Sistemas Distribuídos

INF2545

Sistemas Distribuídos

- o que são: "coleção de máquinas independentes que aparecem para o usuário como um único sistema coerente"
  - que tipo de usuário?
    - » programador é usuário?
  - o que é "coerente"?
    - » o conceito de transparência
- um sistema distribuído é uma coleção de máquinas independentes que são usadas em conjunto para executar uma tarefa ou prover um serviço.

Received: by jumbo.dec.com (5.54.3/4.7.34)
id AA09105; Thu, 28 May 87 12:23:29 PDT
Date: Thu, 28 May 87 12:23:29 PDT
From: lamport (Leslie Lamport)
To: src-t
Subject: distribution

There has been considerable debate over the years about what constitutes a distributed system. It would appear that the following definition has been adopted at SRC:

A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable.

The current electrical problem in the machine room is not the culprit--it just highlights a situation that has been getting progressively worse. It seems that each new version of the nub makes my FF more dependent upon programs that run elsewhere. Having to wait a few seconds for a program to be swapped in is a lot less annoying than having to wait an hour or two for someone to reboot the servers. ...

I will begin the effort by volunteering to gather some data on the problem. If you know of any instance of user's FF becoming inoperative through no fault of its own, please send me a message indicating the user, the time, and the cause (if known).

Leslie

para que queremos SDs?

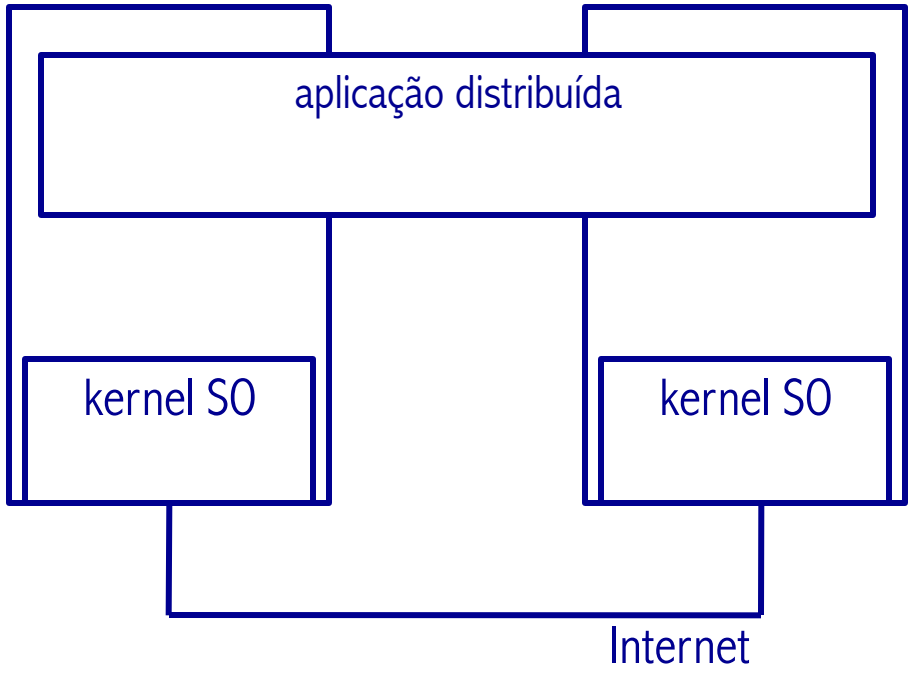
- custo e desempenho
  - paralelismo e computação distribuída
- escalabilidade
  - facilidade de aumentar recursos
- distribuição inerente
  - dispersão geográfica e social
  - compartilhamento de recursos
- confiabilidade
  - redundância

... mas...

- rede tem que ser levada em consideração
  - desempenho e falhas
- segurança
  - distribuição introduz problemas inexistentes em sistemas centralizados
- complexidade de software

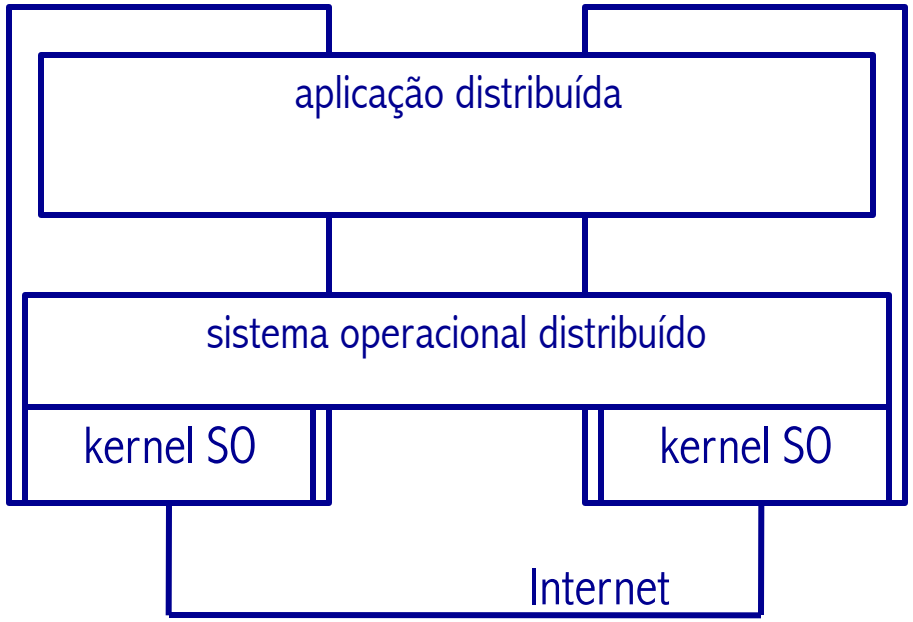
arquiteturas de interesse

- multicomputadores: cada um com sua memória e processador(es?), interligados por redes
  - construção de aplicação distribuída sobre recursos de redes de SO pode ser árdua



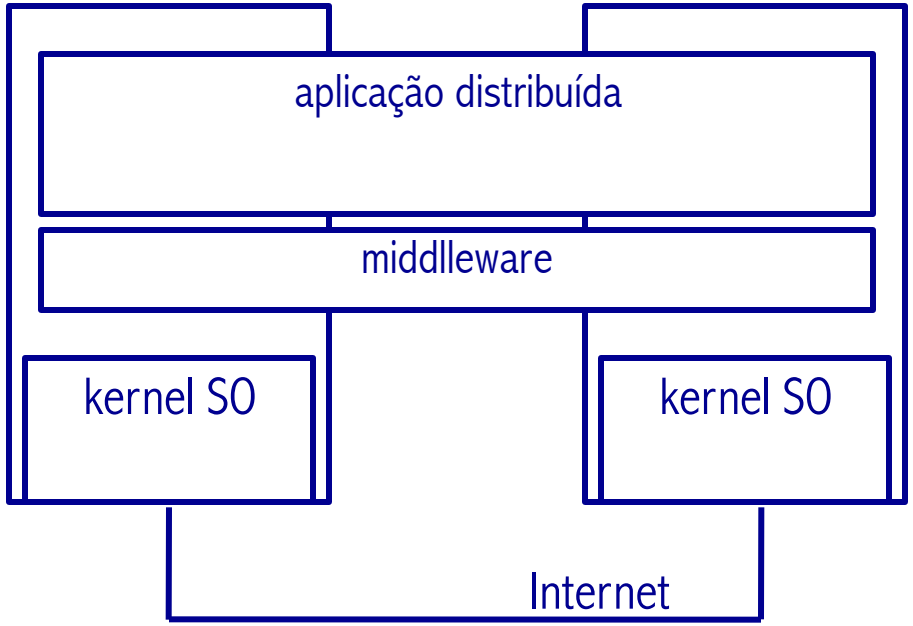
sistemas operacionais distribuídos

- interesse especial nos anos 80
- sistemas que controlariam o conjunto de recursos de várias máquinas
  - remanescentes importantes:
    - » servidores de arquivos, técnicas de segurança, ...



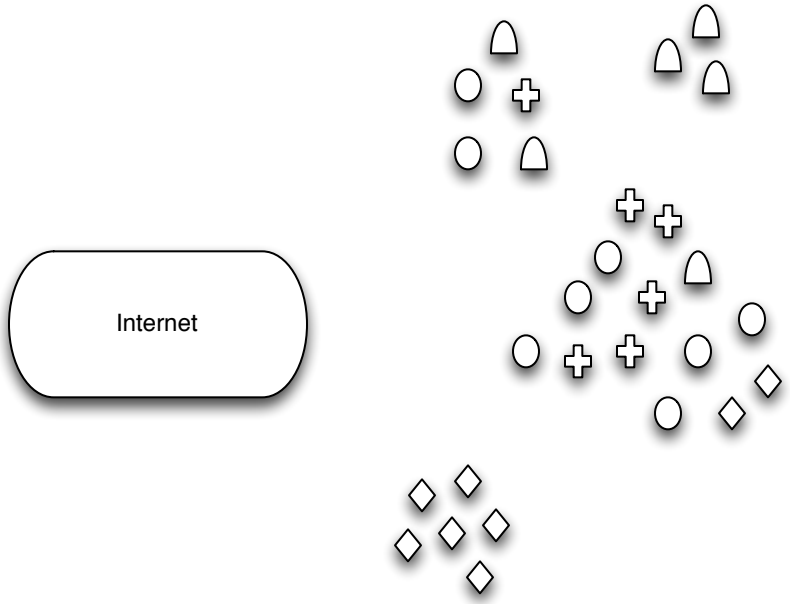
middleware

- serviços e abstrações que facilitam o desenvolvimento de aplicações distribuídas



outros cenários

IoT: arquiteturas



- interesse como sistema distribuído:
  - limitação de recursos traz “mistura de camadas”
  - insegurança
  - falhas

estudo de sistemas distribuídos

- livros clássicos de sistemas distribuídos
  - comunicação entre processos
  - sincronização
  - sistemas de arquivos
  - segurança
  - confiabilidade

- nesse curso, ênfase também na programação de SDs e em graus de acoplamento distintos

programação

- requisitos de diferentes ambientes
  - redes locais e geográficas
  - baixo e alto acoplamento
    - linguagens e bibliotecas
    - protocolos abertos
- facilidades de programação importantes para diferentes classes de aplicações
  - ou mesmo para diferentes interações dentro da mesma aplicação
- necessidades
  - modelos de programação suportados:
    - » cliente-servidor, p2p, computação móvel, ...
  - comportamento diante de falhas
  - escala e acoplamento

programa do curso

- Introdução: processos e concorrência. threads. eventos.
- Comunicacão: troca de mensagens. chamada remota de procedimentos e métodos. publish/subscribe. comunicação em grupo. código móvel
- Arquiteturas: clliente-servidor, p2p, eventos.
- Sincronização e Coordenação: relógios lógicos. ordenação de eventos, exclusão mútua. roteamento
- Replicação e consistência: multicast confiável e ordenado
- Tolerância a Falhas: comunicação confiável. recuperação de falhas.
- Outros:
  - nomes
  - segurança

discussão

- facilidade de desenvolvimento
- desempenho
- transparência
- escalabilidade
- flexibilidade

avaliação

- 3 trabalhos (implementação) – grupo
  - cliente-servidor
  - RPC
  - roteamento/coordenação
- 4 resumos e críticas de artigos – individuais
- 1 trabalho final (implementação, texto e apresentação)

- sobre prazos de entrega:
  - Cada aluno terá direito a 8 dias de atraso por semestre. Depois de gastos esses dias, trabs serão desconsiderados.

Bibliografia

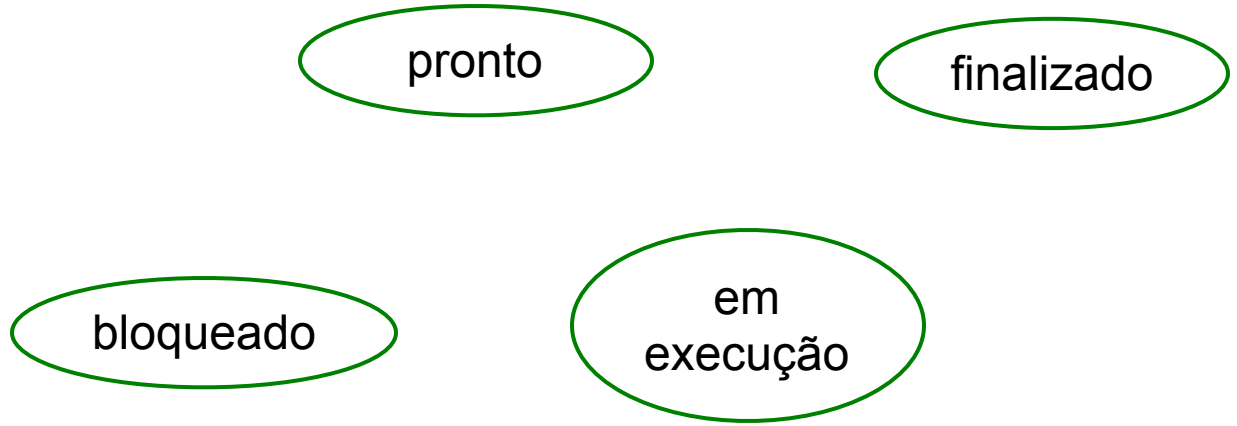
- livros SDs:
  - K. Birman. Reliable Distributed Systems. Springer, 2005.
  - A. Tanenbaum e M. van Steen. Distributed Systems: Principles and Paradigms. Prentice-Hall, 2007.
- surveys e artigos
  - alguns “clássicos”:
    - » Andrews, Gregory. Paradigms for Process Interaction in Distributed Programs. *ACM Computing Surveys* , 23(1), mar 91.
    - » Valeria Cardellini, Emiliano Casalicchio, Michele Colajanni, and Philip S. Yu. 2002. The state of the art in locally distributed Web-server systems. *ACM Computing Surveys,* 34(2), jun 2002.
    - » outros um pouco mais recentes ☺️

introdução

- processos e concorrência

Processos e Concorrência

- processo: modelo de execução sequencial
  - "programa em execução"
  - código, dados globais, pilha de execução
  - estrutura presente em qualquer sistema operacional
    - » escalonamento
    - » diferença para processos de livros de SO



concorrência

- concorrência
  - como lidar com diversas atividades simultaneamente
  - concorrência versus paralelismo
- concorrência e distribuição
  - aplicações distribuídas envolvem a execução concorrente de processos em várias máquinas
  - uso de concorrência local:
    - » atendimento a comunicações concorrentes
    - » sobreposição de comunicação e processamento

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

threads em Java

```
public void MyThread extends Thread {
  public void run() {
    ...
  }
...
Thread thr = new MyThread();
thr.start();
```

threads em Java

```
public void MyThread extends Thread {
  public void run() {
    ...
  }
...
Thread thr = new MyThread();
thr.start();
```

transferência de controle implícita!
(preempção)

condições de corrida (1)

```
class Conta {
  private int saldo;
  public Conta (int ini) {
    saldo = ini;
  }
  public int veSaldo() {
    return saldo;
  }
  public void deposita(int dep) {
    saldo += dep;
  }
}
```

condições de corrida (2)

```
public class ThreadsEnxeridas {
  public static void main(String[] args) {

    Conta cc = new Conta(0);
    (new ThreadEnxerida("1", cc)).start();
    (new ThreadEnxerida("2", cc)).start();
    (new ThreadEnxerida("3", cc)).start();
    (new ThreadEnxerida("4", cc)).start();
  }
}
```

acessos concorrentes à conta

condições de corrida (1.1...)

```
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!!
      saldo++;
    }
  }
}
```

condições de corrida (3)

```
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++) {
            saldo++;
    }
  }
}
```

modelo semelhante em pthreads

```c
#include <pthread.h>
#include <stdio.h>
#include <math.h>

void* umathread (void* arg) {
  int i; double  a;
  for (i=0;i<10000000;i++)
    a = sin(i);
  printf ("alo \n");
}

void* outrathread (void* arg) {
  int i; double a;
  for (i=0;i<10000000;i++)
    a = sin(i);
  printf ("mundo!\n");
}

int main(int argc, char *argv[])
{
  pthread_t pid1, pid2;

  pthread_create(&pid1, NULL, umathread, NULL);
  pthread_create(&pid2, NULL, outrathread, NULL);

  pthread_join(pid1, NULL);
  pthread_join(pid2, NULL);

  return 0;
}
```

alternativas ao modelo multithread clássico

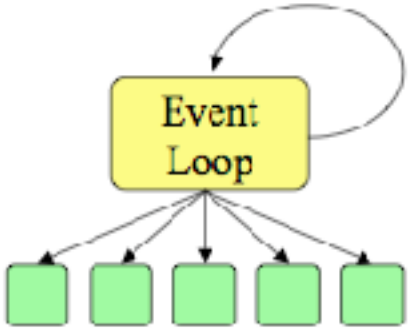
2. modelos orientados a eventos
   - cada evento tratado até o final
   - programa como máquina de estado
3. multitarefa sem preempção
   - co-rotinas
     - Lua e outros
4. multithreading com troca de mensagens
   - Erlang, Scala, ...

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*

co-rotinas em 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

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

- notas de aula Ihor Kuz, Manuel M. T. Chakravarty & Gernot Heiser (intro-notes.pdf, na página do curso*)
- J. Ousterhout. Why threads are a bad idea (for most purposes)
  - conjunto de slides (procurar no google)
- 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).
- capítulo de co-rotinas - livro de Lua
  - Roberto Ierusalimschy. Programming in Lua. 4a edição. lua.org, 2016.
    - » 1a ediçãodisponível em www.lua.org/pil/

*inf.puc-rio.br/~noemi/sd-17/

Tarefa

- Resumo 1:
  - ler os slides de Ousterhout e o trabalho de von Behren (e opcionalmente outras coisas)
  - escrever um resumo (em torno de duas páginas):
    - » explicar em que situações eventos podem substituir threads e como
    - » discutir vantagens e desvantagens de cada estilo de programação
  - enviar via sistema ead (ead.puc-rio.br) até 22/03

Atul Adya, Jon Howell, Marvin Theimer, William J. Bolosky, and John R. Douceur. 2002. Cooperative Task Management Without Manual Stack Management. In Proceedings of the General Track of the annual conference on USENIX Annual Technical Conference (ATEC '02). até 3.2 (inclusive).

» nunca entregar documentos word! pdf por favor!