UNIVALI

Universidade do Vale do Itajaí
Escola Politécnica

Escalonamento de CPU

1

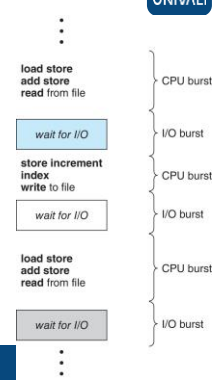
Agenda

- Baseado no capítulo 6 do livro base da disciplina
  - Conceitos Básicos (6.1)
  - Critérios de escalonamento (6.2)
  - Algoritmos de escalonamento (6.3)
  - Escalonamento de threads (6.4)
  - Escalonamento de vários processadores (6.5)
  - Exemplos de sistemas operacionais (6.7)
- Livro base: Abraham Silberschatz, Peter Baer Galvin, Greg Gagne. Fundamentos de sistemas operacionais

2

Conceitos Básicos

- Utilização máxima da CPU obtida com multiprogramação
- Ciclo de intermitência de CPU–I/O – A execução do processo consiste em um ciclo de execução da CPU e espera de E/S
- **Burst da CPU seguido de burst de E/S**
- A distribuição de burst da CPU é a principal preocupação

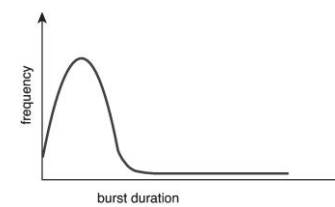


3

Histograma de tempos de burst de CPU

Grande número de rajadas curtas

Pequeno número de rajadas mais longas



4

Escalonamento de CPU

- O escalonador de CPU seleciona entre os processos na fila de aptos e aloca um núcleo de CPU para um deles
  - A fila pode ser ordenada de várias maneiras
- Decisões de escalonamento de CPU podem ocorrer quando um processo:
  1. Alterna do estado de execução para o estado de espera
  2. Alterna do estado em execução para o estado pronto
  3. Muda de espera para pronto
  4. Termina
  - Para as situações 1 e 4, não há escolha em termos de escalonamento. Um novo processo (se existir um na fila pronta) deve ser selecionado para execução.
  - Para as situações 2 e 3, no entanto, há uma escolha.

5

Escalonamento preemptivo e não preemptivo

- Quando o escalonamento ocorre apenas nas circunstâncias 1 e 4, o esquema de escalonamento não é preemptivo
- Caso contrário, é preemptivo
- Em escalonamento não preemptivo, uma vez que a CPU tenha sido alocada a um processo, o processo mantém a CPU até liberá-la encerrando ou alternando para o estado de espera
- Praticamente todos os sistemas operacionais modernos, incluindo Windows, MacOS, Linux e UNIX, usam algoritmos de agendamento preemptivo

6

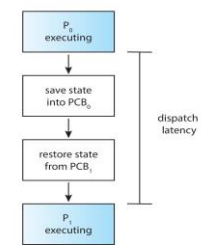
Escalonamento Preemptivo e Condições de Corrida

- O escalonamento preemptivo pode resultar em condições de corrida quando os dados são compartilhados entre vários processos
- Considere o caso de dois processos que compartilham dados
  - Enquanto um processo está atualizando os dados, ele é preemptado para que o segundo processo possa ser executado
  - O segundo processo, em seguida, tenta ler os dados, que estão em um estado inconsistente

7

Dispatcher (despachador)

- O módulo Dispatcher dá o controle da CPU ao processo selecionado pelo escalonador da CPU:
  - Chaveamento de contexto
  - Chaveamento para o modo de usuário
  - Saltar para a localização adequada no programa de usuário para reiniciar esse programa
- **Latência de despacho** – tempo que leva para o dispatcher parar um processo e colocar outro em execução



8

Critérios de escalonamento

- **Utilização da CPU** – manter a CPU o mais ocupada possível
- **Vazão** – Nº de processos que concluem sua execução por unidade de tempo
- **Tempo de resposta** – quantidade de tempo para executar um determinado processo
- **Tempo de espera** – quantidade de tempo que um processo está aguardando na fila pronta
- **Tempo de resposta** – quantidade de tempo que leva desde o momento em que uma solicitação foi enviada até que a primeira resposta seja produzida

9

Critérios de otimização do algoritmo de agendamento

- Utilização máxima da CPU
- Taxa de vazão máxima
- Tempo de resposta mínimo
- Tempo de espera mínimo
- Tempo de resposta mínimo

10

Escalonamento - First- Come, First-Served (FCFS) ou First in, First out (FIFO)

Processo	Tempo de Burst
P_1	24
P_2	3
P_3	3

- Suponha que os processos cheguem na ordem: P_1 , P_2 , P_3
  O Gráfico para o escalonamento é:

- Tempo de espera para P_1 = 0; P_2 = 24; P_3 = 27
- Tempo de espera médio: (0 + 24 + 27)/3 = 17

11

Escalonamento FCFS Scheduling

Suponha que os processos cheguem na ordem:

P_2 , P_3 , P_1

- O gráfico para o escalonamento é:

- Tempo de espera para P_1 = 6; P_2 = 0, P_3 = 3
- Tempo de espera médio : (6 + 0 + 3)/3 = 3
- Muito melhor do que o caso anterior
- **Efeito comboio** - processo curto antes de um processo longo
  - Considere um processo vinculado à CPU e muitos processos vinculados a E/S

12

Escalonamento - Shortest-Job-First (SJF)

- Associe a cada processo a duração do seu próximo burst de CPU
  - Use esses "tempos" para escalonar o processo com o menor tempo
- SJF é ideal – fornece tempo de espera médio mínimo para um determinado conjunto de processos
  - A dificuldade é saber o "tempo" da próxima solicitação de CPU
  - Poderia perguntar ao usuário

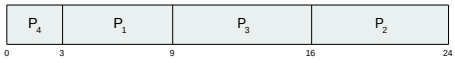
Escalonamento - Shortest-Job-First (SJF)

- Versão preemptiva chamada de **shortest-remaining-time-first**
- Como determinamos o tempo do próximo burst de CPU?
  - Poderia perguntar ao usuário
  - Estimar: como?

Example of SJF

Processo	Tempo de Burst
P_1	6
P_2	8
P_3	7
P_4	3

- Gráfico de escalonamento SJF



- Tempo de espera médio = (3 + 16 + 9 + 0) / 4 = 7

Determinando o "tempo" do próximo burst de CPU

- Só pode estimar o comprimento – deve ser semelhante ao anterior
  - Em seguida, escolha o processo com o próximo burst de CPU previsto mais curto
- Pode ser feito usando o comprimento de intermitências de CPU anteriores, usando a média exponencial
  1. t_n = actual length of n^{th} CPU burst
  2. τ_{n+1} = predicted value for the next CPU burst
  3. $\alpha, 0 \le \alpha < 1$
  4. Define : $\tau_{n+1} = \alpha t_n + (1 - \alpha)\tau_n$.
- Comumente, α definido como 1/2

Escalonamento - Round Robin (RR)

- Cada processo recebe uma pequena unidade de tempo de CPU (tempo quântico q), geralmente 10-100 milissegundos. Após esse tempo decorrido, o processo é antecipado e adicionado ao final da fila pronta
- Se houver n processos na fila pronta e o quantum de tempo for q, então cada processo obtém 1/n do tempo da CPU em pedaços de no máximo q unidades de tempo de uma só vez. Nenhum processo espera mais do que (n-1)q unidades de tempo.
- O temporizador interrompe cada quantum para agendar o próximo processo
  - *q* grande ⇒ FIFO
  - *q* pequeno ⇒ *q deve ser grande em relação ao chaveamento de contexto, caso contrário, a sobrecarga é muito alta*

20

Exemplo de RR com Quantum de Tempo = 4

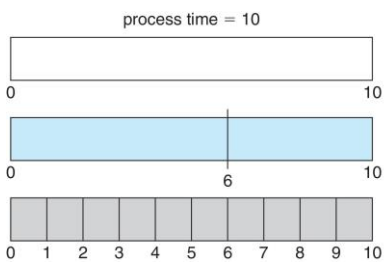
Processo	Tempo de Burst
P_1	24
P_2	3
P_3	3

- O gráfico do RR é:

- Normalmente, maior prazo médio de resposta do que o SJF, mas melhor resposta
- q deve ser grande em comparação com o tempo de troca de contexto
  - q geralmente de 10 milissegundos a 100 milissegundos,
  - Chaveamento de contexto < 10 microssegundos

21

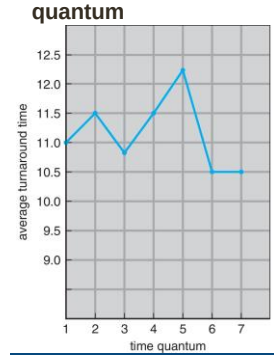
Tempo de Quantum e Tempo de Chaveamento de Contexto



quantum	context switches
12	0
6	1
1	9

22

O tempo de resposta varia com o tempo de quantum



process	time
P_1	6
P_2	3
P_3	1
P_4	7

80% dos bursts de CPU devem ser menores que q

23

Escalonamento por prioridade

- Um número de prioridade (inteiro) é associado a cada processo
- A CPU é alocada para o processo com a prioridade mais alta (menor inteiro ≡ prioridade máxima, porém nem sempre)
  - Preemptivo
  - Não preemptivo
- SJF é um escalonamento de prioridade em que a prioridade é o inverso do próximo tempo previsto de burst na CPU
- Problema ≡ **Starvation (ou Inaniação ou morrer de fome)** – processos de baixa prioridade podem nunca ser executados
- Solução ≡ **Aging (Envelhecimento)** – à medida que o tempo avança, aumenta a prioridade do processo

Exemplo de Escalonamento por Prioridade

Processo	Tempo de Burst	Prioridade
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2

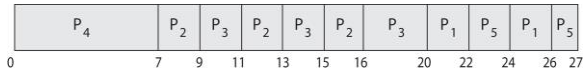
- Gráfico de escalonamento por prioridade

- Tempo de espera médio = 8,2

Escalonamento por prioridade c/ Round-Robin

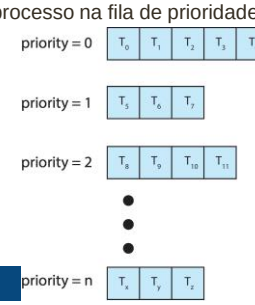
Processo	Tempo de Burst	Prioridade
P_1	4	3
P_2	5	2
P_3	8	2
P_4	7	1
P_5	3	3

- Execute o processo com a prioridade mais alta. Processos com a mesma prioridade são executados em round-robin
- Gráfico com quantum de tempo = 2



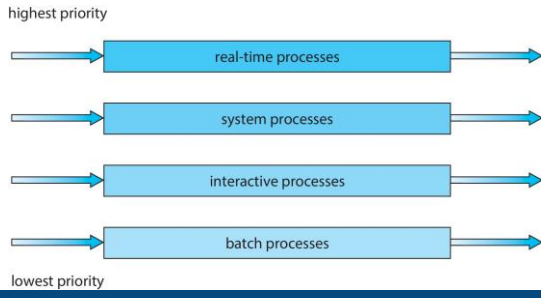
Fila de vários níveis

- Com o escalonamento por prioridade, tenha filas separadas para cada prioridade
- Escalone o processo na fila de prioridade mais alta!



Fila de vários níveis

- Priorização com base no tipo de processo



28

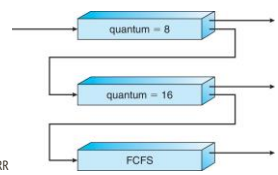
Fila de vários níveis com realimentação

- Um processo pode se mover entre as várias filas.
- Escalonador das filas de vários níveis com realimentação definido pelos seguintes parâmetros:
  - Número de filas
  - Algoritmos de escalonamento para cada fila
  - Método usado para determinar quando atualizar um processo
  - Método usado para determinar quando rebaixar um processo
  - Método usado para determinar qual fila um processo entrará quando esse processo precisar de serviço
- O envelhecimento pode ser implementado usando as filas de vários níveis com realimentação

29

Exemplo de filas de vários níveis com realimentação

- Três filas:
  - Q_0 – RR com tempo quântico 8 milissegundos
  - Q_1 – RR tempo quântico 16 milissegundos
  - Q_2 – FCFS
- Escalonamento
  - Um novo processo entra na fila Q_0 que é escalonado em RR
    - Quando ganha CPU, o processo recebe 8 milissegundos
    - Se não terminar em 8 milissegundos, o processo será movido para a fila Q_1
  - Em Q_1 o trabalho é novamente servido em RR e recebe 16 milissegundos adicionais
    - Se ainda não for concluído, ele será antecipado e movido para a fila Q_2



30

Escalonamento de threads

- Distinção entre threads no nível do usuário e no nível do kernel
- Quando threads são suportadas, threads são agendadas, não processos
- Modelos muitos-para-um e muitos-para-muitos, a biblioteca de threads agenda threads de nível de usuário para execução em LWP
  - Conhecido como **process-contention scope** (**PCS**) uma vez que a competição de escalonamento está dentro do processo
  - Normalmente feito através de prioridade definida pelo programador
- O escalonamento da thread do kernel para a CPU disponível é **system-contention scope** (**SCS**) – competição entre todas as threads do sistema

31

Escalonamento na Pthread

- A API permite especificar PCS ou SCS durante a criação de threads
  - PTHREAD_SCOPE_PROCESS escalona threads usando o escalonamento PCS
  - PTHREAD_SCOPE_SYSTEM escalona threads usando o escalonamento SCS
- Pode ser limitado pelo sistema operacional – Linux e macOS só permitem PTHREAD_SCOPE_SYSTEM

32

Pthread Scheduling API

```
#include <pthread.h>
#include <stdio.h>
#define NUM_THREADS 5
int main(int argc, char *argv[]) {
   int i, scope;
   pthread_t tid[NUM_THREADS];
   pthread_attr_t attr;
   /* get the default attributes */
   pthread_attr_init(&attr);
   /* first inquire on the current scope */
   if (pthread_attr_getscope(&attr, &scope) != 0)
      fprintf(stderr, "Unable to get scheduling scope\n");
   else {
      if (scope == PTHREAD_SCOPE_PROCESS)
         printf("PTHREAD_SCOPE_PROCESS");
      else if (scope == PTHREAD_SCOPE_SYSTEM)
         printf("PTHREAD_SCOPE_SYSTEM");
      else
         fprintf(stderr, "Illegal scope value.\n");
   }
```

33

Pthread Scheduling API

```
   /* set the scheduling algorithm to PCS or SCS */
   pthread_attr_setscope(&attr, PTHREAD_SCOPE_SYSTEM);
   /* create the threads */
   for (i = 0; i < NUM_THREADS; i++)
      pthread_create(&tid[i],&attr,runner,NULL);
   /* now join on each thread */
   for (i = 0; i < NUM_THREADS; i++)
      pthread_join(tid[i], NULL);
}
/* Each thread will begin control in this function */
void *runner(void *param)
{
   /* do some work ... */
   pthread_exit(0);
}
```

34

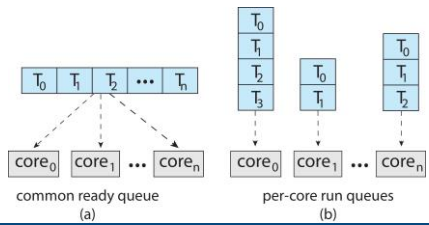
Escalonamento de vários processadores

- Escalonamento de CPU mais complexo quando várias CPUs estão disponíveis
- Multiprocesso pode ser qualquer uma das seguintes arquiteturas:
  - Multicore CPUs
  - Núcleos multithreaded
  - Sistemas NUMA
  - Multiprocessamento heterogêneo

35

Escalonamento de vários processadores

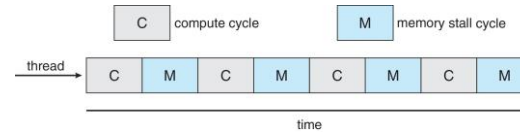
- O multiprocessamento simétrico (Symmetric multiprocessing - SMP) é onde cada processador é auto-escalonado
- Todos os threads podem estar em uma fila pronta comum (a)
- Cada processador pode ter sua própria fila privada de threads (b)



36

Processadores Multicore

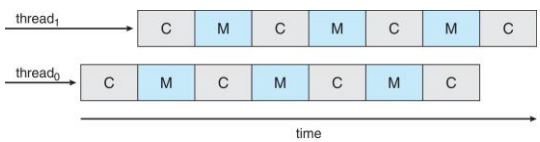
- Tendência recente de colocar vários núcleos de processador no mesmo chip físico
- Mais rápido e consome menos energia
- Vários threads por núcleo também crescem
  - Aproveita a parada de memória para progredir em outro thread enquanto a recuperação de memória acontece



37

Sistema Multicore e Multithread

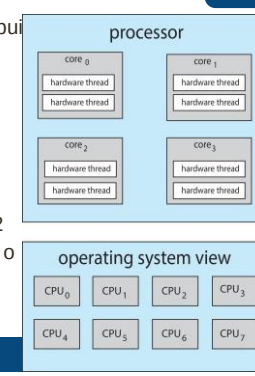
- Cada núcleo tem > 1 threads de hardware
- Se um thread tiver uma parada de memória, mude para outro thread!



38

Sistema Multicore e Multithread

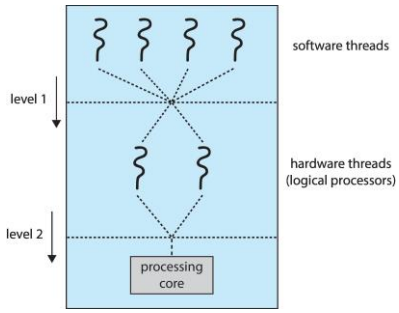
- **Chip-multithreading** (CMT) atribui a cada núcleo vários threads de hardware. (Intel refere-se a isso como hyperthreading – núcleo lógico)
- Em um sistema quad-core com 2 threads de hardware por núcleo, o sistema operacional vê 8 processadores lógicos



39

Multithreaded Multicore System

- Dois níveis de escalonamento:
  1. O sistema operacional que decide qual thread de software executar em uma CPU lógica
  2. Como cada núcleo decide qual thread de hardware executar no núcleo físico



40

Escalonamento de vários processadores – balanceamento de carga

- Se SMP, precisa manter todas as CPUs carregadas para eficiência
  - **Balanceamento de carga** tenta manter a carga de trabalho distribuída uniformemente
  - **Migração por push** – verificações periódicas de tarefas carregam em cada processador e, se encontradas, enviam a tarefa da CPU sobrecarregada para outras CPUs
  - **Migração pull** – processadores ociosos puxam a tarefa de espera do processador ocupado

41

Agendamento de vários processadores – Afinidade do processador

- Quando um thread está sendo executado em um processador, o conteúdo do cache desse processador armazena os acessos à memória por esse thread.
- Referimo-nos a isso como um thread com afinidade com um processador (ou seja, "afinidade com o processador")
- O balanceamento de carga pode afetar a afinidade do processador, pois um thread pode ser movido de um processador para outro para equilibrar cargas, mas esse thread perde o conteúdo do que tinha no cache do processador do qual foi movido.
  - **Afinidade soft** – o sistema operacional tenta manter um thread em execução no mesmo processador, mas sem garantias.
  - **Afinidade hard** – permite que um processo especifique um conjunto de processadores em que ele pode ser executado.

42

Exemplos de sistemas operacionais

- Escalonamento Linux
- Escalonamento Windows

43

Escalonamento do Linux até a versão 2.5

- Antes da versão 2.5 do kernel, executava a variação do algoritmo de escalonamento padrão do UNIX
- Versão 2.5 movida para o tempo de escalonamento de ordem constante O(1)
  - Preventiva, baseada em prioridades com dois intervalos de prioridade: time-sharing e real-time
  - Intervalo em tempo real de 0 a 99 e normal de 100 a 140
  - Mapeie em prioridade global com valores numericamente mais baixos indicando prioridade mais alta
  - Maior prioridade fica maior q e tarefa executável desde que o tempo restante na fatia de tempo (ativa)
  - Se não houver tempo restante (expirado), não poderá ser executado até que todas as outras tarefas usem suas fatias
  - Todas as tarefas executáveis controladas na estrutura de dados runqueue por CPU
    - Duas matrizes de prioridade (ativas, expiradas)
    - Tarefas indexadas por prioridade
    - Quando não estão mais ativos, os arrays são trocados
  - Funcionou bem, mas com tempos de resposta ruins para processos interativos

44

Escalonamento no Linux na Versão 2.6.23 +

- **Completely Fair Scheduler (CFS)**
- **Classes de escalonamento**
  - Cada um tem prioridade específica
  - O Agendador escolhe a tarefa de prioridade mais alta na classe de agendamento mais alta
  - Em vez de quantum baseado em loteamentos de tempo fixos, com base na proporção de tempo da CPU
  - Duas aulas de agendamento incluídas, outras podem ser adicionadas
    - Padrão
    - tempo real

45

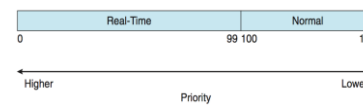
Escalonamento no Linux na Versão 2.6.23 +

- Quantum calculado com base em **nice value** de -20 a +19
  - Valor mais baixo é prioridade mais alta
  - Calcula a latência de destino – intervalo de tempo durante o qual a tarefa deve ser executada pelo menos uma vez
  - A latência de destino pode aumentar se, digamos, o número de tarefas ativas aumentar
- O escalonador CFS mantém o tempo de execução virtual por tarefa na variável vruntime
  - Associado ao fator de decaimento com base na prioridade da tarefa – a prioridade mais baixa é a taxa de decaimento mais alta
  - A prioridade padrão normal produz tempo de execução virtual = tempo de execução real
- Para decidir a próxima tarefa a ser executada, o agendador seleciona a tarefa com o menor tempo de execução virtual

46

Linux Scheduling (Cont.)

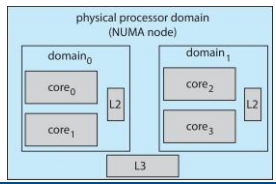
- Real-time scheduling according to POSIX.1b
  - Real-time tasks have static priorities
- Real-time plus normal map into global priority scheme
- Nice value of -20 maps to global priority 100
- Nice value of +19 maps to priority 139



48

Linux Scheduling (Cont.)

- Linux supports load balancing, but is also NUMA-aware.
- **Scheduling domain** is a set of CPU cores that can be balanced against one another.
- Domains are organized by what they share (i.e., cache memory.) Goal is to keep threads from migrating between domains.



49

Windows Scheduling

- Windows uses priority-based preemptive scheduling
- Highest-priority thread runs next
- **Dispatcher** is scheduler
- Thread runs until (1) blocks, (2) uses time slice, (3) preempted by higher-priority thread
- Real-time threads can preempt non-real-time
- 32-level priority scheme
- **Variable class** is 1-15, **real-time class** is 16-31
- Priority 0 is memory-management thread
- Queue for each priority
- If no run-able thread, runs **idle thread**

50

Windows Priority Classes

- Win32 API identifies several priority classes to which a process can belong
  - REALTIME_PRIORITY_CLASS, HIGH_PRIORITY_CLASS, ABOVE_NORMAL_PRIORITY_CLASS,NORMAL_PRIORITY_CLASS, BELOW_NORMAL_PRIORITY_CLASS, IDLE_PRIORITY_CLASS
  - All are variable except REALTIME
- A thread within a given priority class has a relative priority
  - TIME_CRITICAL, HIGHEST, ABOVE_NORMAL, NORMAL, BELOW_NORMAL, LOWEST, IDLE
- Priority class and relative priority combine to give numeric priority
- Base priority is NORMAL within the class
- If quantum expires, priority lowered, but never below base

51

Windows Priority Classes (Cont.)

- If wait occurs, priority boosted depending on what was waited for
- Foreground window given 3x priority boost
- Windows 7 added **user-mode scheduling** (**UMS**)
  - Applications create and manage threads independent of kernel
  - For large number of threads, much more efficient
  - UMS schedulers come from programming language libraries like **Concurrent Runtime** (ConcRT) framework C++

52

Windows Priorities

	real-time	high	above normal	normal	below normal	idle priority
time-critical	31	15	15	15	15	15
highest	26	15	12	10	8	6
above normal	25	14	11	9	7	5
normal	24	13	10	8	6	4
below normal	23	12	9	7	5	3
lowest	22	11	8	6	4	2
idle	16	1	1	1	1	1

53

End

54