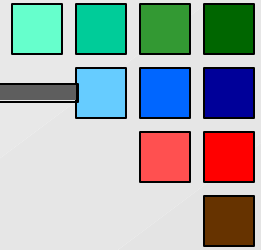
Sistemas Distribuídos

Ricardo Ribeiro dos Santos

ricrs@ec.ucdb.br

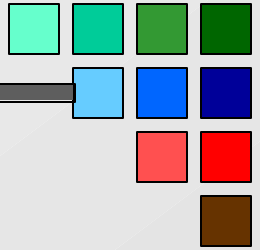
Curso de Engenharia de Computação – UCDB – Agosto/2003

Tópicos



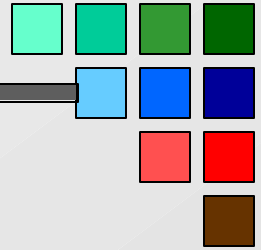
- Processos em SD
- *Threads*

Processos em SD



- Processo=Programa em execução...
- Do ponto de vista do SO:
  - Gerenciamento e Escalonamento
- Em SD há necessidade de tratar ainda outras questões...
  - Organização C/S
  - Migração dos processos

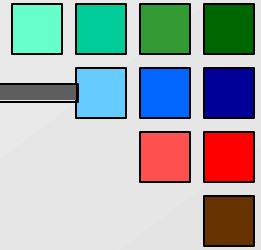
Threads em SD



- **Granularidade** dos processos limita o desempenho

- Necessidade de permitir que um processo possa ser dividido em múltiplas linhas de execução=múltiplas *threads*

Threads em SD

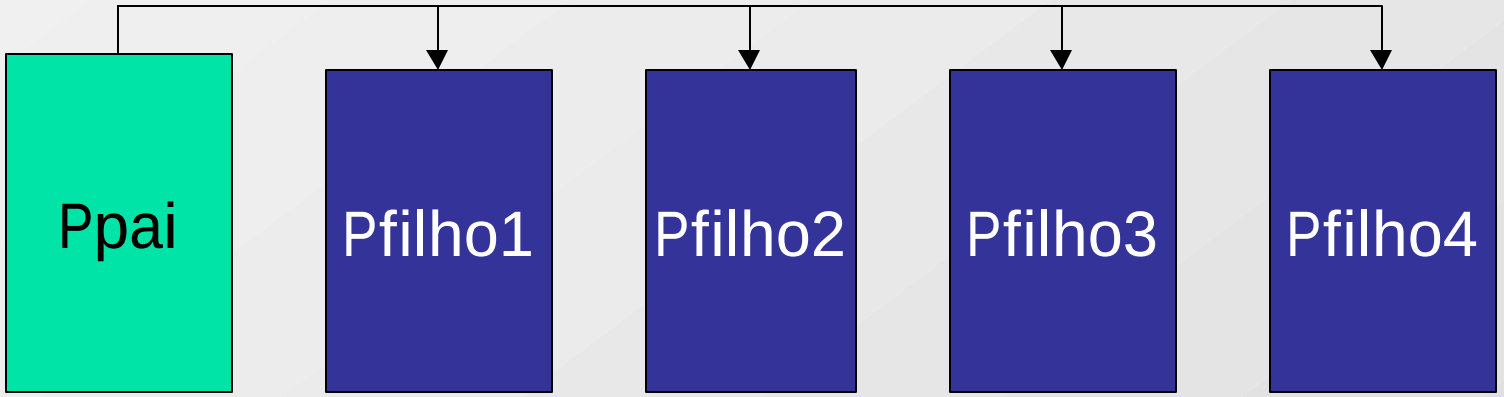


- Antes de *threads*...
  - Concorrência com granularidade “grossa”
    - Chamada *fork()*

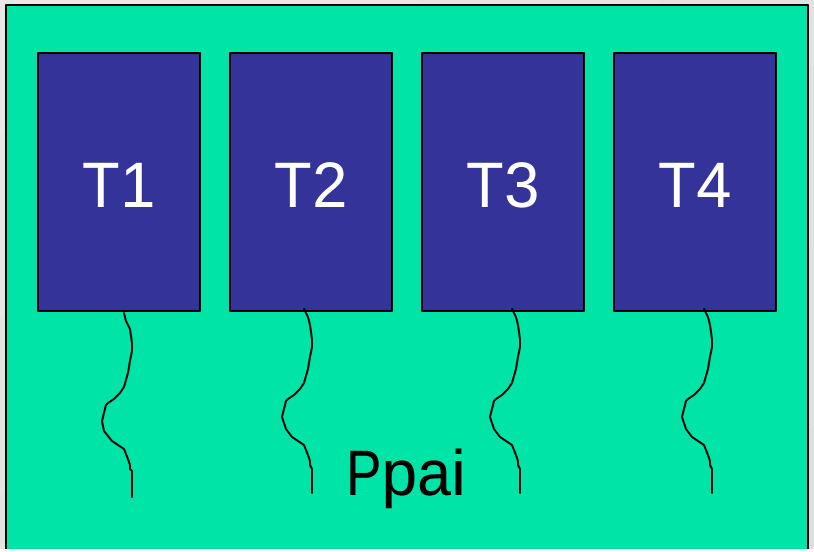
```
#include <unistd.h>
#include <stdio.h>
int main(void){
  int retorno,i;
  for (i=0;i<4;i++){
      retorno=fork();
      if (retorno==0){
          printf("\nSou o filho No.:%d\n",i);
          i=4;
      }
  }  return 0;
}
```

Threads em SD

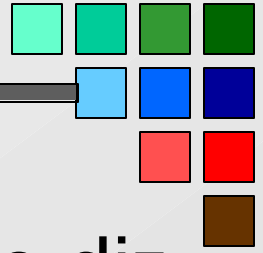
- Concorrência entre processos



- Concorrência entre *threads*



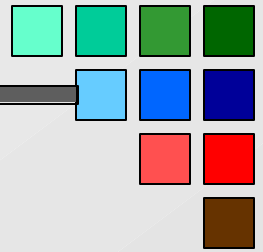
Threads em SD



- *Threads* são similares aos processos no que diz respeito à execução em processadores virtuais
- No entanto, a concorrência pode ser controlada se o desempenho é afetado!

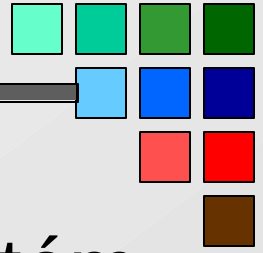
- *Threads* exigem um maior esforço intelectual e capacidade de programação
  - *Threads* não possuem os mecanismos de proteção como os processos
  - Muitas aplicações ficam mais simples se estruturadas como uma coleção de *threads*
    - Ex: Processador de Textos que manipula a entrada do usuário, verificação ortográfica, layout do documento...

Comparação *Threads* vs Múltiplos processos



- Criação de *threads* envolve menos custo computacional do que criar um novo processo;
  - Exemplo Processador CVAX:
    - 11 milisegundos para criar um novo processo
    - 1 milisegundo para criar uma nova *thread*
- *Threads* podem compartilhar recursos dentro de um processo
- Uma *Thread* não é protegida de outras dentro de um mesmo processo

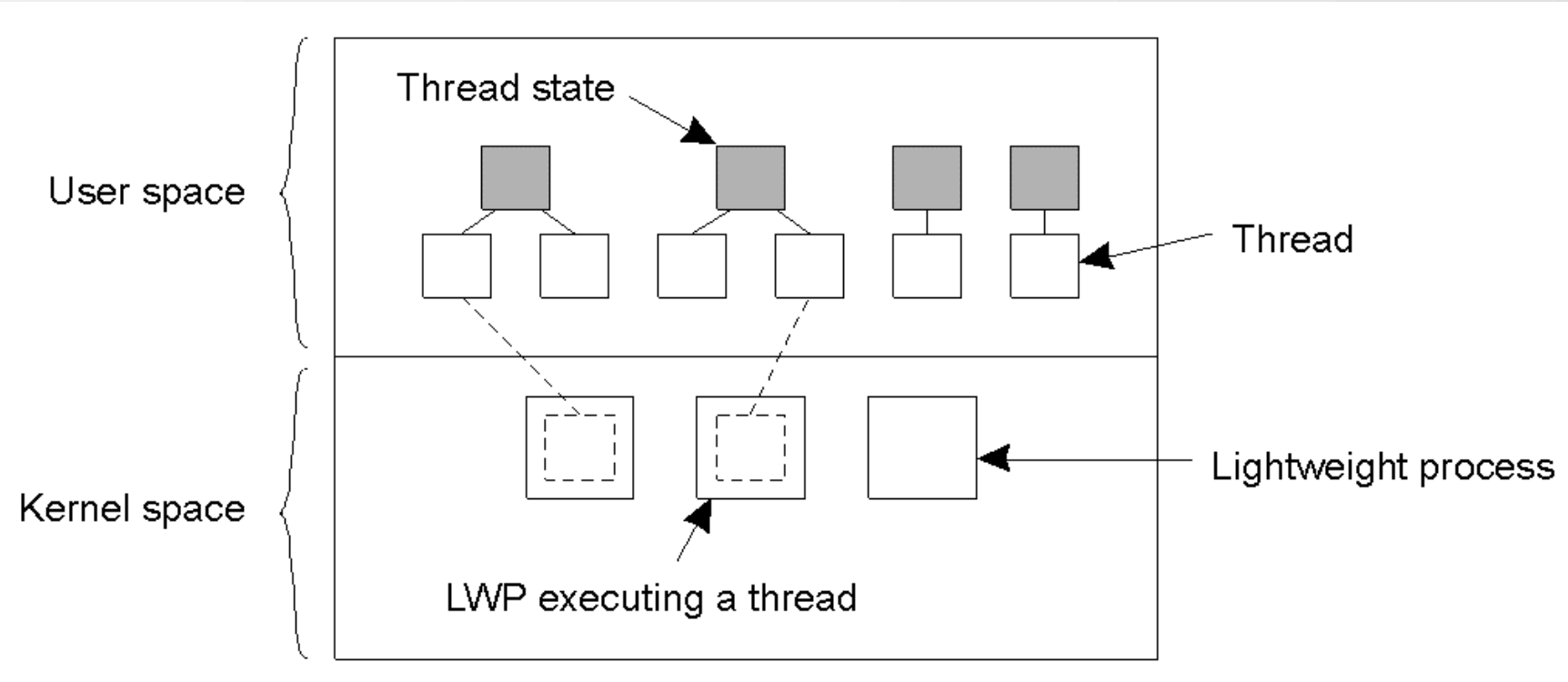
Implementação de *Threads*



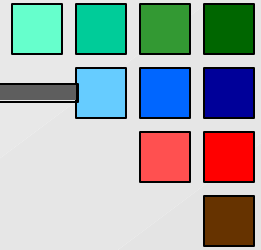
- De forma geral, os pacotes de *threads* contém operações para:
  - Criar e destruir *threads*;
  - Sincronização de *threads*;
- Duas abordagens para implementar *threads:*
  - Bibliotecas de *threads* em nível de usuário
    - Bibliotecas de *Threads:* C threads, pthreads
    - Pela linguagem: Ada, Java
  - *Threads* fornecidos pelo *kernel*
    - Ex: WinNT, Windows 2000, Solaris, Mach
- Quais as vantagens/Desvantagens?

Implementação de *Threads*

- *Threads* no espaço do usuário e no espaço do *kernel*
  - Ex: Solaris 2



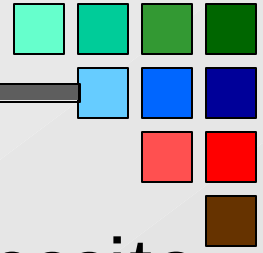
Implementação de *Threads*



- Em Java: Métodos da classe *Thread*

- ***Thread(ThreadGroup group, Runnable target, String name)***
  - Cria uma nova *thread no estado SUSPENDED*, pertencendo para *group* e identificado como *name*; a *thread* executará o método *run()* de *target*.
- ***setPriority(int newPriority), getPriority()***
  - Configurar e retornar a prioridade da *thread*.
- ***run()***
  - Método que deve ser sobreposto pelo objeto alvo
- ***start()***
  - Modifica o estado da *thread de SUSPENDED* para *RUNNABLE*.
- ***sleep(int millisecs)***
  - Faz com que a *thread entre no estado SUSPENDED* por um tempo específico
- ***destroy()***
  - Destrói a *thread*.

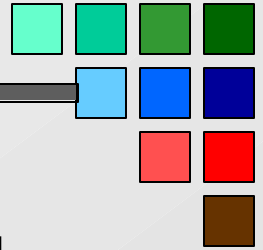
Clientes *Multithreaded*



- Em determinadas aplicações, o cliente necessita buscar e manipular dados...
- Se o processo for completamente seqüencial haverá atrasos significativos
- Concorrência através de *threads* poderá melhorar o desempenho

- Ex: *Web Browsers*
  - Após a página html ter sido buscada, *threads* podem ser ativados para buscar outras partes do documento

Servidores *Multithreaded*

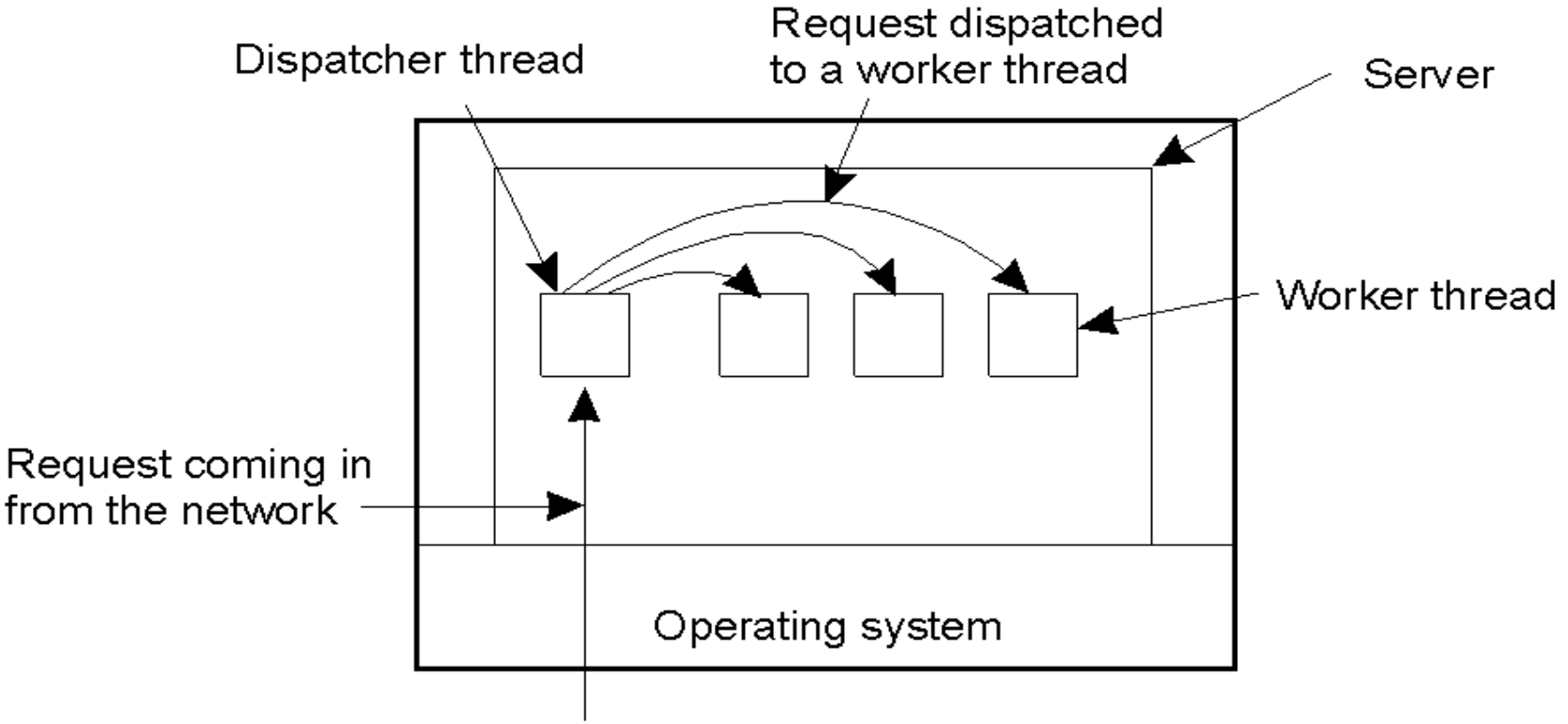


- Principal uso da idéia de *multithreads* reside nos servidores!

- Servidores podem ser:
  - Iterativos:
    - Ex: Servidor de Arquivos
      - Espera por requisições...
      - Processa a requisição
      - Envia a resposta
  - Concorrentes:
    - Recebe a requisição e “passa” a tarefa para uma *thread*

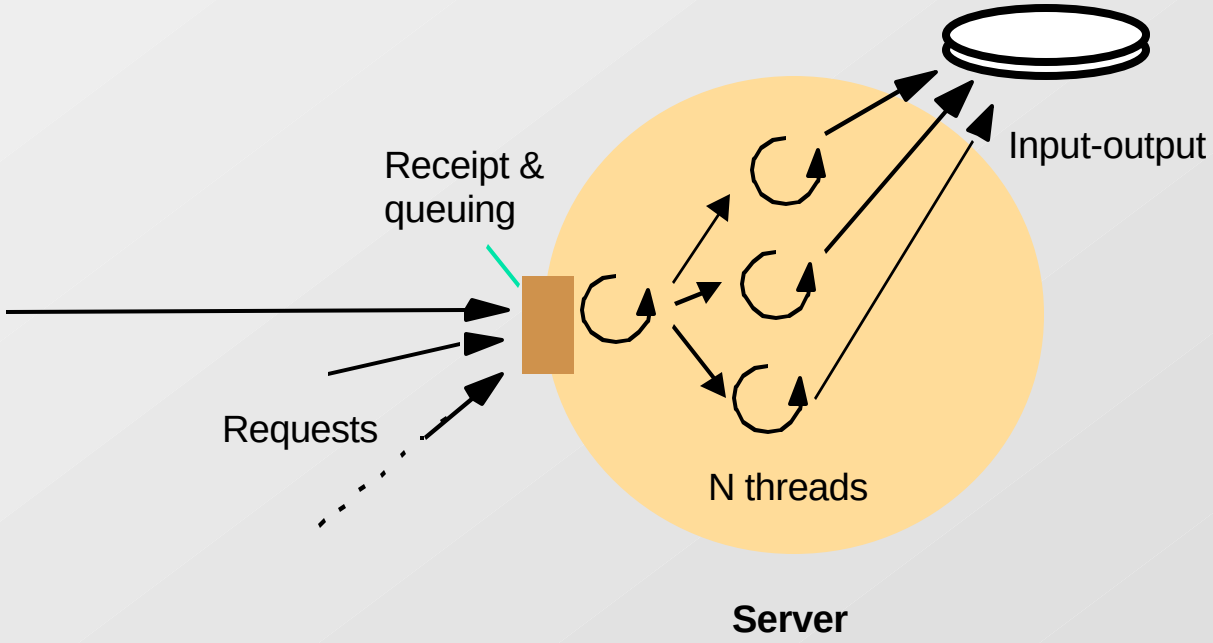
Servidores *Multithreaded*

- Uma possível organização pode fazer uso de *dispatchers*



Servidores *Multithreaded*

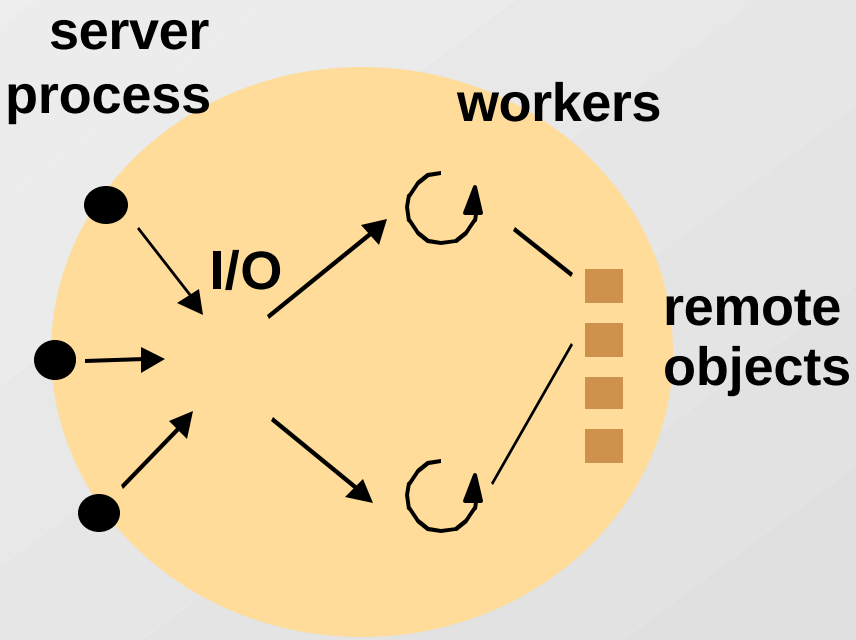
- Arquiteturas
  - Pool de *Threads ou Worker* Pool



- Servidor cria um número fixo de *threads* para processar as requisições
- Quando as requisições chegam, apenas distribui o trabalho

Servidores *Multithreaded*

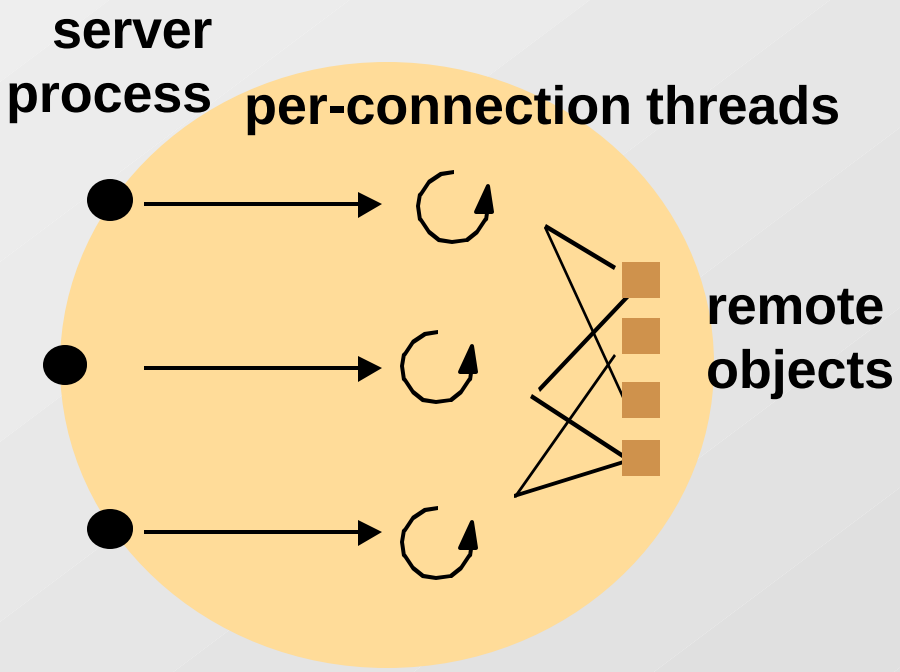
- Arquiteturas (para servidores baseados em OD)
  - *Thread* por requisição



  - *Thread de IO* gera um novo *thread* para cada nova requisição
  - Os *threads* criados são destruídos após realizarem o trabalho

Servidores *Multithreaded*

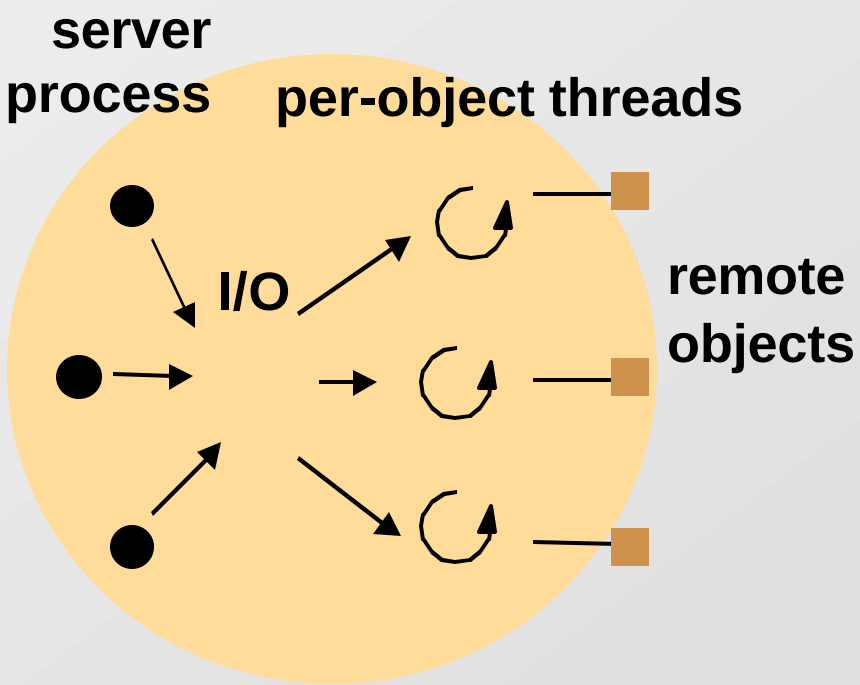
- Arquiteturas (para servidores baseados em OD)
  - *Thread* por conexão



  - Servidor cria uma nova *thread* quando um cliente estabelece uma conexão e destrói essa *thread* quando a conexão é fechada

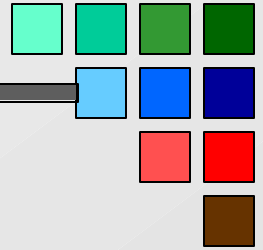
Servidores *Multithreaded*

- Arquiteturas (para servidores baseados em OD)
  - *Thread* por objeto



  - Servidor associa uma *thread* com cada objeto remoto

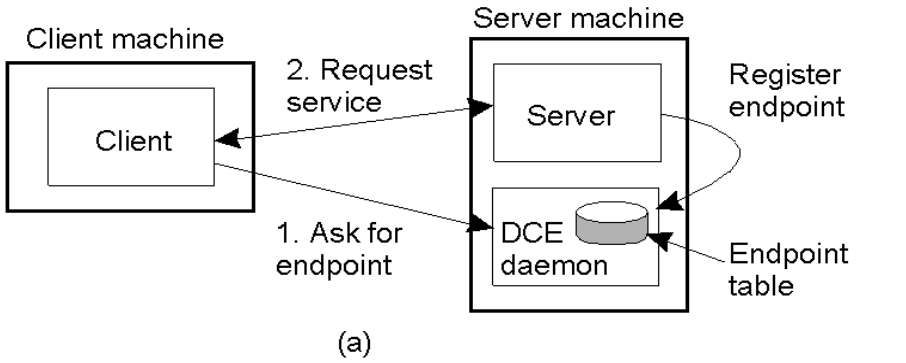
Servidores *Multithreaded*



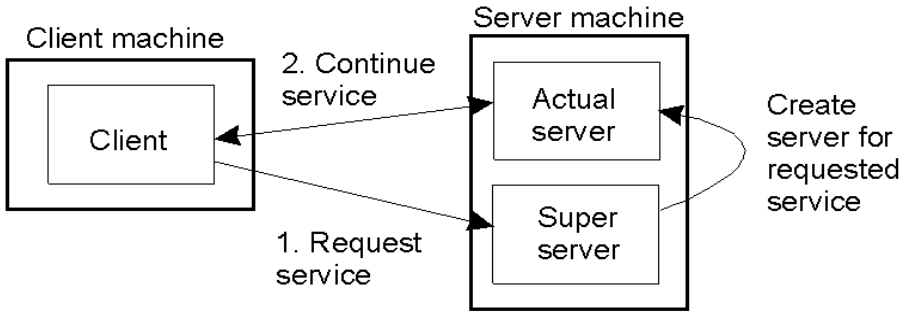
- Algumas outras questões...
  - Como os clientes encontram os servidores?
  - Algumas alternativas:
    - Servidores “escutam” em portas bem conhecidas
      - E se a porta já estiver sendo usada?
    - Servidores utilizam portas livres
      - Como o cliente irá saber qual porta contactar?

Servidores *Multithreaded*

- Algumas outras questões...
  - Como os clientes encontram os servidores?
  - Ex: DCE e *Superserver* em Unix

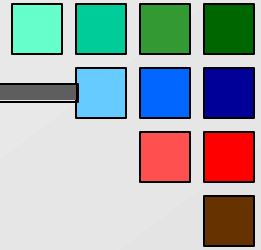


(a)



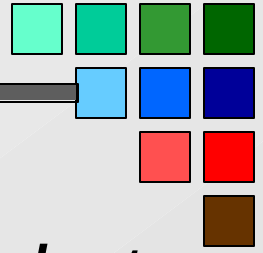
(b)

Servidores *Multithreaded*



- Algumas outras questões...
  - O servidor deve armazenar o estado dos clientes?
  - Servidores *stateless*: Não mantém informações sobre os clientes
    - Ex: Servidores Web
  - Servidores *statefull:* Armazena informações sobre os clientes
    - Ex: Servidores de arquivo

Aplicando *Threads*



- Utilizar *threads* no servidor que utiliza *Sockets*
- Após receber uma conexão, o tratamento da requisição deve ser feito por uma *thread*