Sistemas Distribuídos

Processos

Edeyson Andrade Gomes
www.edeyson.com.br

Fluxos de Execução (THREADS)

- ***Sistemas tradicionais***
  - Processos com único **Espaço de Endereçamento Virtual** (EEV) e único **Fluxo de Execução**
  -
- ***Sistemas distribuídos***
  - EEV único com vários fluxos de execução - THREADS
    - Paralelismo para processos
  - Cooperação de THREADS para a execução de tarefas
    - Servidor de Chat, de Arquivos, etc.

Fluxos de Execução (THREADS)

- **((THREAD))**
  - Seqüência independente de comandos de um programa.
    - Fluxo de Execução

Fluxos de Execução (THREADS)

- **Principais aplicações:**
  - Processamento paralelo
    - CPU bound: ganho com várias CPU
    - I/O bound: ganho sempre
      - Se uma thread está bloqueada a outra pode executar
  - Aplicações que acessam dispositivos secundários lentos (disco local ou remoto) e não querem ficar esperando – síncronos - pela resposta para poder continuar executando.
  - Aplicações que controlam múltiplos servidores ou múltiplos clientes.
  - Melhorar funcionalidade e performance da interface gráfica.

Fluxos de Execução (THREADS)

- **Características:**
  - *Herança* de recursos alocados pelo processo.
    - Menos custo ao SO para instanciação.
      - Arquivos, Portas de Comunicação, Memória, Ponteiros
  - Possuem PC, SP, contexto, prioridade, etc;
  - Disputam, entre sí, pelo processador.
    - Podem executar em paralelismo real – multiprocessamento – ou virtual – multiprogramação.

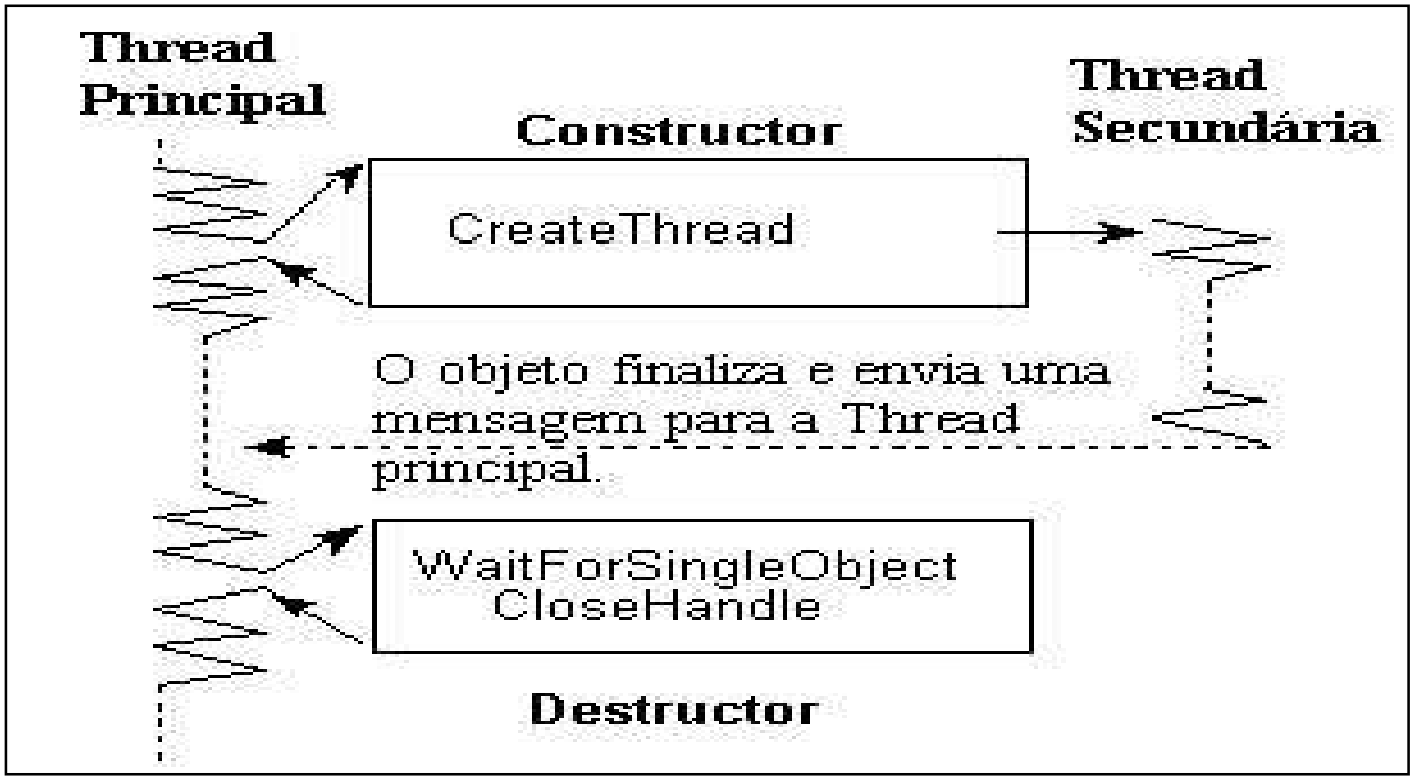
Fluxos de Execução (THREADS)

- ((**THREADS**))
  - Pode criar threads filhas (secundárias).

```
HANDLE hThread; . . .
hThread  =  CreateThread(...);
CloseHandle( hThread );
```

  THREAD secundária
    - Remoção automática após término da função.

Fluxos de Execução (THREADS)



Fluxos de Execução (THREADS)

- Diferenças de Processos Convencionais
  - THREADS de um mesmo processo compartilham o mesmo EEV.
  - Não existe proteção entre threads de um mesmo processo
    - Uma Thread pode ler, escrever, ou até mesmo eliminar uma pilha de outra Thread do mesmo processo
  - Threads de um mesmo processo herdam desse os arquivos abertos, temporizadores, sinais, semáforos, variáveis globais, etc.

- **Uso de THREADS**
  - Paralelismo combinado com execução serial

Fluxos de Execução (THREADS)

- Metas:
  - Performance
  - Vazão (Throughput)

- Performance
  - Tempo total de computação
  - Criação de Threads e Troca de Contexto
    - OVERHEAD
  - CPU-Bound
    - Única Thread ou Múltiplas?
    - Sincronização.

Fluxos de Execução (THREADS)

- Modelos de Alocação
  - **Dinâmica**
    - **Thread é alocada por demanda**
      - **Cria, usa e descarta**
    - **Boa escalabilidade**
      - **Pode criar novas até o limite do SO**
    - **Desempenho é reduzido com tempo de criação da Thread**
      - **D:\Aulas\Programas SO WIN\Threads\Threads Dinâmicas para Primos**

Fluxos de Execução (THREADS)

- Modelos de Alocação
  - **Estática**
    - **Threads são pré-alocadas**
      - **Aumenta o desempenho eliminando o overhead de criação antes do uso**
      - **Baixa escalabilidade pois o número de Threads é predeterminado**
        - **D:\Aulas\Programas SO WIN\Threads\Threads Estáticas para Primos**
  - **Híbrido**
    - **Pré-aloca n Threads**
    - **Se todas forem usadas, aloca mais por demanda**

Fluxos de Execução (THREADS)

- Modelos de Programação
  - **Gerente / Trabalhador**
    - Thread principal – Gerente
    - Atribuição de tarefas às Threads Trabalhadoras
    - Thread Trabalhadora sinaliza estado – Livre ou Ocupada.
    - Filas – Disponibilização, pelo Gerente, de novas tarefas.
  - **Trabalho em Grupo**
  - **Linha de Montagem**

Fluxos de Execução (THREADS)

- Modelos de Programação
  - **Gerente / Trabalhador**
  - **Trabalho em Grupo**
    - Macro-Tarefas são divididas em micro-tarefas (Threads) concorrentes

Fluxos de Execução (THREADS)

- Modelos de Implementação
  - Nível do Kernel
  - Nível do Usuário
  - Multiplexação

  - Um SO monotarefa pode executar aplicações com múltiplas Threads?

Fluxos de Execução (THREADS)

- Modelos de Implementação
  - Nível do Kernel
    - Modelo 1:1
      - (Ex. Windows 2000, XP, Unix)
    - Thread visível ao processo possui uma Thread correspondente no Kernel.
  - Nível do Usuário
    - Modelo N:1
      - Exemplo: DOS + Windows 3.11
    - Threads visíveis ao processo são mapeadas como uma simples Thread no Kernel.
  - Multiplexação
    - M threads de processo multiplexadas para N thread Kernel.

Fluxos de Execução (THREADS)

- **Implementação no espaço de endereçamento do usuário (N:1)**
  - O sistema operacional não precisa suportar threads
    - Por exemplo, SO monoprogramado
  - Threads executam sobre o sistema de run-time
    - O Kernel possui uma única Thread
    - O run-time escalona as Threads visíveis ao programador para a Thread do Kernel

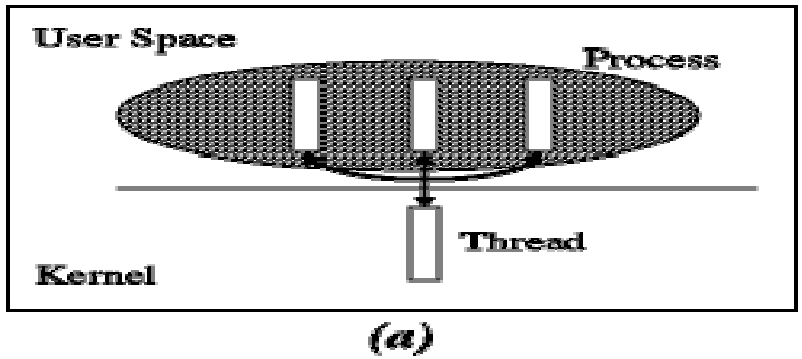
Fluxos de Execução (THREADS)

- **Implementação no espaço de endereçamento do usuário (N:1)**
  - A troca de contexto é extremamente rápida - entre threads de um mesmo processo - até que outro seja escalonado
  - Escalonamento personalizado
  - Melhor escalabilidade
    - Facilidade de alocação de espaço para tabelas e pilhas no espaço do usuário
  - Problemas no bloqueio de threads
  - Problemas na utilização do tempo do processador

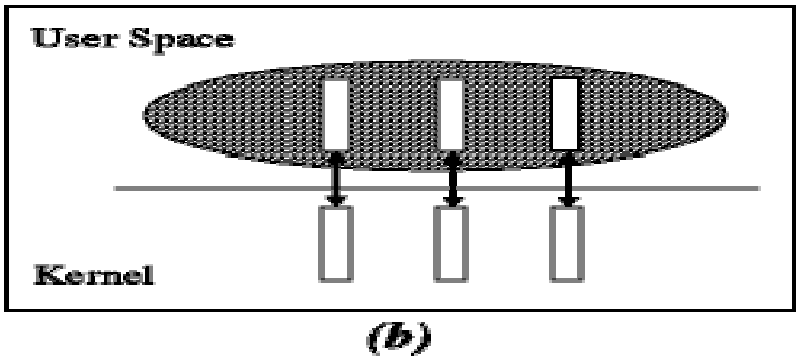
Fluxos de Execução (THREADS)

- Implementação no Núcleo (1:1)
  - Não é necessário um sistema de run-time
  - Uma tabela de threads por processo
  - Troca de contexto pode ser feita entre threads de processos diferentes
  - Bloqueio de threads simplificado
  - Execução preemptiva

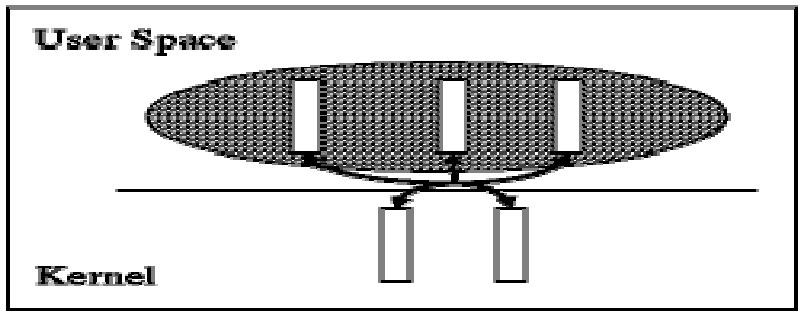
Fluxos de Execução (THREADS)



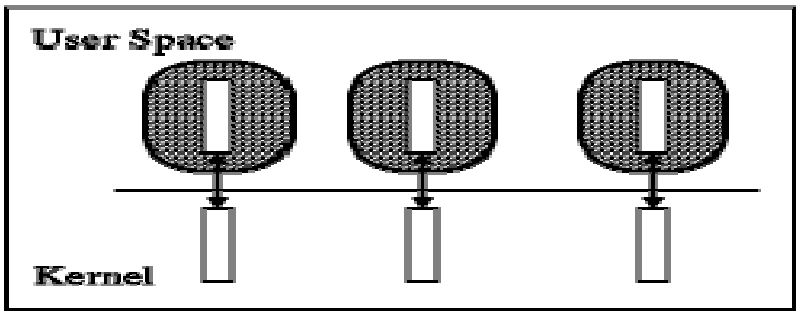
(a)



(b)



(c)



(d)

(a) N: 1

(b) 1:1, com um contexto de processo

(c) M:N

(d) Múltiplos contextos de processo

Exemplos de Utilização

- Implementação de Servidor de Arquivos.
  - Thread única
    - Requisições atendidas em série
    - Baixo Throughput
  - Múltiplas Threads
    - Modelo Gerente/Trabalhador
    - O Gerente recebe as requisições dos clientes e as encaminha para as Threads de trabalho
    - Cada Thread pode atender a uma requisição diferente

Exemplos de Utilização

- Implementação de Servidor de Arquivos.
  - Múltiplas Threads
    - Várias Threads podem executar ao mesmo tempo, compartilhando o processador
    - Enquanto uma thread está bloqueada, outra thread pode ser posta para executar
    - Executam em paralelo em sistemas multiprocessados
    - Todas as Threads compartilham uma mesma cache de arquivo

Sincronização de Threads

- Sincronização via sinais ou semáforos
  - Thread fica bloqueada esperando sinais
  - O bloqueio da Thread – caso essa não seja a única do processo – NÃO bloqueia o processo
  - Modelo Produtor-Consumidor
    - Paralelismo Inerente
    - Compartilhamento de recurso
      - Buffer de mensagens
    - Utilização de Threads maximiza a utilização do Buffer

Gerência de Threads

- Threads Estáticas
  - O número de threads é definido quando o programa é escrito ou compilado
  - Uma pilha de tamanho fixo é associada a cada thread
  - Menor flexibilidade
- Threads Dinâmicas
  - Threads criadas e destruídas em tempo de execução
    - Alocação por demanda
  - O tamanho da pilha é passado como parâmetro para a chamada de sistema de criação de thread

Compartilhamento de Dados

- Área de dados global do Processo é compartilhada por suas Threads
  - Dados armazenados num espaço de endereçamento comum
- Regiões Críticas controladas através de Exclusão Mútua
  - Utilização de semáforos binários - mutex
  - Operações sobre a variável mutex:
    - UP e Down - modelo padrão
    - *LOCK*, UPLOCK e TRYLOCK (falha no down)
  - Variáveis condicionais para alocação de recursos
    - Sincronizadores de eventos.
    - Wait (condição)
    - WakeUp (condição)

Exemplos

Threads Concorrentes

```
lock mutex;
   verifica estruturas de dados;
   while (ocupado(R1))
        wait (C1)
   R1 = ocupado
unlock mutex;
```

```
lock mutex;
    R1 = livre;
unlock mutex;
wakeup (C1);
```

While força o TESTE.

Exemplos

Threads Concorrentes – MS Visual C++

```
HANDLE hMutex;
HANDLE hEmptySem;
HANDLE hFullSem;

hMutex = CreateMutex(NULL, FALSE, “Buffer_Mutex");
hEmptySem = CreateSemaphore(NULL,  MAX_ITEM,
                    MAX_ITEM,  "Empty_Sem")
hFullSem = CreateSemaphore(NULL,   0,
                    MAX_ITEM,  "Full_Sem")
```

Exemplos

Threads Concorrentes

((PRODUTOR))

```
WaitForSingleObject(hEmptySem, INFINITE);
WaitForSingleObject(hMutex, INFINITE);
        enter_item(item);
ReleaseMutex(hMutex);
ReleaseSemaphore(hFullSem, 1, NULL);
```

Exemplos

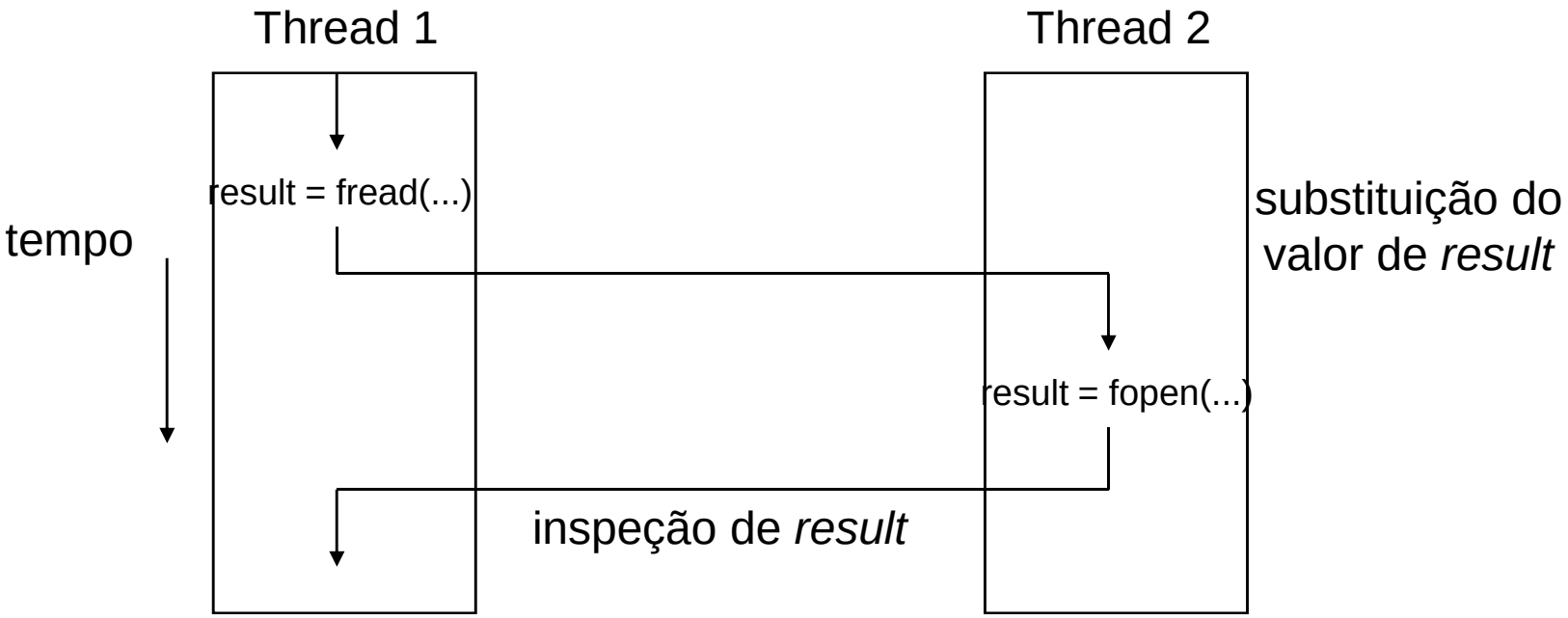
Threads Concorrentes

((CONSUMIDOR))

```
WaitForSingleObject(hFullSem, INFINITE);
WaitForSingleObject(hMutex, INFINITE);
        remove_item(item);
ReleaseMutex(hMutex);
ReleaseSemaphore(hEmptySem, 1, NULL);
```

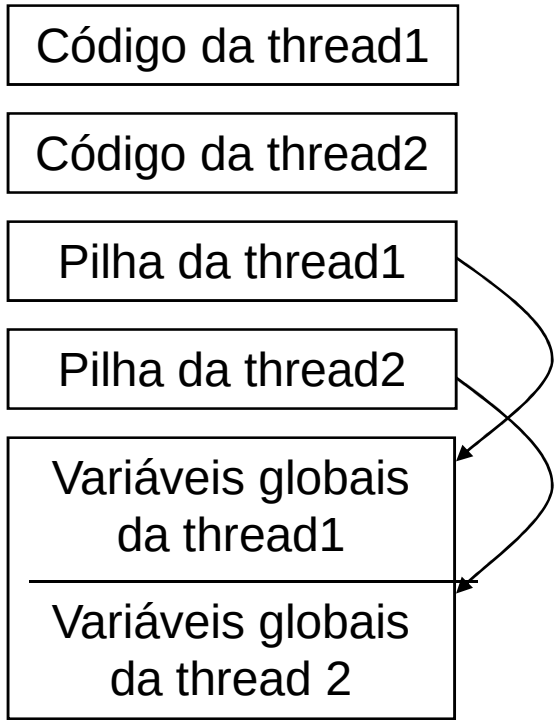
Compartilhamento de Dados

- Variáveis globais de uma thread
  - O compartilhamento de variáveis globais pode gerar conflitos (result controla o resultado da operação)



Compartilhamento de Dados

- Soluções para problemas de compartilhamento de variáveis globais
  - Proibição do uso de variáveis globais
  - Variáveis privadas globais de cada thread
    - Limitação de linguagens - Cópia



Escalonamento

- Mesmos algoritmos de escalonamento de processos convencionais
  - prioridade
  - round-robin
- Normalmente definido pelo usuário
- Exemplos:

```
hThread1 = CreateThread (NULL, 0,
      (LPTHREAD_START_ROUTINE)ThreadProc,  (LPVOID)pColor1, 0,
      (LPDWORD)&ThreadID1);

SetThreadPriority (hThread1, THREAD_PRIORITY_IDLE);

SuspendThread (hThread1);

ResumeThread (hThread1);
```

Modelos de Sistemas

Modelos de Sistemas

- **Organização de processadores em Sistemas Distribuídos.**
  - Modelos:
    - Estações de Trabalho.
    - Pool de Processadores.

- ***Modelo de Estações de Trabalho***
  - O sistema é constituído de estações de trabalho conectadas pela rede de comunicação

Modelos de Sistemas

- ***Modelo de Pool de Processadores***
  - O sistema é constituído por um servidor de processamento (pool) com múltiplas CPUs.

- ***Modelo híbrido***

Modelo de Estações de Trabalho

- **Tipos de Estações de Trabalho**
  - Dedicadas ou compartilhadas
    - Com usuário ou ociosa.
  - Com disco ou sem disco

Modelo de Estações de Trabalho

- **Estações sem Disco**
  - ***Vantagens:***
    - Menor Custo
    - Manutenibilidade
      - Distribuição de novas versões de software
      - Backup centralizado
    - Simetria e flexibilidade
      - Uso igual em qualquer estação
      - Transparência.

Modelo de Estações de Trabalho

- **Estações com Disco**
  - Maior Custo

- **Utilização dos Discos:**
  - Paginação e arquivos temporários
    - A memória virtual é local, evitando a requisição de SWAP via rede.
    - Redução de carga na rede.
    - Arquivos temporários podem ser criados localmente : compilações, etc.

Modelo de Estações de Trabalho

- **Estações com Disco**
  - Maior Custo

- **Utilização dos Discos:**
  - + Arquivos binários.
  - Código executável de aplicativos é armazenado localmente
  - Reduz ainda mais a carga na rede
  - Complexidade adicional na atualização de software:
    - Distribuição automática de versões
    - Controle de versões

Modelo de Estações de Trabalho

- **Utilização dos Discos:**
  - *Memória Cache de Blocos*
    - Problemas de consistência de cache
    - Maior performance
  - Sistema de arquivos completo
    - Minimiza tráfego na rede
    - Elimina a necessidade de servidores de arquivo
    - Sistema operacional de rede: possibilidade de acesso a arquivos remotos
    - Perda de transparência

Modelo de Estações de Trabalho

- **Vantagens do Modelo**
  - Simplicidade
  - Tempo de resposta garantido
    - Poder de computação fixo para cada usuário
  - Autonomia do usuário
  - Maior disponibilidade do sistema com discos locais
    - Uma falha no servidor de arquivos não inviabiliza a utilização do sistema

Modelo de Estações de Trabalho

- **Desvantagens do Modelo**
  - Desperdício de Recursos
    - Estações ociosas.
    - O poder computacional de estações ociosas não é utilizado por usuários (ou processos) que necessitam de capacidade extra.

Utilização de Estações Ociosas

- Em períodos de pico, a ociosidade de estações chega a 30% ou mais.

- Execução remota de tarefas
  - comando *rsh* do Unix BSD:

```
rsh máquina comando
```

Utilização de Estações Ociosas

- **Problemas com essa abordagem**
  - O usuário faz o balanceamento da carga
    - Informa qual máquina usar.
  - O programa executa no ambiente remoto
    - Ambientes possivelmente diferentes.
  - Máquina remota passa a ser utilizada por usuário
    - O que fazer com o processo remoto?

Utilização de Estações Ociosas

- **Problemas a resolver:**
  - Como achar uma estação ociosa?
  - Como executar um processo remoto de forma transparente?
  - O que fazer quando o usuário da estação volta a utilizá-la?

- **Achando uma estação ociosa**
  - O que é uma estação ociosa?
  - Cada sistema tem uma visão diferente:
    - Estação sem nenhum usuário conectado?
    - uma estação sem nenhum processos de usuário executando?
    - Diferenças na carga de duas estações ociosas

Utilização de Estações Ociosas

- **Algoritmos para localizar estações ociosas:**

- **Dirigidos pelos servidores**
  - A máquina ociosa anuncia seu estado
  - ***Registro***
    - Replicação do registro
    - REMOTE command
    - Executa o comando numa máquina ociosa registrada

Utilização de Estações Ociosas

- **Algoritmos para localizar estações ociosas:**

- **Dirigidos pelos servidores**
  - ***Disseminação de ociosidade via broadcast***
    - Todas as máquinas mantém o arquivo de máquinas ociosas
    - Localização de máquina ociosa numa tabela local

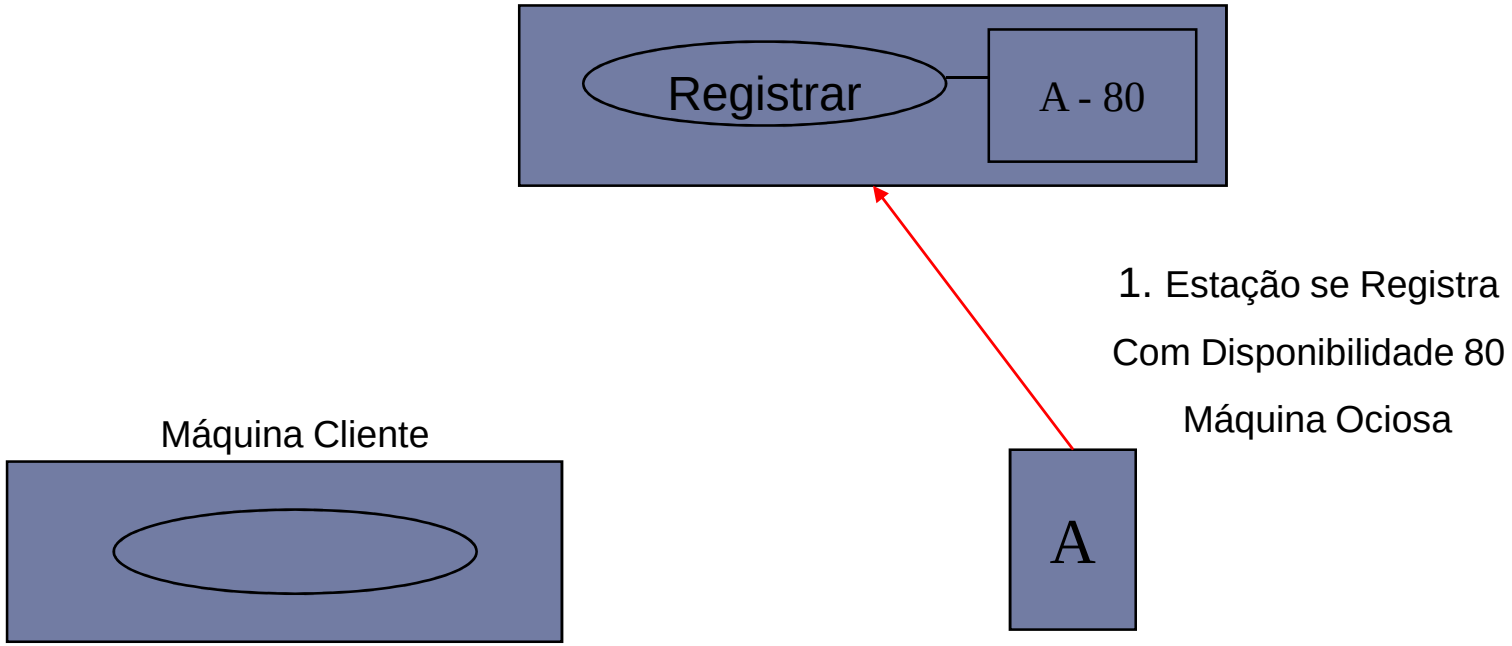
  - **Possibilidade de condições de corrida**
    - *REMOTE* apaga *o* registro da máquina ociosa escolhida do registro local sempre que seu processador for utilizado

Utilização de Estações Ociosas

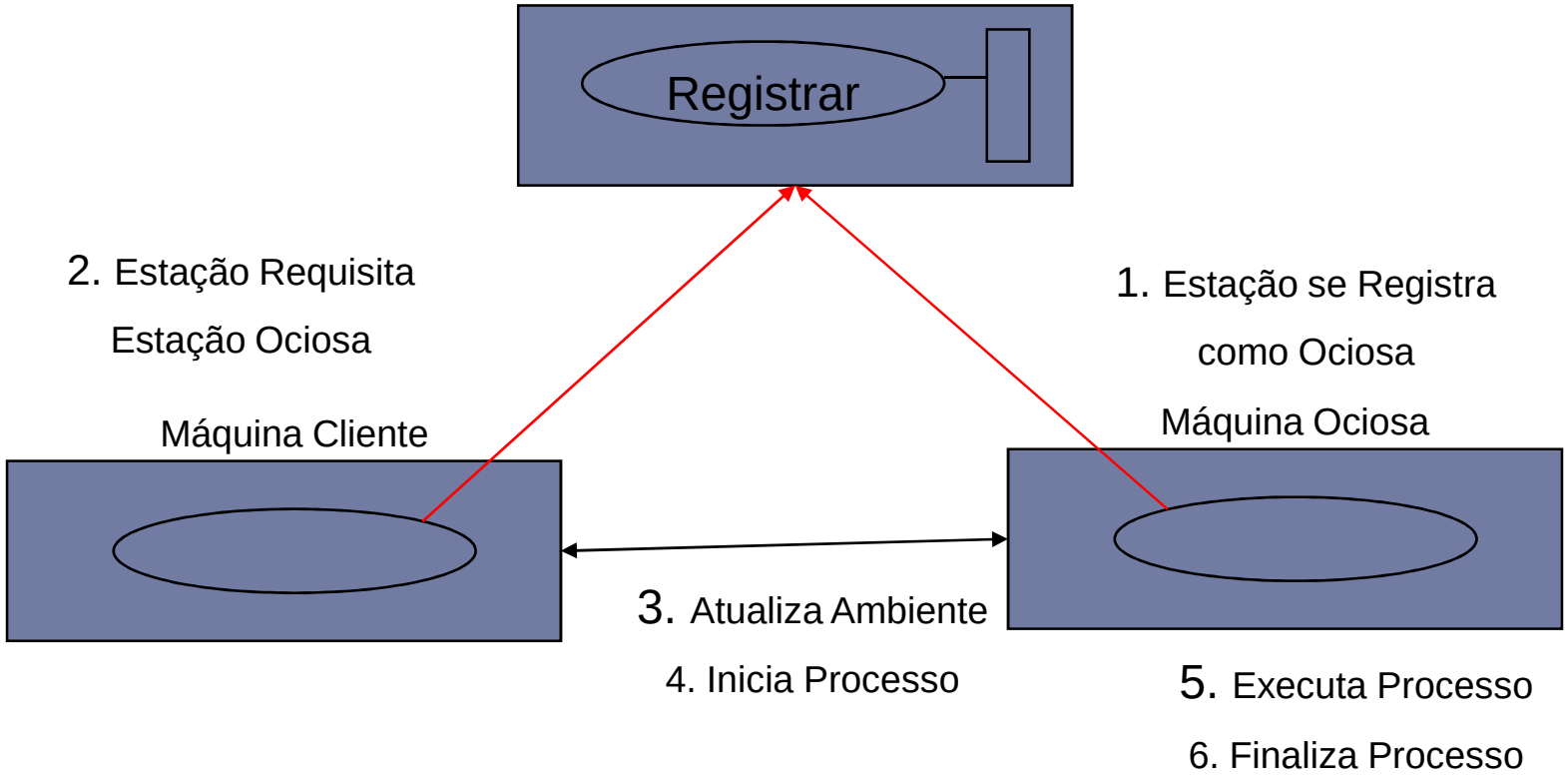
- **Algoritmos para localizar estações ociosas:**

- Dirigidos pelos Clientes
  - *REMOTE* dissemina pedidos para estações ociosas
  - em sistemas heterogêneos, os pedidos podem conter os requisitos (memória, co-processador, etc) para a execução do processo e as máquinas que cumprirem tal requisito respondem
  - Pode-se retardar a resposta das máquinas ociosas - quanto maior a carga de trabalho, maior o atraso da resposta-.
    - Respostas de máquinas mais ociosas chegam primeiro

Utilização de Estações Ociosas



Utilização de Estações Ociosas



Utilização de Estações Ociosas

- **Executando um processo remoto**
  - Movimentação do código
  - Igualar o ambiente de execução
    - Visão do sistema de arquivos
    - Diretório de trabalho
    - Variáveis de ambiente
  - Chamadas ao Sistema (System Call)
    - READ? O que o Kernel faz?
    - Sistema de arquivo centralizado X distribuído
    - Clientes com ou sem disco
      - processamento local

Utilização de Estações Ociosas

- **Evitando carga remota indesejada**
  - Desprezar
    - Acaba com a noção de estação pessoal
  - Cancelar tarefas remotas
    - Perda de processamento já realizado
    - Possibilidade de inconsistências
    - Tratamento de interrupções: um sinal é enviado ao processo remoto, solicitando que ele termine
    - Necessidade de deixar a máquina como encontrou
  - Migração de tarefas remotas
    - Complicado: como migrar as estruturas do núcleo?

Modelo do *Pool* de Processadores

- Alocação dinâmica de processadores a processos
- Usuários necessitam de interface gráfica de alta qualidade e bom desempenho

- **Motivação:**
  - Custo mais acessível de processadores, podendo existir um número maior de processadores do que de usuários
  - A idéia de centralizar o processamento da mesma forma que foi feito com o armazenamento
- Melhor relação custo/desempenho
- Facilidade de crescimento incremental

Modelo do *Pool* de Processadores

- Teoria das filas
  - TE - Taxa de Entrada (requisições/segundo)
  - TS - Taxa de Saída (requisições/segundo)
  - Para um sistema estável, necessita-se de:
    - TS > TE
  - Tempo médio de resposta é: T = 1 / (TS - TE)

- Qual o tempo de resposta de um *pool* de processadores com *n* processadores?
  - $T_n = 1 / (n\,TS - n\,TE) = T / n$

Modelo do *Pool* de Processadores

- O resultado anterior mostra que:
  - O pool de processadores pode dar um ganho linear de performance proporcional ao número de processadores.
- Por que ter então um sistema distribuído?
  - custo
  - menor variância no tempo de resposta
- Um *pool* de *n* processadores não é a mesma coisa que um processador *n* vezes mais rápido