COPPE UFRJ

Multiprocessadores & Multicomputadores

Pedro Cruz

EEL770 – Sistemas Operacionais

Datas

GTA / UFRJ

- 13 de junho
  - Entrega das lista
- 18 de Junho
  - P2
- 20 e 25 de junho
  - Apresentações finais dos trabalhos e vistas de prova
- 27 de junho
  - Prova final
- 02 de julho
  - Prova de 2ª chamada

Motivação

GTA / UFRJ

- Queremos mais operações por tempo
  - Aumentar a frequência do relógio não adianta mais
    - Relatividade
    - Dissipação

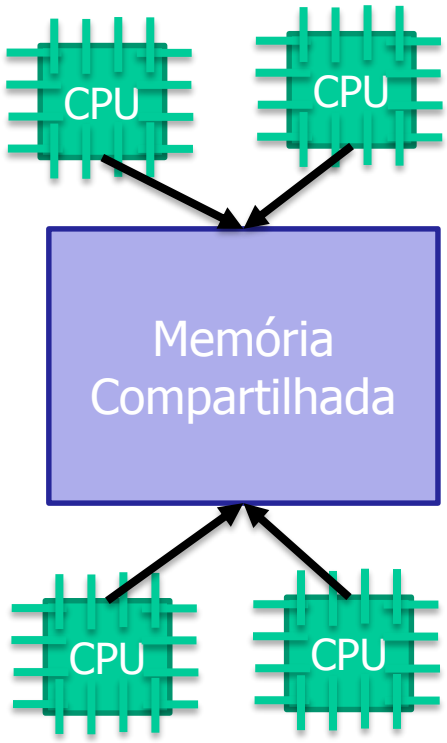
Arquitetura

- Multiprocessador de memória compartilhada
- Multicomputador com troca de mensagens
- Sistema distribuído em rede de longa distância

Multiprocessadores

GTA / UFRJ

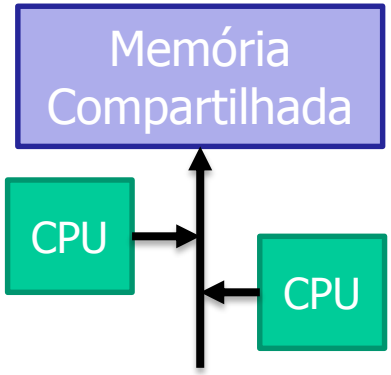
- Múltiplas CPUs compartilham memória
  - Memória compartilhada pode servir de área de comunicação
  - SO também deve realizar operações de processador único



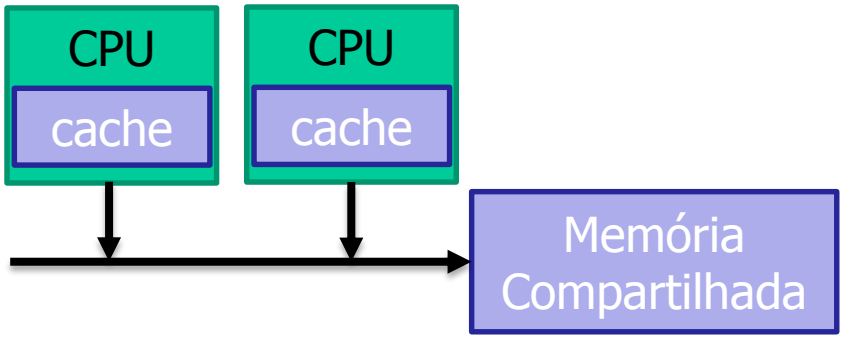
Arquitetura de multiprocessadores

GTA / UFRJ

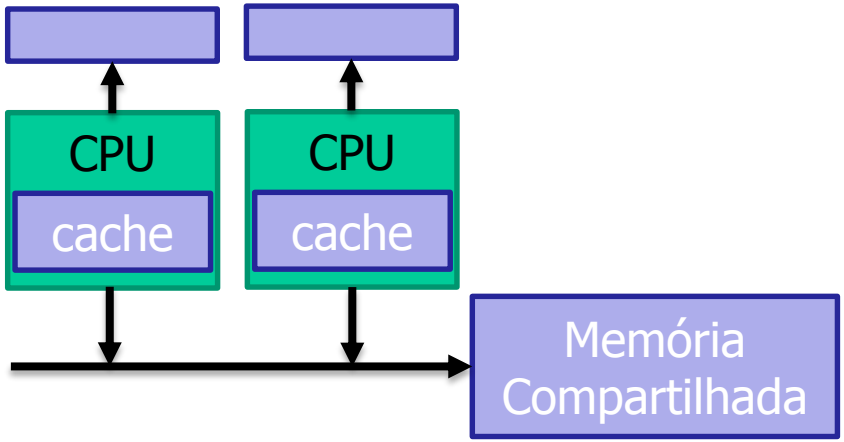
- Sem cache



- Com cache



- Com cache e área privada



Tempo de acesso à memória

GTA / UFRJ

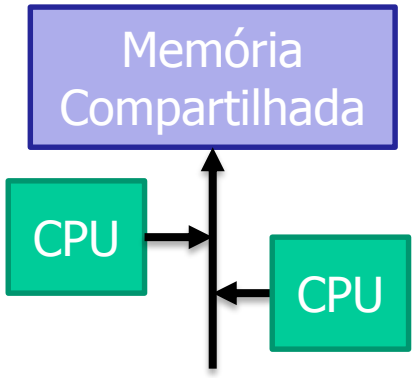
- *Uniform Memory Access* (UMA)
  - Tempo de acesso à memória é uniforme
    - Tempo de acesso igual para todas palavras de memória
- *Nonuniform Memory Access* (NUMA)
  - Tempo de acesso à memória não é uniforme
    - Tempo de acesso diferente a palavras distintas

Multiprocessadores sem cache

GTA / UFRJ

- Uma CPU deseja acessar memória
  - Verifica se barramento está ocupado
  - Envia comando para a memória
  - Aguarda a memória
  - Lê o barramento

  - Se barramento estiver ocupado, aguarda
    - Métodos de acesso ao meio

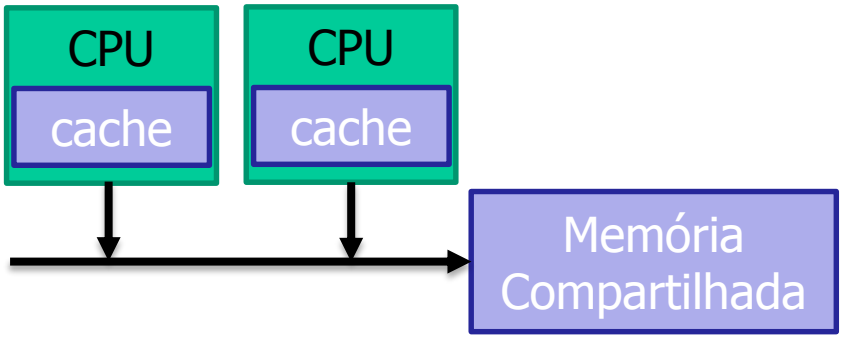


Solução: cache

Multiprocessadores com cache

GTA / UFRJ

- CPUs possuem cache
  - Memória "escondida"
- CPUs copiam parte da memória compartilhada
  - Redução do tráfego do barramento



Considerações sobre cache

GTA / UFRJ

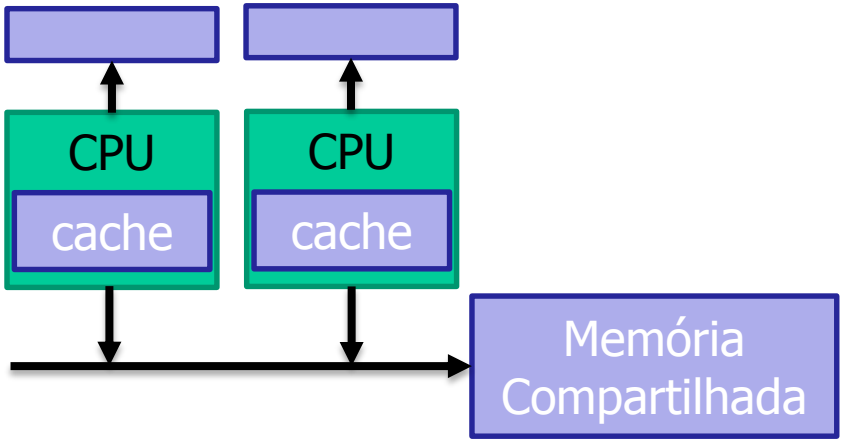
- Cache é acessado em linhas de cache
  - Mais de uma palavra

- Coerência de cache
  - Cache é uma cópia de parte da memória
    - Se uma CPU escreve em memória que é compartilhada
      - Inicia atualização de outros caches que fazem referência àquela memória

CPUs com cache e área privada

GTA / UFRJ

- CPUs possuem área significativa de área privada
  - Área compartilhada
    - Comunicação entre CPUs
  - Área privada
    - Código
    - Variáveis locais

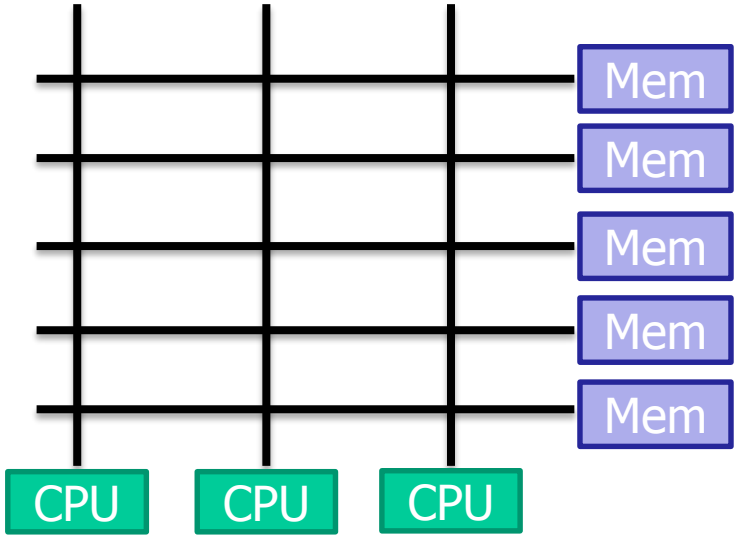


- Compilador dos programas possui papel fundamental

Barramento cruzado

GTA / UFRJ

- n CPUs e k módulos de memória
  - Malha conecta todos
  - Interseções são chaveáveis
- Número de chaves é n*k
  - Pode ser grande demais
- Barramento não bloqueia CPU
  - Acesso a mesmo módulo pode bloquear CPUs



Redes de comutação multiestágio

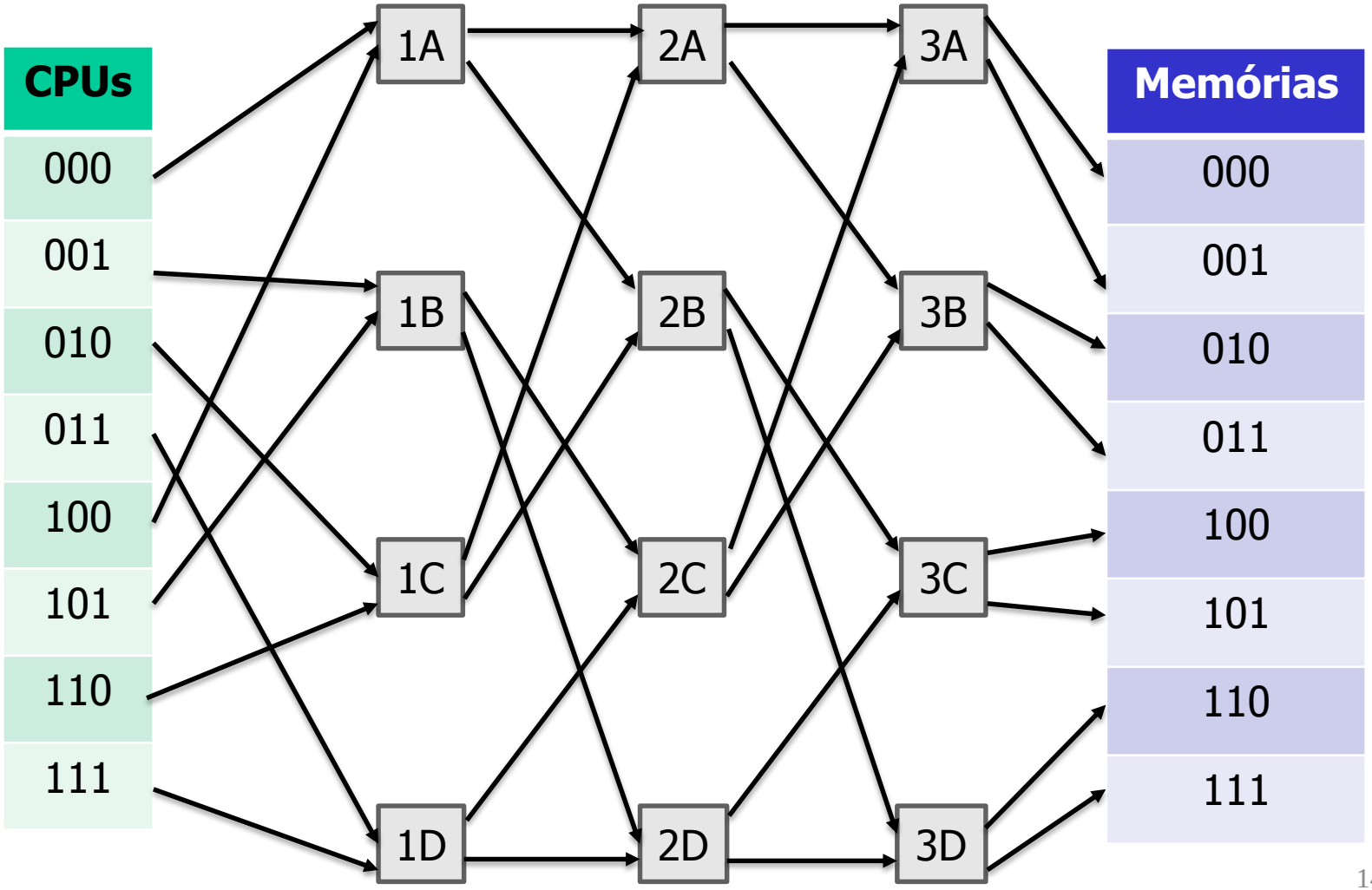
GTA / UFRJ

- Elemento de comutação

- Mensagem ao barramento contém dados de comutação
  - Qual saída deve ser ativada

Rede de comutação ômega

GTA / UFRJ



Rede de comutação ômega

GTA / UFRJ

- Pode haver bloqueio de CPU por causa do barramento?
  - Como?

Entrelaçamento de memória

GTA / UFRJ

- Acesso à memória costuma ser em endereços consecutivos
  - Armazenar endereços consecutivos em módulos diferentes
    - Aumento no nível de paralelismo

Multiprocessadores NUMA

GTA / UFRJ

- CPUs possuem memória local
  - Memória local de uma CPU é visível para outra CPU
- Acesso à memória local é mais rápido
- Acesso à memória remota é feito por instruções especiais
  - LOAD
  - STORE

- Cache pode ser utilizado
  - Deve haver coerência de cache

Cache-Coherent NUMA

GTA / UFRJ

- Multiprocessador baseado em diretórios
  - Banco de dados mapeia o cache local e seu estado
- Escritas na memória são encaminhadas a nó que tem aquele endereço em cache

Sistemas operacionais para multiprocessadores

- Abordagem simplória
- Abordagem mestre-escravo
- Abordagem de multiprocessadores simétricos

Abordagem simplória

GTA / UFRJ

- Um Sistema Operacional pra cada CPU
  - CPUs recebem fatia fixa da memória
  - CPUs não compartilham processos
  - CPUs não compartilham memória
  - CPUs não compartilham usuários
  - CPUs compartilham dispositivos

Abordagem mestre-escravo

GTA / UFRJ

- CPU mestre roda o operacional
  - Recebe todas as chamadas de sistema
  - Aloca memória para as outras CPUs
  - Distribui tarefas para as outras CPUs

Abordagem de multiprocessadores simétricos

- Existe uma cópia do SO em memória
  - Qualquer CPU pode executá-lo
- Chamada de sistema chaveia a CPU para modo núcleo
  - CPU que recebe a chamada é a mesma que a atende
- Executar o núcleo requer exclusão mútua
  - Grande trava de núcleo
    - Execução do núcleo é protegida por **mutex**
    - Só uma CPU executa o núcleo por vez
  - Múltiplas regiões críticas independentes
    - Cada região crítica do SO é protegida por um **mutex**
      - Potencial de erros: over 9000

Test and Set Lock

GTA / UFRJ

- Instrução TSL realiza leitura e escrita atômica
  - Atomicidade só do ponto de vista da CPU que a executa
    - Para o barramento e memória, são duas operações
    - TSL trava o barramento
- CPUs requisitantes fazem teste contínuo
  - Desperdício de tempo
  - Carga sobre o barramento
    - TSL é uma escrita
      - Escritas invalidam cache
        - » Palavra de memória tem que ser reescrita nos caches de todas as CPUs que participam do mutex

Redução de tráfego no barramento

- Antes de executar TSL, executar uma leitura
  - Se variável estiver livre, executar TSL
    - Não garante nada, mas reduz reescrita de cache
- Recuo exponencial binário (binary *exponential backoff*)
  - Se tentativa de TSL falha, reduz frequência de testes por 2
    - Até um mínimo estabelecido

Chaveamento vs teste contínuo

GTA / UFRJ

- Se há uma única CPU e thread está bloqueada
  - A própria CPU deve trabalhar para destravar thread
    - Chaveamento é necessário
- Se há muitas CPUs e thread está bloqueada
  - Outra CPU pode desbloquear a thread
    - Chaveamento não é necessário
      - Custos de chavear nem sempre valem a pena

- Se bloqueio demora -> chaveamento vale o custo
- Se bloqueio é rápido -> chaveamento não vale o custo
- Se histórico for utilizado, saberemos quais bloqueios valem o custo de chaveamento

Escalonamento em multiprocessadores

GTA / UFRJ

- *Threads* de usuário e de núcleo

- Escalonamento em multiprocessadores
  - Qual thread escalonar
  - Qual CPU deve receber a thread

Compartilhamento de tempo

GTA / UFRJ

- Threads são ordenadas por prioridade
  - CPU ociosa escolhe a primeira da fila
  - Vantagem
    - Balanceamento de carga “de graça”
  - Desvantagem
    - Competição pelo uso da fila
    - Exclusão mútua das threads

Compartilhamento de tempo

GTA / UFRJ

- Escalonamento inteligente
  - Se thread adquire trava, ganha um pouco mais de tempo
- Escalonamento por afinidade
  - SO procura enviar threads para o mesmo processador
- Escalonamento de dois níveis
  - Nível 1
    - Threads são distribuídas pelas CPUs
  - Nível 2
    - CPUs escolhem quais threads de seu conjunto executar

Compartilhamento de espaço

GTA / UFRJ

- Threads que se comunicam muito devem ser executadas ao mesmo tempo
  - Abordagem simples
    - Quando threads são inicializadas juntas, cada uma é alocada para uma única CPU
      - Inicialização deve esperar até que haja CPUs suficientes
  - Abordagem mais curta primeiro
    - Threads que vão demorar menos são executadas primeiro
  - Abordagem de teoria dos jogos
    - Threads "gananciosas" são penalizadas quando número de CPUs é insuficiente

Escalonamento em bando

- É bom que threads relacionadas executem juntas
  - Solução
    - Todas as threads relacionadas serem escalonadas juntas

Multicomputadores

GTA / UFRJ

- Diversos computadores fortemente acoplados
  - Não há memória compartilhada

Hardware de multicomputadores

- CPU
- Memória privada
- Placa de rede

Interconexão

GTA / UFRJ

- Nós dos multicomputadores devem ser conectados entre si
  - Estrela
    - Necessita de comutador
  - Grade ou malha
    - Múltiplos saltos

Comutação

GTA / UFRJ

- Comutação de circuitos

- Comutação de pacotes

Comutação de circuitos

GTA / UFRJ

- Caminho entre dois nós é reservado
  - Conexão estabelecida
- Caminho é utilizado até o fim da conexão

- Existe caminho fixo
- Existe reserva de recurso

Comutação de pacotes

GTA / UFRJ

- Mensagens são divididas em pacotes
- Nós armazenam e encaminham pacote para o próximo nó

- Não existe caminho fixo
- Não existe reserva de recurso

Placas de rede

GTA / UFRJ

- Devem ser capazes de receber e enviar dados de forma síncrona
  - Não podem ser travadas por competição pelo barramento
    - Devem ter RAM própria
  - Devem transferir muitos dados para a memória principal
    - Devem dar suporte à DMA
  - Podem ou não tratar problemas de rede
    - Ter uma CPU própria
      - Transmissão confiável
      - Multicasting
      - Compactação
      - Criptografia
      - ....

Software de comunicação

GTA / UFRJ

- Muitas cópias que atrapalham desempenho
  - RAM origem -> placa origem
  - Placa origem -> placa destino
  - Placa destino -> RAM destino
- Softwares devem ler/escrever na placa
  - Mapear RAM da placa no espaço do usuário
    - Com múltiplos processos, como garantir exclusão mútua?
    - Quando um pacote chegar, pra quem entregar?
  - Mapear RAM da placa no espaço de núcleo
    - Chamada de sistema pra cada operação é custoso

Software de comunicação

GTA / UFRJ

- Múltiplas placas no mesmo nó
  - Uma pra cada usuário/processo
- Placas de rede com múltiplas filas
  - Uma fila pra cada usuário/processo
    - TCP e UDP oferecem 'portas'

Software de comunicação a nível de usuário

GTA / UFRJ

- Usuário quer enviar mensagens
  - Interface pode ser fornecida pelo SO
    - Send(dest, &mptr)
    - Receive(addr, &mprt)
- Usuário precisa saber endereçar o processo destinatário
  - Habita um nó destinatário
    - Endereçamento hierárquico
      - Endereço = (id_nó, id_processo)

Chamadas bloqueantes/ não bloqueantes

GTA / UFRJ

- Enviar mensagem não necessita de bloqueio
  - Mas usuário não pode escrever em &mptr até fim do envio
    - Soluções
      - Cópia para buffer de núcleo
      - Interrupção ao fim do envio
      - Buffer com cópia na escrita

Chamada de rotina remota

GTA / UFRJ

- *Remote Procedure Call* – RPC
  - Paradigma cliente-servidor
    - *Stub* do cliente
      - Fica no cliente e se comporta como o servidor
    - *Stub* do servidor
      - Fica no servidor e se comporta como o cliente

Problemas com RPC

GTA / UFRJ

- Passagem de ponteiros
  - Os *stubs* podem resolver isso
    - Para tipos simples de dados
- Convenções do programador
  - Se vetores possuem terminador, *stub* tem problemas
- Tipos de parâmetros nem sempre são óbvios
- Compartilhamento de variáveis globais

Memória compartilhada distribuída - DSM

- Memória virtual compartilhada (*Distributed Shared Memory* – DSM)
  - Sistema possui globalmente um número de páginas
  - Páginas são distribuídas por máquinas do sistema
  - Chamadas de LOAD e STORE trazem toda uma página
    - Página é removida da máquina de origem

DSM – uma outra visão

GTA / UFRJ

- Endereçamento é feito em duas partes: página e palavra
- Quando página referenciada não está presente localmente
  - Sistema de DSM realiza captura
  - Página remota é localizada
  - Página remota é trazida para memória local
  - Processo que realizou a referência é reiniciado

- Problema?

Replicação de páginas em DSM

GTA / UFRJ

- Páginas podem ser copiadas por algumas máquinas
  - Inconsistência
    - Copiar apenas páginas "somente leitura"
      - E se não for o suficiente?
        - » Copia mais e mantém consistência

Página efetiva

GTA / UFRJ

- Páginas do DSM não precisam ser as páginas da memória virtual
  - Precisam ser múltiplos inteiros, para usar a MMU
    - Página efetiva
- Realizar conexão é custoso
  - Vantajoso enviar muitas páginas por vez

Falso compartilhamento

GTA / UFRJ

- Página efetiva contendo muitas páginas
  - Probabilidade de atualização de variáveis aumenta
    - Necessidade de troca de mensagens aumenta
  - Se páginas são do mesmo processo, compilador resolve
    - Mas não necessariamente são

Consistência

GTA / UFRJ

- Tentativas de escrita iniciam processo de consistência
  - CPU adquire trava sobre página
    - Todas as outras CPUs descartam a página
      - Se estiver em memória
  - CPU libera trava sobre página
    - Todas as outras CPUs copiam a página
      - Se estiver em memória

Escalonamento

GTA / UFRJ

- Processos convivem em espaços diferentes de memória
  - Não vale a pena processo da CPU1 ser executado na CPU2
- Inicialização de processos pode balancear a carga
- Escalonamento em bando requer acordo de sincronização

Alocação de processadores

GTA / UFRJ

- É necessário alocar uma máquina para cada processo
  - Algoritmo determinístico de teoria dos grafos
  - Algoritmo determinístico iniciado por emissor
  - Algoritmo heurístico distribuído iniciado por receptor

Algoritmo determinístico de teoria dos grafos

GTA / UFRJ

- Grafo
  - Vértices: processos
  - Arestas: fluxo de dados entre processos
- Meta
  - Encontrar agrupamentos no grafo
    - Fluxo de dados intra agrupamento é máximo
    - Fluxo de dados inter agrupamento é mínimo
    - Requisitos dos processos são obedecidos
      - CPU
      - Memória
      - ...

Algoritmo determinístico iniciado por emissor

- Ao iniciar um processo
  - CPU verifica a própria carga
    - Se menor do que limiar, executa
    - Se maior do que limiar, busca outra CPU para executar

- Se sistema estiver muito sobrecarregado?

Algoritmo heurístico distribuído iniciado por receptor

- Quando uma máquina está ociosa, envia mensagem pedindo trabalho

COPPE UFRJ

Multiprocessadores & Multicomputadores

Pedro Cruz

EEL770 – Sistemas Operacionais