FACULDADE DE CIÊNCIAS E TECNOLOGIA
DEPARTAMENTO DE INFORMÁTICA

Fundamentos de Sistemas de Operação

LEI - 2023/2024

Vitor Duarte
Mª. Cecília Gomes

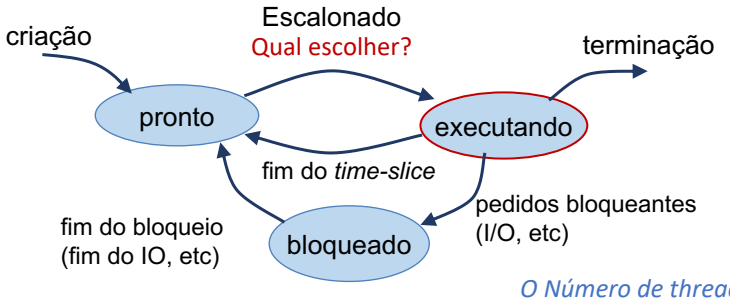
1

Aula 20

- Escalonamento de processos
- OSTEP: cap. 7

2

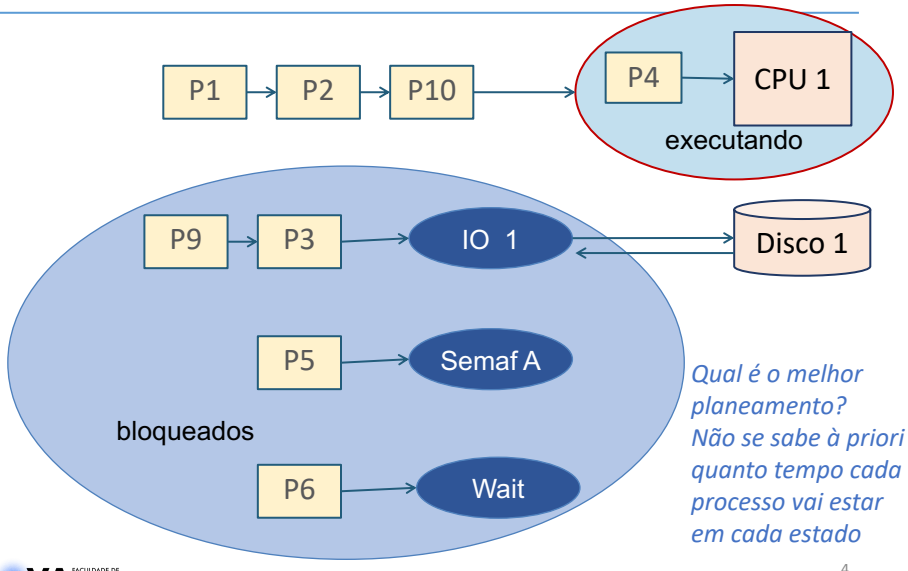
Estados de um processo/thread



O Número de threads realmente em execução simultânea depende do nº de CPU/Cores

3

Como escalonar todos os pedidos?



Qual é o melhor planeamento?
Não se sabe à priori quanto tempo cada processo vai estar em cada estado

4

Vários escalonamentos

- Escalonamento de processos e threads para o/s CPU
- Escalonamento de cada pedido em cada recurso (p.e. IO)

- Escalonamento de curto prazo, médio prazo e de longo prazo
  - que processos executar a cada "instante"
  - que processos suspender/swapout
  - para o dia, semana, mês, ... em batch ou por agendamento
- Escalonamento interno ao processo, pelo programador
  - Exemplo de mau escalonamento pelo programador: abrir ficheiros antes de ter a certeza que o programa vai precisar destes...

- O escalonamento no CPU pode dominar o desempenho de todo o sistema

Grandes objetivos

- Justiça no acesso aos recursos
  - todos têm possibilidade de usar o recurso de que necessitam (p.e. CPU)
- Impor politica/disciplina de atendimento
  - pretende-se obter um plano para determinados objetivos: fazer mais tarefas, atender mais depressa, atender mais tarefas, usar melhor os recursos, etc
- Equilibrio na utilização dos recursos
  - pretende-se que todos os recusos estejam em uso – para efetuar mais trabalho em vez de um recurso ficar desocupado por causa de mau planeamento

Modelos para os sistemas

- Modelo para o estudo:
  - existem processos ou tarefas → define a carga de trabalho (o workload)
    - consideram-se várias carateristicas das tarefas: uso de cpu, io, tempo de execução, etc
  - estuda-se o escalonador (algoritmo) que define o plano de atendimento considerando determinados objetivos
  - definem-se metricas para os objetivos pretendidos, a usar na avaliação/comparação
- Exemplos de sistemas:
  - BATCH – atendimento sequencial de tarefas
  - Processos concorrentes – vários processos partilham o uso de CPU, memória, IO, etc

Métricas de escalonamento

- Taxa de utilização de recurso:
  - De cada periférico e, principalmente, do CPU

$$\frac{\textit{tempo utilização}}{\textit{tempo total}}$$

(o ideal é 100%)

- Débito de trabalhos ou pedidos:

$$\frac{\textit{trabalhos/pedidos concluídos}}{\textit{tempo}}$$

(quanto mais melhor)

Métricas de escalonamento (2)

- Tempo de resposta/conclusão (**turnarround time**)
  - Tempo que decorre desde a submissão do trabalho até à sua conclusão

    Tempo conclusão - Tempo chegada

  (quanto menor melhor para o utilizador)
- Em sistemas interativos é mais relevante o tempo de resposta em termos de cada ação (**response time**) – visto como latência
  - Tempo que decorre para o sistema responder a uma ação do utilizador

    Tempo da resposta - Tempo pedido ação

  - Pode ser definido como o tempo até começar a sua execução

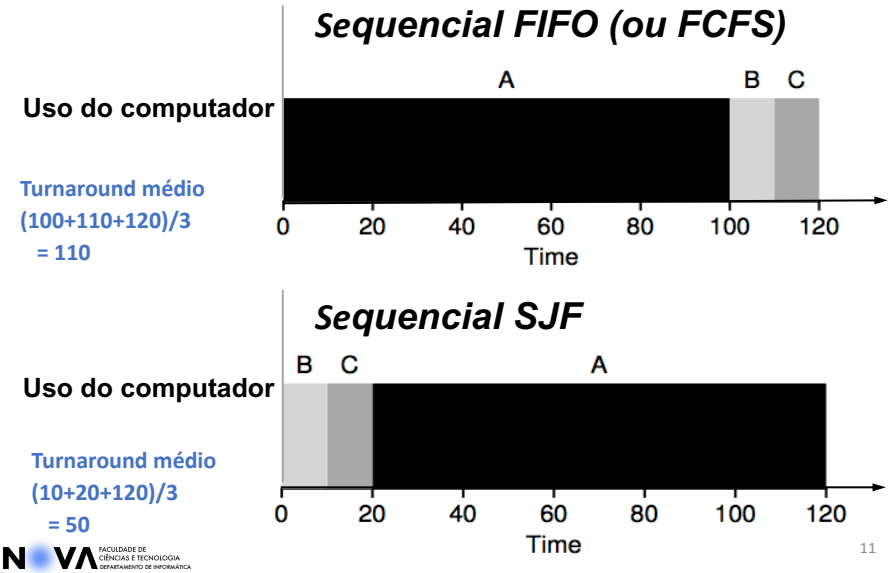
    Tempo início de execução - Tempo pedido ação

  (quanto menor melhor para o utilizador)
- Nota: Melhorar a resposta média pode aumentar injustiças...

Exemplo

- Consideremos a execução de tarefas (p.e. processos)
- Simplificamos considerando
  - temos um conjunto de tarefas pré-definido (workload)
  - não temos em conta se usam CPU, IO, chamadas ao SO, etc
  - atendimento sequencial, como em batch, executando uma tarefa de cada vez
    - só troca de contexto no fim de cada tarefa
  - sabemos o tempo total de execução para cada tarefa
- workload: 3 tarefas no instante 0 que duram 100, 10 e 10 unidades tempo

Exemplo: 2 possíveis ordenações

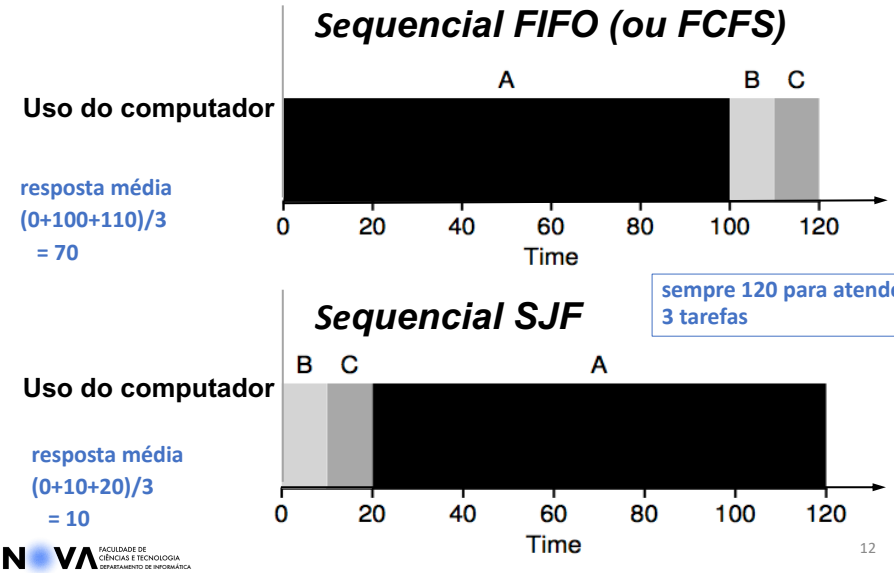


Turnaround médio
(100+110+120)/3
= 110

Turnaround médio
(10+20+120)/3
= 50

11

Exemplo: 2 possíveis ordenações



resposta média
(0+100+110)/3
= 70

sempre 120 para atender as 3 tarefas

resposta média
(0+10+20)/3
= 10

12

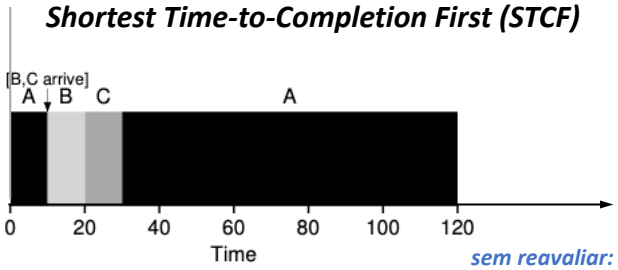
Problemas?

- Todo o computador é atribuído a um programa
  - Má rentabilização dos recursos
  - Maus tempos para atender todos os pedidos, baixos débitos
- Normalmente não é possível conhecer/estimar o futuro
- Se as tarefas vão chegando
  - o plano devia ser refeito
- Não serve para sistemas interativos
  - Nestes queremos bom *response time*

Ajustando nas chegadas

- A cada novo pedido, se pode suspender a tarefa corrente, reavalia o plano
  - prioridade ao que falta menos tempo para terminar

Shortest Time-to-Completion First (STCF)



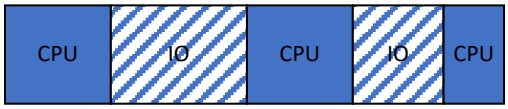
- B e C só no instante 10

sem reavaliar:
Turnaround médio = 110
resposta média = 70

Turnaround médio = (20+30+120)/3 = 57 apr.
resposta média = (0+10+20)/3 = 10

Perfil de um processo (ready vs block)

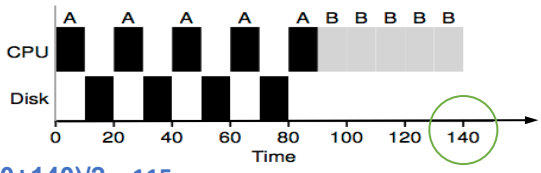
- Um processo executa diferentes ações que definem o seu perfil
- CPU, IO, sincronização (wait, semáforos, etc), ...
  - Comuta entre diferentes estados (running, ready, blocked)
- O CPU e IO podem ser melhor usados → mais trabalho feito



15

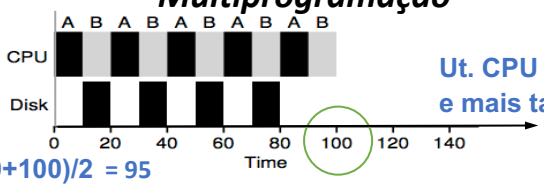
Exemplo: Sobrepondo IO com CPU

Sequencial



Turnaround médio = (90+140)/2 = 115
resposta média = (0+90)/2 = 45

Multiprogramação



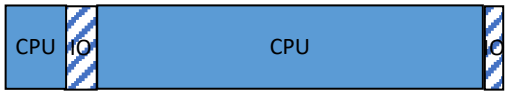
Ut. CPU = 100%
e mais tarefas/t

Turnaround médio =(90+100)/2 = 95
resposta média = (0+10)/2 = 5

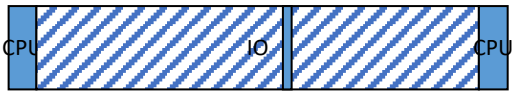
16

Recordando: Perfis dos processos

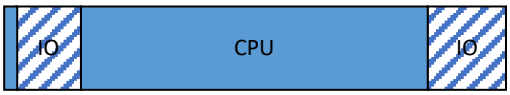
- Dois exemplos de perfis:
  - CPU bound – é limitado pelo tempo de uso do CPU



  - IO bound – é limitado pela espera nas ações de IO ou sincronização



- Os processos podem ter diferentes fases:

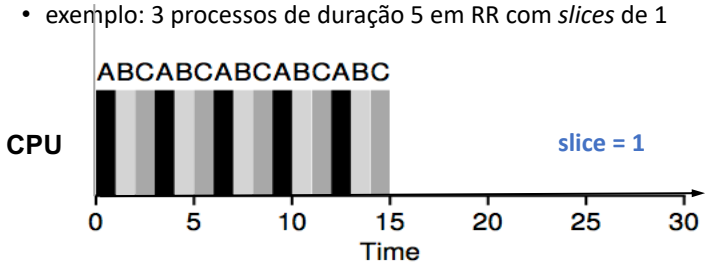


Sem conhecer o tempo de cada tarefa

- Idealmente o SO gere processos com diferentes perfis, em diferentes estados (*running, ready, blocked*)
  - Permite sobrepor ações diferentes de diferentes processos
- Procura que todos os processos progridam, mantendo os recursos sempre ocupados
  - Quando um se bloqueia é uma oportunidade para outro (ready) usar o CPU (passar a Running) → troca de contexto do CPU
  - Evita a monopolização do CPU por um processo: a cada chamada ao SO e usando *time-slicing*
- Que processo escolher para executar em cada oportunidade?
  - Como ter uma melhor utilização do CPU e dos outros recursos (p.e. IO)?
  - Como ter uma boa sobreposição CPU/IO?
  - **Como obter bons tempos de *turnaround* e *response*?**

CPU: Round-Robin com preempção

- Lidar com processos em períodos CPU bound
  - Apreender-lhes o CPU ao fim de um tempo limite - *time-slice* (ou *quantum*)
  - permite atender nova tarefa em cada *time-slice*
- Fazer rodar, à vez, o CPU pelos vários processos READY
  - exemplo: 3 processos de duração 5 em RR com *slices* de 1



Análise

- Troca de contexto (*context swtch*) tem custos
  - suspender um processo + retomar outro processo
  - e ainda voltar a popular as caches, TLB, etc para o novo processo
- Round-Robin tem *overhead* relacionado com o time-slice
  - podemos amortizar reduzindo o número de trocas de contexto
  - se quatum = 10 e troca de context = 1 → 10% overhead
  - se quantum = 100 e troca de contexto =1 → 1% overhead (logo mais tempo para os processos, mas pior resposta)

Vantagens vs Desvantagens

- Sobreposição de IO com CPU
  - Melhora o uso dos recursos e o tempo para terminar um conjunto de processos: **melhor uso de recursos e melhor débito de tarefas**
  - No fim de cada IO o processo passa a ready → pode executar
- RR com time-slice no uso do CPU
  - Permite partilhar o CPU entre todos os processos *ready*, protegendo dos CPU-bound: **melhor *response time***
  - Tem custos com a troca de contextos
- O tempo de conclusão de todas as tarefas tende a ser atrasado
  - Num sistema com vários tipos de processos, os curtos e os IO-bound ficam prejudicados: **pior *turnaround time***
- Não é necessário saber o futuro (as tarefas podem entrar a meio)

Prioridade ao IO-bound?

- Objectivos:
  - Melhorar o *turnarround time* dos processos, especialmente os IO-bound (interatividade é IO)
  - Garantir bom *response time* na interação com utilizadores. Evitar injustiças e impedir *starvation*
- Se um processo é IO-bound deve ter prioridade no uso do CPU?
  - De qualquer modo vai deixar de usar o CPU brevemente... → *Shortest Job First*
- Como saber o perfil de cada processo?
  - Não sabemos o futuro mas podemos usar o passado recente como indicador do futuro
  - Se terminou IO dar-lhe oportunidade de passar à frente do que têm estado a usar o CPU

Multi-Level Queue

- Em vez de uma fila de processos READY, várias filas, cada uma para diferente prioridade
  - Cada processo READY está numa só fila
- O escalonador atribui o CPU ao primeiro da fila com maior prioridade (usa RR se vários)
  - necessita de políticas para fazer subir e descer processos nas prioridades