

# Processos

Paulo Sérgio Almeida

Grupo de Sistemas Distribuídos  
Departamento de Informática  
Universidade do Minho

2005/2006



- Processos
  - Conceito de processo
  - Operações sobre processos
  - Escalonamento de processos



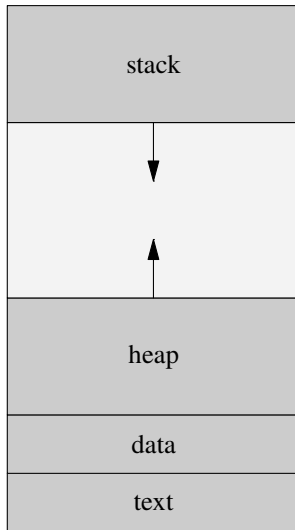
# Conceito de processo

- Sistemas operativos executam:
  - sistemas **batch - jobs**
  - sistemas com **time-sharing** – programas ou tarefas
- Termos **job** (mais antigo) e **process** (moderno) são semelhantes
- Processo: programa em execução
- Processo necessita de contexto de execução:
  - código
  - dados (variáveis globais, heap, stack)
  - estado do processador (registos)
  - ficheiros abertos
  - ...



# Processo em memória

max



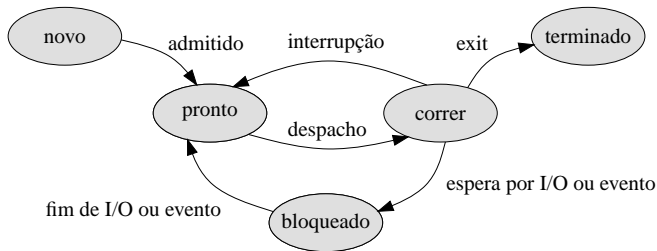
0



# Estados de um processo

Um processo ao executar passa por vários estados:

- **novo**: acaba de ser criado
- **pronto**: à espera de lhe ser atribuído processador
- **correr**: está a ser executado pelo processador
- **bloqueado**: à espera que algum evento aconteça
- **terminado**: terminou execução



# Bloco de controlo de um processo (PCB)

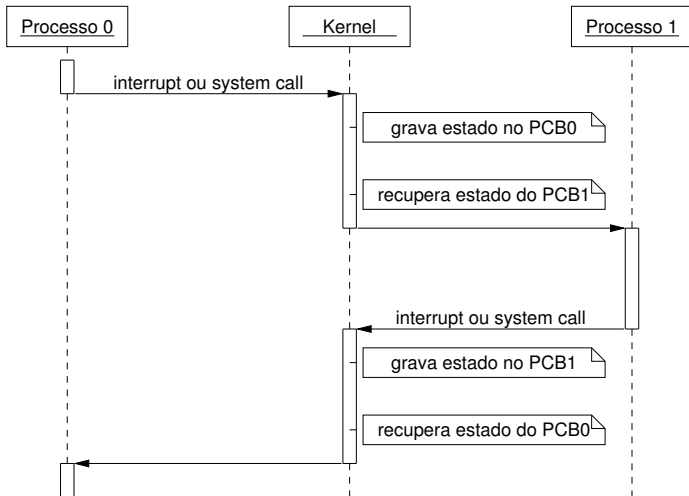
A informação sobre cada processo é guardada no **PCB** - **process control block**:

- estado do processo
- program counter
- registos do processador
- informação de escalonamento
- informação de gestão de memória
- informação contabilística
- estado relativamente a I/O

estado do processo
número do processo
program counter
outros registos
limites da memória
ficheiros abertos
...



# Comutação entre processos



# Criação de processos

- Padrão normal: processo **pai** cria processo **filho**
- Se repetido pode levar a árvore de processos
- Partilha de recursos; pai e filho podem:
  - partilhar todos,
  - alguns ou
  - nenhum recurso
- Relativamente ao espaço de endereçamento:
  - o do filho é uma duplicação do do pai
  - o do filho é um programa diferente desde a criação
- Pai pode:
  - continuar independentemente ou
  - esperar que o filho termine





# Criação de processos em Unix

- Chamada ao sistema **fork** cria filho
- **fork** “retorna duas vezes”, no pai e no filho
- Ambos continuam a executar o mesmo **programa**
- Filho herda duplicado do pai mas continua independentemente: com a sua memória, . . .
- Para substituir o programa a executar usa-se **exec**; muda o programa mas continua o processo corrente



# Terminação de processos

- Quando um processo decide terminar:
  - invoca uma chamada ao sistema: **exit**
  - o sistema operativo liberta os recursos alocados ao processo
  - o pai pode obter informação sobre o filho (**wait**)
- Pai pode decidir terminar o filho:
  - a tarefa atribuída já não é necessária
  - filho excedeu recursos
  - quando o pai termina e não está interessado que o filho continue
  - alguns sistemas não permitem que um filho sobreviva ao pai:  
**cascading termination**



# Escalonamento

- Os processos necessitam recursos
- Há tipicamente mais processos que recursos; tal é útil
- Portanto: os **processos competem por recursos**
- O sistema operativo deve fazer o **escalonamento** dos processos
- Os recursos devem ser atribuídos pela ordem correspondente às **políticas de escalonamento**



# Objectivos

- Conveniência:
  - justiça
  - redução de tempo de resposta
  - previsibilidade
- Eficiência:
  - débito (throughput)
  - maximizar utilização do CPU e outros dispositivos

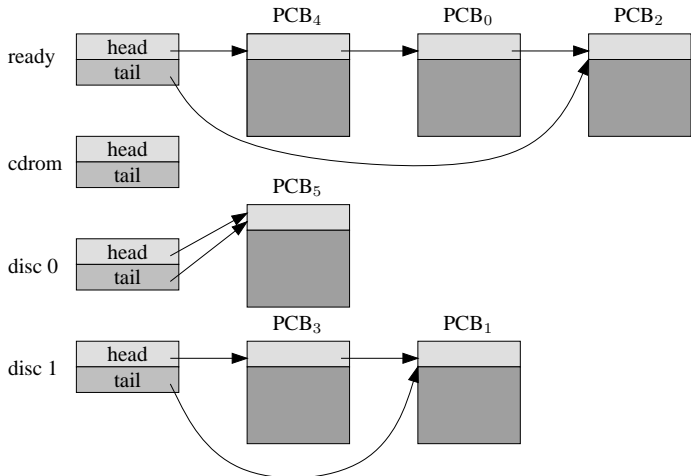


# Filas de espera de escalonamento

- O sistema mantém várias filas de espera para processos:
  - **job queue**: todos os processos do sistema
  - **ready queue**: todos os processos em memória prontos e à espera de executar
  - **device queues**: contém os processos à espera de I/O num dispositivo (uma para cada dispositivo)
- Os processos migram entre as várias filas de espera



# Filas de espera de escalonamento



# Comutação de contexto

É útil distinguir dois tipos de comutação:

- **Comutação de modo** do processador para modo kernel:
  - usado simplesmente para servir uma interrupção
  - sem efectuar comutação de processo
  - o mesmo processo continuará logo em seguida
  - operação relativamente **leve**; é guardado pouco contexto (program counter, alguns registos)
- **Comutação de processo**
  - um outro processo ficará com o processador quando voltar ao modo utilizador
  - é necessário guardar o contexto completo no PCB e carregar contexto do novo processo antes de voltar ao modo utilizador
  - é necessário actualizar estados e listas de processos
  - operação mais **pesada**



# Comutação de processos e o sistema operativo

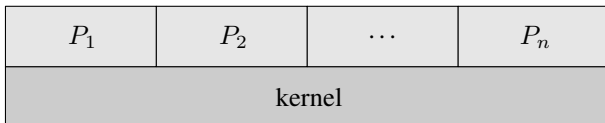
- Diferentes sistemas operativos podem ter diferentes concepções de processos e implementações diferentes:
  - Kernel como não-processo
  - SO a correr dentro de processos utilizador
  - SO baseado em processos





# Kernel como não processo

- O conceito de processo só se aplica a programas em modo utilizador
- O código do SO é uma entidade única e separada que corre em modo protegido
- O código do SO nunca corre dentro de um processo



# SO a correr dentro de processos utilizador

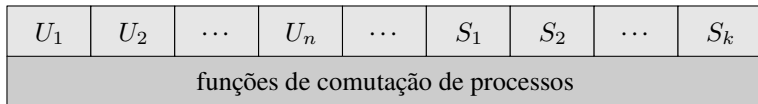
- SO visto como um conjunto de rotinas invocadas pelo utilizador
- Grande parte do código do SO corre dentro do contexto de processos utilizador
- Interrupções levam à comutação para o modo protegido, mas a rotina corre dentro do processo utilizador
- Funções de comutação correm fora dos processos
- Adequado para máquinas pequenas; e.g. PDAs

$P_1$	$P_2$	...	$P_n$
funções SO	funções SO	funções SO	funções SO
funções de comutação de processos			



# SO baseado em processos

- O SO é uma colecção de processos do sistema, fora do espaço de endereçamento dos processos utilizador
- Cada serviço do SO corre num processo separado
- Funções de comutação correm fora dos processos



# Swapping

## Problemas:

- A necessidade de memória de um processo varia, podendo variar bastante
- Mesmo com memória virtual, demasiados processos em memória conduzem a má performance
- Processos podem estar bastante tempo no estado bloqueado, a ocupar memória

## Solução:

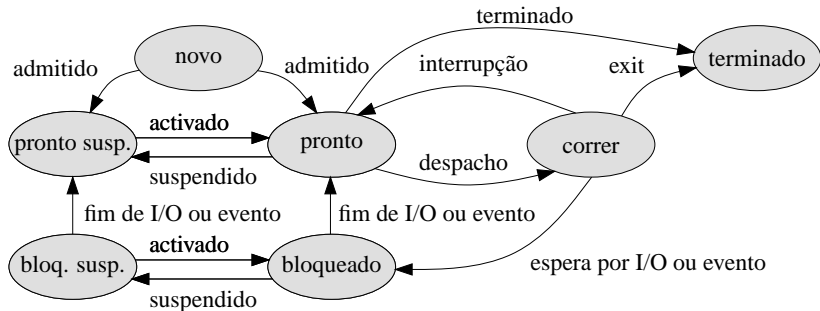
- O sistema operativo pode **suspender** um processo; guardando a sua imagem em disco (**swap out**)
- Pode aplicar-se a processos bloqueados, ou prontos
- Mais tarde os processos podem ser trazidos de volta (**activados, swap in**)



# Estados de um processo – modelo com 7 estados

O swapping introduz mais 2 estados possíveis:

- pronto suspenso
- bloqueado suspenso



# Transições de estados

- **novo** → **pronto**
  - admitido e trazido para memória para a fila ready
- **novo** → **pronto suspenso**
  - admitido como suspenso; uma boa maneira de introduzir novos processos sem mudar o grau de multiprogramação
- **pronto** → **correr**
  - o CPU scheduler escolhe o processo para correr
- **pronto** → **terminado**
  - e.g. quando é morto pelo processo pai
  - é retirado directamente da fila de ready
- **pronto** → **pronto suspenso**
  - quando há demasiados processos em memória e
  - não há um processo bloqueado para fazer swap out



# Transições de estados

- **correr** → **pronto**
  - o processo esgotou a sua fatia de tempo ou
  - o processo é desafectado devido a processo de mais alta prioridade que ficou pronto
- **correr** → **bloqueado**
  - o processo pede um serviço ao SO
  - e.g. inicia I/O, acesso a recurso não disponível
- **correr** → **terminado**
  - processo invoca voluntariamente exit
- **bloqueado** → **pronto**
  - acontece evento pelo qual o processo esperava



# Transições de estados

- **bloqueado** → **bloqueado suspenso**
  - o escalonador decide libertar memória fazendo swap out
  - esta pode ser usada para trazer processo pronto suspenso
  - e manter o grau de multiprogramação
- **bloqueado suspenso** → **pronto suspenso**
  - acontece evento pelo qual o processo suspenso esperava
- **bloqueado suspenso** → **bloqueado**
  - quando o uso de memória diminuiu e
  - não há nenhum outro processo pronto suspenso
- **pronto suspenso** → **pronto**
  - o processo é swapped in
  - quando o uso de memória diminuiu e o escalonador decide aumentar o grau de multiprogramação
  - ou quando é trocado por outro processo swapped out





# Classes de escalonadores

- Escalonador de longo-prazo (**job scheduler**):
  - selecciona os processos que devem vir para a fila **ready**
  - executa pouco frequentemente (segundos ou minutos)
  - pode demorar tempo a tomar decisões
  - controla o **grau de multiprogramação**
- Escalonador de curto-prazo (**cpu scheduler**):
  - selecciona a qual processo deve ser atribuído o processador
  - executa muito frequentemente (milisegundos)
  - tem que tomar decisões rápidas; senão ...
- Escalonador de médio-prazo:
  - usado para controlar o grau de multiprogramação via swapping
  - processos (inteiros) são **swapped** para disco e mais tarde trazidos de volta quando as condições forem apropriadas



# Critérios usados no escalonamento

- Para o escalonamento é importante a divisão:
  - processo **I/O-bound**: gasta mais tempo em I/O do que em computação; muitos períodos curtos de uso do CPU
  - processo **CPU-bound**: gasta mais tempo em computação; poucos períodos longos de uso do CPU
- O escalonador de longo prazo deve conseguir uma boa mistura destes dois tipos de processo
- Outras aspectos relevantes:
  - interactivo ou não (batch, background)
  - urgência de resposta (e.g. tempo real)
  - comportamento recente (utilização de memória, CPU)
  - necessidade de periféricos especiais



# Tipos de escalonamento

- Escalonamento **cooperativo** (non-preemptive)
  - uma vez atribuído a um processo, o CPU nunca lhe é retirado
  - é o processo que cede o CPU voluntariamente
  - risco de o processo monopolizar CPU
  - pouco apropriado para interacção
  - usado em hardware com poucos recursos
- Escalonamento com **desafecção forçada** (preemptive)
  - o cpu é retirado de um processo se acabou fatia de tempo ou apareceu outro de maior prioridade
  - sistema responde melhor interactivamente
  - desvantagem: comutação de contexto tem overhead

