

Escalonamento do CPU

Paulo Sérgio Almeida

Grupo de Sistemas Distribuídos
Departamento de Informática
Universidade do Minho



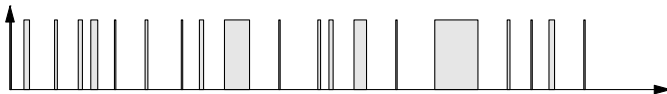
Escalonamento do CPU

- Função: escolher qual o processo pronto que corre em seguida
- Invocado possivelmente aquando:
 - interrupções de relógio (fim de fatia de tempo)
 - interrupções de I/O
 - chamadas ao sistema



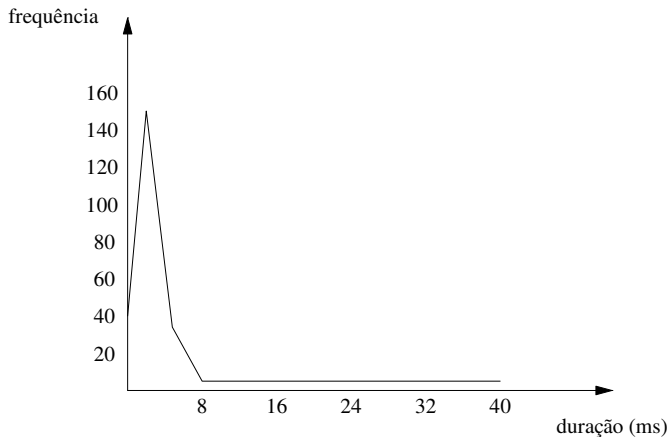
CPU-bursts

- Cada processo a executar alterna entre
 - uso do CPU e
 - espera por I/O
- Para o escalonamento é relevante a duração de cada período de utilização do CPU pelos processos:



Histograma da duração dos CPU-bursts

- Muitos bursts de curta duração
- Poucos bursts de longa duração



Dispatcher

- O Dispatcher dá controlo do processador ao processo escolhido pelo escalonador de CPU; envolve:
 - mudança de contexto
 - comutação para modo utilizador
 - saltar para a localização apropriada e recomeçar o programa
- A **latência de despacho** é o tempo que o dispatcher demora a parar um processo e recomeçar a correr outro



Critérios para escalonamento

- Maximizar **utilização do CPU**: manter o CPU ocupado (em tarefas úteis)
- Maximizar **throughput**: número de processos executados por unidade de tempo
- Minimizar **turnaround time**: tempo que demora a executar um processo
- Minimizar **tempo de espera**: tempo que processo passa na ready queue
- Minimizar **tempo de resposta**: tempo que demora desde a submissão de um pedido até ser produzida a primeira resposta



Critérios para escalonamento: utilizador versus sistema

- Orientados ao utilizador:
 - turnaround time: para sistemas batch
 - tempo de resposta: para sistemas interactivos
- Orientados ao sistema:
 - utilização do CPU
 - throughput
 - tempo de espera



Duas componentes das políticas de escalonamento

- Função de selecção: qual processo da ready queue seleccionar para correr em seguida
- Modo de selecção: quando é que a selecção é exercida
 - Cooperativo (non-preemptive)
 - Com desafecção forçada (preemptive)



Algoritmos de escalonamento

- FCFS: first-come, first-served
- SJF: shortest job first
- SRT shortest remaining time
- Escalonamento por prioridade
- RR: round-robin
- Multilevel queue
- Multilevel feedback queue
- ...



FCFS: first-come, first-served

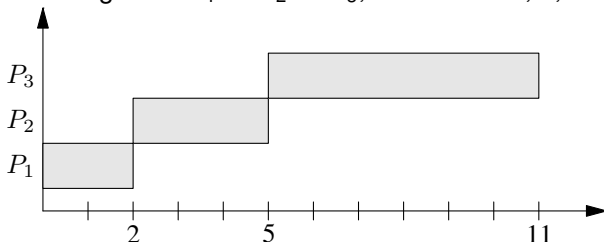
Função de selecção

O processo que está há mais tempo na ready queue

Modo de selecção

Cooperativo: processos correm até se bloquearem voluntariamente

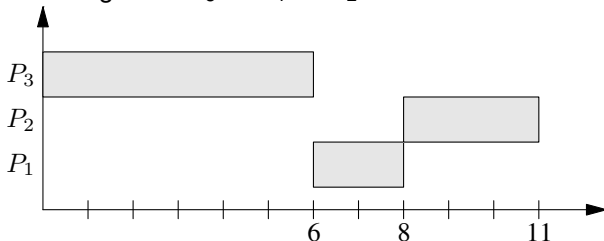
Se ordem de chegada é $P_1 \rightarrow P_2 \rightarrow P_3$, com bursts 2, 3, e 6:



FCFS: first-come, first-served

- Muitos inconvenientes; favorece processos CPU-bound
- Processo que não faz I/O monopoliza processador
- Um processo demorado que chegue primeiro faz todos os outros esperarem muito

Se ordem de chegada é $P_3 \rightarrow P_1 \rightarrow P_2$:



FCFS: first-come, first-served

Calculemos o tempo médio de espera dos processos

- Suponhamos que P_1, P_2, P_3 chegam aproximadamente em $t = 0$
- Pela ordem $P_1 \rightarrow P_2 \rightarrow P_3$:
 - tempos de espera: $t_1 = 0, t_2 = 2, t_3 = 5$
 - tempo médio de espera: $(0 + 2 + 5)/3 = 7/3$
- Pela ordem $P_3 \rightarrow P_1 \rightarrow P_2$:
 - tempos de espera: $t_1 = 6, t_2 = 8, t_3 = 0$
 - tempo médio de espera: $(6 + 8 + 0)/3 = 14/3$

Conclusão

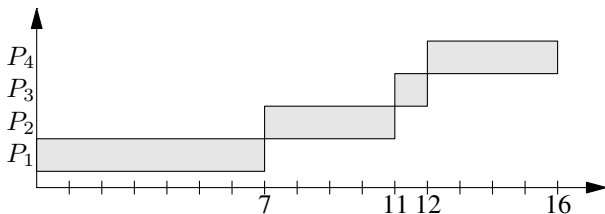
- tempo médio de espera (e turnaround) é maior se processos demorados chegam primeiro
- muito sensível à ordem pela qual chegam os processos



FCFS: first-come, first-served

Outro exemplo, com os seguintes tempos de chegada e bursts:

Processo	Chegada	Burst
1	0	7
2	2	4
3	4	1
4	5	4



$$\text{Tempo médio de espera} = (0 + 5 + 7 + 7)/4 = 4.75$$



SJF: shortest job first (cooperativo)

Função de selecção

O processo com o mais curto burst de CPU esperado

Modo de selecção

Cooperativo: processos correm até se bloquearem voluntariamente

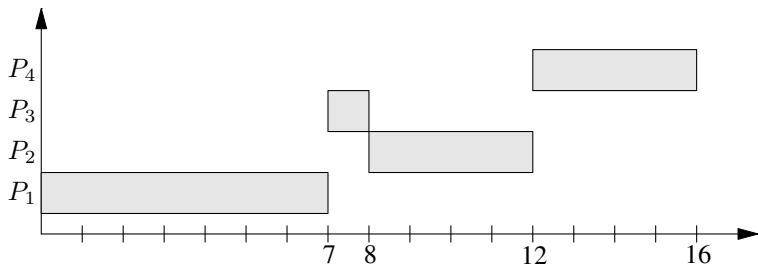
- Processos I/O bound terão tipicamente preferência
- Processos com burst longo podem sofrer **starvation** se continuarem a aparecer processos curtos
- É óptimo: minimiza o tempo médio de espera
- Problema: necessidade de estimar tempo de burst dos processos com base no comportamento passado



SJF: shortest job first (cooperativo)

Suponhamos os mesmos tempos de chegada e bursts:

Processo	Chegada	Burst
1	0	7
2	2	4
3	4	1
4	5	4



$$\text{Tempo médio de espera} = (0 + 6 + 3 + 7)/4 = 4$$



SRT: shortest remaining time (SJF com desafecção)

Função de selecção

O processo com o mais curto burst (restante) de CPU esperado

Modo de selecção

Com desafecção: se chegar um processo com burst menor, o que está a correr é desafectado

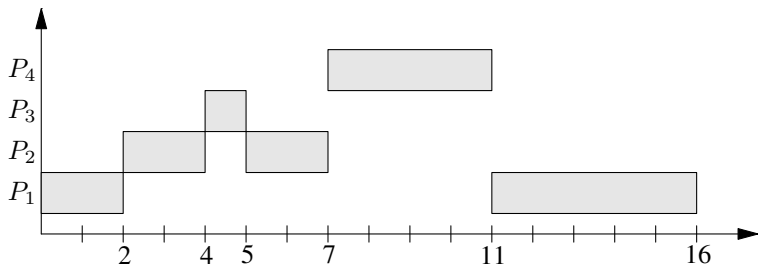
- Evita que processos longos monopolizem só porque chegaram antes
- Melhora turnaround em relação ao SJF: tarefas curtas têm preferencia imediata
- Necessidade de guardar tempo de burst restante



SRT: shortest remaining time (SJF com desafecação)

Suponhamos os mesmos tempos de chegada e bursts:

Processo	Chegada	Burst
1	0	7
2	2	4
3	4	1
4	5	4



$$\text{Tempo médio de espera} = (9 + 1 + 0 + 2)/4 = 3$$



Estimação dos próximos bursts

- SJF e SRT necessitam da duração dos próximos bursts
- Evento futuro; duração exacta desconhecida
- Estimativa do futuro feita com base no passado
- Pode ser feita uma média ponderada dos bursts passados
- Abordagem simplista: média simples de todas as durações passadas
- Podem ser consideradas execuções passadas do mesmo programa



Estimação com ponderação exponencial

- Comportamento recente tem maior probabilidade de ser parecido com o futuro próximo
- Pode ser tido em conta com ponderação exponencial:
 - ao passado recente é atribuída maior importância
 - o factor de ponderação decai exponencialmente com o tempo
- Pode ser calculado de um modo simples com:

$$E_{n+1} = \alpha D_n + (1 - \alpha)E_n$$

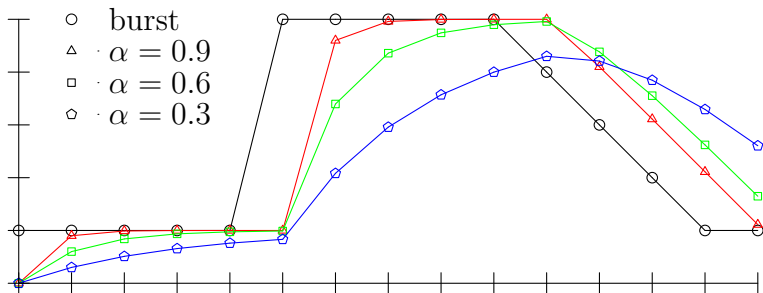
Em que E : estimativa; D : duração medida; $0 \leq \alpha \leq 1$

- Se $\alpha = 1$: estimativa igual ao último burst
- Se α perto de 0: parecido com média simples



Estimação com ponderação exponencial

- $E_{n+1} = \alpha D_n + (1 - \alpha)E_n$
- Para α perto de 1, a estimativa adapta-se mais rapidamente a variações de comportamento



Escalonamento por prioridades

- Número de prioridade associado a cada processo: e.g. números menores – prioridade alta
- Processador atribuído ao processo com a prioridade mais alta
- Variantes: cooperativa ou com desafecção forçada
- SJF: a prioridade é o tempo previsto de burst
- Problema: **starvation** – processos de baixa prioridade continuamente ultrapassados e nunca mais executam
- Solução: **aging** – aumentar a prioridade do processo com o passar do tempo



Round-robin

Função de selecção

O processo que está há mais tempo na ready queue (como FCFS)

Modo de selecção

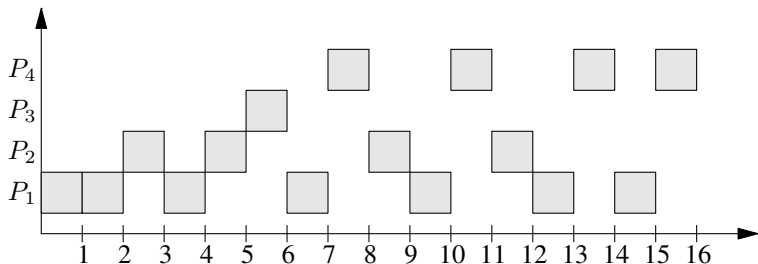
- Com desafecção: é atribuída uma fatia pequena de tempo de processador (**time quantum**); e.g. 10–100 ms
- depois de fatia de tempo esgotada o processo é desafectado e volta para o fim da ready queue
- Com n processos e quantum q :
 - cada processo fica com $1/n$ do tempo de processador
 - executa no máximo q sem interrupção
 - espera no máximo $(n - 1)q$
- Parameterização:
 - q grande \rightarrow FCFS
 - q muito pequeno \rightarrow comutação de contexto significativa



Round-robin

Suponhamos mesmo exemplo e quantum = 1:

Processo	Chegada	Burst
1	0	7
2	2	4
3	4	1
4	5	4



$$\text{Tempo médio de espera} = (8 + 6 + 1 + 7)/4 = 5.5$$



Comparação FCFS, SJF, SRT, RR

- No mesmo exemplo:

Processo	Chegada	Burst
1	0	7
2	2	4
3	4	1
4	5	4

- Os tempos médios de espera na queue são:
 - FCFS: $(0 + 5 + 7 + 7)/4 = 4.75$
 - SJF: $(0 + 6 + 3 + 7)/4 = 4$
 - SRT: $(9 + 1 + 0 + 2)/4 = 3$
 - RR: $(8 + 6 + 1 + 7)/4 = 5.5$
- As variantes de SJF levam a menores tempos de espera
- A variante de SJF com desafecção (SRT) leva ao menor tempo
- Mas SJF, SRT dependem de se conseguir estimativar bursts
- O RR resulta tipicamente em maiores tempos de espera (e turnaround) médios



Vantagens do Round Robin

- Grande vantagem do RR: não necessita de estimação da duração dos bursts
- Escolhendo fatia de tempo pequena, mas onde caibam grande percentagem dos bursts:
 - processos CPU-bound nunca monopolizam processador por muito tempo
 - processos I/O-bound completam burst e iniciam I/O – bom para o tempo de resposta
- RR constitui um modo simples de se ter bons tempos de resposta – critério importante para sistemas interactivos



Virtual Round Robin

- O round robin “normal” ainda favorece num sentido processos cpu-bound:
 - processo I/O-bound tipicamente não usa a fatia toda e bloqueia
 - processo CPU-bound usa fatia toda e volta à fila, à frente dos bloqueados
- Virtual Round Robin resolve o problema:
 - quando operação de I/O termina, o processo bloqueado vai para uma fila auxiliar
 - fila auxiliar tem preferência sobre a fila ready principal
 - processo escalonado da fila auxiliar corre menos tempo: o que lhe restava quando libertou o processador voluntariamente



Fila multi-nível

- Em vez de uma única ready queue, temos várias filas; e.g.:
 - foreground – interactivos
 - background – batch
- Cada fila tem o seu algoritmo de escalonamento; e.g.:
 - foreground – RR
 - background – FCFS
- Escalonamento das filas:
 - Por prioridade fixa:
 - foreground tem prioridade;
 - servir primeiro todos em foreground;
 - problema: starvation
 - Fatia de tempo desigual entre as filas; e.g.:
 - 80% tempo para foreground
 - 20% tempo para background



Fila multi-nível com feedback

- Várias filas; de prioridades decrescentes
- Processo pode mudar de fila ao longo do tempo
- Muito flexível; parameterizavel em:
 - número de filas
 - algoritmos de escalonamento por fila
 - escolha de como promover/despromover processos
 - escolha em que fila um processo começa



Fila multi-nível com feedback; exemplo

- Três filas:
 - Q0 – RR, fatia=8ms
 - Q1 – RR, fatia=16ms
 - Q2 – FCFS
- Prioridades decrescentes:
 - Q1 servida só se Q0 vazia
 - Q2 servida só se Q0 e Q1 vazias
 - se chega processo a fila de maior prioridade, o corrente é desafectado
- Processo novos começam em Q0
- Quando acaba fatia, processo passa para fila abaixo (ou continua em Q2)
- Aging: se processo espera muito em fila é promovido



Fila multi-nível com feedback; exemplo

