# Sistemas Distribuídos

INF2545

2015.1

# alternativas p/ concorrência

1. multithreading
   - várias linhas de execução compartilham globais com escalonamento preemptivo
   - surgido de estudos de sistemas operacionais
   - dificuldades de sincronização
   - exemplos: threads em C (posix) e em Java

# exemplo em Java

```
public class ThreadsDorminhocas {
  public static void main(String[] args) {
    new ThreadDorminhoca("1");
    new ThreadDorminhoca("2");
    new ThreadDorminhoca("3");
    new ThreadDorminhoca("4");
  }
}
```

# threads em Java

```
class ThreadDorminhoca extends Thread {
 int tempo_de_sono;
 public ThreadDorminhoca(String id) {
  super(id);
  tempo_de_sono = (int) (Math.random() * 5000);
  System.out.println("Tempo de sono da thread "+id+
                ": "+tempo_de_sono+"ms");
  start();
 }
 public void run() {
  try {
   sleep(tempo_de_sono);
  } catch(InterruptedException exception) {
   System.err.println(exception);
  }
  System.out.println("thread "+getName()+" acordou!");
 }
}
```

transferência de controle implícita!
(preempção)

# threads e Java– variáveis compartilhadas

```
class ThreadDorminhoca extends Thread {
  int tempo_de_sono; Conta minhaConta;
  public ThreadDorminhoca(Conta c) {
    minhaConta = c;
    start();
  }
  public void run() {
    //  realiza um monte de operações envolvendo minhaConta;
}
```

# condições de corrida

```
class Conta {
  private int saldo;
  public Conta (int ini) {
    saldo = ini;
  }
  public int veSaldo() {
    return saldo;
  }
  public void deposita(int dep) {
      saldo = saldo+dep; // problema!!! – difícil de observar
  }
}
```

# condições de corrida

```
class Conta {
  private int saldo;
  public Conta (int ini) {
    saldo = ini;
  }
  public int veSaldo() {
    return saldo;
  }
  public void deposita(int dep) {
    for (int i=0; i<dep; i++) { // artificial!! – facilita observação
      saldo++;
    }
  }
}
```

# condições de corrida

```
class Conta {
 private int saldo;
 public Conta (int ini) {
  saldo = ini;
 }
 public int veSaldo() {
  return saldo;
 }
 public void deposita(int dep) {
  for (int i=0; i<dep; i++) {
   try {
    Thread.sleep(10); //+ artificial – facilita observação
   }
   catch(InterruptedException exception) {
    System.err.println(exception);
   }
   saldo++;
  }
 }
}
```

# condições de corrida

```
class Conta {
 private int saldo;
 public Conta (int ini) {
  saldo = ini;
 }
 public int veSaldo() {
  return saldo;
 }
 synchronized public void deposita(int dep) {
  for (int i=0; i<dep; i++) {
   try {
    Thread.sleep(10); //+ artificial:  para escalonador agir!
   }
   catch(InterruptedException exception) {
    System.err.println(exception);
   }
   saldo++;
  }
 }
}
```

# alternativas ao modelo multithread clássico

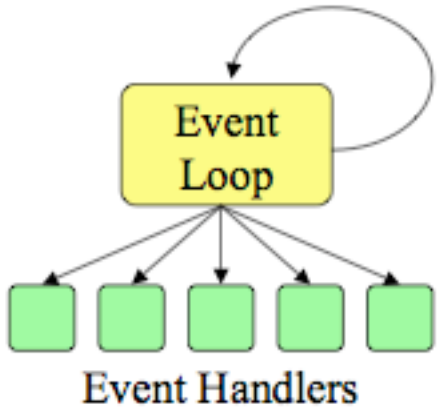
2. modelos orientados a eventos
   - cada evento tratado até o final
   - programa como máquina de estado

# alternativas p/ concorrência

- eventos - descrição Ousterhout:

## Event-Driven Programming

- One execution stream: no CPU concurrency.
- Register interest in events (callbacks).
- Event loop waits for events, invokes handlers.
- No preemption of event handlers.
- Handlers generally short-lived.




*Why Threads Are A Bad Idea* *September 28, 1995, slide 9*

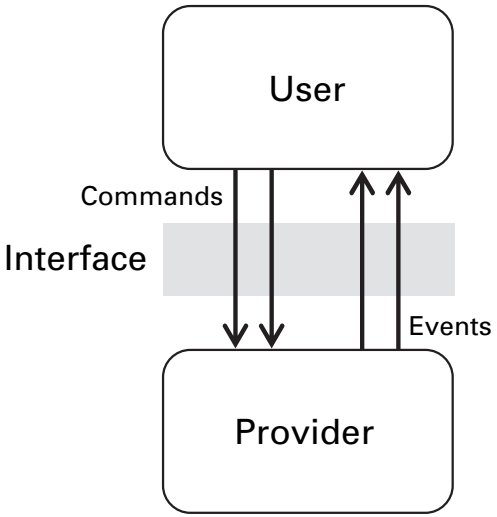
PUC RIO

# tratadores de eventos

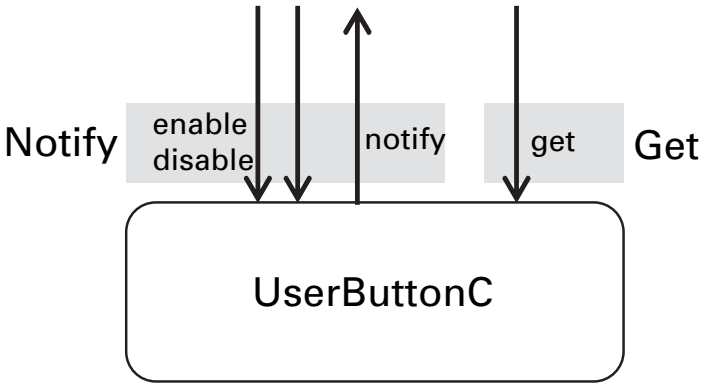
- lógica "invertida": programa reage a eventos
- conceito geral com várias variantes
  - canal de eventos
  - bindings diretos entre geradores e consumidores

# tinyOS

- modelo do sistema é comunicação por comandos e eventos
  - comandos disparam ações que podem retornar através de eventos




```
interface Notify<val_t> {
  command error_t enable();
  command error_t disable();
  event void notify(val_t val);
}
```




# threads X eventos

- why threads are a bad idea...
  - custo (recursos e tempo)
  - uso de primitivas de sincronização
  - dificuldade de programação
  - escalonamento embutido e pouco maleável

- why events are a bad idea...
  - fluxo de controle fica escondido (máquina de estado)
    - stack ripping
  - dificuldade com coleta de lixo
  - dificuldade de programação

# threads X eventos

- why threads are a bad idea...
  - custo (recursos e tempo)
  - uso de primitivas de sincronização
  - dificuldade de programação
  - escalonamento embutido e pouco maleável

PREEMPÇÃO

- why events are a bad idea...
  - fluxo de controle fica escondido (máquina de estado)
    - stack ripping
  - dificuldade com coleta de lixo
  - dificuldade de programação

# alternativas ao modelo multithreading clássico

3. multitarefa sem preempção: multithreading no nivel da aplicação
   - fibers
   - co-rotinas
   - ...

# co-rotinas simétricas: Modula-2

```
MODULE M;
 CONST
  WKSIZE = 512;
 VAR
  wkspA, wkspB : ARRAY [1..WKSIZE]
      OF BYTE;
  main, cA, cB : ADDRESS;
  x : ADDRESS;

PROCEDURE A;
BEGIN
 LOOP
  ...
  TRANSFER(x,x);
  ...
 END;
END A;
```

```
 PROCEDURE B;
 BEGIN
  LOOP
   ...
   TRANSFER(x,x);
   ...
  END;
 END B;


 BEGIN (* M *)
    (* create two processes out of
procedure A and B *)
   NEWPROCESS( A, ADR(wkspA),
WKSIZE, cA );
   NEWPROCESS( B, ADR(wkspB),
WKSIZE, cB );
   x := cB;
   TRANSFER(main,cA);
 END M;
```

# co-rotinas assimétricas: Lua

```
function boba ()
  for i=1,10 do
    print("co", i)
    coroutine.yield()
  end
end
co = coroutine.create(boba)

coroutine.resume(co) -> co 1
coroutine.resume(co) -> co 2
...
coroutine.resume(co) -> co 10
coroutine.resume(co) -> nada... (acabou)
```

transferência de controle explícita!
menos problemas com condições de corrida.
por outro lado...

- só usamos um processador
- temos que controlar a transferência: flexibilidade mas programação mais baixo nível....

# threads + eventos

- sistemas híbridos procuram combinar vantagens de threads e eventos

  - Salmito, Moura, Rodriguez. Understanding Hybrid Concurrency Models. Revista Bras. de Redes e Sistemas Distribuídos, 2011.

# alternativas ao modelo multithreading clássico

4. multithreading com troca de mensagens
   - Erlang

# erlang

- troca de mensagens:
- envio:

```
pid! msg
```

- recebimento:

```
receive
    padrão -> ação
end
```

# erlang

```
loop(... ) ->
 receive
  {From, Request} ->
     Response = F(Request),
     From ! {self(), Response},
     loop(...)
 end.
```

# Referências


- J. Ousterhout. Why threads are a bad idea (for most purposes)
- von Behren, R., Condit, J., and Brewer, E. 2003. Why events are a bad idea (for high-concurrency servers). In Proceedings of the 9th Conference on Hot Topics in Operating Systems - Volume 9 (Lihue, Hawaii, May 18 - 21, 2003).
- L. Mottola and G.-P. Picco. 2011. Programming wireless sensor networks: Fundamental concepts and state of the art. ACM Comput. Surv. 43, 3, (April 2011) seções 1 e 2
- capítulo de co-rotinas - Roberto Ierusalimschy. Programming in Lua. lua.org, 2013 (1ª edição disponível em www.lua.org/pil/).