Instituto de Informática - UFRGS

Sistemas Operacionais

Gerência do processador

Escalonamento

Aula xx

Introdução

- O escalonador é a entidade do sistema operacional responsável por selecionar um processo* apto para executar no processador
- O objetivo é maximizar o uso do processador
- Típico de sistemas multiprogramados: *batch, interativos* ou tempo real
  - Requisitos e restrições diferentes em relação a utilização da CPU
- Duas partes:
  - Escalonador: implementa um mecanismo e uma política de seleção
  - *Dispatcher:* efetua a troca de contexto

*válido também para threads

Escalonador

- É um mecanismo que define:
  - QUEM, entre os processos aptos, receberá o direito de usar a CPU
    - Existência de uma política que pode levar em conta prioridades, regras de desempate (tie break) entre processos
  - QUANDO deve ser executado
    - Modos não preemptivo e preemptivo (tempo ou prioridade)
- Prioridades
  - Pode ser estática ou dinâmica
  - Um processo no estado "executando" deve ter prioridade maior ou igual que qualquer outro processo no estado apto
    - Pressupõe preempção (poder de retirar um processo da CPU)
  - Possível haver escalonadores não preemptivos, mas com prioridades

Escalonadores não preemptivos e preemptivos

- Uma vez selecionado, um processo utiliza o processador até que:
  - Não preemptivo:
    - Término de execução do processo
    - Execução de uma requisição de E/S ou sincronização bloqueante
    - Liberação voluntária do processador a outro processo (*yield*)
  - Preemptivo:
    - Término de execução do processo
    - Execução de uma requisição de entrada/saída ou sincronização
    - Liberação voluntária do processador a outro processo (*yield*)
    - Interrupção de relógio (preempção por tempo)
    - Processo de mais alta prioridade esteja pronto para executar (preempção por prioridade)

Níveis de escalonamento

- Na verdade, existem diferentes níveis de escalonamento
  - Curto prazo
  - Médio prazo
  - Longo prazo
- O escalonador de curto prazo é o mais importante

Escalonador longo prazo

- Determina quais processos o sistema permite que disputem ativamente os recursos do sistema
  - Controle de admissão
- Controla o grau de multiprogramação do sistema, isto é, o número de processos em um determinado instante de tempo
  - Quanto maior o número de processos, menor a porcentagem de tempo de uso do processador por processo, mas aumenta a chance do processador sempre estar ocupado (eficiência de uso)

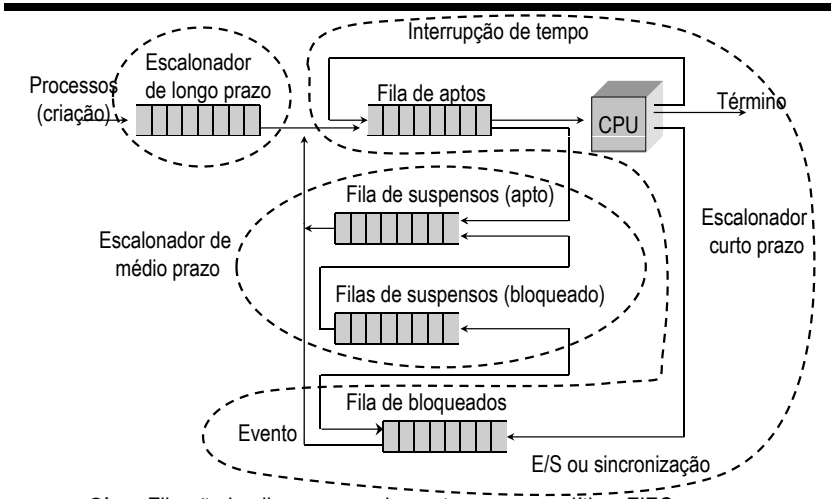
Escalonador médio prazo

- Determina quais processos admitidos têm permissão para competir pelo uso do processador
- Responsável por suspender/resumir processos
  - Associado a gerência de memória (participa do mecanismo de *swapping)*
- Suporte adicional a multiprogramação
  - Grau de multiprogramação efetiva (diferencia aptos dos aptos-suspensos)

Escalonador de curto prazo

- Executado mais frequentemente (mais importante)
  - Quando a CPU se torna livre
  - Processo de maior prioridade no estado de apto (escalonador preemptivo)
- Determina qual processo apto deverá utilizar o processador
- Executado sempre que ocorre eventos importantes:
  - Chamadas de sistema (E/S, sincronização, término, passagem voluntária)
  - Sinais (interrupção software)
  - Interrupções de hardware (tempo, conclusão de E/S, falta de páginas, proteção)
  - Interrupções de exceções (overflow, underflow)
  - Término de processos

Diagrama de escalonamento



Obs.: Fila não implica necessariamente em uma política FIFO

Objetivos gerais do escalonamento

- Maximizar a utilização do processador
- Maximizar a produção do sistema (*throughput* ou vazão)
  - Número de processos completados por unidade de tempo
- Minimizar o tempo de execução ou retorno (*turnaround*)
  - Intervalo entre a submissão até a conclusão de um determinado processo
- Minimizar o tempo de espera
  - Tempo que um processo permanece na lista de aptos
- Minimizar o tempo de resposta
  - Tempo decorrido entre uma requisição e sua realização (sistemas interativos)
  - Em sistemas interativos priorizar a variância à média
- Justiça:
  - Processos semelhantes devem ser tratados da mesma forma
  - Evitar adiamento indefinido (postergação ou *starvation*)

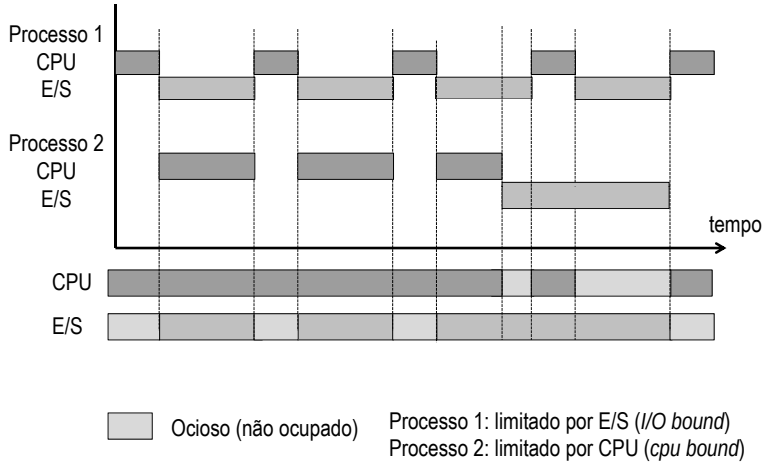
Categorias de sistemas operacionais

- Ambientes diferentes, têm objetivos e requisitos diferentes
- Três grandes categorias
  - Em lote (batch)
    - Vazão, tempo de retorno e utilização da cpu
  - Interativos
    - Tempo de resposta
    - proporcionalidade
  - Tempo real
    - Cumprimento de prazos
    - Previsibilidade

Processos CPU *bound* e I/O *bound*

- Um processo é dito CPU bound quando:
  - Ciclo de processador >> ciclo de E/S
- Um processo é dito I/O bound quando:
  - Ciclo de E/S >> ciclo de processador
- Sem quantificação exata
- Situação ideal: misturar processos CPU bound com I/O bound
  - Executar os CPU bounds quando os processos I/O bound estão bloqueados a espera de E/S→ superposição de tarefas de processamento e de E/S

Execução em ambientes com multiprogramação



Algoritmos de escalonamento

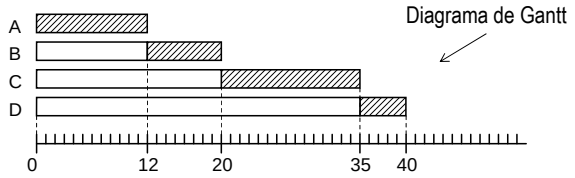
- Implementam a política de seleção do processo a ser alocado na CPU
  - Algoritmos não preemptivos (cooperativos)
    - *First-In First-Out* (FIFO) ou *First-Come First-Served* (FCFS)
    - *Shortest Job First* (SJF)
    - *Shortest Process Next* (SPN) ou *Shortest Request Next* (SRN)
    - *High Response Ratio Next* (HRRN)
  - Algoritmos preemptivos
    - Round robin (circular)
    - Baseado em prioridades
  - Existem outros algoritmos de escalonamento (tempo real)
    - *Rate Monotonic* (RM)
    - *Earliest Deadline First* (EDF)
    - etc...

FIFO - *First In First Out*

- *First-Come, First-Served* (FCFS)
- Adequado para sistemas em lote (batch)
- Simples de implementar (fila)
- Funcionamento:
  - Processos são organizados por ordem de chegada no estado apto
    - Processos que se tornam aptos são inseridos no final da fila
  - Processo que está na frente da fila é o próximo a executar
  - Processo executa até que:
    - Libere explicitamente o processador (operação yield())
    - Realize uma chamada de sistema bloqueante
    - Termine sua execução

FIFO - *First In First Out* (*cont.*)

Processo	Tempo
A	12
B	8
C	15
D	5

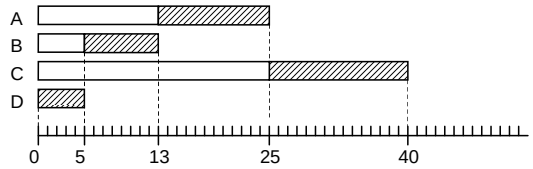


- Tempo médio de espera na fila de execução:
  - Ordem A-B-C-D = (0 + 12 + 20 + 35) / 4 = 16.75 u.t.
  - Ordem D-A-B-C = (0 + 5 + 17 + 25) / 4 = 11.75 u.t.
- Tempo médio de retorno (turnaround) na fila de execução:
  - Ordem A-B-C-D = ((0+12)+ (12+8) +(20+15) + (35+5)) / 4 = 26.75 u.t.
  - Ordem D-A-B-C = ((0+5) + (5+12) + (17+8) + (25+15)) / 4 = 21.75 u.t.
- Desvantagem: Prejudica processos *I/O bound*

SJF - *Shortest Job First*

- Sistemas em lote
- Seleciona processo apto com menor tempo de execução estimado
  - Introduz uma noção de prioridade (não preemptivo)
- Algoritmo ótimo (tempo de espera), isto é, fornece o menor tempo médio de espera para um conjunto de processos

Processo	Tempo
A	12
B	8
C	15
D	5



Tempo de espera médio: (0 + 5 + 13 + 25)/4 = 10,75 u.t
Tempo de retorno médio: ((0+5)+(5+8)+(13+12)+(25+15))/4 = 20,75 u.t.

SPN - *Shortest Process Next*

- *Shortest Job First* foi concebido para sistemas em lote (batch)
  - Ambientes em lote se tem uma estimativa de tempo de execução dos jobs
  - Adaptação para sistemas interativos considerando cada ciclo de CPU como se fosse um job → SPN
- Também denominado de *Shortest Request Next* (SRN)
- Problema: como determinar (estimar) o tempo de execução futuro?
  - Resposta: prever o futuro com base no comportamento passado

Prevendo o futuro...

- Pode ser feito utilizando os tempos de ciclos já passados e realizando uma média exponencial

1. t_n = tempo do enésimo ciclo de CPU
2. τ_{n+1} = valor previsto para o próximo ciclo de CPU
3. τ = armazena a informação dos ciclos passados (n - 1)
4. $\alpha, 0 \le \alpha \le 1$
5. Define - se : $\tau_{n+1} = \alpha\, t_n + (1-\alpha)\tau_n.$

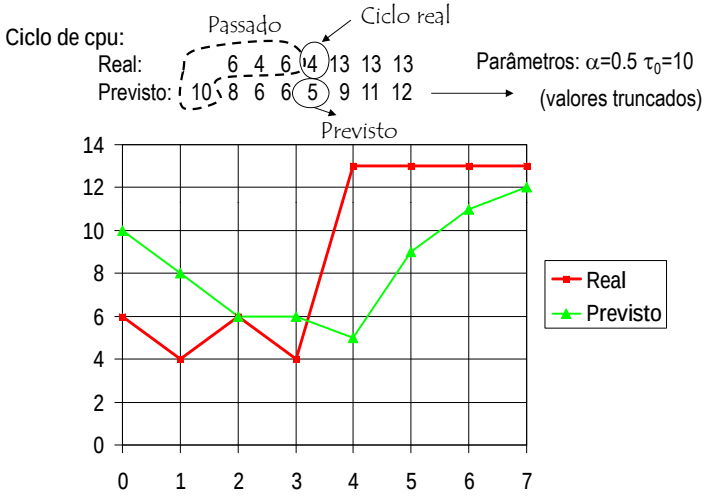
- Forma simplificada:
  - Tempo_futuro_estimado = tempo_ultimo_ciclo + tempo_passado
- Fator α fornece importância para o passado recente (último ciclo) ou para a história (ciclos passados)

Prevendo o futuro... (cont.)

- Não considera o último ciclo de processador, só o passado (α =0)
  - $\tau_{n+1} = \tau_n$
- Considera apenas o último ciclo de processador (α = 1)
  - $\tau_{n+1} = t_n$
- α =0.5
  - Tem o efeito de considerar o mesmo peso para a história recentel e passada

$$\tau_{n+1} = \alpha\, t_n + (1-\alpha)\,\alpha\, t_{n-1} + \ldots + (1-\alpha)^j\,\alpha\, t_{n-j} + \ldots + (1-\alpha)^{n+1}\,\tau_0$$

Exemplo de “previsão do futuro”



Escalonamento não preemptivo e prioridades

- Prioridade é usada como critério de seleção APENAS quando a CPU fica livre
  - Exemplo: SPF é um forma de priorizar processos, pois a prioridade é o inverso do próximo tempo previsto para ciclo de CPU
- Processos de igual prioridade podem ser executados de acordo com uma política FIFO
- Problema de adiamento indefinido ou inanição (*starvation*)
  - Processo de baixa prioridade não é alocado a CPU por sempre existir um processo de mais alta prioridade a ser executado
- Solução: envelhecimento (*aging*)
  - Elevação temporária da prioridade de um processo

High Response Ratio Next (HRRN)

- Forma de implementar envelhecimento (*aging*)

$$response\_ratio = \frac{Tempo\_espera + Tempo\_serviço}{Tempo\_serviço}$$

- Onde:
  - Tempo de espera = tempo passado no estado apto desde sua entrada nele
  - Tempo de serviço = tempo necessário a execução (ciclo de CPU)

Escalonadores preemptivos

- Escalonamento preemptivo significa que o sistema pode retirar o processador de um processo* para entregá-lo a outro processo
- Preempção pode ser por:
  - Tempo: um processo esgotou um tempo máximo de ciclo de processador
  - Prioridade: um processo de mais alta prioridade ficou pronto para executar
- Portanto um escalonador preemptivo executa quando:
  - O processo em execução termina
  - O processo em execução se bloqueia (requisição de E/S ou sincronização)
  - Libera voluntariamente o processador (chamada de sistema *yield*)
  - Ocorre interrupção de relógio (verificação se esgotou ou não o tempo máximo)
  - Um processo de mais alta prioridade entrou no estado apto

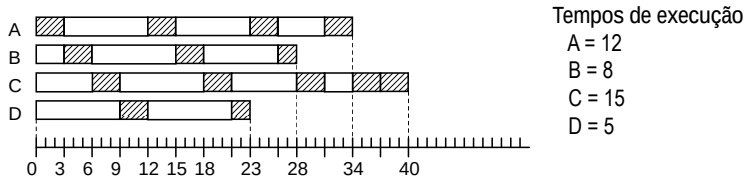
*Vale para threads também!

Algoritmos de escalonamento preemptivos

- Preempção por tempo
  - Round-Robin (RR)
- Baseado em prioridades
  - Shortest Remaining Time Next (SRTN)
  - Múltiplas filas
  - Prioridades dinâmicas (múltiplas filas com realimentação)

RR - *Round Robin*

- Similar ao algoritmo FIFO, só que:
  - Cada processo recebe um tempo limite máximo (*time-slice*, *quantum*) para executar um ciclo de processador
- Fila de processos aptos é uma fila circular
- Necessidade de um relógio para delimitar as fatias de tempo
  - Interrupção de tempo



RR - *Round Robin* (*cont*.)

- Por ser preemptivo, um processo perde o processador quando:
  - Liberar explicitamente o processador (*yield*)
  - Realizar uma chamada de sistema (bloqueado)
  - Terminar sua execução
  - Quando esgotar sua fatia de tempo
- Se *quantum* $\rightarrow \infty$ obtém-se o comportamento de um escalonador FIFO

Problemas com o *Round Robin*

- Problema 1: Dimensionamento do *quantum*
  - Compromisso entre *overhead* e tempo de resposta em função do número de usuários (*1/k* na presença de *k* usuários)
  - Compromisso entre tempo de chaveamento e tempo do ciclo de processador (*quantum*)
- Problema 2: Processos *I/O bound* são prejudicados
  - Esperam da mesma forma que processos *CPU bound*, porém quando ganham a CPU, muito provavelmente, não utilizam todo o seu *quantum*
  - Solução:
    - Prioridades: Associar prioridades mais altas aos processos *I/O bound* para compensar o tempo gasto no estado de espera (apto)

Shortest Remaining Time Next (SRTN)

- Versão preemptiva do *Shortest Job/Process First* (SJF-SPF)
- Princípio:
  - Escolhe o processo cujo tempo de execução restante seja o menor
  - Quando um processo entra no estado apto seu tempo é comparado com aquele que está em "executando"
    - Se for menor, preempta o que está executando
- Privilegia tarefas "curtas" ou aquelas que tem o menor ciclo de CPU
  - Forma de priorizar processos I/O bound

Prioridades

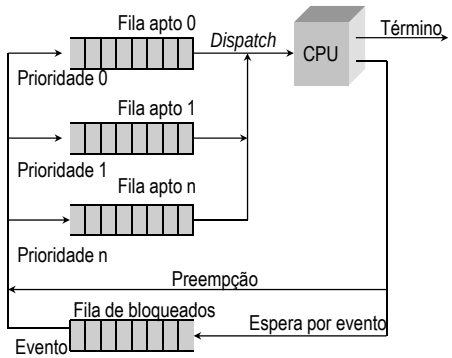
- Se existir, é um critério para selecionar processos
- Prioridade pode ser:
  - Estática: definida no momento da criação (alterada via chamada de sistema)
  - Dinâmica: modifica durante a execução do processo em função de condições de utilização (carga e recursos) do sistema
    - e.g: Processo que detém recurso importante pode ter prioridade aumentada, processo que executa demais pode ter prioridade reduzida etc
- Pode ser incluído em políticas preemptivas e não preemptivas:
  - Preemptiva: "poder" de retirar processo de menor prioridade da CPU
  - Não-preemptiva: A prioridade é considerada apenas no momento da selecionar um dos processos que está no estado de apto para passar a executando.

Escalonamento com prioridades preemptivo

- Quando um processo de maior prioridade que o processo em execução entrar no estado apto deve ocorrer uma preempção
- Havendo mais de um processo com uma mesma prioridade se aplica uma política de escalonamento entre eles:
  - Round-Robin
  - FIFO (FCFS)
  - SJF (SPF)

Implementação de escalonador com prioridades

- Múltiplas filas associadas ao estado apto
- Cada fila uma prioridade
  - Pode ter sua própria política de escalonamento (FIFO, SJF, RR)



Exemplo: *pthreads*

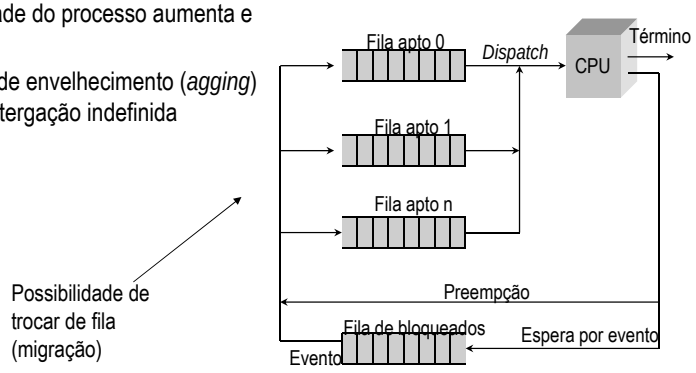
- É um exemplo de implementação: Não é regra geral!!
- A política de escalonamento FIFO com prioridade considera:
  - Quando um processo em execução é preemptado ele é inserido no ínicio de sua fila de prioridade
  - Quando um processo bloqueado passa a apto ele é inserido no final da fila de sua prioridade
  - Quando um processo troca de prioridade ele é inserido no final da fila de sua nova prioridade
  - Quando um processo em execução "passa a vez" para um outro processo ele é inserido no final da fila de sua prioridade

Problemas com prioridades

- Um processo de baixa prioridade pode não ser executado
  - Postergação indefinida (*starvation*)
- Processo com prioridade estática pode ficar mal classificado e ser penalizado ou favorecido em relação aos demais
  - Típico de processos que durante sua execução trocam de padrão de comportamento (*CPU bound* a *I/O bound* e vice-versa)
- Solução:
  - Múltiplas filas com realimentação

Múltiplas filas com realimentação

- Baseado em prioridades dinâmicas
- Em função do tempo de uso da CPU a prioridade do processo aumenta e diminui
- Sistema de envelhecimento (*agging*) evita postergação indefinida



Escalonamento por fração justa (*fair share*)

- Um escalonador deve ser justo com os usuários do sistema
  - Tomar decisões baseados só em processos leva a injustiças
    - Ex: usuário 1 com 9 processos e usuário 2 com 1 processo. O usuário 1 potencialmente usaria 90% do processador.
- Solução:
  - Considerar o proprietário do processo como parte do procedimento da política de escalonamento
    - Dois usuários devem receber 50% do processador independente do número de processos que cada um deles detém

Escalonamento em múltiplos processadores

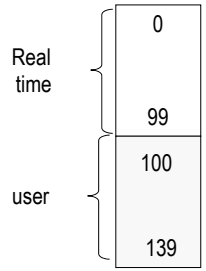
- Duas categorias
  - Assimétricos: uma CPU executa o sistema operacional, as demais executam aplicações de usuários
  - Simétricos: todas CPUs executam o sistema operacional e aplicações usuário
    - Estratégias: Fila de aptos única para todas CPUs ou uma fila por CPU
- Noção de afinidade
  - Tentativa de manter o processo/thread executando na mesma CPU
    - Pode ser afinidade rígida ou flexível
    - Objetivo é reaproveitar informação já presente nos níveis de cache
- Balanceamento de carga
  - Com estratégia "uma fila por CPU" é preciso manter todas ocupadas
  - Migração entre filas de aptos se opõe a noção de afinidade

Processadores multicore e com *hyperthreading*

- São vistos pelo sistema operacional como multiprocessadores
- Hyperthreading é a definição de processadores lógicos em processadores físicos
  - O sistema operacional "enxerga" os processadores lógicos para efeitos de escalonamento
    - Ex.: Intel i3 (dual core com HT): cada core tem dois lógicos (HT), então aparece para o sistema operacional como quatro processadores

Estudo de caso: escalonamento Linux 2.6

- Duas classes em função do tipo de processos (*threads*)
  - User: Processos interativos e *batch* (regular)
  - RT: Processos de tempo real
- 140 níveis diferentes de prioridade
  - Escalona sempre o processo (*thread*) de maior prioridade
- Valores de *quantum* por nível de prioridade
  - Maior a prioridade, maior o *quantum*
  - *Quantum* é o número de ciclos que um processo pode continuar em execução
    - Unidade de ciclo é 1ms (denominado *jiffy*)



Escalonamento Linux (classe *tempo real*)

- Prioridades entre 0 e 99
- Esquema de prioridade fixa
  - Definida por um usuário com privilégios especiais
- Seleciona o processo de maior prioridade para executar
  - Múltiplas filas sem realimentação
- Políticas de escalonamento (padrão POSIX)
  - SCHED_FIFO: escalonador FIFO com prioridade estática
  - SCHED_RR: escalonador Round-robin com prioridade estática
- Na verdade NÃO são escalonadores de tempo real
  - Não garantem prazo!!
  - São simplesmente mais prioritários que a classe USER

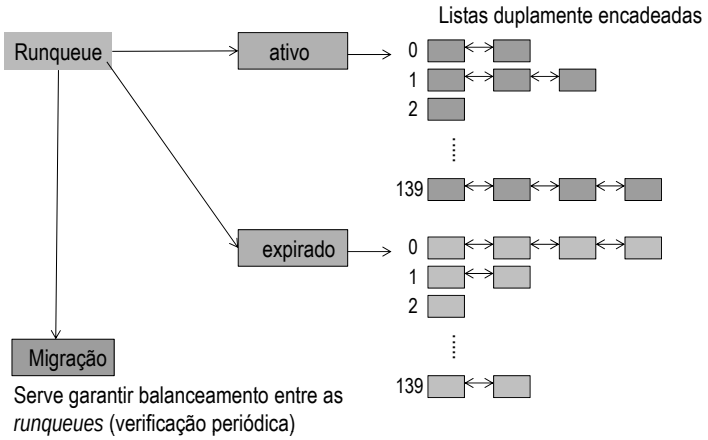
Escalonamento linux (classe *USER*)

- Prioridades entre 100 e 139
- Múltiplas filas com realimentação (prioridade dinâmica)
- Valor *nice*
  - Uma espécie de prioridade estática a ser adicionada a dinâmica
    - Varia entre -20 a +19 respeitando os limites (100 a 139)
- Prioridade dinâmica
  - Recalculado periodicamente e no esgotamento do *quantum*
    - Processos interativos (I/O bound) recebem bônus de prioridade (até -5)
    - Processos CPU/bound são penalizados na prioridade (até +5)
    - Maior a prioridade, maior o *quantum*
  - Idéia: permitir que os I/O bound executem logo porque eles ficarão bloqueados em operações de E/S

Fila de execução (*Runqueue*)

- Estrutura de dados, por processador, que representa o estado "apto"
- Possui ponteiros para dois vetores e uma função (migração)
  - Vetores Ativo e Expirado (140 elementos cada)
  - Cada elemento do vetor é o cabeça de uma lista encadeada
    - Processos de prioridade *i* estão encadeados a partir desse elemento *i*
- Princípio de funcionamento
  - Seleciona o processo de mais alta prioridade do vetor de ativos
  - Se *quantum* do processo esgotar, é movido para o vetor de expirados
    - Se FIFO não há *quantum*
    - Classe USER pode ser colocado em um nível de prioridade diferente
  - Quando não houver mais processos na lista de ativos, inverte o valor dos ponteiros "ativo" e "expirados"

Esquema da estrutura *runqueue*



Serve garantir balanceamento entre as *runqueues* (verificação periódica)

Estudo de caso: escalonamento windows

- Unidade de escalonamento é a *thread*
- Escalonador preemptivo com prioridades
- Prioridades organizadas em duas classes:
  - Tempo real: prioridade estática (níveis 16-31)
  - Variável: prioridade dinâmica (níveis 0-15)
- Cada classe possui 16 níveis de prioridades
  - Cada nível é implementado por uma fila em uma política *round-robin*
    - Múltiplas filas: classe de tempo real
    - Múltiplas filas com realimentação: classe de tempo variável
  - Threads da classe tempo real tem precedência sobre as da classe variável

Escalonamento windows (classe variável)

- Dois parâmetros definem a prioridade de uma *thread:*
  - Valor de prioridade de base do processo
  - Prioridade inicial que indica sua prioridade relativa dentro do processo
- Prioridade da *thread* varia de acordo com uso do processador
  - Preemptada por esgotar o *quantum*: prioridade reduzida
  - Preemptada por operação de E/S: prioridade aumentada
  - Nunca assume valor inferior a sua prioridade de base, nem superior a 15
- Fator adicional em máquina multiprocessadoras: afinidade!
  - Tentativa de escalonar uma *thread* no processador que ela executou mais recentemente

Leituras complementares

- A. Tanenbaum. *Sistemas Operacionais Modernos* (3ª edição), Pearson Brasil, 2010.
  - Capítulo 2: seções 2.4.1 a 2.4.3
- A. Silberchatz, P. Galvin; *Sistemas Operacionais*. (7ª edição). Campus, 2008.
  - Capítulo 5 (seções 5.1, 5.2 e 5.3)
- R. Oliveira, A. Carissimi, S. Toscani; *Sistemas Operacionais*. Editora Bookman 4ª edição, 2010
  - Capítulo 4 (seções 4.4 e 4.5)