MC504 - Sistemas Operacionais

História e Escalonamento de Processos

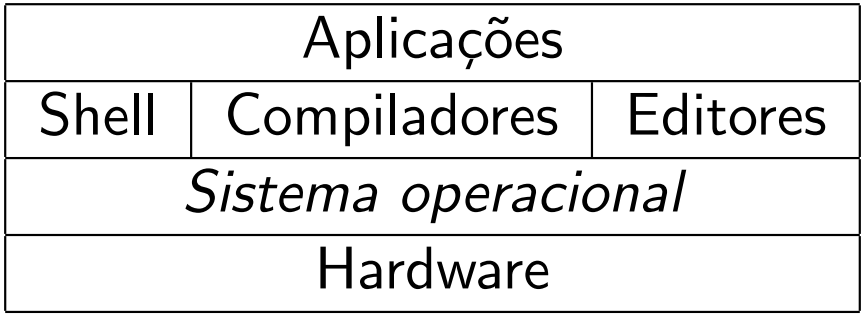
Islene Calciolari Garcia

Instituto de Computação - Unicamp

Primeiro Semestre de 2014

Sumário

Sistema operacional



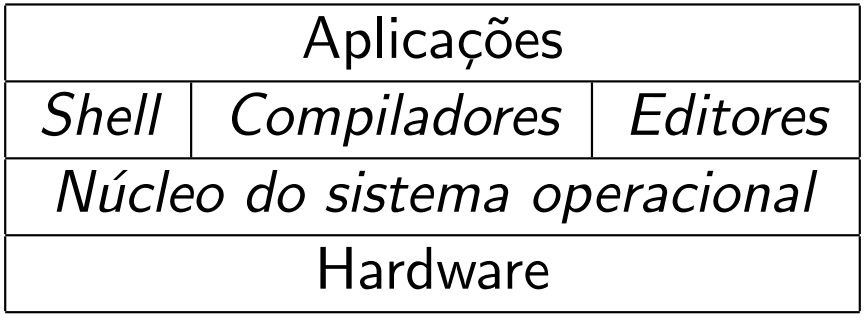
O sistema operacional isola o hardware das camadas superiores em um sistema computacional

Sistema operacional

- *Gerenciador de recursos*
  fornece uma alocação controlada de processadores, memória e dispositivos de entrada/saída
- *Máquina estendida*
  oferece uma máquina virtual mais simples de programar do que o hardware
- *Precisamos de muito mais do que o kernel (exemplo gnu coreutils)*

Sistema operacional completo

Núcleo e software básico

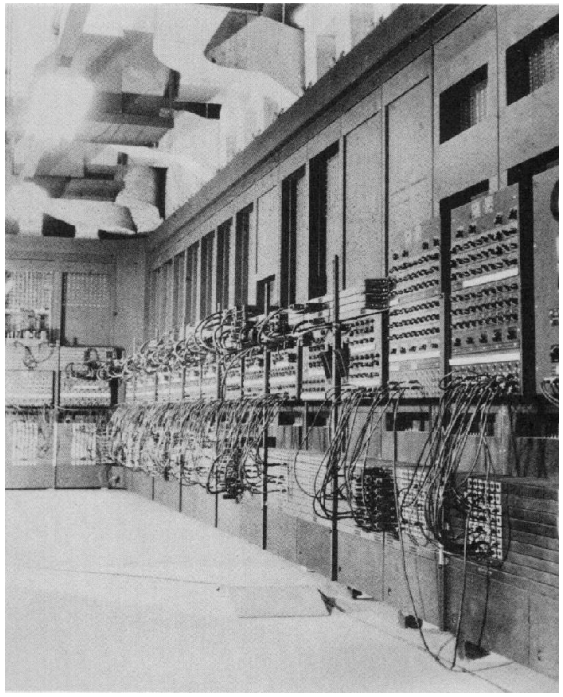


História dos sistemas operacionais

ENIAC

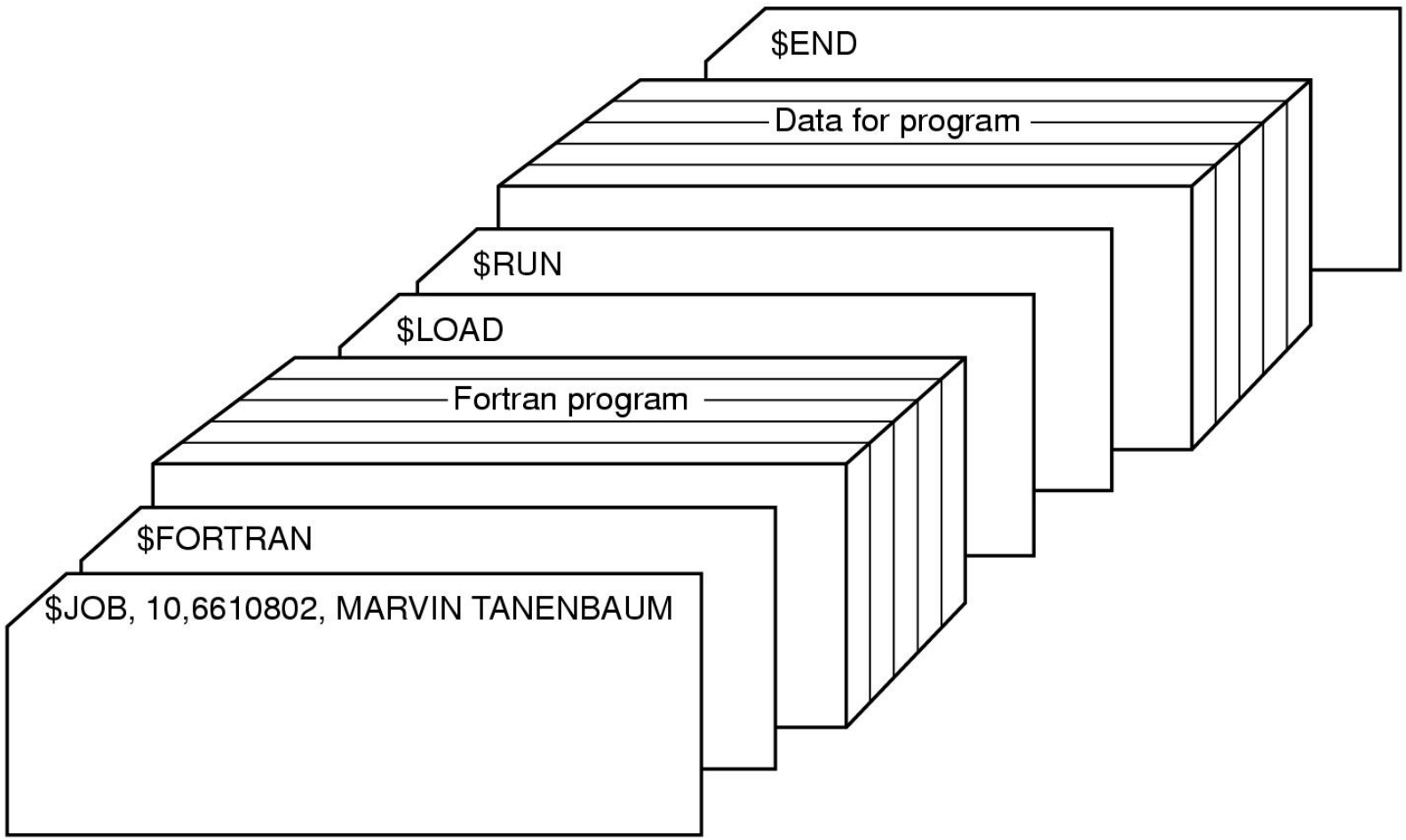
Anos 40

Não tinha SO



História dos sistemas operacionais

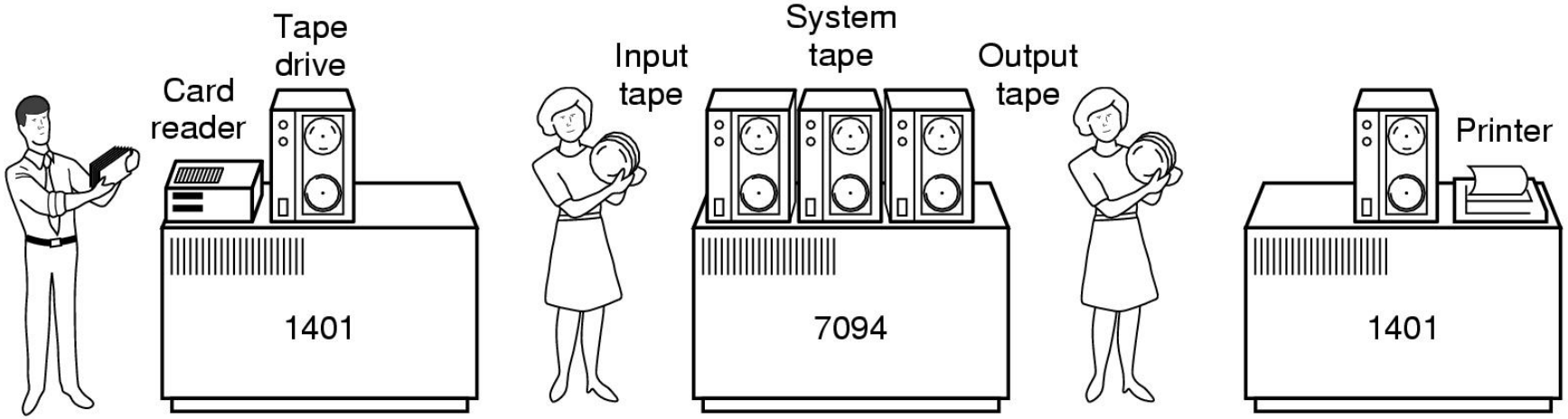
Cartões perfurados



Tanenbaum: Figura 1.3

História dos sistemas operacionais

- Segunda geração 1955–1965
- Transistores e sistemas batch



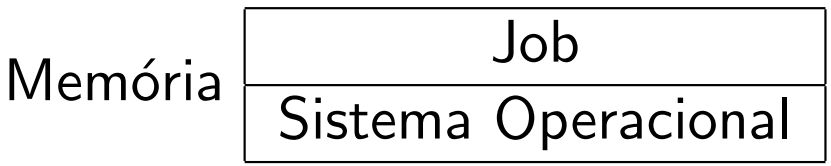
Tanenbaum: Figura 1.2

História dos sistemas operacionais

- Terceira geração 1965–1980
- Circuitos intregados e multiprogramação
- System/360: família de computadores compatíveis

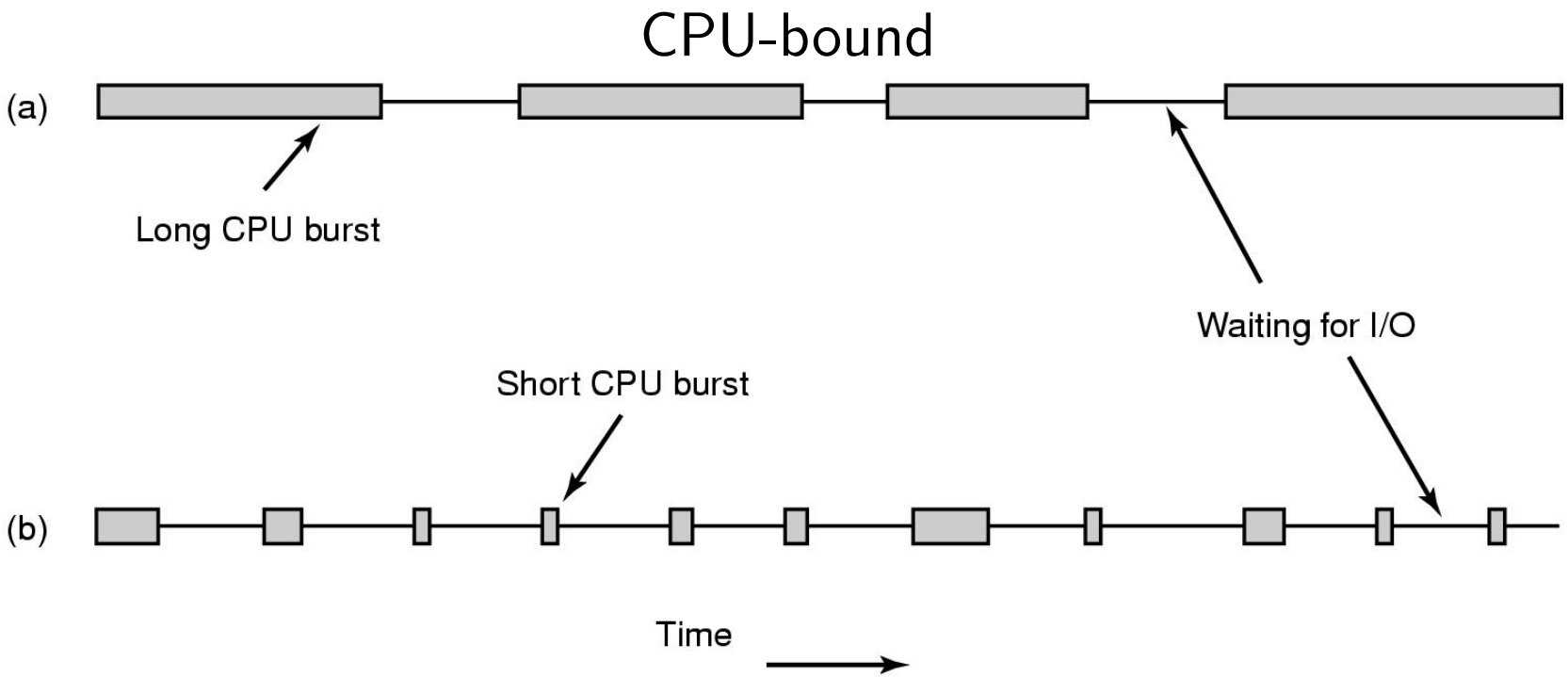
História dos sistemas operacionais

Monoprogramação



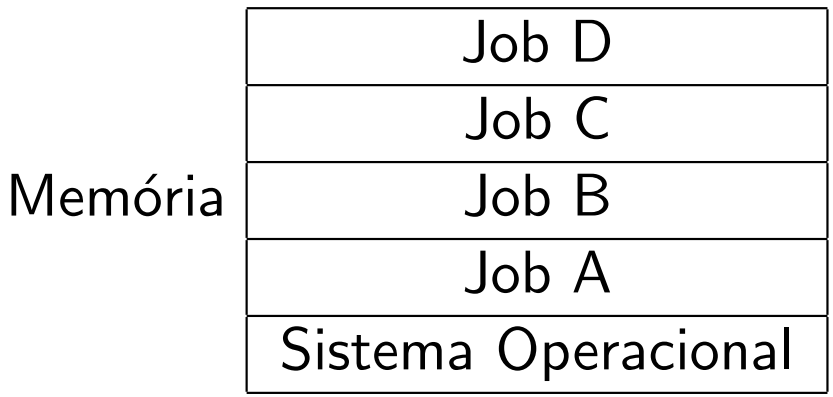
Com apenas um job em memória
a CPU fica ociosa durante operações de E/S

História dos sistemas operacionais
Tipos de jobs



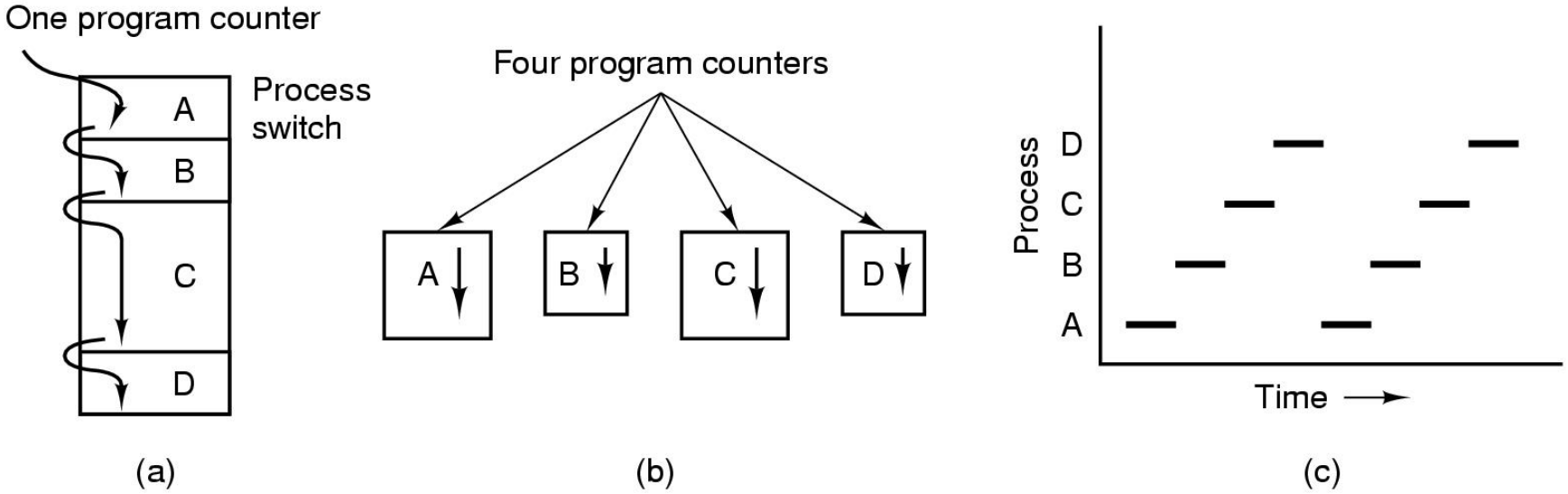
Tanenbaum: Figura 2.37

História dos sistemas operacionais
Multiprogramação



Com vários jobs em memória
a CPU pode ser melhor aproveitada

História dos sistemas operacionais
Multiprogramação



Tanenbaum: Figura 2.1

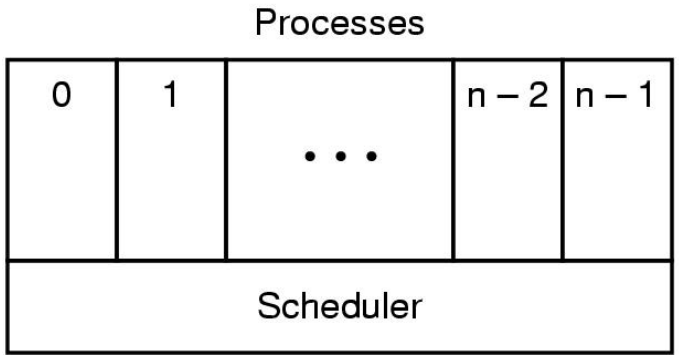
História dos sistemas operacionais
SPOOLing

- Simultaneous Peripheral Operation OnLine
- Leitura dos cartões passou a ser feita em paralelo à execução de outros programas
- Os computadores auxiliares puderam ser aposentados

História dos sistemas operacionais
Tempo-compartilhado

- Vários terminais conectados a um mainframe
- Os usuários exigem resposta rápida

Escalonador



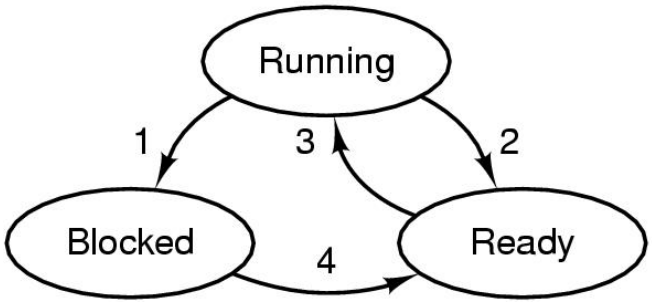
Tanenbaum: Figura 2.3

A função do escalonador é escolher qual deve ser o próximo processo a ser executado.

Quando escalonar

- Quando um processo é criado
- Quando um processo termina
- Quando um processo faz uma operação de I/O
- Interrupção de relógio (sistemas preemptivos)

Estado dos processos



1. Process blocks for input
2. Scheduler picks another process
3. Scheduler picks this process
4. Input becomes available

Tanenbaum: Figura 2.2

Por que algumas arestas estão faltando?

Campos gerenciados por processo

- Gerência de processos: registradores, contador de programa, *program status word*, estado, prioridades, identificador de processos, sinais
- Gerência de memória: apontadores para os segmentos de dados, texto e pilha.
- Gerência de arquivos: diretório raiz e corrente, descritores de arquivos, identificadores de usuário e grupo
- ∗ Quais recursos são compartilhados pelas threads?

Mudança de contexto

1. Hardware stacks program counter, etc.
2. Hardware loads new program counter from interrupt vector.
3. Assembly language procedure saves registers.
4. Assembly language procedure sets up new stack.
5. C interrupt service runs (typically reads and buffers input).
6. Scheduler decides which process is to run next.
7. C procedure returns to the assembly code.
8. Assembly language procedure starts up new current process.

Tanenbaum: Figura 2.5

Objetivos dos algoritmos de escalonamento

- Justiça
  - Cada processo deve receber a sua parte da CPU
- *Policy enforcement*
  - Respeito às políticas estabelecidas
- Equilíbrio
  - Todas as partes do sistema devem estar operando

Escalonamento em sistemas batch

- *Throughput*
  - o número de jobs por hora deve ser maximizado
- *Turnaround time*
  - o tempo entre a submissão e o término de um job deve ser minimizado
- Utilização da CPU
  - A CPU deve ficar ocupada o tempo todo

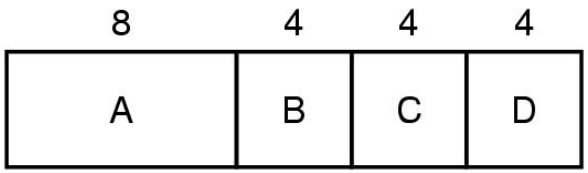
Escalonamento em sistemas batch

First-Come First-Served

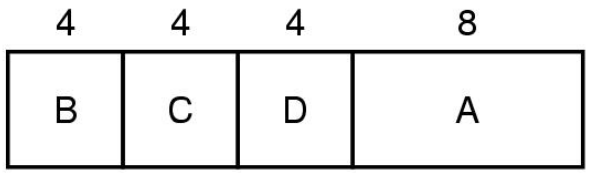
- Processos obtêm a CPU na ordem de requisição
- Não preemptivo
- Aproveitamento ruim da CPU

Escalonamento em sistemas batch

Shortest Job First



(a)



(b)

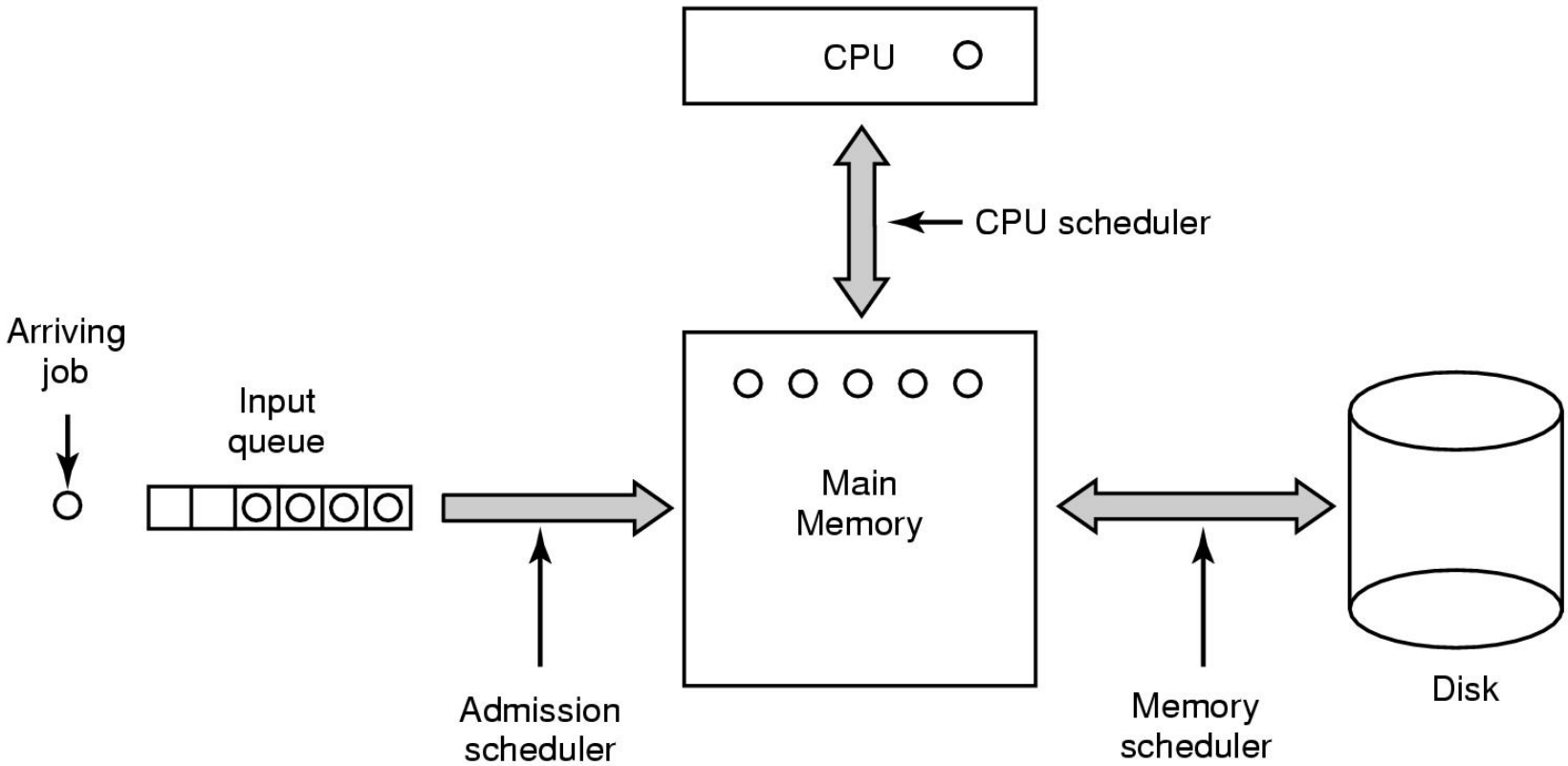
- Vazão (throughput) excelente
- Turnaround time
  - (a) $(8 + 12 + 16 + 20)/4 = 14$
  - (b) $(4 + 8 + 12 + 20)/4 = 11$

Escalonamento em sistemas batch

Shortest Job First

- Todos jobs precisam ser conhecidos previamente
  - Processos no tempo 0: 8 10
  - Processos no tempo 3: 4 4 8 10
- Se jobs curtos chegarem continuamente, os jobs longos nunca serão escalonados
  - Processos no tempo 100: 4 4 4 4 4 4 4 4 4 8 10

Escalonamento em três níveis

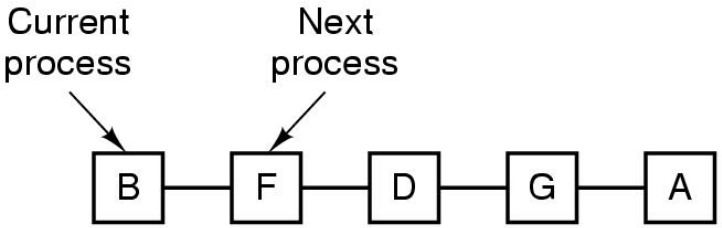


Tanenbaum: Figura 2.40

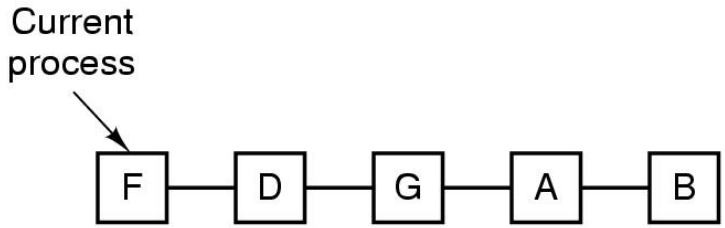
sistemas interativos

- Tempo de resposta
  - O usuário quer respostas rápidas
- Proporcionalidade
  - É necessário respeitar as expectativas de tempo (tarefas fáceis versus tarefas difícies)

Round-Robin



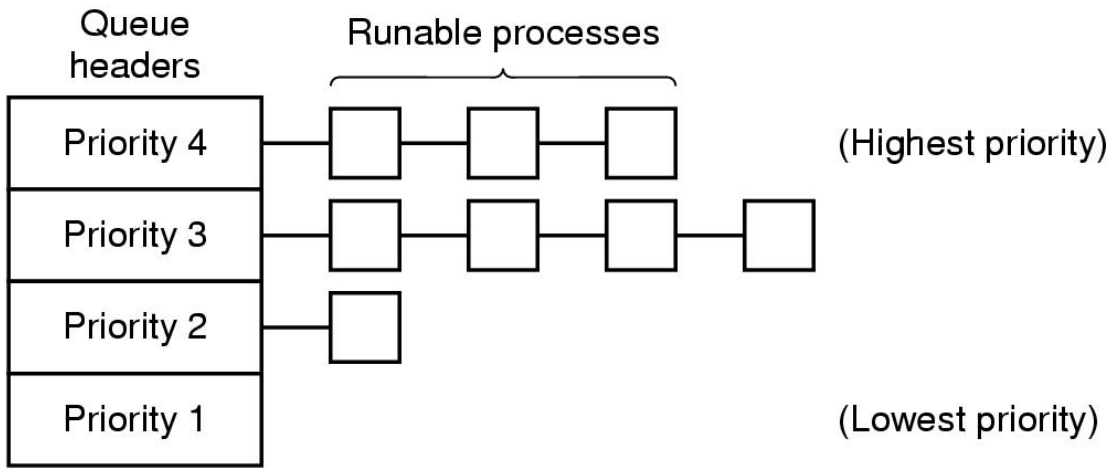
(a)



(b)

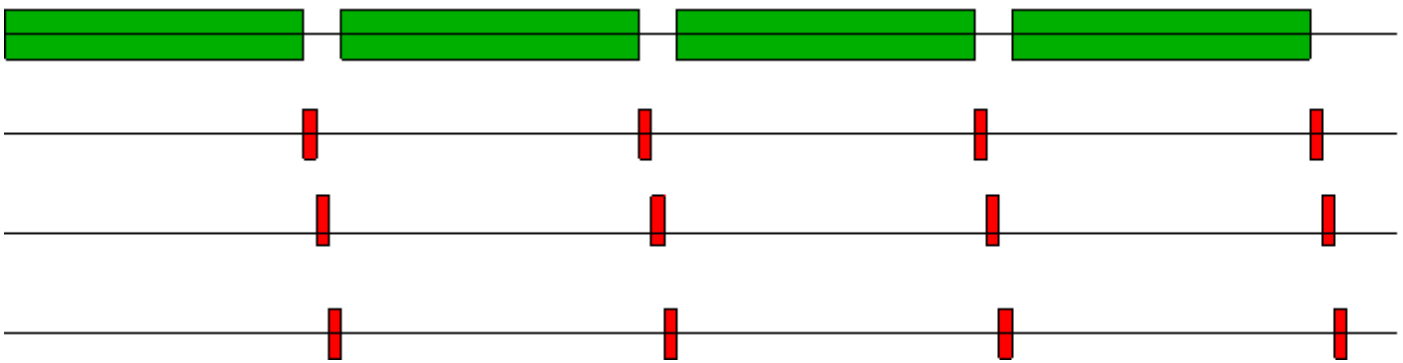
- Preemptivo
- Time quantum
  Como saber o valor ideal?

Prioridades para escalonamento



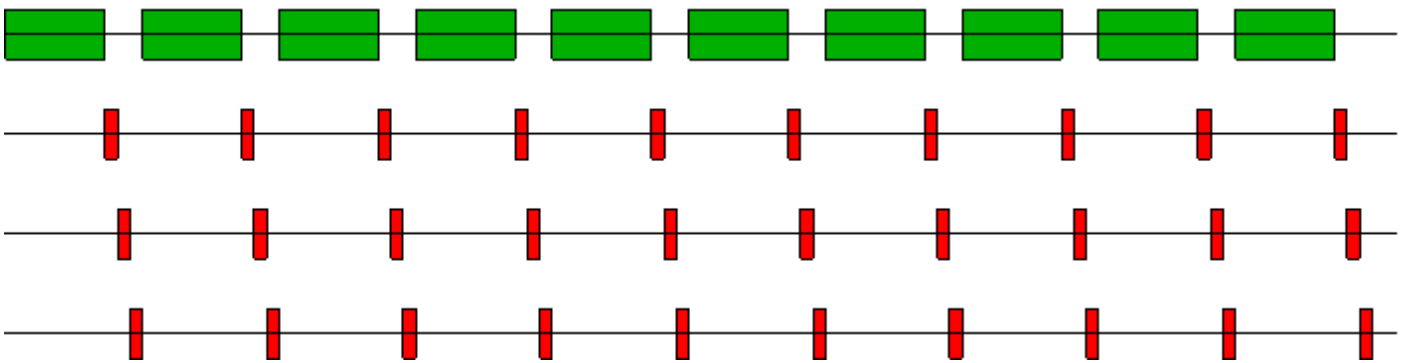
- Prioridades estáticas ou dinâmicas
- Comando `nice`

Aproveitamento da CPU



Processos I/O-bound conseguem poucos ciclos

Aproveitamento da CPU



Processos I/O-bound conseguem mais ciclos

CTSS
Compatible Time Sharing System

- É mais eficiente rodar programas CPU-bound raramente por períodos longos do que frequentemente por períodos curtos
- Como determinar a classe de um processo?

Classe 0 (1 quantum)	→	P1 P2 P5 P7
Classe 1 (2 quanta)	→	P0 P3
Classe 2 (4 quanta)	→	P4
Classe 3 (8 quanta)	→	P6

Shortest Process Next

- Baseado no algoritmo shortest job first
- Comandos $\equiv$ jobs
- Estimativas de tempo (*aging*)
  - T_0
  - $T_0/2 + T_1/2$
  - $T_0/4 + T_1/4 + T_2/2$
  - $T_0/8 + T_1/8 + T_2/4 + T_3/2$

Justiça em sistemas interativos

- Escalonamento garantido
  - O SO faz promessas e deve mantê-las (e.g. 1/n CPU)
- Loteria
  - Baseado na distribuição de tickets
  - Fácil dar pesos distintos aos processos
- *Fair-share*
  - Cada usuário receberá uma parte adequada do poder de processamento da CPU

Escalonamento em sistemas de tempo real

- Respeitar deadlines
- Previsibilidade
- Hard real time e soft real time
- Tratamento dos eventos periódicos $\sum_{i=1}^{m} \frac{C_i}{P_i} \leq 1$

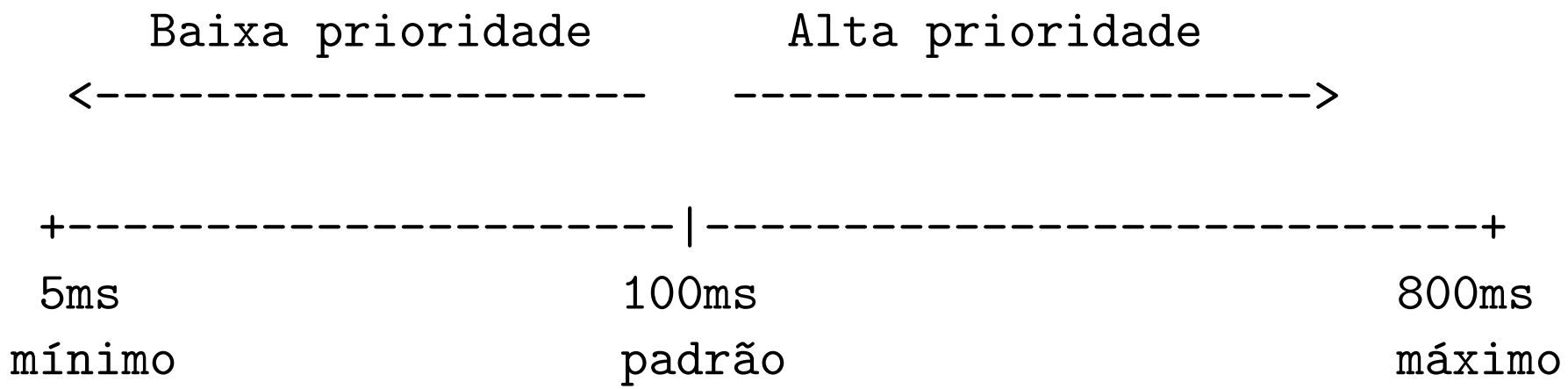
Escalonamento com várias CPUs

- Agrupar tarefas em uma CPU para executá-las
- Migrar tarefas de CPU para balancear filas de execução
- Manter eficiência

Linux implementa dois tipos de prioridade

- Estática (nice)
  - -20..19
  - $>$ valor, $<$ prioridade
  - $<$ valor, $>$ prioridade, $+$ CPU
  - `ps -el`
- Dinâmica
  - 0..99
  - $>$ valor, $>$ prioridade
  - favorece threads interativas

Fatias de tempo no linux



- Nice: 19→ 20 ⇒ timeslice: 10ms → 5ms
- Nice: 0→ 1 ⇒ timeslice: 100ms → 95ms

Até o kernel 2.4

- $O(n)$, não escalava bem
- apenas uma run-queue que era percorrida considerando-se prioridades
- suporte ruim para vários processadores
- enquanto uma CPU escolhia o processo as outras precisavam esperar
- cold cache quando um processo era escalonado para outro processador

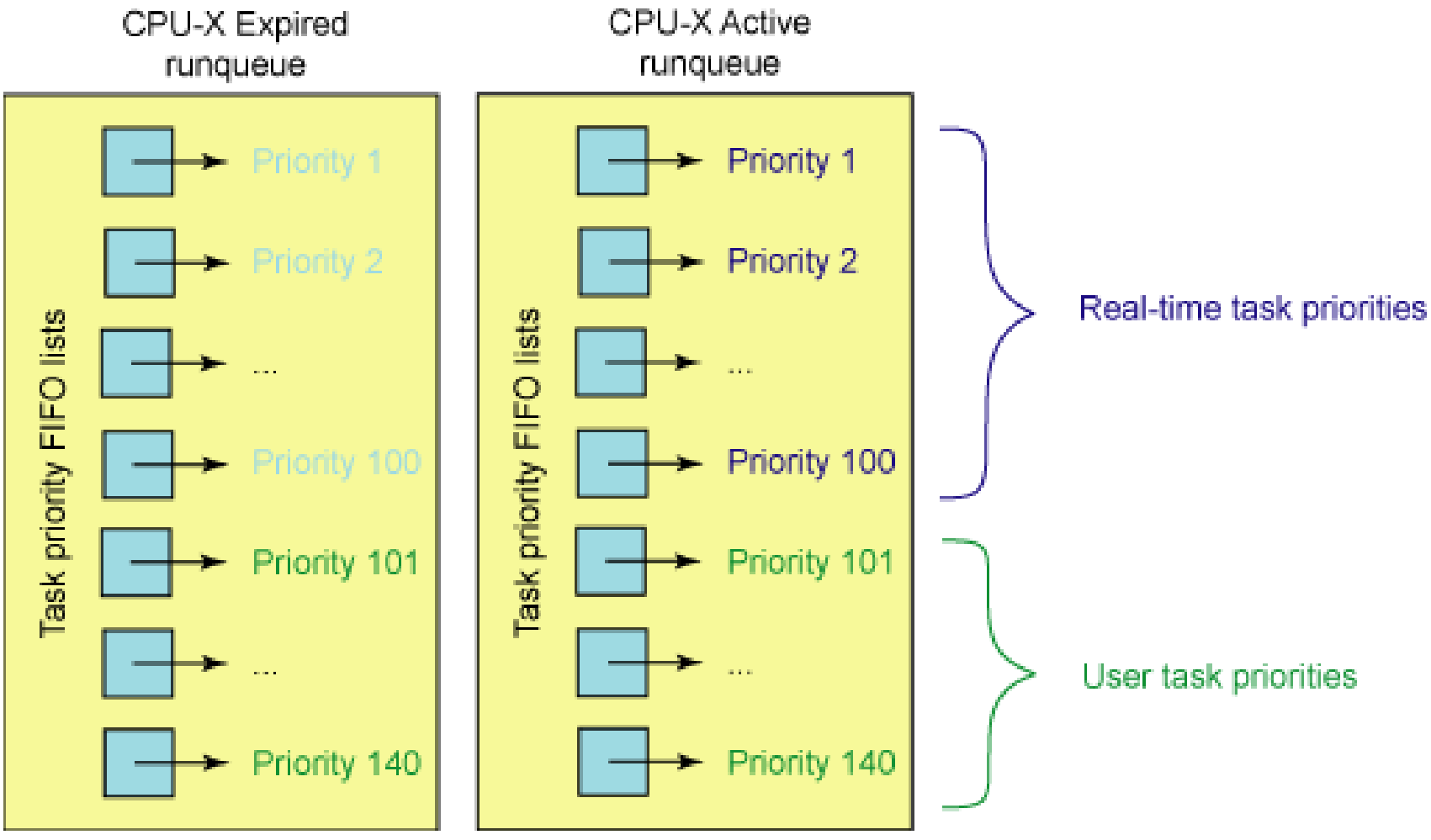
Kernel 2.5 até 2.6.22

- $O(1)$
- 140 listas de prioridades
- uma run-queue por processador
- processos migram apenas para balanceamento das run-queues
- bom desempenho para servidores, mas não tão bom para desktops
- prioridade interativa difícil de ser compreendida
- by Ingo Molnar, mantenedor

Como o $O(1)$ é garantido?

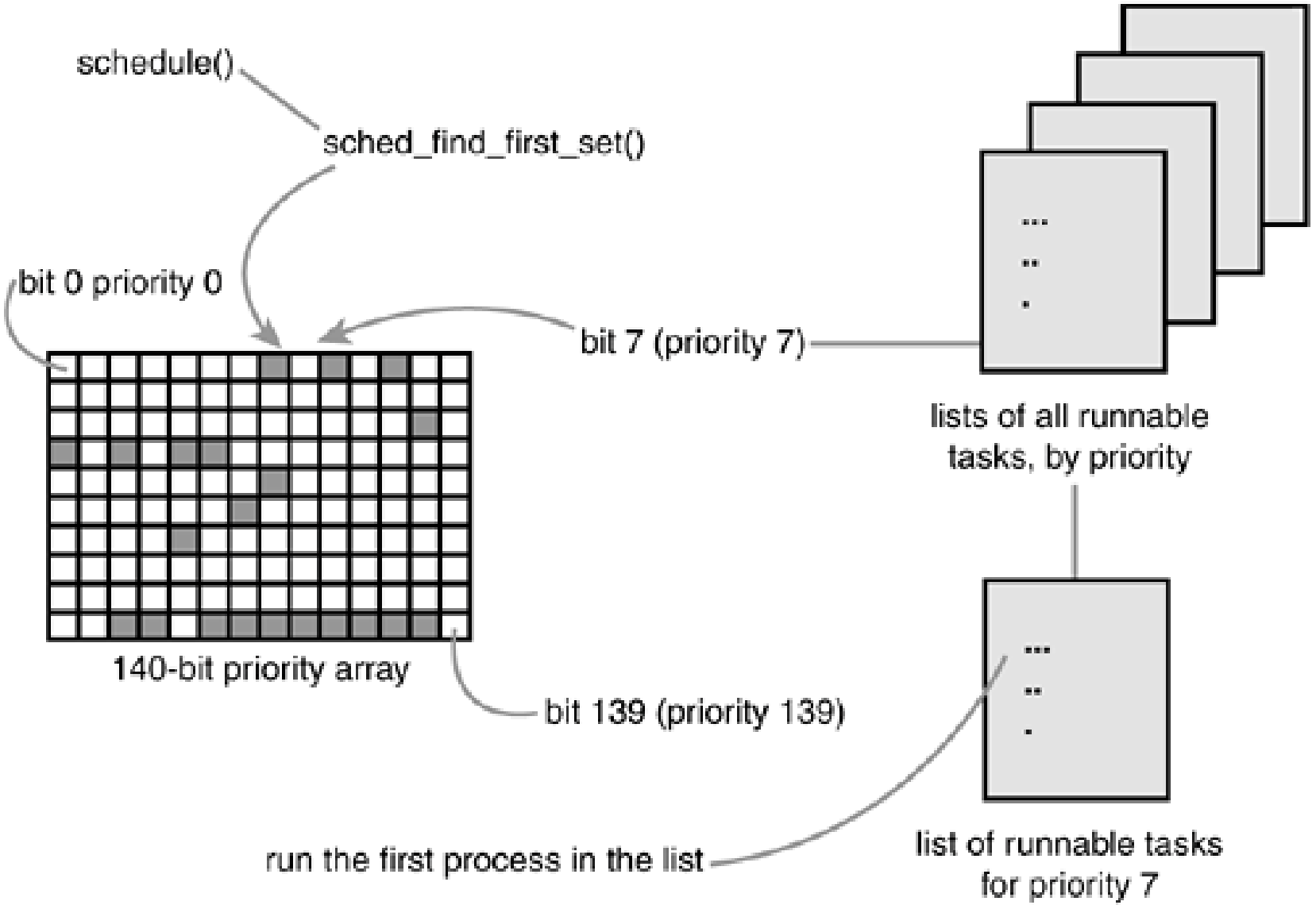
- Número de prioridades é estático = 140
- Bit map: 5 palavras de 32 bits = 160 bits
- busca pelo primeiro bit 1, que indicará a posição na tabela em que temos o processo com maior prioridade
- temos duas tabelas: uma de processos ativos e outra de processos que já consumiram sua fatia de tempo

Tabelas de prioridade



Inside the Linux Scheduler

Tabelas de prioridade



2004: Rotating Staircase Scheduler

- by Con Kolivas, médico anestesista
- Código enxuto (200 versus 498 linhas de código)
- Ideias mais claras, mais fácil de se entender
- Out-of-tree
- Contribuição não foi aceita

Interactivity, what is it?

Con Kolivas, Mon Jul 11 17:29:21 2005

There has been a lot of talk about what makes up a nice feeling desktop under linux. It comes down to two different but intimately related parameters which are not well defined. We often use the terms responsiveness and interactivity in the same sentence, but I'd like to separate the two. As there is no formal definition I prefer to define them as such:

Responsiveness: *The rate at which your workloads can proceed under different load conditions.*

Interactivity: *The scheduling latency and jitter present in tasks where the user would notice a palpable deterioration under different load conditions.*

Responsiveness would allow you to continue using your machine without too much interruption to your work, whereas interactivity would allow you to play audio or video without any dropouts, or drag a gui window across the screen and have it render smoothly across the screen without jerks.

2007: Rotating Staircase Deadline Scheduler (RSDL)

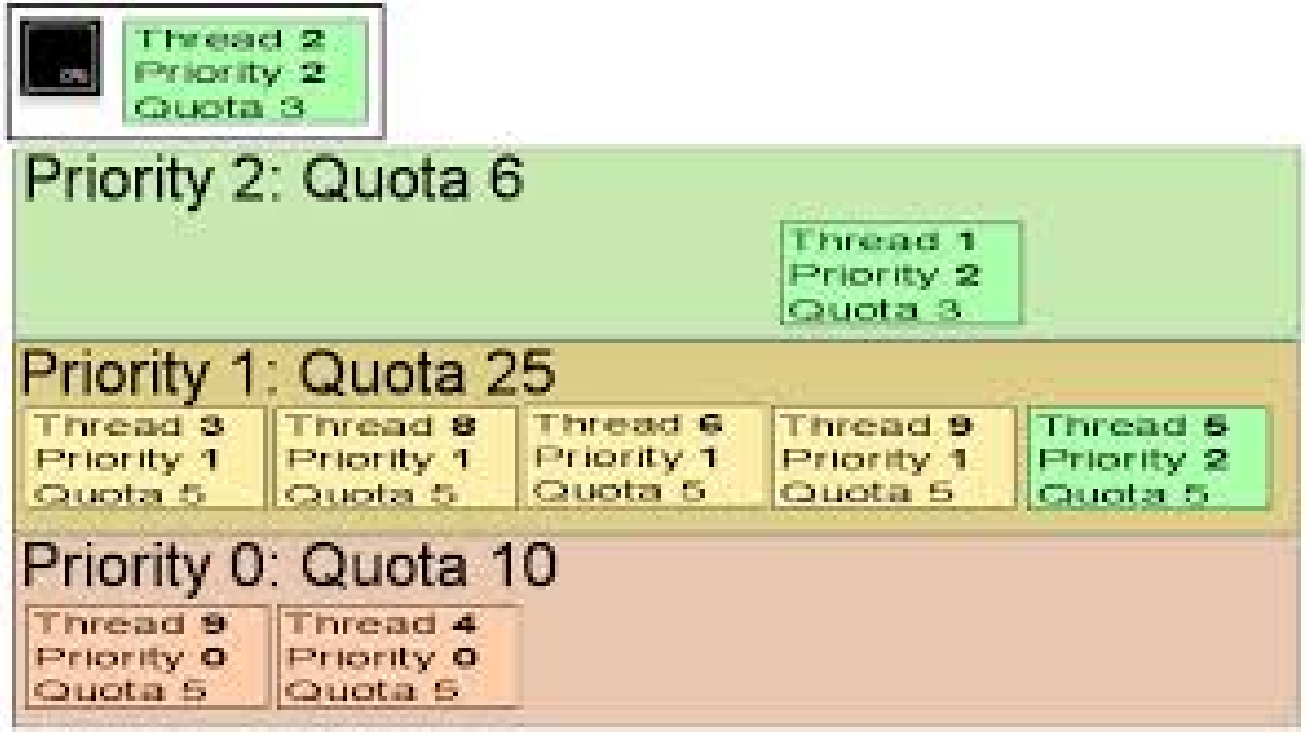
Con Kolivas

A starvation free, strict fairness O(1) scalable design with interactivity as good as the above restrictions can provide. There is no interactivity estimator, no sleep/run measurements and only simple fixed accounting. The design has strict enough a design and accounting that task behaviour can be modelled and maximum scheduling latencies can be predicted by the virtual deadline mechanism that manages runqueues. The prime concern in this design is to maintain fairness at all costs determined by nice level, yet to maintain as good interactivity as can be allowed within the constraints of strict fairness.

2007: Rotating Staircase Deadline Scheduler (RSDL)

- Lista de prioridades ativas
- Processos têm uma cota para rodar em uma determinada prioridade
- Quando esta cota termina, devem passar para um nível com prioridade mais baixa
- A fila de prioridade também tem cota
- Quando a cota termina, todos os processos são rebaixados
- Limite de tempo para um processo de baixa prioridade rodar

Tabelas de prioridade



www.cse421.net

Out-of-tree and maintainership

Do NOT fall into the trap of adding more and more stuff to an out-of-tree project. It just makes it harder and harder to get it merged. There are many examples of this.– Andrew Morton

The fact is, maintainership does _not_ mean ownership. It means that you should be _responsible_ for the code, and you get credit for it, but if problems happen you do NOT "own" it. Not at all. –Linus Torvalds

Histórico

http://www.linux-kongress.org/2010/slides/KN.lk2010_jon_corbet_fail.pdf

- 2007-03-04: First post
- 2007-03-05: Linus amenable to merging
- 2007-03-19: Linus gets irritated
- 2007-04-13: Molnar posts CFS
- 2007-07-10: CFS merged for 2.6.23
- 2007-07-25 Con leaves the kernel community

So, I've had enough. I'm out of here forever. I want to leave before I get so disgruntled that I end up using windows. – Con Kolivas

Completely Fair Scheduler

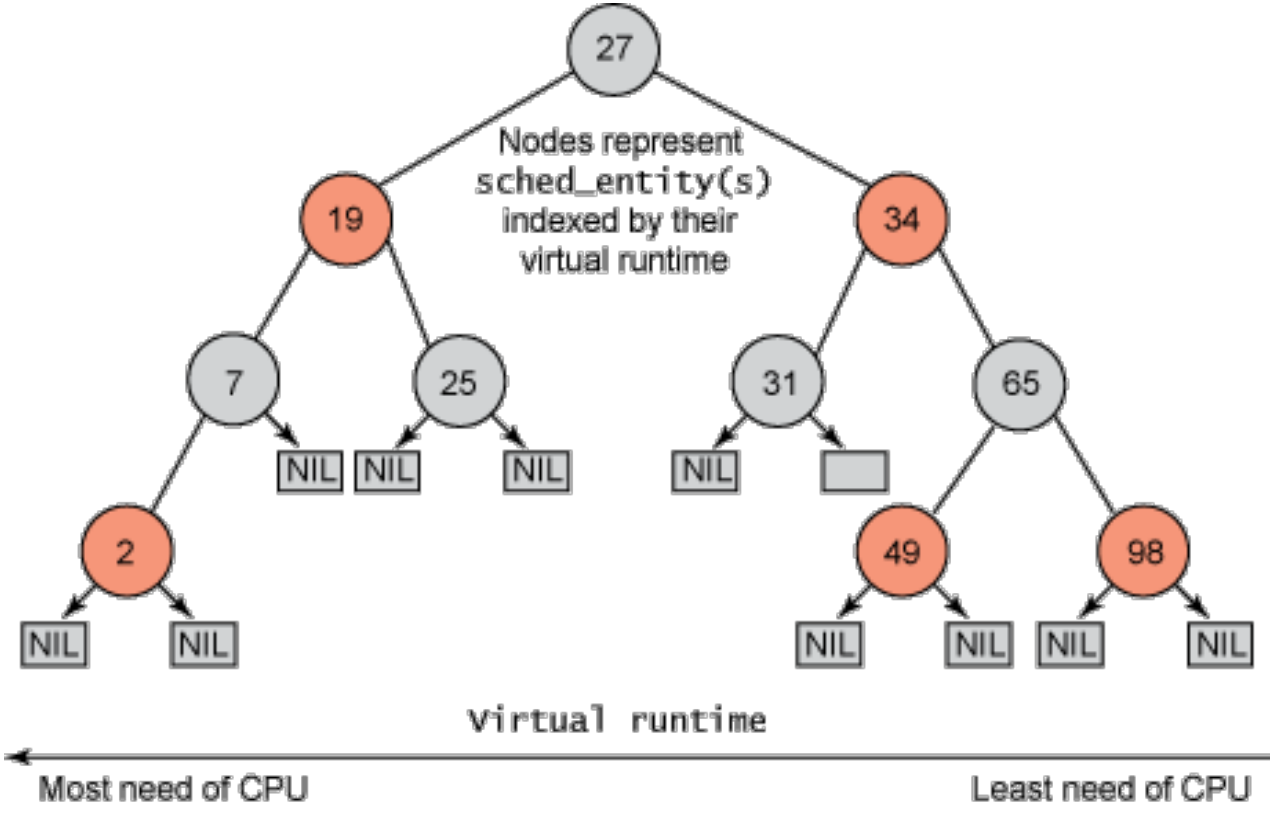
- Em uma CPU multitask perfeita
  - n processos poderiam rodar ao mesmo tempo
  - cada um receberia $1/n$ do poder de processamento
- Em uma CPU real
  - um processo roda e os outros esperam
- Primeira versão foi escrita em 62 horas

CFS: Implementação

- `p->wait_runtime`: tempo que a thread deve rodar para alcançar a justiça na distribuição
- em um ambiente ideal, `p->wait_runtime` seria sempre zero
- thread escolhida tem sempre o maior valor de `p->wait_runtime`
- enquanto uma thread está rodando `p->wait_runtime` é decrementada

  p->wait_runtime = p->wait_runtime - time_running
- a thread sofre preempção quando atinge o menor `p->wait_runtime`

CFS: Árvore Rubro-Negra



Inside the Linux 2.6 Completely Fair Scheduler

CFS

- Virtual time: tempo corre mais devagar do que no mundo real
- Velocidade é inversamente proporcional ao número de processos

Um cartoon inspirou Con Kolivas

Con Kolivas Introduces New BFS Scheduler, Linux Magazine

2009: BFS

*BFS is the Brain F*ck Scheduler. It was designed to be forward looking only, make the most of lower spec machines, and not scale to massive hardware. ie it is a desktop orientated scheduler, with extremely low latencies for excellent interactivity by design rather than "calculated", with rigid fairness, nice priority distribution and extreme scalability within normal load levels. —Con Kolivas*

Referências

- Modern Operating Systems, Andrew Tanenbaum, Second Edition
- Inside the Linux 2.6 Completely Fair Scheduler, Tim Jones
- Inside the Linux Scheduler, Tim Jones
- Linux Kernel Development Second Edition, Robert Love
- 25 Feb 2013: Linux and Linux Scheduling, Geoffrey Challen
- Kernel Development: How things can go wrong (and why you should participate anyway, Jonathan Corbet
- *Process Scheduling*, Cristianno, Robson e Rodrigo, MO806 2008
- *CFS Scheduler*, Fabio e Andre, MO806 2009