Interrupção, Filas e Pilhas
- Loop de Mensagens
- Multiprogramação Não Preemptiva
- Multiprogramação Preemptiva
- Conclusão

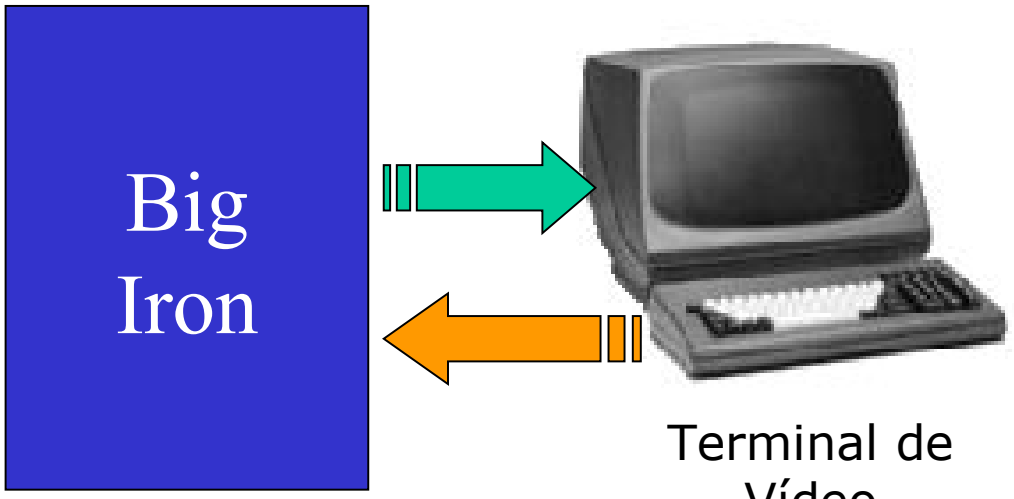
O Problema da Simultaneidade



Em uma aplicação embarcada é comum o software ter que tratar várias coisas “ao mesmo tempo”

O Problema da Simultaneidade

Um exemplo simples



O Problema da Simultaneidade

- Em um sistema embarcado podem existir várias entradas que exijam tratamento
- Estas entradas costumam ser assíncronas (ocorrem em momentos independentes)
- Alguns tratamentos podem ser demorados

Loop Infinito de Polling

```
while (1)
{
  if (TemTecla())
    TrataTecla();
  if (TemRx())
    TrataRx();
}
```

- É a solução mais óbvia e comum
- Não requer grandes teorias...
- mas é bastante limitada

Loop Infinito de Polling

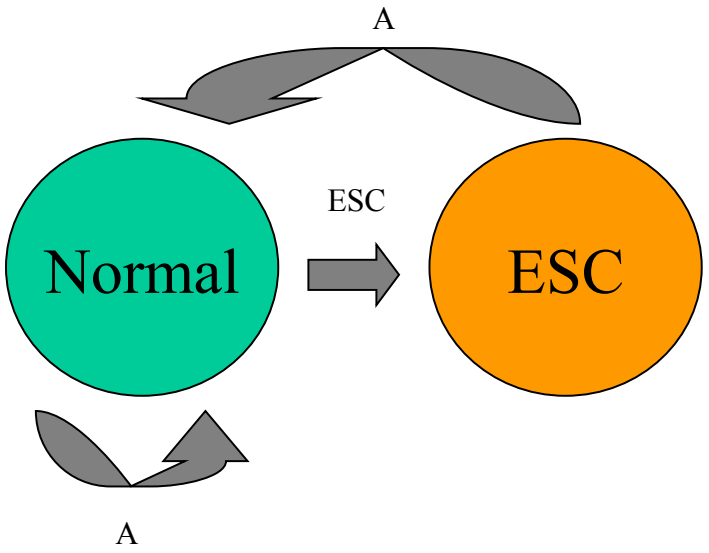
- Um problema é quando as entradas para uma determinada tarefa vem em partes
- Exemplo: comandos de um terminal de vídeo
  - ‘A’ escreve um A na tela
  - ESC ‘A’ sobe o cursor uma linha

Loop Infinito de Polling

```
while (1)
{
  if (TemTecla())
    TrataTecla();
  if (TemRx())
  {
    if (Rx == ESC)
    {
      while (!TemRx())
        ;
      if (Rx == 'A')
        SobeCursor();
    }
    else
      Mostra(Rx);
  }
}
```

Máquinas de Estado

- Em cada momento o software está em um estado dentre vários possíveis
- Uma entrada pode causar uma mudança de estado e/ou a execução de uma ação



Máquinas de Estado

```
enum { EST_N, EST_ESC }
estado;

estado = EST_N;
while (1)
{
  if (TemTecla())
    TrataTecla();
  if (TemRx())
    if (estado == EST_N)
    {
      if (Rx == ESC)
        estado = EST_ESC;
      else
        Mostra (Rx);
    }
```

```
      else // EST_ESC
      {
        if (Rx == 'A')
          SobeCursor();
        estado = EST_N;
      }
}
```

Máquinas de Estado

- Podemos ter várias máquinas de estado em um programa
- Implementação
  - If (poucos estados e entradas)
  - Switch (mais legível se tem muitos estados ou entradas)
  - Tabela de estado anterior, entrada, novo estado, ação (mais lento se procurar sequencialmente)

Interrupção, Filas e Pilhas

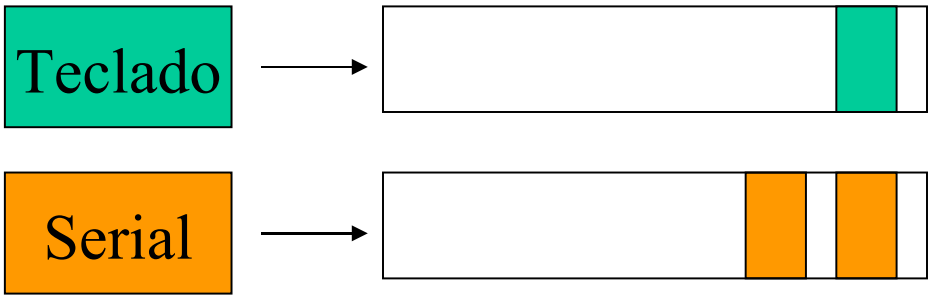
- Nem sempre dá para esperar a vez antes de registrar uma entrada
  - 9600 bps = 1 byte cada ms !
- Interrupção garante atendimento "imediato" mas não pode fazer tratamento demorado
- Entrada precisa ser guardada em um lugar temporário

Interrupção, Filas e Pilhas

- Uma fila (FIFO) permite tratar as entradas na ordem em que ocorreram
- Estrutura típica é a fila circular
- Uma fila pode ser usada para acomodar diferenças temporárias de velocidade
- Uma pilha (LIFO) pode guardar uma tarefa temporariamente enquanto uma mais prioritária é executada

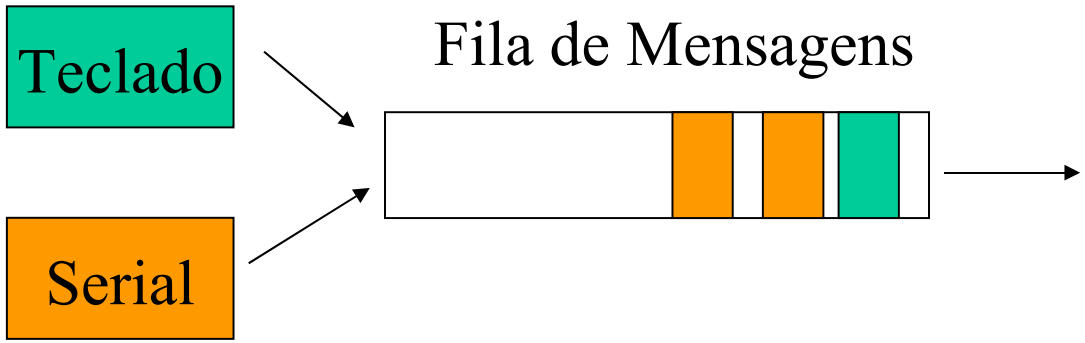
Interrupção, Filas e Pilhas

- Filas e Pilhas podem ser usadas com todas as técnicas descritas
  - quem disse que TemTecla() olhava no hardware e não em uma fila?
- Podemos ter uma fila para cada entrada...



Loop de Mensagens

- Ou usar uma fila “genérica” para armazenar diversos tipos de entradas
- Cada entrada gerar uma “mensagem” (ou evento) na fila, com tipo e dados



Loop de Mensagens

```
while (1)
{
  if (TemMsg(&msg))
  {
    switch (msg->tipo)
    {
      case MSG_TEC:
        TrataTec();
        break;
      case MSG_RX:
        TrataRx();
        break;
    }
  }
}
```

Resumo Até Agora

- Polling, máquinas de estado e filas resolvem a maioria dos problemas, mas...
- Máquinas de estado complexas reduzem a legibilidade do código
- Código que será executado seqüencialmente no tempo precisa ser quebrado no fonte

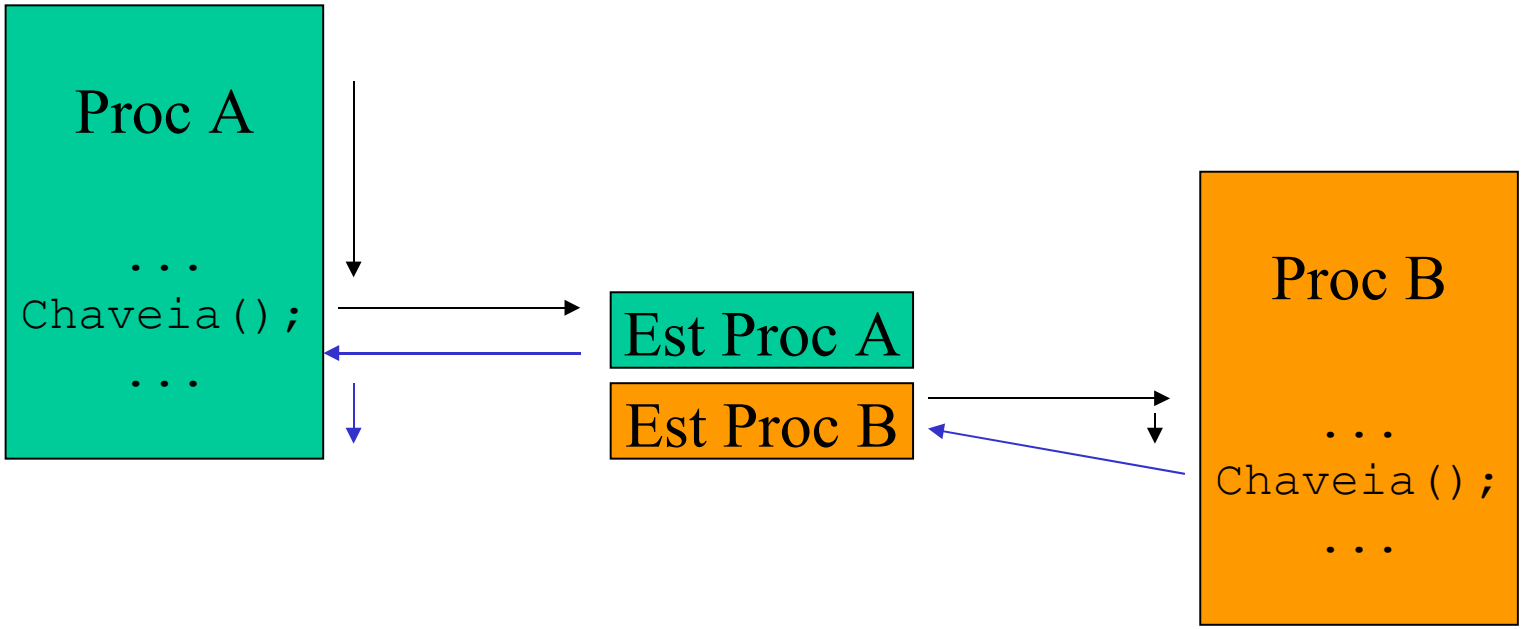
Multiprogramação Não Preemptiva

- Programa é dividido em ‘tarefas’ executadas independentemente (‘em paralelo’)
- Uma tarefa executa até que passar o controle para outra
- As tarefas precisam ser ‘bem comportadas’
- Minimiza problema de regiões críticas

Multiprogramação Não Preemptiva

- Em uma chamada normal de rotina o estado do processador é salvo na entrada e restaurado na saída, normalmente usando uma pilha
- O chaveamento de processos é feito chamando uma rotina que salva o estado do processo atual e restaura o estado do processo seguinte

Multiprogramação
Não Preemptiva



Multiprogramação Não Preemptiva

- Uma fila (ou algo mais complexo) anota os processos prontos para executar
- Primitivas
  - Chaveia() passa para o processo seguinte, processo atual entra na fila de prontos
  - Dorme(tempo) passa para o processo seguinte, processo atual só entra na fila de prontos após o tempo

Multiprogramação Não Preemptiva

- Primitivas (semáforos contadores)
  - p() solicita um 'recurso', se não disponível passa para o processo seguinte, processo atual entra na fila de espera do recurso
  - v() libera o 'recurso', se tinha alguém na fila do recurso coloca na fila de processos prontos para execução
  - implementados através de contador + fila
  - variação: ptemp(tempo) p() com timeout para recurso ficar disponível

Multiprogramação
Não Preemptiva

```
ProcTeclado()
{
  while (1)
  {
    p(semTecla);
      TrataTecla();
  }
}
```


```
ProcRx()
{
  while (1)
  {
    p(semRx);
    if (Rx == ESC)
    {
      p(semRx);
      if (Rx == 'A')
        SobeCursor();
    }
    else
      Mostra(Rx);
  }
}
```

Multiprogramação Preemptiva

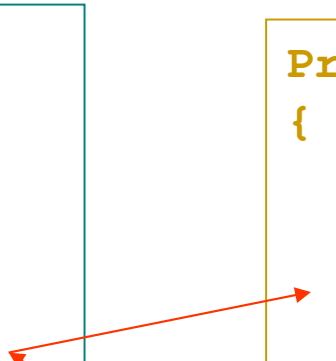
- Programa é dividido em ‘tarefas’ executadas independentemente (‘em paralelo’)
- Uma tarefa pode ser interrompida para passar o controle para outra
- Resiste melhor a tarefas ‘mal comportadas’
- Surge o problema de regiões críticas

Multiprogramação Preemptiva

```
int var;
```

```
ProcA ()
{
   int aux;
   ...
   aux = var;
   aux = aux+1;
   if (aux > MAX)
      aux = 0;
   var = aux;
   ...
}
```

```
ProcB ()
{
   int aux;
   ...
   aux = var;
   aux = aux-1;
   if (aux < 0)
      aux = MAX;
   var = aux;
   ...
}
```



Multiprogramação Preemptiva

- Programação fica mais complexa
- Implementação é bem mais complexa (contexto é maior, chaveamento feito em tempo de interrupção)
- Aplicações embarcadas requerem tempos reduzidos e determinísticos para operar em ‘tempo real’

Multiprogramação Preemptiva

- Normalmente se utilizam soluções prontas:
  - proprietários
  - derivados do Linux
  - Windows CE
- Requer um microcontrolador poderoso
- SOs trazem recursos adicionais

http://en.wikipedia.org/wiki/List_of_real-time_operating_systems

Conclusão

- A simultaneidade está sempre presente em aplicações embarcadas
- Polling, máquinas de estado e filas são adequados para projetos pequenos e médios
- Multiprogramação não preemptiva é uma solução para sistemas médios
- Multiprogramação preemptiva é para projetos maiores e mais complexos

Obrigado!

Perguntas?