

Processos - Parte 1

Sistemas Operacionais

2017-1

Flavio Figueiredo (<http://flaviovdf.github.io>)

Processos

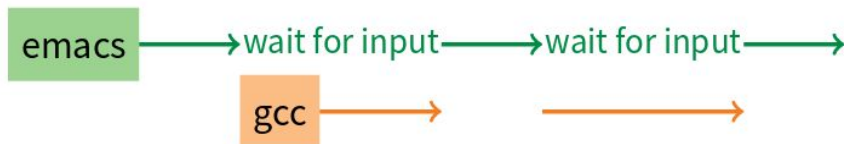
Processo é um programa em execução

- Fluxo de controle de execução de instruções
- Estado
 - Contador de programa
 - Heap, pilha
 - Informações do sistema operacional
- vim, emacs, gcc todos são processos em execução
 - Posso rodar o firefox enquanto compilo código com gcc

Qual a vantagem de permitir múltiplos processos?

Multiprocessamento

- Melhor uso da CPU



- Menor Latência

- Running *A* then *B* requires 100 sec for *B* to complete



- Running *A* and *B* concurrently makes *B* finish faster



Como garantir a menor latência (slide anterior)?

Qual a diferença entre o multiprocessamento aqui visto e o pipelining de arquitetura?

Requisitos de um sistema operacional

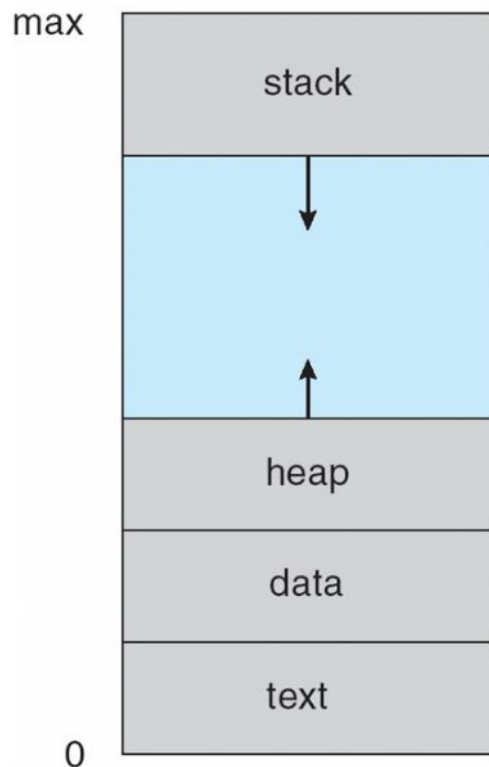
- Multiplexação de recursos
- Isolamento
- Interação, cooperação

Requisitos de um sistema operacional

- Multiplexação de recursos
- Isolamento
- Interação, cooperação
- **Abstração**

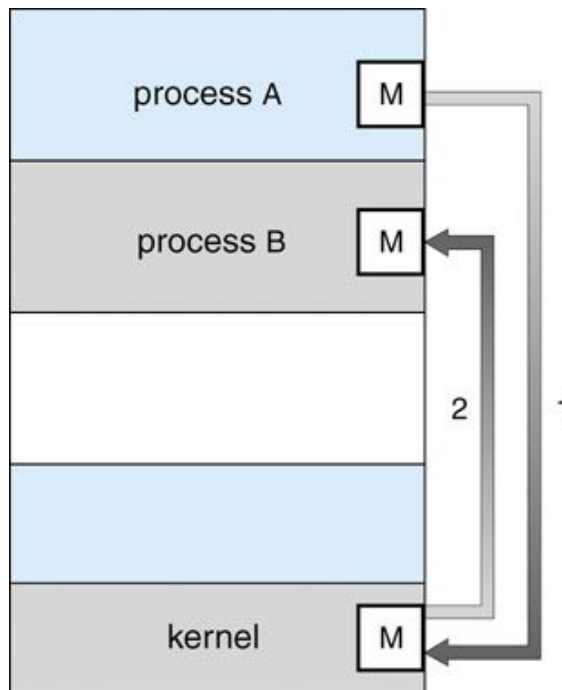
Vista do mundo de um processo

- Cada processo tem uma visão isolada da máquina
 - Endereçamento
 - Arquivos
 - CPU virtual
- `*(char *)0xc000` é diferente entre dois processos
 - Locais diferentes da memória
- Simplifica tudo
 - Abstração

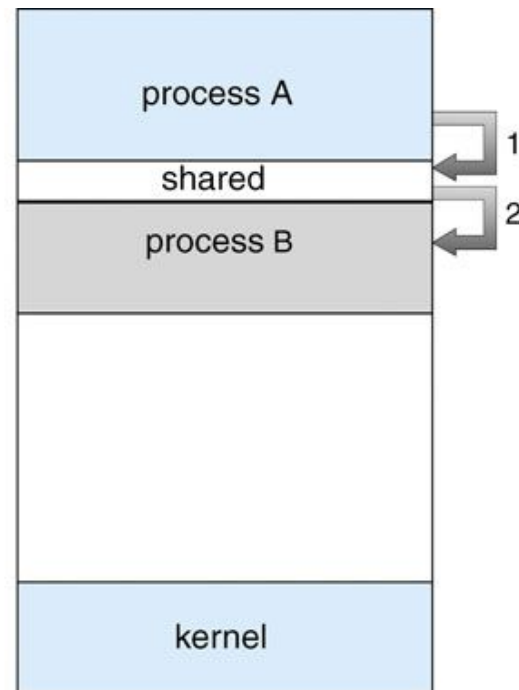


Comunicação entre processos

- Através do Kernel
- Através de Sinais
- Através de memória compartilhada



(a)



(b)

Etapas de um processo

Um processo possui três etapas principais

- Criação (fork, exec)
- Execução
- Término (exit)
- Ver:
 - Paper original do UNIX <http://www.scs.stanford.edu/17wi-cs140/sched/readings/unix.pdf>
 - *proc.c* e *vm.c* do xv6 <https://github.com/mit-pdos/xv6-public/>
 - Funções de fork e exec

Precisamos de fork?

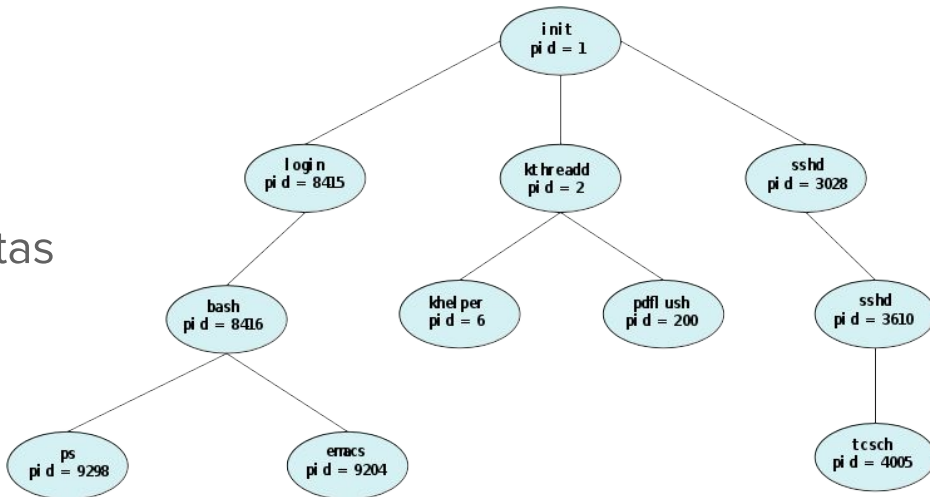
- Provavelmente será seguido de um exec
- Simplicidade no UNIX
 - Fork é bem mais simples, copiamos tudo
 - Também útil para guardar estado
 - Podemos criar um filho usando fork e reduzir a prioridade do mesmo
- Windows permite exec direto

Qual a diferença entre um processo e uma thread?

Cada processo pode criar n-filhos. Qual estrutura de dados representa a hierarquia de processos?

Árvore de Processos

- [Geralmente] Pais podem afetar os filhos
 - Matar os filhos
 - Preemptar
 - Quando eu fecho o shell por exemplo
- Alguns sistemas não permitem florestas
 - No UNIX todo mundo é filho do init
 - Se um processo ficar órfão (pai morrer) o init adota o mesmo
 - Quando desligamos a máquina, todos os processos são finalizados



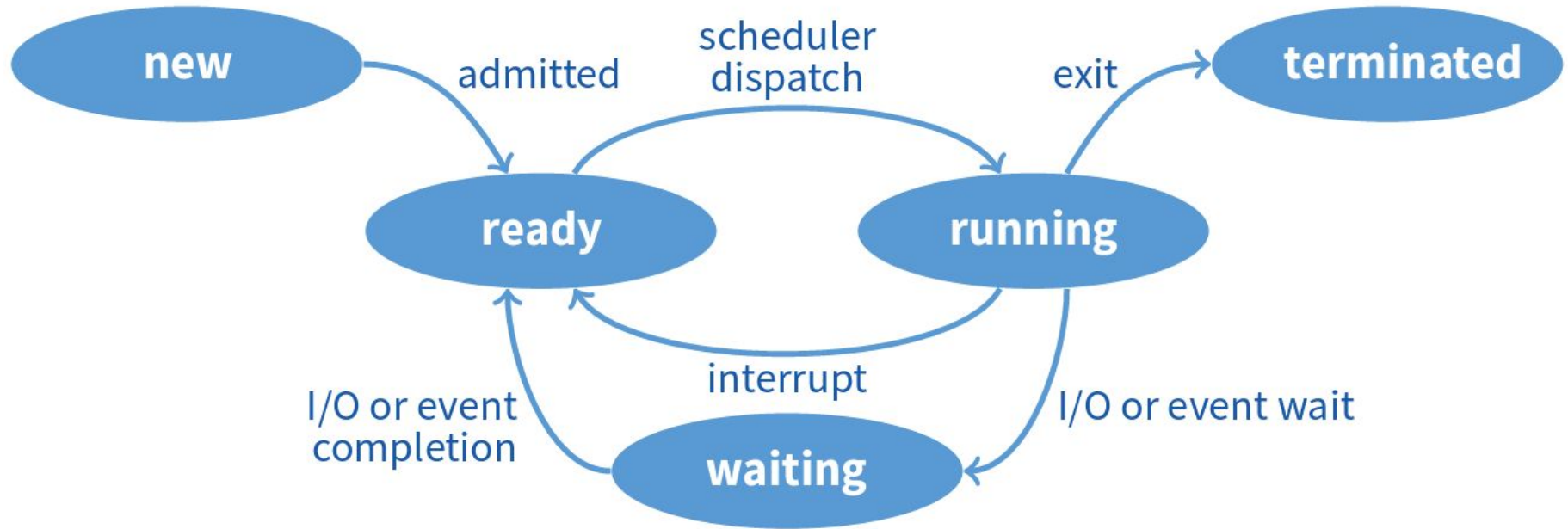
Como um Kernel vê o processo

Armazena informações que o sistema operacional precisa sobre um processo

- Estado
- Recursos
- Contabilização
 - Escalonamento

process state
process number
program counter
registers
memory limits
list of open files
...

Estados de um processo (execução)



Escalonamento

Sistema operacional precisa decidir qual processo deve rodar a cada instante

- Maximizar utilização dos recursos
- Reduzir tempo de resposta
- Reduzir tempo para terminar um processo
- Satisfazer *deadlines*

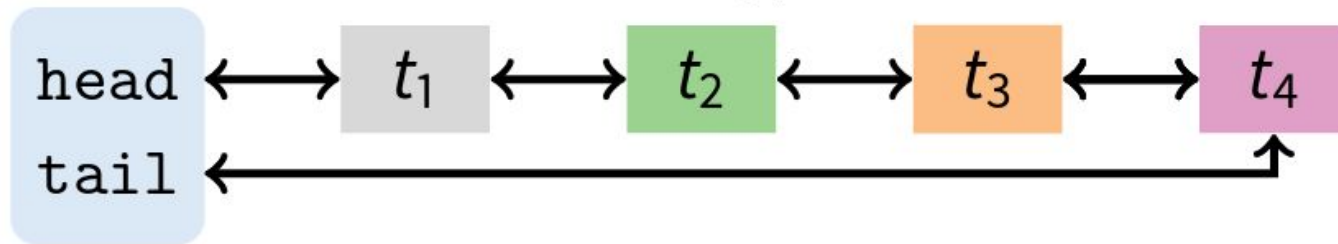
Escalonamento

- [Problema] Como escolher o processo que vou executar?

Escalonamento

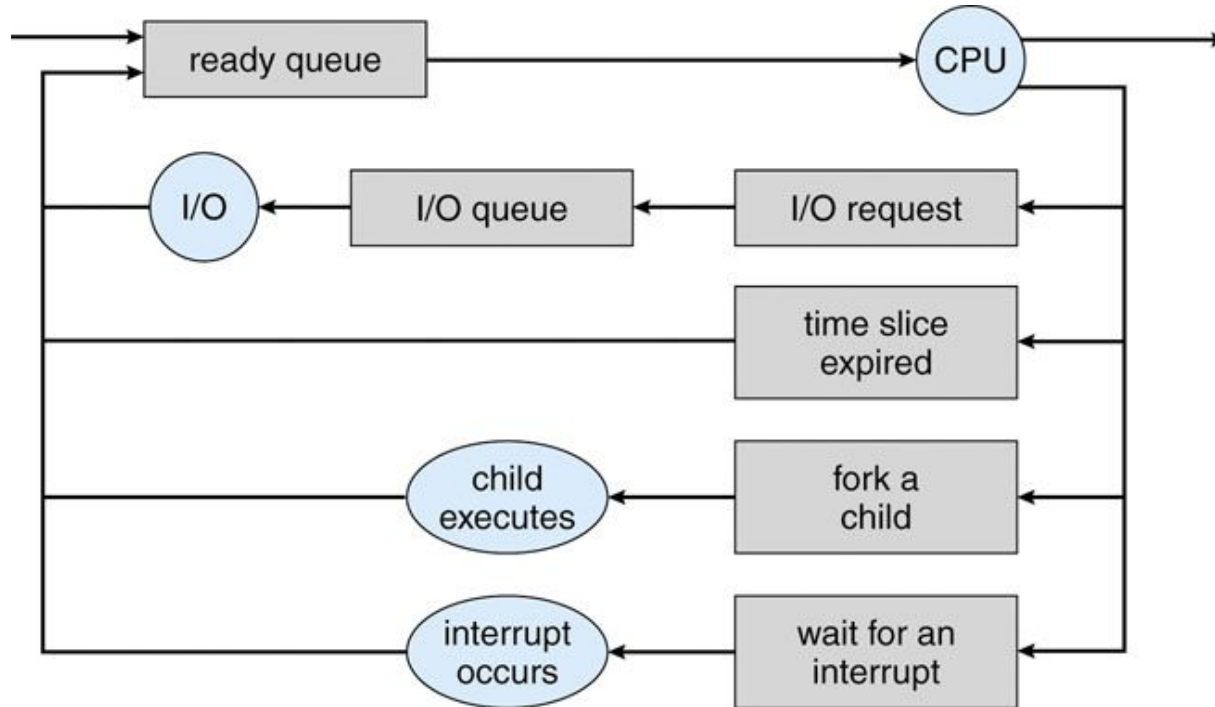
- [Problema] Como escolher o processo que vou executar?

- FIFO?

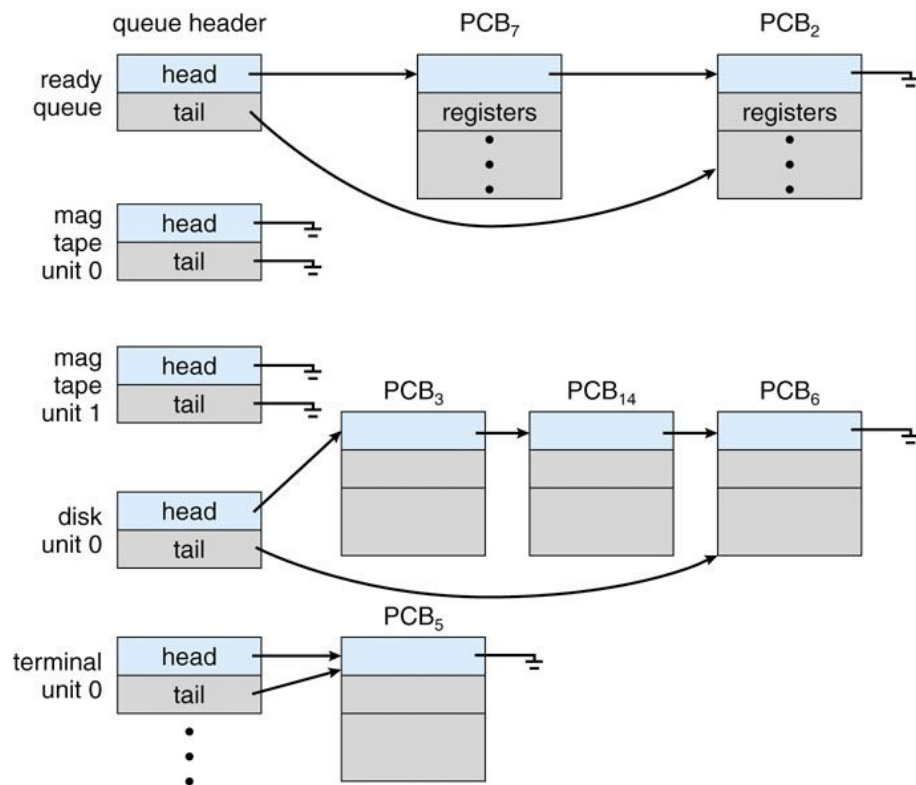


- Prioridade?
- Longest job first?

Passos do escalonador



Filas de processos

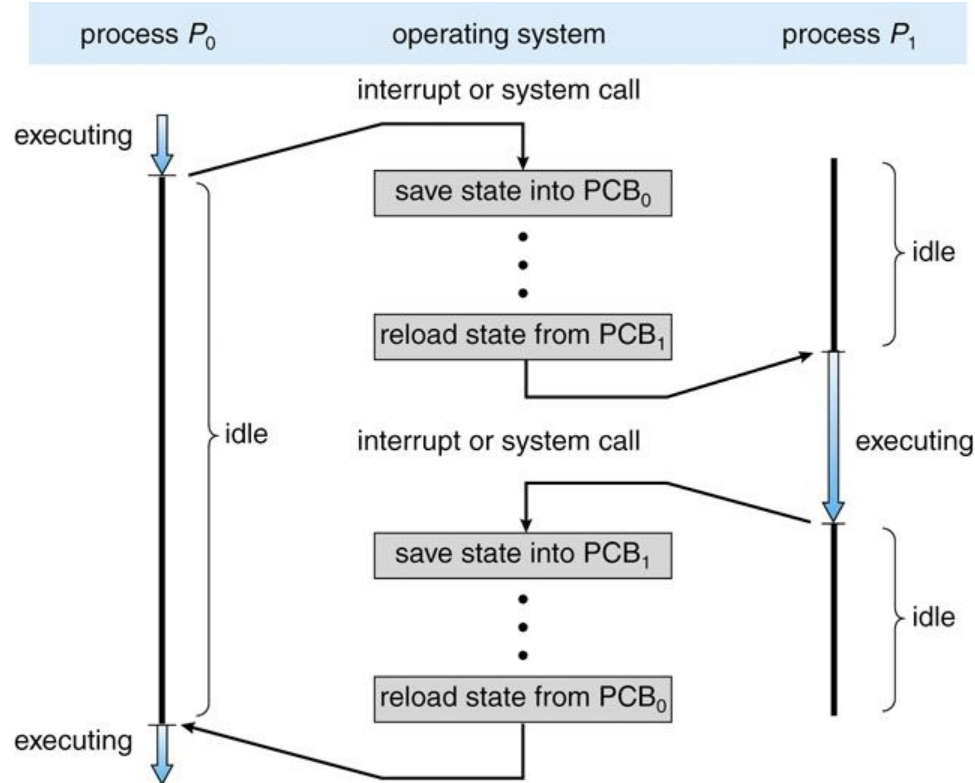


Troca de contexto

Kernel tem a capacidade de *preempt* (preemptar) processos

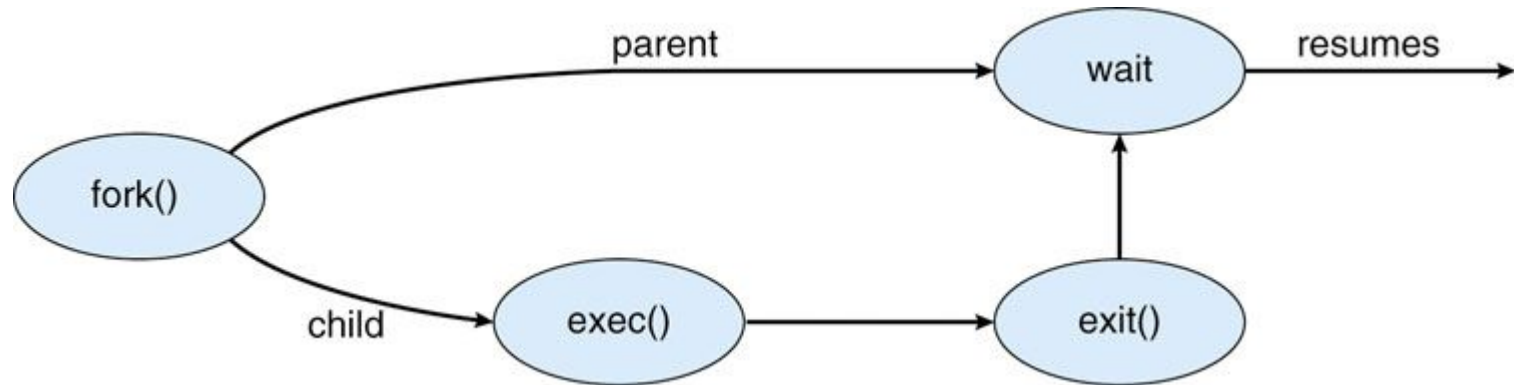
- Uso de chamadas de sistema como vimos
 - Quero ler do disco
 - Mande tantos bytes pela rede
 - Adquirir lock
- Quando um *quantum* expira
 - Garantindo menor latência
 - Todo mundo executa por 1 *quantum* (tantos ciclos da cpu)

Troca de contexto



Criação de processos

Processos criam outros processos utilizando fork



Terminação de processos

Sistema operacional recupera todos os recursos obtidos por um processo durante sua execução

