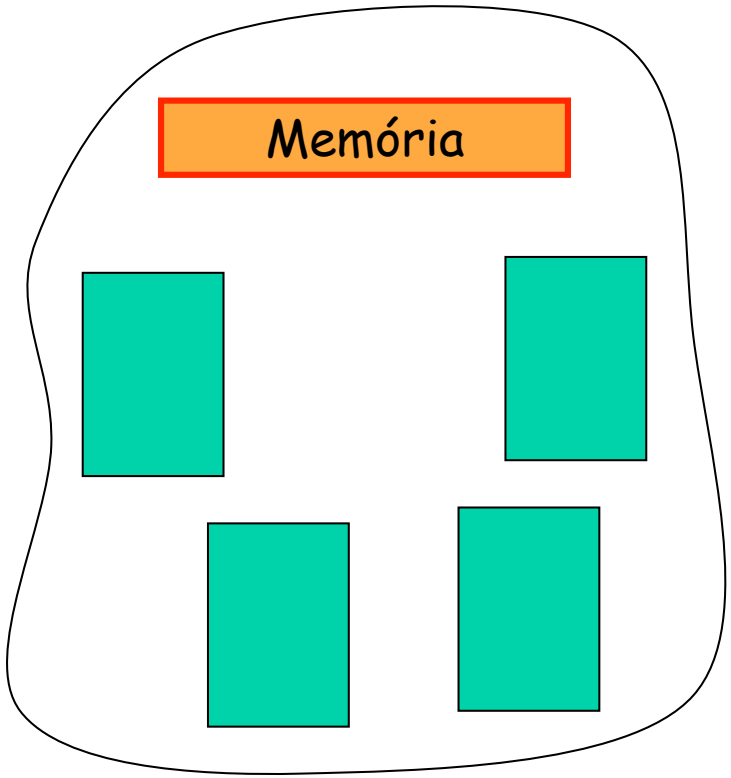
Sistemas Operacionais

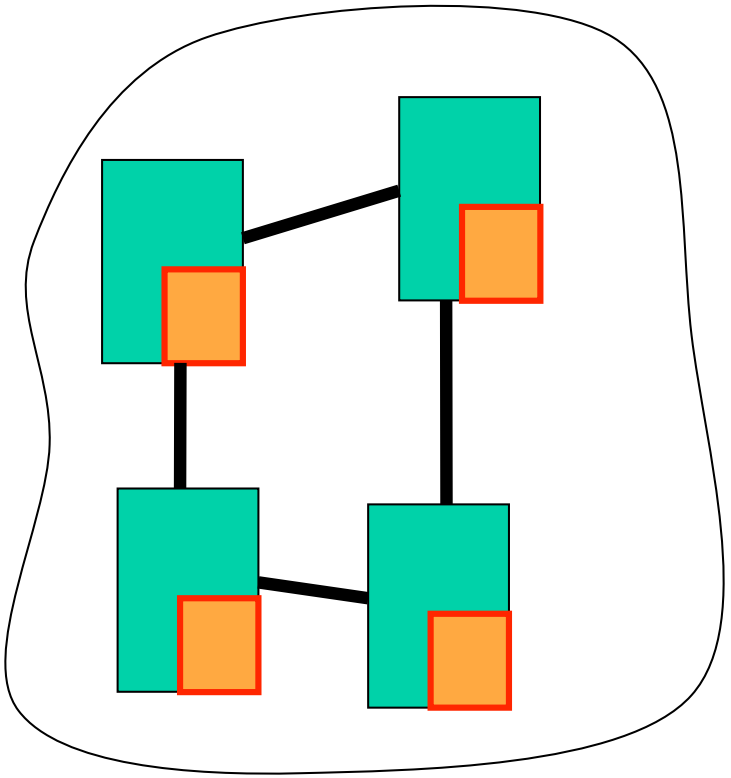
Escalonamento de Processos em Multiprocessadores

Capítulo 10

Processamento Paralelo e Distribuído

Multiprocessadores X Multicomputadores





Processamento Paralelo e Distribuído

- fracamente acoplados
  - geralmente um conjunto de computadores conectados de alguma forma
  - processadores especializados – servidores
  - cluster de processadores
- fortemente acoplados
  - memória compartilhada por vários processadores

Tipos de Paralelismo

- várias aplicações (*jobs*) podem ser executados em paralelo
  - processos independentes
- uma aplicação paralela constituída de vários processos para solucionar um problema
  - processos se comunicam para trocar resultados parciais do problema

Tipos de Paralelismo

- granularidade/granulosidade
  - relação entre a carga computacional e de comunicação
- paralelismo de granularidade grossa
  - processos se comunicam raramente
- paralelismo de granularidade média
  - exemplo: aplicação = conjunto de *threads*
- paralelismo de granularidade fina
  - muita comunicação entre processos

Tipos de Paralelismo

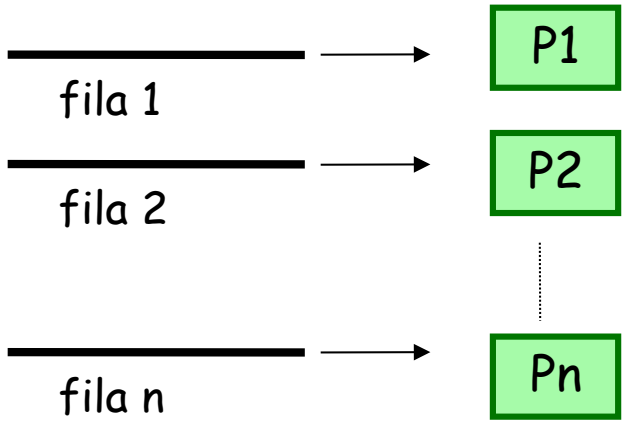
- granularidade/granulosidade
  - relação entre a carga computacional e de comunicação
- paralelismo de granularidade grossa
  - processos se comunicam raramente
- paralelismo de granularidade média
  - exemplo: aplicação = conjunto de *threads*
- paralelismo de granularidade fina
  - muita comunicação entre processos

Escalonamento de Processos

Estático

- conhecimento de características associadas às aplicações antes da execução desta (estimativas)
- relação de precedência entre os componentes da aplicação ou aplicações independentes

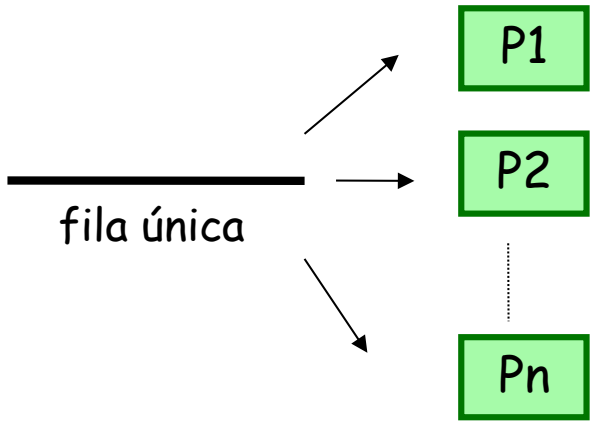
- uma fila por processador
- processador pode ficar ocioso (se estimativas não forem corretas)



Escalonamento de Processos

Estático

- fila global para evitar ociosidade de processador
- contexto do processo disponível para todos os processadores (depende muito do projeto)



Escalonamento de Processos

Dinâmico

- estimativas são conhecidas antes da execução e não as características reais.
- a especificação do escalonamento é feita ao longo da execução da aplicação
  - balanceamento de carga, por exemplo

Onde está o escalonador

Abordagem mestre-trabalhador (cliente-servidor)

- mestre escalona os processos (trabalhadores)
  - localiza-se em um dos processadores
- trabalhadores requisitam necessidades ao mestre
- Problemas
  - se o mestre falhar
  - mestre é um gargalo
  - bom limitar serviços oferecidos pelo mestre

Onde está o escalonador

Abordagem distribuída (arquitetura peer-to-peer)

- um escalonamento distribuído entre processadores
- cada processador escalona processos de um *pool* de processos
- sincronização complicada
  - dois escalonadores não podem escolher um mesmo processo para ser escalonado em algum processador
  - escalonadores tem que trocar conhecimento

Escalonamento de Processos em vários processadores

Diferentes questões devem ser analisadas:

- uma fila com todos os processos prontos
- um conjunto de filas de diferentes prioridades

Exemplo de estudo comparativo

S1 - sistema com um processador

S2 - sistema duo-processador

- Supondo
  - taxa de processamento de cada processador em S2 = $\frac{1}{2}$ taxa processamento do processador de S1
  - *Ou seja: é melhor ter dois processadores mais lentos do que um mais rápido?*

Escalonamento de Processos em vários processadores

- comparação entre FCFS e round-robin
  - RR: quantum bem maior que tempo de troca de contexto
  - RR: quantum com valor pequeno quando comparado ao tempo médio de serviço dos processos
- resultado da análise: depende do **coeficiente de variação em relação ao tempo de serviço**

$$C_s = \frac{\sigma_s}{\overline{T}_s}$$

ou seja, o quanto varia o tempo de serviço entre os diferentes processos

Escalonamento de Processos em vários processadores

- σ_s - desvio padrão do tempo de serviço
- $\overline{T_s}$ - tempo médio de serviço
- C_s
  - se Zero: os tempos de serviços são similares
  - pode ser alto: muita variação entre os tempos de serviço

Escalonamento de Processos em vários processadores

- FCFS pode ser problemático quando comparado com RR em um processador. Mas....
- FCFS em S2 (duo processado) é amenizado:
  - enquanto um processo longo que chegou primeiro está sendo executado por um processador, outros processos são executados no outro
  - pode ser tão bom quanto *round-robin*, principalmente se o número de processadores aumentar

Escalonamento de *Threads*

- processo = conjunto de *threads*
- em um processador
  - *threads* são vantajosas devido a E/S
- em vários processadores
  - dividir a funcionalidade do processamento entre processadores leva a um maior ganho
- *threads* + multiprocessamento = exploração do grau de paralelismo da aplicação
  - granularidade fina → paralelismo não tão vantajoso
  - alto grau de interação entre *threads*

Escalonamento de *Threads* em *Multiprocessadores*

- Escalonamento é mais complexo quando várias CPUs se tornam disponíveis

- **Processadores Homogêneos** dentro de um mesmo multiprocessador
  - no entanto, já se inicia uma tendência de heterogeneidade: *big and small* CPUs

Escalonamento de *Threads* em *Multiprocessadores*

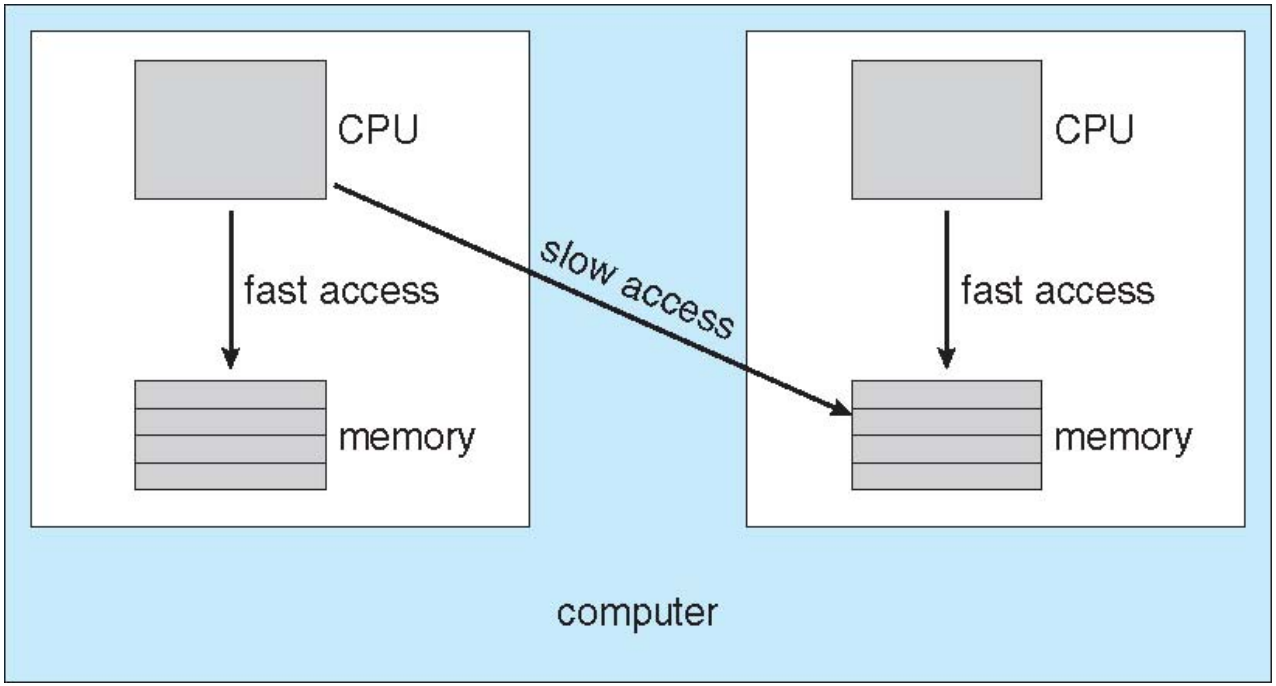
Asymmetric multiprocessing

- **escalonamento mestre trabalhador** – somente um processador decide pelo escalonamento (executa SO) e os outros executam atividades dos usuários
  - menos carga quando só um core acessa as estruturas de dados do sistema

Symmetric multiprocessing (SMP)

- cada processador tem o seu escalonador
  - neste caso, uma fila comum pode ser compartilhada e os escalonadores vão retirar processos dessa fila
  - sincronização necessária

Escalonamento de *Threads* em *Multiprocessadores*



Multicore Processors

- Vários processadores no mesmo chip físico
- Mais rápido com menor consumo de energia
- Uso de múltiplos threads por core está aumentando
  - enquanto um pedido de T1 de acesso a memória está sendo atendido (memoru stall) , uma outra thread pode ser executada

Escalonamento de *Threads*

Compartilhamento de carga

- uma fila global de *threads* e vários processadores
- processador ocioso executa um *thread* pronto
- Algumas políticas
  - FCFS
  - jobs de menor número de *threads*: prioridade dada aos *threads* de um *job* de menor número de *threads*
  - de menor número de *threads* preemptivo

Escalonamento de *Threads*

Compartilhamento de carga

- FCFS
  - cada *thread* de um *job* é inserida ao final da fila global
  - quando ocioso, o processador seleciona o primeiro *thread* pronto da fila
  - não preemtivo

Escalonamento de *Threads*

Compartilhamento de carga

- jobs de menor número de *threads*
  - fila global é uma lista ordenada por prioridade: um *job* de menor número de *threads* é prioritário
  - não preemptivo
- jobs de menor número de *threads* preemptivo
  - mesmo esquema anterior, mas quando um job com menor número de *threads* chega, os *threads* de um job menos prioritário que estão sendo executados são interrompidos
  - custo de gerenciamento pode ser alto

Escalonamento de *Threads*

Compartilhamento de carga

- estudos mostraram que FCFS é o melhor caso
- é a política mais comumente utilizada

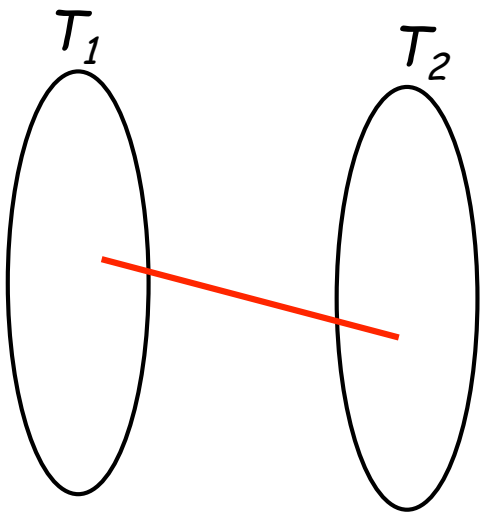
Desvantagens de compartilhamento de carga

- fila centralizada acessada por exclusão mútua
- quanto mais processadores, pior o gargalo
- *threads* preemptivas podem ser executadas em diferentes processadores: dados temporários tem que ser copiados entre as respectivas *caches*

Escalonamento de *Threads*

Gang Scheduling

- escalonamento em grupos de processos nos processadores
- *threads* no grupo geralmente se relacionam (comunicação)



Escalonamento de *Threads*

Gang Scheduling

- se executados em paralelo
  - custos relacionados a sincronização, troca de processos podem diminuir
  - uma decisão de escalonamento para um grupo, logo, menos decisões são tomadas pelo escalonador
  - os *threads* correlatos tem que ser identificados
  - memória compartilhada
- similar a co-escalonamento
  - aplicados a *threads* pequenos
  - em cluster de processadores

Escalonamento de *Threads*

Gang Scheduling

- menor custo de troca de processos
- T1 e T2 são duas *threads* de um mesmo processo
- T1 está sendo executada: precisa sincronizar com T2
- T1 fica em espera até que T2 seja executado em algum processador
  - seria melhor T2 já estar executando em outro processador

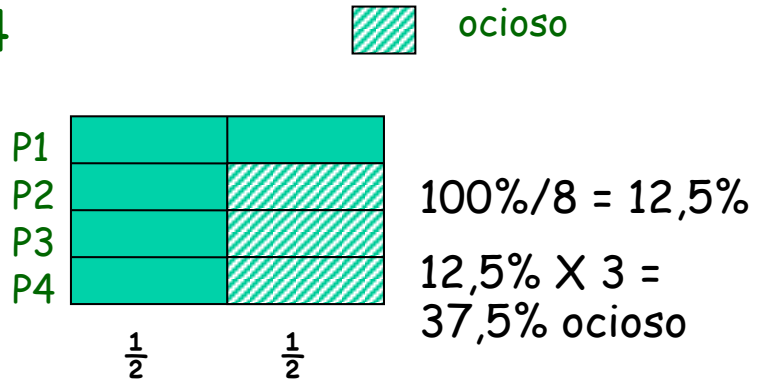
Escalonamento de *Threads*

Gang Scheduling

- N processadores, M aplicações com N ou menos *threads* cada aplicação
  - cada aplicação poderia receber 1/M de fatia de tempo, utilizando os N processadores
  - nem sempre isso é eficiente

A1	4 threads
A2	1 thread

N = 4

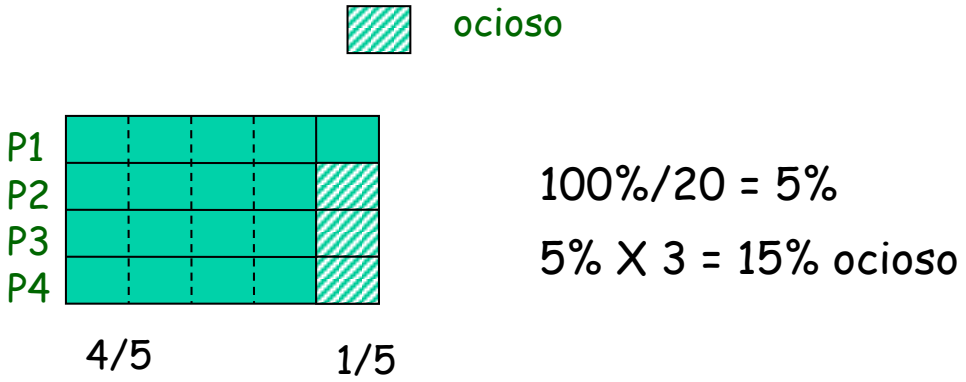


  - alocação de tempo uniforme:

Escalonamento de *Threads*

Gang Scheduling

- uma solução: dar pesos aos processos de acordo com o número de *threads*
  - considerando todos os processos, A1 tem 4/5 das *threads*



100%/20 = 5%

5% X 3 = 15% ocioso

Escalonamento dinâmico

- especificação da alocação durante a execução
- preempção pode ser permitida para migração de processos/threads
- pode acontecer
  - migração de código (custos associados, guardar contexto para execução em outro processador)
  - modificação da alocação entes do início da execução
  - mapeamento de tempos em tempos
- escalonamento nos processadores realizado de acordo com a carga disponível nos processadores

Escalonamento em Tempo Real

- Uma definição:
  - o resultado correto de um problema deve ser produzido, mas também, o tempo que leva para ser produzido é essencial
  - é um sistema composto de tarefas em que algumas são urgentes

- Para esse classe de aplicações, o SO, e particularmente o escalonador são de crucial importância

Atualidade em Sistema de Tempo Real

- laboratórios de controle
- robótica
- trafego aéreo
- telecomunicações
- sistemas de controle

Próxima geração de Sistema de Tempo Real

(... que já é atual)

- carros autonômicos
- controladores de robôs com juntas elásticas
- fábricas inteligentes
- exploração de águas profundas
- etc.

Sistema de Tempo Real

- as tarefas ou processos podem ser urgentes ou não
- associado a cada tarefa
  - tempo de fim
  - deadline
- hard real time task: deadline tem que ser respeitado

X

- soft real time task: deseja-se atingir o deadline

Características de SO para Sistema de Tempo Real

- tempo de resposta de seus serviços
  - tempo contabilizado a partir de sua requisição
  - depende:
    - tempo da rotina de tratamento de interrupção
    - iniciar a rotina do serviço (troca de contexto)
    - tempo de execução do seviço
    - mais interrupções, se ocorrerem

Características de SO para Sistema de Tempo Real

- Controle do usuário
  - nestes sistemas, o usuário tem maior interação com o SO para alimentar dados da aplicação a ser executada
- Confiabilidade
  - muito importante em Sistemas de Tempo Real
  - falhas devem ser tratadas de forma transparente ao usuário
  - tolerâncias a falhas via software
  - salvamento de dados para recuperação de processo
    - manter arquivos de entrada, etc

Características de SO para Sistema de Tempo Real

Confiabilidade

- com mecanismos de tolerância a falhas, problemas podem ser contornados e a execução continua
- esta operação pode ter *estabilidade*
  - quando ainda com falha, os deadlines são atingidos

Escalonamento em Tempo Real

- aspectos que podem ser considerados:
  - existe uma análise do comportamento do sistema
  - a análise pode ser estática ou dinâmica
  - de acordo com a análise, o escalonamento é produzido

Escalonamento em Tempo Real

análise estática

- determina a alocação das tarefas em tempo de execução
- não necessariamente determina o escalonamento, mas as prioridades entre as tarefas
- o sistema pode ser preemptivo, realocar, de acordo com as essas prioridades

análise dinâmica

- considerando as restrições do sistema para atingir o melhor desempenho
- *deadlines* das tarefas são considerados
  - se um processo não atinge seu *deadline* é abortado

Escalonamento em Tempo Real

análise estática

- bom para aplicações que são executadas repetidamente
- problema: estimativas precisas

análise estática, que determina prioridades

- uma análise a priori pode auxiliar na especificação da importância das tarefas
- *deadlines* podem ser considerados

dinâmica

- quando o sistema tem um caráter dinâmico, com tarefas chegando aleatoriamente
- deadlines devem ser obedecidos

Características de SO para Sistema de Tempo Real

Escalonador de curto prazo – papel crucial

- importante que as tarefas críticas (*hard real time tasks*) sejam executadas não ultrapassando deadlines
- o máximo de tarefas não críticas devem ser executadas

Maioria de Sistemas de Tempo real

- dificuldade de atingir deadlines
- quando um deadline está para ser atingido, a tarefa é rapidamente escalonada

Escalonamento baseado em Deadlines

Nos sistemas de tempo real, as seguintes informações são utilizadas:

- tempo que o processo/tarefa fica pronta
  - caso de tarefas periódicas – esta seqüência de tempos pode ser pré-conhecida
- deadline de início e de fim
  - tempo que uma tarefa tem que (a partir do qual deve) começar e tempo que a tarefa deve estar terminada
- tempo de processamento
  - em caso de desconhecimento, o SO pode usar algum modelo de previsão

Escalonamento baseado em Deadlines

- conjunto de recursos requisitados pelas tarefas (sem ser processadores)
- prioridade
  - tarefas críticas podem ter prioridade absoluta (tem que ser executadas o mais rápido)
    - caso de falha: sistema aborta
    - se executado de qualquer modo, prioridades são reavaliadas
- estrutura da tarefa
  - uma tarefa pode ser um conjunto de subtarefas, algumas sendo críticas e outras não

Escalonamento baseado em Deadlines

Seleção e decisões

- qual tarefa deve ser a próxima a ser escalonada
- mais do que acabar mais rápido, é mais importante que escalone as tarefas tal que as aplicações terminem de acordo com os deadlines
- não preempção
  - quando deadline de início devem ser utilizados, faz mais sentido visto que preempção não é permitida. Assim, dá para assegurar que a data limite de início será atingida

Escalonamento baseado em Deadlines

Seleção e decisões

- preempção ser permitida para atingir deadline
  - mais apropriado quando tarefas tem deadlines de fim
  - Processo X está sendo executado e Y está pronto – os dois tem deadlines de fim associados
    - dependendo, X sofre interrupção para que Y seja escalonado para atingir seu deadline, tal que X não perca seu deadline

Escalonamento baseado em Deadlines

Um exemplo: tarefas periódicas – se repetem com freqüência

- Tarefa A
  - deadlines a cada 20ms
  - duração de 10ms (cada período)
  - chega a cada 20ms
- Tarefa B
  - deadline a cada 50ms
  - duração de 25ms
  - chega a cada 50ms
- preempção é permitida

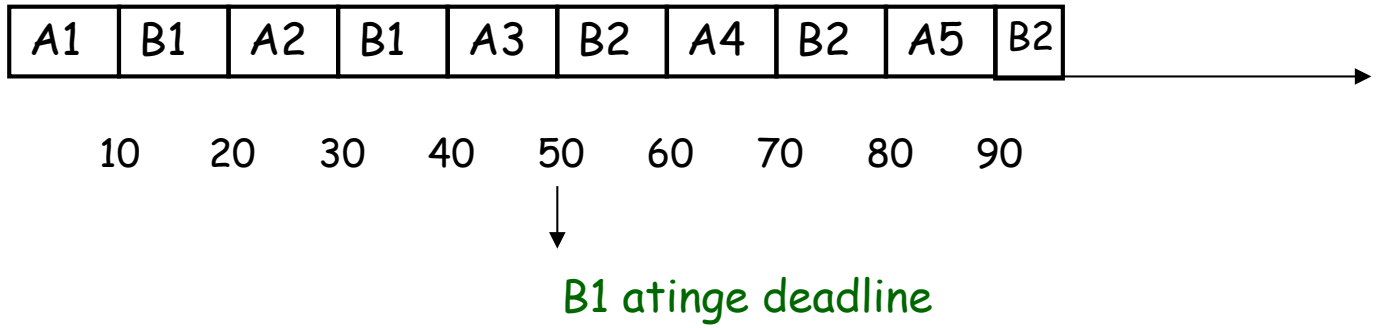
Escalonamento baseado em Deadlines

Processo periódicos	Chegada	Tempo de Execução	Deadline de fim
A(1)	0	10	20
A(2)	20	10	40
A(3)	40	10	60
A(4)	60	10	80
A(5)	80	10	100
.....	.....	.....	.....
B(1)	0	25	50
B(2)	50	25	100
.....	.....	.....	.....

Escalonamento baseado em Deadlines

- Decisões são efetuadas a cada 10ms, e escalonamento por prioridade

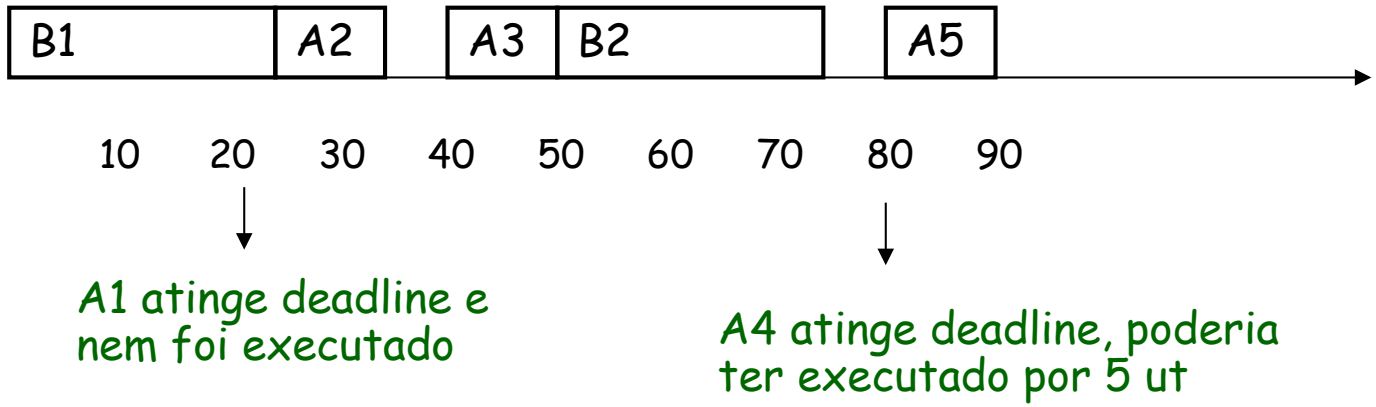
A tem prioridade



Escalonamento baseado em Deadlines

- Decisões são efetuadas a cada 10ms, e escalonamento por prioridade

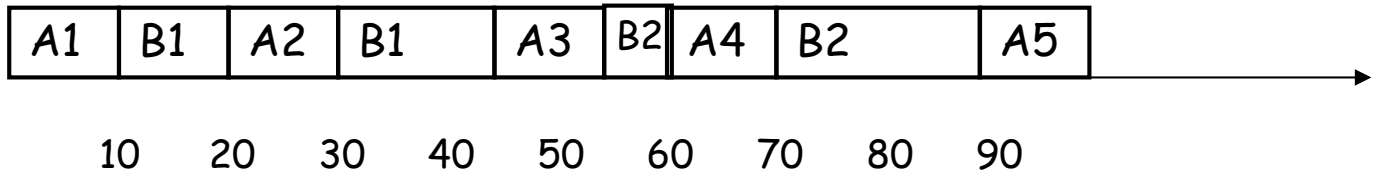
B tem prioridade



Escalonamento baseado em Deadlines

- Decisões são efetuadas a cada 10ms, e escalonamento por prioridade: menor deadline de fim

Menor deadline de fim



Escalonamento baseado em Deadlines

Tarefas Aperiódicas – sem preempção

Processo	Chegada	Ts	Deadline início
A	10	20	110
B	20	20	20
C	40	20	50
D	50	20	90
E	60	20	70

- earliest deadline:
  - A será logo escalonado e B não poderá ser executado

Escalonamento baseado em Deadlines

Tarefas Aperiódicas – sem preempção

Processo	Chegada	Ts	Deadline início
A	10	20	110
B	20	20	20
C	40	20	50
D	50	20	90
E	60	20	70

- menor deadline, conhecimento a priori, mesmo que o processador fique ocioso:
  - B é escalonado antes, mas o processador fica ocioso até este ser submetido (pois senão não será atendido devido a não preempção)

Escalonamento com taxa monotônica

Rate Monotonic Scheduling (RMS)

- muito usado em escalonamento de tempo real em tarefas que acontecem periodicamente
- Escalonar primeiro a tarefa de maior prioridade
  - Prioridade: escolha da tarefa com menor tempo de serviço
- considerando todas as tarefas prontas, ao ordenarmos as tempos de serviço, se observa que este tempo cresce monotonicamente

Escalonamento com taxa monotônica

- Em caso de tarefas periódicas, parâmetros relevantes para esse tipo de escalonamento:
  - tempo entre chegadas entre cada instância da tarefa = período T
    - o inverso deste período de tempo é a frequência (em Hz)
  - tempo de execução da tarefa C
    - C <= T
  - se a tarefa sempre é executada até seu término, a utilização do processador é

$$U = C/T$$

Escalonamento com taxa monotônica

- a política garante ou não que tarefas críticas encontrem seus deadlines
- Suponha que existam n tarefas. Para que todas encontrem seus deadlines é preciso que:

$$U(n) = C_1/T_1 + C_2/T_2 + ... + C_n/T_n <= 1$$

  - quer dizer, a soma de utilização de processador não pode ultrapassar de 1
  - quando igual a 1: o processador está sendo totalmente utilizado

Escalonamento com taxa monotônica

- Limite superior UL(n)
- Liu e Layland (1973) mostraram que para um conjunto de n tarefas periodas um escalonamento viável que alcança os deadlines é possível, se a utilização da CPU está de acordo com o limite

$$UL(n) = C_1/T_1 + C_2/T_2 + ... + C_n/T_n <= n\,(2^{1/n} - 1)$$

- isso depende do número de tarefas

Escalonamento com taxa monotônica

$$UL(n) = C_1/T_1 + C_2/T_2 + ... + C_n/T_n <= n\,(2^{1/n} - 1)$$

n	UL(n) = n (2^1/n – 1)
1	1.0
2	0,828
3	0,779
4	0,756
5	0,743
6	0,734
.....	.....
∞	ln 2 = 0,693

Escalonamento com taxa monotônica

- Suponha que existam três tarefas periódicas tal que $U_i = C_i/T_i$

Tarefa	C_i	T_i (período)	U_i
P1	20	100	0,2
P2	40	150	0,267
P3	100	350	0,286
		U(3) =	0,753

- O limite superior para que as três tarefas sejam escalonáves, usando escalonamento monotônico RMS, é :

$$UL(3) = C_1/T_1 + C_2/T_2 + C_3/T_3 <= 3\,(2^{1/3} - 1) = 0,779$$

- Como a utilização total de processador para as três tarefas é menor que o limite superior para o RMS então é possível executá-las de acordo com o RMS

Escalonamento com taxa monotônica

- Também pode ser comprovado que o limite superior da utilização de processador pode ser utilizado para escalonamento com a prioridade:
  - Deadline mais cedo

Escalonamento com taxa monotônica

- Considere as tarefas P_i com (C_i , T_i) associados

  {(1,3),(1,5),(1,6),(3,10)}

- Qual a utilização de CPU:

- Qual o limite superior

- Conclusão?

Escalonamento com taxa monotônica

- Considere as tarefas Pi com (Ci , Ti) associados

  {(1,3),(1,5),(1,6),(3,10)}

- Qual a utilização de CPU:
  - U (4) = 1/3 + 1/5 + 1/6 + 3/10 = 0.899
- Qual o limite superior
  - UL(4) = 4 ($2^{1/4}$– 1) = 0.756
- Conclusão?
  - as quatro tarefas não são escalonáveis

Escalonamento com taxa monotônica

- Considere as tarefas Pi com (Ci , Ti) associados

  {(1,3),(1,5),(1,6),(3,10)}

- E as três primeiras tarefas?

Escalonamento com taxa monotônica

- Considere as tarefas Pi com (Ci , Ti) associados

  {(1,3),(1,5),(1,6),(3,10)}

- E as três primeiras tarefas?
  - U(3)= 1/3 + 1/5 + 1/6 = 0.699
  - UL (3) = 0.779
- Logo: as três primeiras tarefas são escalonáveis.