Monitores

Setembro de 2009

Monitores

- mecanismo de sincronização clássico
  - referência
  - influência de conceitos de programação estruturada
  - C. A. R. Hoare, Monitors: an operating system structuring concept, *Communications of the ACM*, v.17 n.10, p.549-557, Oct. 1974
  - Per Brinch Hansen, Structured multiprogramming, *Communications of the ACM*, v.15 n.7, p.574-578, July 1972

  - material baseado no Cap. 5 de FMPDP (Andrews)

o que são?

- construção sintática: compilador "entende" a encapsulação por monitor
- construção `monitor` garante exclusão mútua para as operações encapsuladas

```
monitor meusdados {
  int pegaDado ();
  void poeDado (int);
}
```

- para cooperação, uso de *condições*

```
cond umaCond;
...
while (!B) wait(umaCond);
```

Exclusão Mútua

```
monitor Buffer {
  int buf[SIZE]; int nxtfree = 0; int nxtdata = 0;
  cond hasfree, hasdata;
  void deposit (int data) {
    while ((nxtfree+1)%SIZE == nxtdata) wait(hasfree);
    buf[nxtfree] = data;
    nxtfree = (nxtfree+1)%SIZE;
    signal(hasdata);
  }
  int fetch () {
    int data;
    while (nxtfree == nxtdata) wait(hasdata);
    data = buf[nxtdata];
    nxtdata = (nxtdata+1)%SIZE;
    signal(hasfree);
  }
}
```

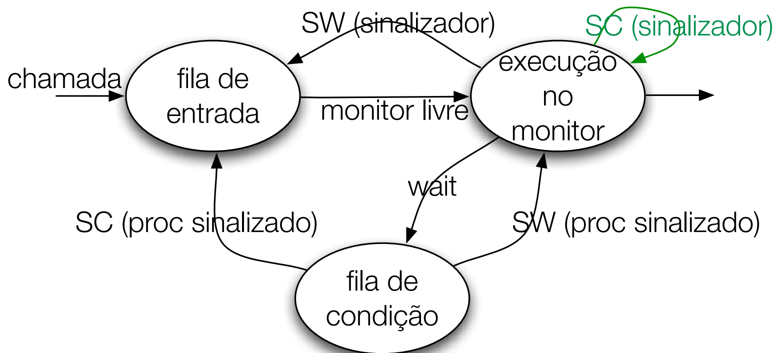
Espera, Sinalização e Exclusão Mútua

- espera por condição não pode manter exclusão mútua
  - caso contrário a condição provavelmente nunca será verdadeira
- primitiva *wait* deve liberar EM
- e o que acontece quando um outro processo executa *signal*?
  - dois processos dentro do monitor?

Políticas propostas

1. sinaliza e continua (SC)
2. sinaliza e espera (SW)
3. sinaliza e sai

Políticas de Sinalização



Políticas de Sinalização

- com SW, processo sinalizado pode ter certeza que a condição é verdadeira
  - política próxima à proposta inicialmente por Hoare
- com SC, o próprio processo sinalizador ou outros processos da fila de entrada podem ter revertido a condição
  - sinalização passa a ser apenas "dica"
  - teste de condição deve, quase sempre, ficar dentro de loop
  - mais fácil de entender!

Políticas de Sinalização

- a implementação de um semáforo é um bom exemplo da diferença entre as políticas:

```
monitor Semaphore {
  int s = 0; ## s >= 0
  cond pos;  # signaled when s > 0
  procedure Psem() {
    while (s == 0) wait(pos);
    s = s-1;
  }
  procedure Vsem() {
    s = s+1;
    signal(pos);
  }
}
```

Passagem de Condição

- técnica um pouco semelhante à passagem de bastão pode ser usada:

```
monitor Semaphore {
  int s = 0; ## s >= 0
  cond pos;  # signaled when s > 0
  procedure Psem() {
    if (s == 0) wait (pos);
    else s = s-1;
  }
  procedure Vsem() {
    if (empty(pos)) s = s+1;
    else signal(pos);
  }
}
```

- mais difícil de entender...

Leitores e Escritores

```
monitor RW_Controller {
  int nr = 0, nw = 0; ## (nr == 0 or nw == 0) and nw <= 1
  cond oktoread; # signaled when nw == 0
  cond oktowrite; # signaled when nr == 0 and nw == 0
  procedure request_read() {
    while (nw > 0) wait(oktoread);
    nr = nr + 1;
  }
  procedure release_read() {
    nr = nr - 1;
    if (nr == 0) signal(oktowrite); # awaken one writer
  }
  procedure request_write() {
    while (nr > 0 || nw > 0) wait(oktowrite);
    nw = nw + 1;
  }
  procedure release_write() {
    nw = nw - 1;
    signal(oktowrite); # awaken one writer and
    signal_all(oktoread); # all readers
  }
```

Outros problemas

- barreiras
- reserva de recursos

Cuidados...

- sinalização não embute contagem
  - condição sempre deve ser testada
- ordens de entrada totalmente arbitrárias

```
...
## tentativa de otimizar produtor/consumidor
int fetch () {
    ...
    data = buf[nxtdata]; nxtdata = (nxtdata+1)%SIZE;
    if (estavaCheia) signal(hasfree); # NÃO FUNCIONA COM WC
```

- aninhamento e deadlocks
- reentrância

Monitores com testes implícitos

- ao invés de filas de condições, esperas por predicados

```
monitor ReadersWriter {
    int rcnt, wcnt;

  public:
    ReadersWriter() { rcnt = wcnt = 0; }
    void StartRead() {
:;          waituntil( wcnt == 0 );
        rcnt += 1;

    }
    void EndRead() {
        rcnt -= 1;

    }
    void StartWrite() {
        waituntil( wcnt == 0 && rcnt == 0 );
        wcnt = 1;
    }
    void EndWrite() {
        wcnt = 0;


    }
};
```

Construções com testes implícitos

- programação mais fácil
  - sinalização é um grande problema na abstração de monitores
- implementação mais difícil
  - especialmente em termos de eficiência (quanto testar cada predicado pendente?)
- além de monitores, já foram feitas várias propostas com esperas implícitas

Monitores com sinalição implícita

- Buhr, P. A. and Harji, A. S. 2005. Implicit-signal monitors. *ACM Trans. Program. Lang. Syst.* 27(6), Nov. 2005, 1270-1343.

... but this *[referring to condition waiting]* may be too inefficient for general use in operating systems, because its implementation requires reevaluation of the expression B after every exit from a procedure of the monitor. [Hoare 1974, pp. 556–557]

This beautiful concept *[referring to condition waiting]* requires a process to reevaluate a boolean expression B periodically until it yields the value true. The fear that this reevaluation would be too costly on a single processor motivated the introduction of queues (also called signals or conditions) as an engineering compromise between elegance of expression and efficiency of execution. [Brinch Hansen 1981, p. 371]

PUC RIO

Implementações de Monitores

- exemplos clássicos: Concurrent Pascal, Modula, Mesa
- baseadas em monitores: Java e pthreads

Monitores em pthreads

- locks explícitos para exclusão mútua
  tipo `pthread_mutex_t` com operações `pthread_mutex_lock` e `pthread_mutex_unlock`
- variáveis de condição
  tipo `pthread_cond_t`
- operações explicitamente ligam condições a locks
  `pthread_cond_wait(&cond, &mutex)`

Monitores em pthreads: exemplo

- Soma de elementos de matriz (FMPDP)

- classe Thread e interface Runnable

```
class Simple implements Runnable {
  public void run () { System.out.println ("alo alo");}
}
Runnable s = new Simple;
new Thread(s).start();
...
```

- métodos podem ser declarados como synchronized
  - lock associado a cada objeto
- uma fila de condição por objeto

Monitores em Java: exemplo

```
// concurrent read or exclusive write
class ReadersWriters extends RWbasic {
  private int nr = 0;
  private synchronized void startRead() {
    nr++;
  }
  private synchronized void endRead() {
    nr--;
    if (nr == 0) notify(); // awaken waiting Writers
  }
  public void read() {
    startRead();
    System.out.println("read:  " + data);
    endRead();
  }
  public synchronized void write() {
    while (nr > 0)   // delay if any active Readers
      try { wait(); }
        catch (InterruptedException ex) {return;}
    data++;
    System.out.println("wrote:  " + data);
    notify();    // awaken another waiting Writer
  }
}
```