

Fundamentos de Sistemas de Operação

Unix Windows NT Netware Mac OS DOS/V/VS Vax/VMS
Linux Solaris HP/UX AIX Mach Chorus

*Escalonamento do CPU:
O escalonador Linux CFS*

Escalonamento proporcional (1)

□ O conceito

- Garantir que cada job recebe uma determinada percentagem do tempo de CPU (ignorando métricas como o turnaround)
- Os recursos (neste caso o CPU) são alocados aos consumidores (neste caso, processos) em função da proporção a que estes têm direito; por exemplo:
 - O processo A tem em sua posse um conjunto de senhas que lhe garantem 75% do CPU, enquanto o B tem apenas 25%
 - A “longo prazo”, quando se observa o sistema, deve ver-se que a fatia de CPU dada a A se aproxima dos 75% e a B dos 25%

Escalonamento proporcional (2)

□ *Escalonamento proporcional por lotaria*

- *Digamos que foram distribuídos 100 bilhetes por A e B de forma que A tem os bilhetes de 0 a 74 e B de 75 a 99.*
- *No fim de cada timeslice, é feito um sorteio (**nºs aleatórios**)*
- *O vencedor é o processo que tem o bilhete com o número sorteado*
- *Exemplo: saíram sucessivamente*

63	85	70	39	76	17	29	41	36	39	10	99	68	83	63	62	43	0	49
A	B	A	A	B	A	A	A	A	A	A	B	A	B	A	B	A	A	A

O que, neste exemplo representa 80% As e 20%Bs mas, se o processo continuar, aproximar-se-à dos 75/25.

Escalonamento proporcional (3)

- *Escalonamento proporcional por stride (menção)*
 - É um outro algoritmo de distribuição proporcional de um recurso, determinístico – i.e., **não é usado um gerador de nºs aleatórios**
- *Estes algoritmos (fair-share e stride)*
 - Estão aqui apenas para mencionar a sua existência, não serão (mais) estudados

O escalonador CFS do Linux (2)

- *CFS = Escalonador Completamente Justo*
 - Abandona a ideia de timeslice fixo; substitui-a por um Contador do tempo virtual de execução: **vruntime**
- *Estratégia:*
 - Há medida que um processo corre, o seu *vruntime* vai sendo acumulado (no caso mais simples, *vruntime* segue o incremento do relógio físico).
 - Quando é altura de “escalonar um novo”, o *CFS* selecciona o processo com menor *vruntime*

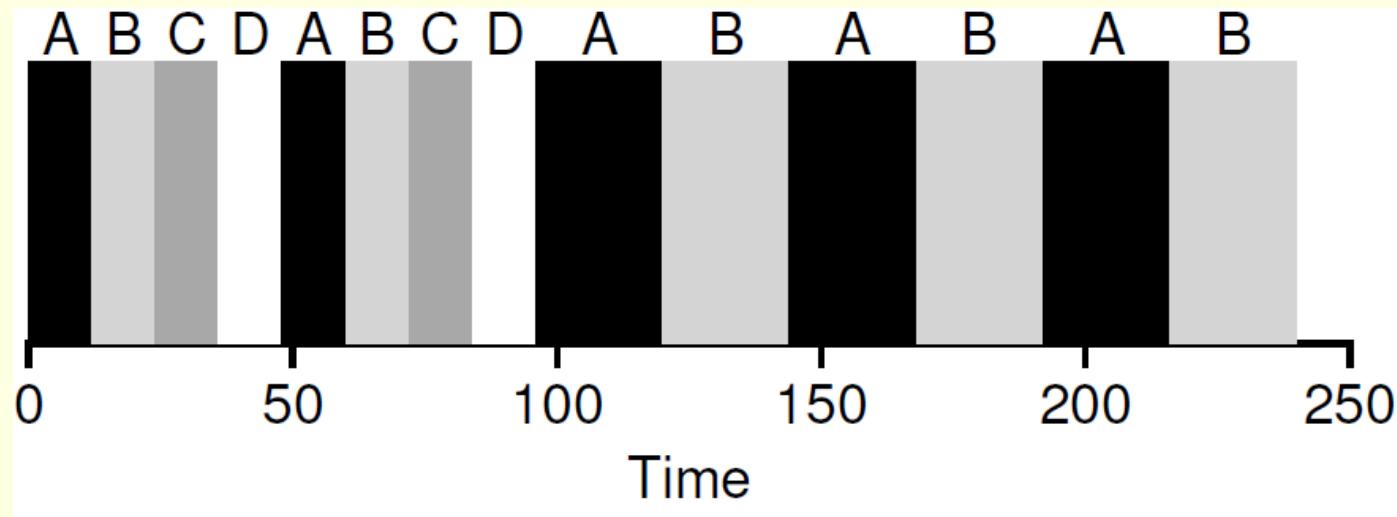
O escalonador CFS do Linux (3)

□ CFS: Quando escalar?

- Se está “sempre” a escalar, perde muito tempo, embora a justeza aumente...
- Supunhamos que num dado instante há N processos ready
 - “faria de tempo” = `sched_latency / N`
 - cada processo vai correr “uma fatia”, quando acabar o CFS procura um outro com `vruntime` menor (do que o do processo que estava a correr) e executa-o
 - assim, os `vruntime` dos N processos aproximam-se

O escalonador CFS do Linux (4)

- CFS: distribuição do CPU pelos “ready”



4 processos a serem escalonados: *fatias de tempo iguais*. 2 terminam: *fatias iguais, mas maiores*

O escalonador CFS do Linux (5)

□ CFS: fatia mínima...

- Se há um elevado n^o de processos ready, as fatias podem ficar tão pequenas que o CFS passa a vida a comutar ...
- O parâmetro **min_granularity** dita que uma fatia nunca é menor que esse valor:
$$timeslice = \max(\min_granularity, \text{ sched_latency}/N)$$
- Os valores citados no OSTEP para estes parâmetros são
 $\text{sched_latency} = 48 \text{ ms}$, $\min_granularity = 6 \text{ ms}$
- Na VM distribuída para as aulas (Ubuntu 16.04.6) meu CPU 2GHz
 $\text{sched_latency} = 12 \text{ ms}$, $\min_granularity = 1.5 \text{ ms}$

O escalonador CFS do Linux (6)

□ *CFS: controle da prioridade de um processo*

- *Baseado no sistema utilizado no Unix: um processo, com uma dada prioridade, pode alterá-la com base no factor nice*
- *Por omissão, um processo ao ser criado tem o parâmetro nice com o valor 0. Esse valor pode ser alterado; a gama permitida é entre -20 e +19, sendo que os valores positivos representam prioridades inferiores*
- *O valor do nice é transformado num outro valor, peso (weight), que é usado para calcular o timeslice*

$$\text{time_slice}_k = \frac{\text{weight}_k}{\sum_{i=0}^{n-1} \text{weight}_i} \cdot \text{sched_latency}$$

O escalonador CFS do Linux (7)

□ *CFS: cálculo do peso*

```
static const int prio_to_weight[40] = {  
    /* -20 */ 88761, 71755, 56483, 46273, 36291,  
    /* -15 */ 29154, 23254, 18705, 14949, 11916,  
    /* -10 */ 9548, 7620, 6100, 4904, 3906,  
    /* -5 */ 3121, 2501, 1991, 1586, 1277,  
    /* 0 */ 1024, 820, 655, 526, 423,  
    /* 5 */ 335, 272, 215, 172, 137,  
    /* 10 */ 110, 87, 70, 56, 45,  
    /* 15 */ 36, 29, 23, 18, 15,  
};
```

Exemplos: weight[0]=1024; weight[1]=820; weight[2]=655; weight[4]=423;
weight[-1]=1277; weight[-5]=3121...

O escalonador CFS do Linux (8)

□ *CFS: usando o nice...*

- Sejam 2 processos, A com *nice*= -5 e B com *nice*= 0. Ou seja, $weight_A=3121$ e $weight_B=1024$
- Aplicando a fórmula,
$$timeslice_A = (3121/1024+3121) / 48\text{ms} \sim 36\text{ms}$$
$$timeslice_B = (1024/1024+3121) / 48\text{ms} \sim 12\text{ms}$$

□ *Concluindo...*

- Quando 2 processos têm o mesmo *nice* (e são os únicos a correr) recebem 50% cada um (as fatias são $48/2 = 24\text{ms}$)
- Quando um deles sobe (ou diminui) o *nice*, a sua fatia cresce (ou diminui) e no o outro a fatia é actualizada de modo inverso, mantendo-se o total constante

O escalonador CFS do Linux (9)

□ *CFS: ajustando o vruntime...*

- A actualização do vruntime é também “mediada” pelo nice, seguindo a fórmula

$$\text{vruntime}_i = \text{vruntime}_i + \frac{\text{weight}_0}{\text{weight}_i} \cdot \text{runtime}_i$$

- No caso dos 2 processos, A com nice= -5 e B com nice= 0 (ou seja, weight_A=3121 e weight_B=1024) o runtime_A real é “desagravado” de um factor de ~1/3 (ou seja, 1024/3121) aquando do cálculo do vruntime.

O escalonador CFS do Linux (10)

- É ainda mais complexo...
 - Classes de escalonamento (mais prioritária à esquerda):
 - STOP, REALTIME, DEADLINE, CFS, IDLE
 - ...
- Na classe CFS...
 - 3 políticas distintas:
 - SCHED_NORMAL (a que apresentamos)
 - SCHED_BATCH
 - SCHED_IDLE

O escalonador CFS do Linux (11)

□ Para finalizar...

- O CFS é um escalonador muito complexo (apesar de altamente optimizado) pois depende de muitos parâmetros que podem ser afinados (*tuned*) para ambientes específicos
- A complexidade a que nos referimos aqui não é relativa ao tempo de execução (“overhead”, mas sim “intelectual”. De facto, o CFS é muitíssimo eficiente, tanto em configurações mono como multi-processador (ou multi-core)
- Infelizmente, como em muitos produtos open-source, a documentação é desactualizada, ou inexistente, ou está espalhada nos mais variados locais (sites, ficheiros de texto, PDFs, etc.) o que é uma forma de evitar(?) a massificação do know-how
- Solução, para quem quer mesmo saber: RTFC ☺ ☺ ☺

Últimas palavras...

□ Mesmo!

- O facto de ainda existirem vários schedulers distintos mostra que não há uma solução única. O livro (OSTEP) refere um artigo que compara o CFS (Linux) com o MLFQ (FreeBSD). Como seria de esperar, numas situações “ganhou” um, noutras o outro...
- Um aspecto importante é sempre saber até que ponto se pode usar 90% ☺ da potencialidade do sistema sem estar a estudar o tuning deste ou daquele mecanismo ou contratar peritos (\$\$\$) [Nota: não esquecer nunca – estudar = tempo = \$\$\$\$]. Se for possível, provavelmente sai muito menos caro comprar outro sistema do que estar a tentar “espremer” os restantes 10%...

Fundamentos de Sistemas de Operação

O SO,

*O Escalonador do CPU,
e os Processos: discussão*