Programação Concorrente e Paralela

Capítulo 1 – Introdução

Marcial Porto Fernández
marcial@larces.uece.br

Semestre 2018.2

Agradecimentos

- Os slides desse curso foram adaptados dos slides do livro “Concurrent Programming in Java” de Doug Lea
- Curso do Prof. André Luis Meneses Silva (UFPE) pela tradução dos termos

Objetivos do Curso

- Relembrar conceitos de concorrência em Sistemas Operacionais
- Desenvolver programas com o conceito de concorrência
- Introduzir conceitos de Sistemas Distribuídos
- Apresentar técnicas de programação concorrente em C, Python e Java
- Introduzir sistemas de computação paralela

Sumário

- O que é Programação Concorrente?
- Motivação
- Conceitos
- Processos
- Threads
- Propriedades
  - Segurança (Safety)
  - Progresso (Liveness)

Sumário

- O que é Programação Concorrente?
- Motivação
- Conceitos
- Processos
- Threads
- Propriedades
  - Segurança (Safety)
  - Progresso (Liveness)

O que é Programação Concorrente?

"Um programa concorrente é um conjunto de programas sequenciais comuns que são executados em um paralelismo abstrato"

(M.Bem-Ari)

O que é Programação Concorrente?

"Um programa concorrente especifica dois ou mais processos que cooperam para realizar uma tarefa. Processos cooperam através de comunicação; da utilização de variáveis compartilhadas ou troca de mensagens"

(G. R. Andrews)

Sumário

- O que é Programação Concorrente?
- Motivação
- Conceitos
- Processos
- Threads
- Propriedades
  - Segurança (Safety)
  - Progresso (Liveness)

Motivação Programação Concorrente

- Melhorar o desempenho das aplicações
- Aproveitar hardware com múltiplos processadores
- Atender a vários usuários simultaneamente
- Aumentar a disponibilidade de serviço para o usuário
- Implementar programas complexos (paralelamente)

Sumário

- O que é Programação Concorrente?
- Motivação
- Conceitos
- Processos
- Threads
- Propriedades
  - Segurança (Safety)
  - Progresso (Liveness)

Conceitos

- Paralelismo
  - Processamento simultâneo físico
- Concorrência
  - Processamento simultâneo lógico (aparente)
  - Entrelaçamento (interleaving) de ações
- Processo
  - Execução de uma parte do programa
- Programa Concorrente
  - Programa constituído por vários processos que cooperam para a realização de uma tarefa

Conceitos

- Comunicação
  - Variáveis compartilhadas
  - Passagem de mensagens
- Sincronização
  - Exclusão mútua de seções críticas
  - Sincronização por condição
- Estado de um programa concorrente depende:
  - Valores das variáveis (explícitas/implícitas)
  - Ponto de execução (comando que muda o estado)
- Ações atômicas
  - Conjunto de ações indivisível

Sumário

- O que é Programação Concorrente?
- Motivação
- Conceitos
- Processos
- Threads
- Propriedades
  - Segurança (Safety)
  - Progresso (Liveness)

Processos

- Um processo é um programa que está em algum estado de execução
- Tem espaço de endereçamento próprio, que é mapeado pelo S.O. na memória física
- Possui um fluxo de controle único
  - Mantém um contador de programa (PC) que indica o endereço da próxima instrução
- Entradas e saídas bem definidas.

Processos

Espaço de Endereçamento Lógico de um Processo

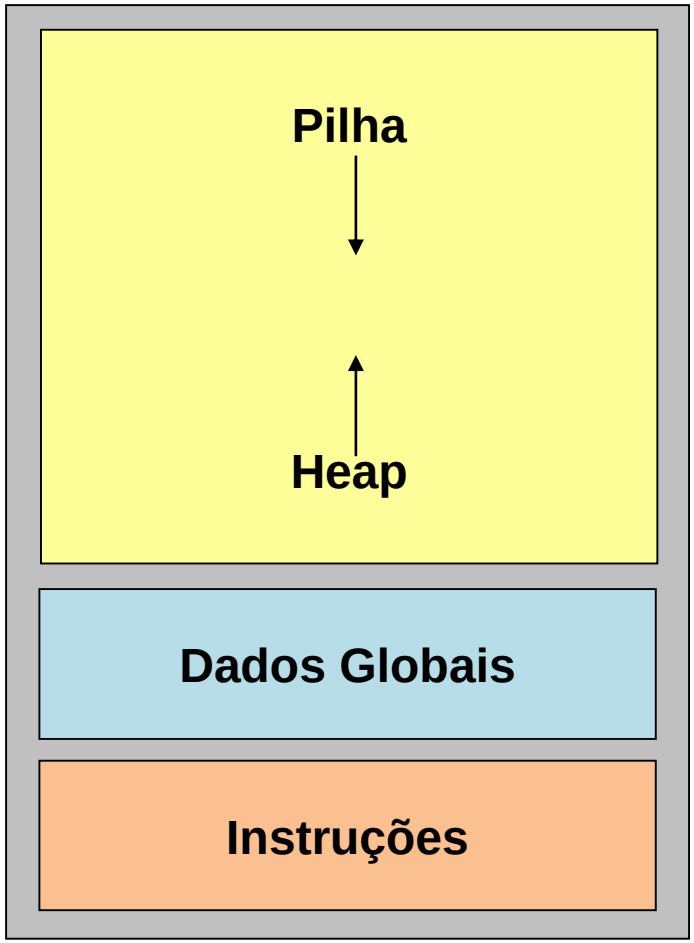


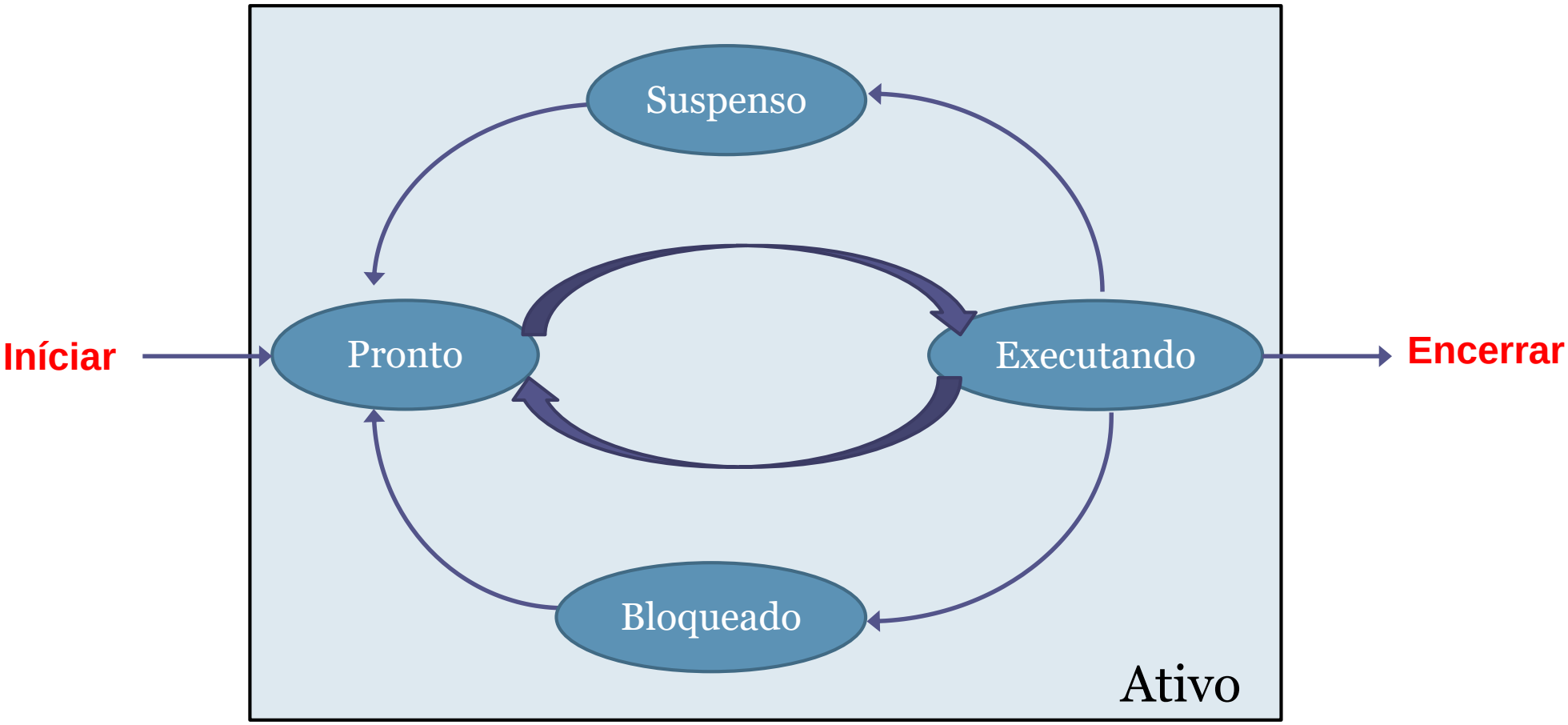
Tabela de Processos

- Estado do processo
- Valores dos registradores
- Arquivos abertos
- Alocação de memória
- PID (Process ID)
- UID (User ID)
- GID (Owner's Group ID)

Estados de um processo

- **Executando (Running):** Utilizando a CPU
- **Executável ou Pronto (Runnable ou Ready)**: Esperando para ser escalonado para usar a CPU
- **Suspenso (Suspended):** Recebeu um sinal para ser suspenso (para dar chance a outro processo)
- **Bloqueado (Blocked):** Esperando pela conclusão de algum serviço solicitado ao S.O. (acesso ao hardware em exclusão mútua)

Máquina de Estado de um Processo



Criação de Processos em Sistemas Distribuídos

- A criação de um processo em um Sistema Distribuído é semelhante a criação em uma máquina, com a diferença que é necessário definir o host destino:
  - Escolha de um Host de destino (transparente ao programador e ao usuário)
  - Executar processos no host escolhido
  - Esperar a mensagem com a resposta

Criação de Processos em Sistemas Distribuídos

- O gerenciador que distribui a carga para execução dos processos distribuídos pode ser:
  - Centralizado (único componente gerenciador de carga)
  - Hierárquico (vários nós gerenciadores de carga organizados em estrutura de árvore)
  - Descentralizado (nós trocam informações diretamente uns com os outros a fim de tomar decisão de alocação)

Criação de Processos em Sistemas Distribuídos

- Decisão de distribuição dos processos pode depender das cargas relativas dos nós, das suas arquiteturas e dos seus recursos.
- Criação de um novo ambiente de execução:
  - Espaço de endereçamento com conteúdos inicializados.
    - Estático: host define área fixa
    - Dinâmico: host define a área necessária e suficiente.
  - Outros recursos, como arquivos de uso.

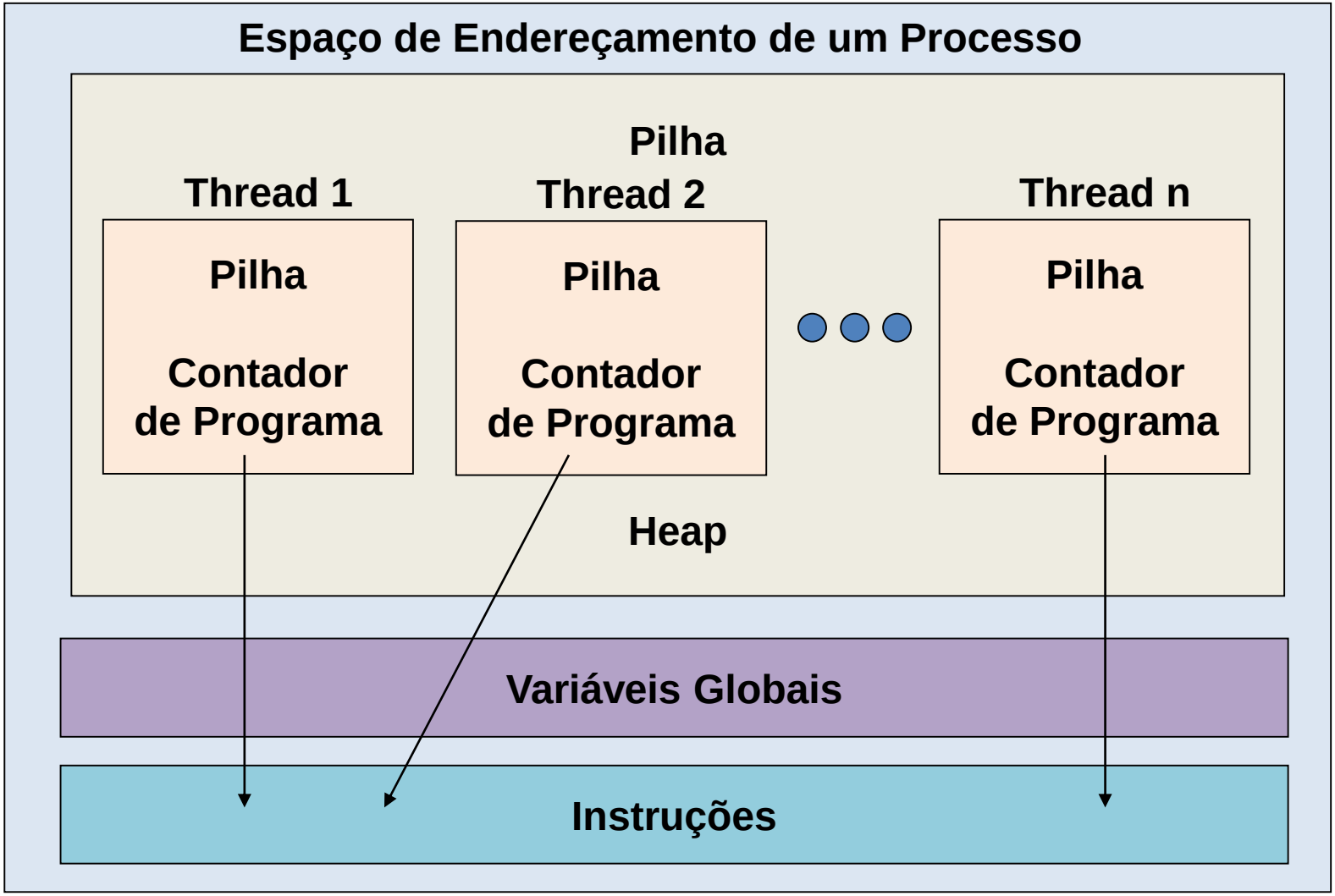
Sumário

- O que é Programação Concorrente?
- Motivação
- Conceitos
- Processos
- Threads
- Propriedades
  - Segurança (Safety)
  - Progresso (Liveness)

Threads

- Um processo pode ter mais de uma Thread (Linha)
- Cada Thread possui contador de programa e pilha próprios
- Quando um processo é escalonado para ser executado, uma das Threads entra em execução
- As Threads compartilham as variáveis globais do processo

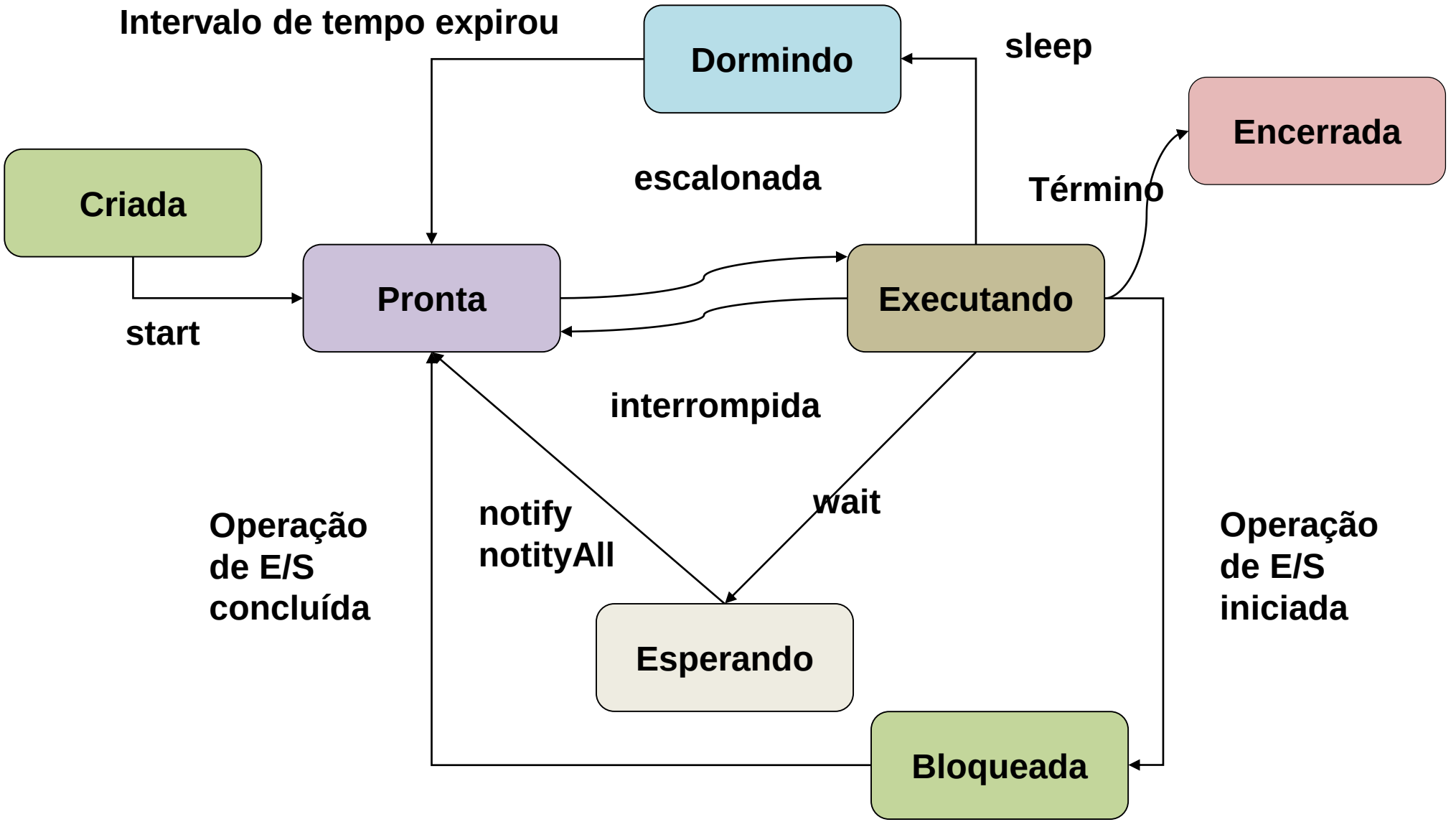
Threads


Espaço de Endereçamento de um Processo
Pilha
Thread 1
Thread 2
Thread n
Pilha
Contador de Programa
Pilha
Contador de Programa
Pilha
Contador de Programa
Heap
Variáveis Globais
Instruções

Threads

- Vantagens sobre processos compartilhando memória
  - São muito mais leves de serem criadas
  - A troca de contexto é mais suave pois compartilha instruções, pilha e variáveis globais
  - Facilita o compartilhamento de memória

Máquina de Estado: Java Thread



Sincronização de Threads

- Programação com múltiplos Threads requer muito cuidado:
  - Acesso/atualização de variáveis compartilhadas
  - Starvation
  - Deadlock
  - Acesso a estados inválidos de outros objetos

Sincronização de Threads

- Problema de acesso a variáveis compartilhadas
  - Várias Threads (representando as operações dos clientes) querendo atualizar (depositar um valor) uma mesma conta.
  - Quando uma está lendo o saldo atual, uma outra Thread pode já ter lido esse valor e calculado o novo saldo, mas ainda não ter gravado o novo valor.
  - Se nesse momento a primeira Thread gravar o valor, o saldo final será igual ao calculado pela Thread que gravar o novo valor por último (desconsiderando assim o depósito da Thread que encerrou primeiro).
  - O ideal é impedir que uma operação de depósito seja iniciada antes da outra encerrar

Sincronização de Threads

- A sincronização baseia-se na ideia de que para acessar um método sincronizado ou um entrar em um bloco sincronizado, é preciso obter (ou já ter) o *lock* (bloqueio) desse objeto.
- A Thread que conseguir esse *lock* é a única autorizada a acessar os **recursos protegidos** através de sincronização.

Threads em Sistemas Distribuídos

- Da mesma forma que processos em Sistemas Distribuídos, a criação de um Thread em um Sistema Distribuído é semelhante a criação em um processado, com a diferença que é necessário definir o host destino.
- Para diferenciar a função, definimos o lado cliente (que solicita um serviço) e o lado servidor (que executa o serviço).
- Threads podem ocorrer em ambos os lados:
  - Clientes Multithreads
  - Servidores Multithreads

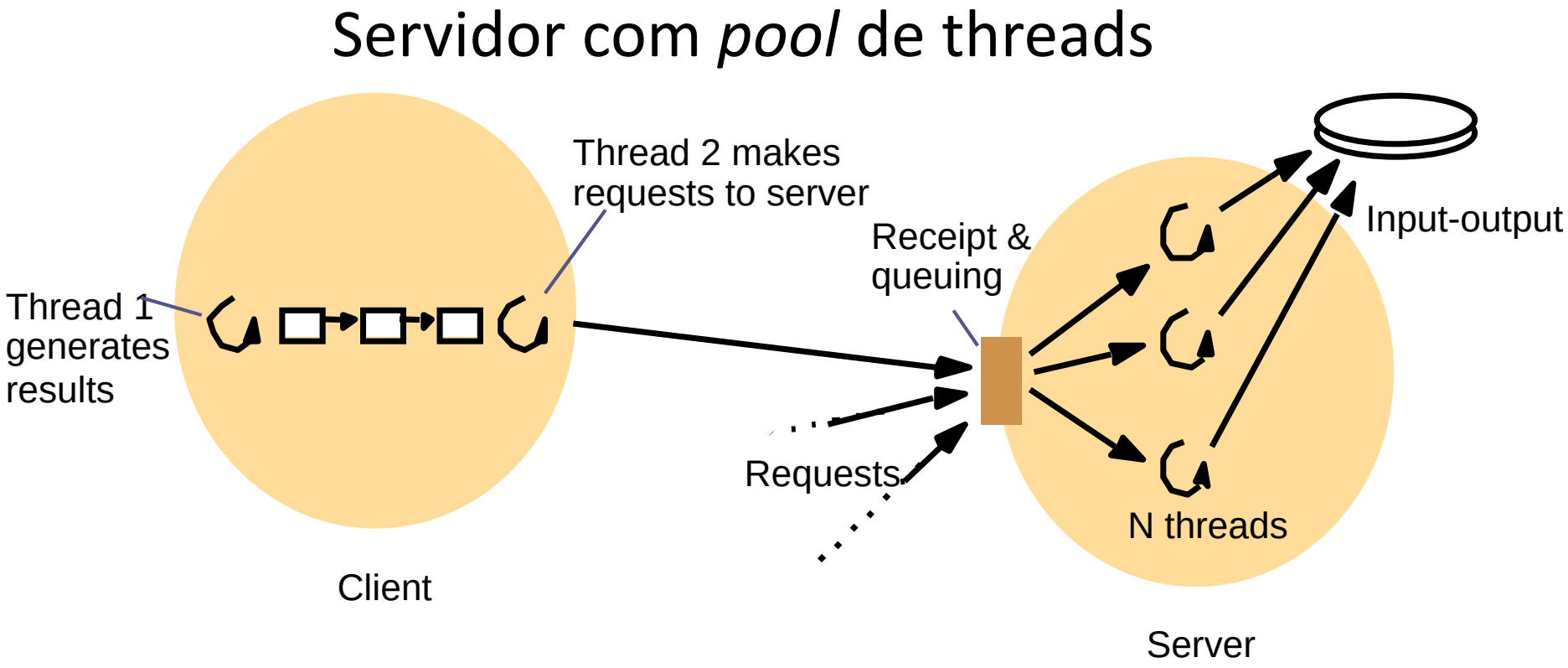
Clientes Multithreads

- Aplicação: clientes necessitam ocultar o atraso de transporte em redes de longa distância
- Técnicas de ocultação de latência
  - Após início da conexão, cliente executa outras tarefas paralelas através do uso de threads
- Ex: Busca de documentos web
  - Browser executa vários processos de busca de objetos e conexões TCP paralelas
    - Uso de várias conexões http paralelas para receber vários objetos simultaneamente.

Servidores Multithreads

- Sistemas monoprocessadores
  - Problema: num processador monothread, o processo é bloqueado quando uma chamada bloqueadora do sistema for executada
- Sistemas multiprocessadores
  - Exploração do verdadeiro paralelismo
  - Cada thread é designada para uma CPU diferente (thread de núcleo)

Clientes e Servidores com threads



Sumário

- O que é Programação Concorrente?
- Motivação
- Conceitos
- Processos
- Threads
- Propriedades
  - Segurança (Safety)
  - Progresso (Liveness)

Propriedades: Introdução

- Vários processos ou threads executando concorrentemente em um mesmo processador pode causar interferência.
- Programas concorrentes devem possuir duas propriedades principais:
  - **Segurança (Safety):** Nada de mau acontecerá durante a execução do programa.
  - **Vivacidade ou Progresso (Liveness):** Algo diferente acontecerá durante a execução do programa.

Propriedades: Introdução

- Em geral se dá mais atenção sobre Segurança do que Progresso.
  - É melhor que o programa não faça nada a ter um comportamento errado aleatório ou até perigoso
- Progresso está relacionado apenas se o programa continua executando.
  - Se garantirmos que um programa não executará nada errado, não precisamos nos preocupar.

Propriedades: Introdução

- Porém em alguns casos é melhor obter uma informação imprecisa do que não receber
  - Sistemas de Suporte a Decisão, Sistemas de controle automático e Simulações de processos.
  - Eventos de falha de progresso podem causar tragédias, e não podemos desprezar.
- **Problema:** qualquer medida para melhorar o Progresso de um sistema implica em reduzir a Segurança, e vice-versa.

Propriedades de Programação Concorrente (1)

- **Segurança (Safety)**: O programa nunca entra em um estado errado
- **Vivacidade ou Progresso (Liveness):** Em algum momento o programa vai para outro estado
- **Ausência de Deadlock:** Nunca todos os processos estarão bloqueados
- **Balanceamento de Carga:** Os processos dividem mais ou menos por igual os recursos de processamento

Propriedades de Programação Concorrente (2)

- **Correção Total:** O programa sempre termina e produz o resultado correto
- **Correção Parcial:** Se o programa terminar, o resultado está correto. Caso contrário, não podemos garantir o resultado correto
- **Término:** O programa termina eventualmente

Sistemas Sem Estado (Stateless)

- Um sistema sem estado (stateless) pode ser visto como uma caixa preta.
- Em qualquer ponto no tempo, o valor da saída do depende apenas do valor da entrada (após um determinado tempo de processamento).

Sistemas Com Estado (Statefull)

- Em um sistema com informações de estado (statefull), o valor da saída do depende do valor da entrada e de um estado interno.
- Basicamente é como uma máquina de estado com "memória", onde o mesmo conjunto de valores de entrada pode gerar uma saída diferente, dependendo da entrada anterior recebida pelo sistema.

Sistemas Stateless x Statefull

- Em programação concorrente, um **sistema sem estado (stateless),** se adequadamente implementado, pode ser executado por vários segmentos/tarefas ao mesmo tempo sem qualquer problema de concorrência.
- Um **sistema com estado (statefull)** vai exigir que os vários segmentos de acesso devem atualizar o estado interno do sistema de forma exclusiva, por isso haverá a necessidade de pontos de sincronização.

Sumário

- O que é Programação Concorrente?
- Motivação
- Conceitos
- Processos
- Threads
- Propriedades
  - Segurança (Safety)
  - Progresso (Liveness)

Segurança (Safety)

- Em programação concorrente, o controle de interferência é implementado em cada processo (classe).
- Objetos/variáveis podem entrar temporariamente em um estado inconsistente.
- Um objeto/variável é considerado seguro quando:
  - Seu estado é sempre consistente.
  - Outras operações não são realizadas enquanto o objeto está em um estado inconsistente.

Segurança (Safety)

- Há 3 estratégias para projetar sistemas seguros:
  - **Imutabilidade:** evitar mudanças de estado (stateless).
  - **Sincronização:** garantir acesso exclusivo dinamicamente através de mecanismos de sincronização.
  - **Contenção:** garantir acesso exclusivo estruturalmente (usando um padrão).
- Lembre-se: aumentar a Segurança pode reduzir o Progresso

Segurança (Safety): Imutabilidade

- Usar variáveis de instância constantes
- Encapsulamento de variáveis
- Não usar variáveis compartilhadas.
- Criar sistema sem estado (stateless)
  - Como não há necessidade de guardar o estado, não há interferência.

Segurança (Safety): Sincronização

- Um objeto/variável sempre está pronto para sofrer modificações, mesmo quando ainda está no meio do processamento.
- Utiliza algum mecanismos de sincronização para evitar riscos de segurança (semáforo/monitor).
- Acesso a estados inconsistentes devem ser evitados.

Segurança (Safety): Sincronização

- No contexto de programação concorrente, podemos ter:
  - Sincronização Total
    - Objetos totalmente sincronizados. Podem fazer apenas uma operações por vez
  - Sincronização Parcial
    - Parte do objeto é sincronizado. Somente métodos críticos são travados

Segurança (Safety): Contenção

- Baseia-se na ideia de manter referências únicas para objetos internos isolados dos demais.
  - Evitar variáveis globais
- Objetos são acessados apenas através de métodos sincronizados do objeto no qual estão contidos, estando, portanto, protegidos.

Segurança (Safety): Contenção

- Recursos Exclusivos
  - Garante que somente um objeto por vez é "proprietário" do recurso exclusivo.
- Semelhanças com objetos físicos:
  - Se você tem um, então pode usá-lo
  - Se você tem um, ninguém mais pode tê-lo
  - Se você dá um para alguém, então deixa de tê-lo
  - Se você destrói um, ninguém nunca mais o terá

Sumário

- O que é Programação Concorrente?
- Motivação
- Conceitos
- Processos
- Threads
- Propriedades
  - Segurança (Safety)
  - Progresso (Liveness)

Progresso (Liveness)

- Projetos baseados em combinações de Imutabilidade, Sincronização e Contenção são abordagens simples, rápidas e de fácil entendimento para o desenvolvimento de programas concorrentes
- Porém, essas técnicas nem sempre são a melhor solução
- Utilização de sincronização excessiva pode provocar problemas de vivacidade e eficiência.
- Lembre-se: aumentar o Progresso pode reduzir a Segurança

Falhas de Progresso (Liveness)

- São tão sérias quanto falhas de Segurança
- São mais difíceis de identificar e evitar que as falhas de segurança
  - O fato de ocorrerem aleatoriamente dificulta a identificação durante o teste

Falhas de Progresso (Liveness) (1)

- Tipos de Falhas de progresso:
  - Contenção: um processo, apesar de estar pronto para executar, não executa porque outro processo tomou recursos (também conhecida como starvation ou adiamento infinito)
  - Dormência: um processo falha ao tentar passar para esse estado (wait com condição impossível).

Falhas de Progresso (Liveness) (2)

- Tipos de Falhas de progresso:
  - Deadlock: dois ou mais processos bloqueiam-se em um ciclo vicioso enquanto tentam acessar travas sincronizadas necessárias para continuar seu processamento.
  - Término prematuro: um processo é parado ou encerrado prematuramente.

Progresso (Liveness)

- Através de análise de variáveis e métodos, é possível identificar oportunidades para reduzir a sincronização.
- Muitas vezes, a proteção implementada é exagerada causando problemas de progresso.

Progresso (Liveness)

- Acesso
  - Métodos de acesso podem deixar de ser sincronizados para permitir leitura durante a escrita
- Condições para remover a sincronização:
  - A variável não pode assumir valores ilegais durante a execução dos métodos
  - Atribuição atômica
    - Usar um método para atribuir um valor a uma variável

Progresso (Liveness)

- Atualizações podem abdicar da sincronização quando satisfaz, além das condições para acesso:
  - Valor da variável permanece consistente em todas as possíveis situações.
    - Ex: atributo temperatura sendo atualizado constantemente por múltiplos processos
  - Não requer ações sincronizadas durante um período.
    - Ex: reconstruir uma estrutura de dados após atingir um tamanho

Fim do Capítulo 1

- Esse capítulo apresentou os conceitos fundamentais de Programação Concorrente.
- Resolver os exercícios da Lista.