FACULDADE DE CIÊNCIAS E TECNOLOGIA
DEPARTAMENTO DE INFORMÁTICA

Fundamentos de Sistemas de Operação

LEI - 2023/2024

Vitor Duarte
Mª. Cecília Gomes

1

Aula 9

- Exemplo com variáveis de condição: Produtores/Consumidores
- Threads em Java

- OSTEP: cap. 30.1, 30.2

2

Variáveis de condição

- Permitem controlar o teste concorrente de condições e aguardar
- Usada junto com um *mutex* para exclusão mútua enquanto se testa a condição e para se bloquear
  - Início:

```
pthread_cond_t cond;
pthread_cond_init( &cond, NULL);
pthread_mutex_t mutex;
pthread_mutex_init( &mutex, NULL);
```

Bloquear aguardando alteração da condição:

```
pthread_cond_wait(&cond, &mutex)
```

liberta mutex

Desbloquear um dos bloqueados na condição ou todos:

```
pthread_cond_signal(&cond)
pthread_cond_broadcast(&cond)
```

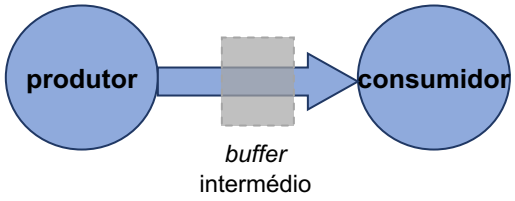

Exemplo de uso

```
pthread_mutex_lock(&lock);
while (!notificado)  // condição a testar
   pthread_cond_wait(&cond, &lock);
notificado = FALSE;
pthread_mutex_unlock(&lock);
```

```
pthread_mutex_lock(&lock);
notificado = TRUE;
pthread_cond_signal(&cond);
pthread_mutex_unlock(&lock);
```

Produtor(es) – Consumidor(es)

- Modelo comunicação por uma fila FIFO (limitada).
  - Caso de comunicação entre threads ou processos
- Consideremos Thread Produtor: produz dados que põe na fila
- Thread Consumidor: retira do buffer os dados anteriores



buffer
intermédio

Produtores e consumidores

- Operações: por(), tirar() (ou put(), take())
- Exemplos de produtores e consumidores:

```
produtor() {
      for (int i=0; i<max; i++)
            por( poduz(i) );
}
```

```
consumidor() {
      for (int i=0; i<max; i++)
            consome( tirar() );
}
```

- O produtor e o consumidor executam aos seus ritmos, mas:
  - tirar() deve aguardar se buffer vazio
  - por() deve aguardar se buffer cheio
  - A concorrência não deve ser problema
  - Podem ser vários produtores e/ou consumidores

Uma implementação

- Baseada num buffer partilhado (como um buffer circular):
  - Permite suportar uma comunicação em *stream* ou troca de mensagens (sejam objetos ou blocos de bytes)
  - Gerido pelos próprios threads e com menos cópias
- Tem contador de posições ocupadas
  - vazias = capacidade-ocupadas
- Condição a considerar pelo por(): buffer cheio?
  - se ocupadas==capacidade bloqueia-se à espera
- Condição a considerar pelo tirar(): buffer vazio?
  - se ocupadas==0 bloqueia-se à espera

Que problemas de sincronização?

- O produtor e consumidor alteram estruturas de dados comuns (p.ex. o buffer e contador de ocupadas)
  - é necessário acesso exclusivo a algumas variáveis?
- tirar() :
  - se buffer vazio, bloqueia-se aguardando por dados
  - o produtor é que pode desbloquear um consumidor quando põe algo no buffer
- por() :
  - se buffer cheio, bloqueia-se aguardando que tirem dados do buffer
  - o consumidor é que pode desbloquear um produtor quando tirar algo do buffer

buffer circular base

```
T buffer[capacidade];
int p=0; // onde por
int g=0; // de onde tirar
int ocupadas=0;
//vazias==capacidade-ocupadas

bool bufempty() {
  return ocupadas==0;
}

bool bufffull() {
  return ocupadas==capacidade;
}
```

```
buffget() {
   T tmp = buffer[g++];
   g = g%capacidade;
   ocupadas--;
   return tmp;
}

buffput(T e) {
    buffer[p++] = e;
    p = p%capacidade;
    ocupadas++;
}
```


1ª tentativa

```
por( T e) {
  while (bufffull())
        ;
  bufput( e );
}
```

```
T tirar() {
  while (bufempty())
        ;
  return bufget();
}
```

Desperdício de CPU

Nem funciona para 1 produtor e 1 consumidor
partilham 'ocupadas'

2ª tentativa

```
por( T e) {
  while (bufffull())
        ;
  lock(m);
  bufput( e );
  unlock(m);
}
```

```
T tirar() {
  while (bufempty())
        ;
  lock(m);
  T e = bufget();
  unlock(m);
  return e;
}
```

Desperdício de CPU

Só funciona para 1 produtor e 1 consumidor

3ª tentativa

```
por( T e) {
  lock(m);
  while (bufffull())
        ;
  bufput( e );
  unlock(m);
}
```

```
T tirar() {
  lock(m);
  while (bufempty())
        ;
  T e = bufget();
  unlock(m);
  return e;
}
```

Desperdício de CPU

Não funciona. Deadlock

4ª tentativa

```
por( T e) {
  lock(m);
  while (bufffull())
     wait(cond, m);
  bufput( e );
  signal(cond);
  unlock(m);
}
```

```
T tirar() {
  lock(m);
  while (bufempty())
     wait(cond, m);
  T e = bufget();
  signal(cond);
  unlock(m);
  return e;
}
```

Não funciona. Há duas condições distintas.

Solução

```
por( T e) {
  lock(m);
  while (bufffull())
     wait(space, m);
  bufput( e );
  signal(elem);
  unlock(m);
}
```

```
T tirar() {
  lock(m);
  while (bufempty())
     wait(elem, m);
  T e = bufget();
  signal(space);
  unlock(m);
  return e;
}
```

Java e threads...

- Suporte no *run-time* para *threads* e controlo da concorrência
  - Classe **`Thread`** para gestão dos *threads*
  - Declaração "**`synchronized`**" para marcar zonas de exclusão mútua
  - Qualquer objecto pode ser usado para exclusão mútua
  - Existem classes para controlo da concorrência (locks, ...)
  - Existem classes de estruturas de dados para uso por múltiplos threads concorrentes:
    - HashMap vs. ConcurrentHashMap
    - LinkedList vs ConcurrentLinkedQueue
    - etc
    - ver pacotes java.util e java.util.concurrent

java.lang.Thread

- Tipo de objecto que representa um thread
- Métodos:
  - run() – método a executar no novo thread
  - start() – criar/arrancar com o novo thread
  - join() – aguardar pela terminação do thread
  - …
- Podemos criar objetos **`Thread`** a partir de qualquer outro objeto que contenha o método **`run()`** (interface **`Runnable`**)

Exemplo 1

```
class MyThread extends Thread {
  public void run() {
    System.out.println("Novo thread: " + this);
  }
}

class Prog {
public static void main( String[] args )
                              throws Exception {
   MyThread t1 = new MyThread();
   MyThread t2 = new MyThread();
   t1.start();
   t2.start();
   t1.join();
   t2.join();
   System.out.println("threads terminados");
 }
}
```


Exemplo 2

```
class MyThread extends Thread {
  static long counter = 0;  //shared
  private long max;

  MyThread( long l ) {
    max = l;
    start();
  }

  public void run() {
    for (long i=0; i<max; i++) {
      counter = counter + 1;
    }
  }
}
```

```
class Prog2 {
  public static void main(String[] args)
                throws Exception {
    long l = Long.parseLong( args[0] );

    MyThread p1 = new MyThread( l );
    MyThread p2 = new MyThread( l );

    p1.join();
    p2.join();
    System.out.println( "counter= "
                        + MyThread.counter );
  }
}
```


Exclusão mútua e Java.lang.Object

- Raiz de todos os objetos
  - Qualquer objecto serve de "sincronizador" (eq. mutex) e de variável de condição
- Métodos oferecidos:
  - wait(); - bloquear o thread aguardando por uma notificação (eq. *cond_wait*)
  - notify(); - desbloquear um thread que aguarda neste objecto (eq. *cond_signal*)
  - notifyall(); - desbloquear todos os threads aguardando neste objecto (eq. *cond_broadcast*)

Exclusão mútua

```
class MyThread extends Thread {
  static long counter = 0;
  static Object mtx = new Object(); //or  MyThread.class
  private long max;
  MyThread( long l ) {
    max = l;
    start();
  }
  public void run() {
    for (long i=0; i<max; i++) {
      synchronized ( mtx ) {
        counter = counter + 1;    // Critical Section
      }
    }
  }
}
```

NOVA FACULDADE DE CIÊNCIAS E TECNOLOGIA DEPARTAMENTO DE INFORMÁTICA

21