Escalonador de processos

Adriano Santos
Felipe Vasconcelos

Processo - Introdução

- Processo é uma abstração de um programa em execução
- Cada processo possui seu contador de programa
- Processos compartilham o código do mesmo programa
- Qual é a diferença entre programa e processo?

Processo - Classe de processos

- Processo Interativo
  - Interagem constantemente com seus usuários.
  - Exemplo: Editor de texto, aplicações gráficas.

- Processo Batch
  - Não precisam de interação do usuário, portanto, eles são executados em segundo plano.

- Processo em tempo real
  - Nunca devem ser bloqueados por processos de baixa prioridade.
  - Devem possuir baixa variação no tempo de resposta.
  - Exemplos são: programas que usam aplicações de som e vídeo, controladores de robos e programas que coletam informações de sensores físicos.

Processo - Estados

1. O processo bloqueia aguardando uma entrada
2. O escalonador seleciona outro processo
3. O escalonador seleciona esse processo
4. A entrada torna-se disponível

Processos

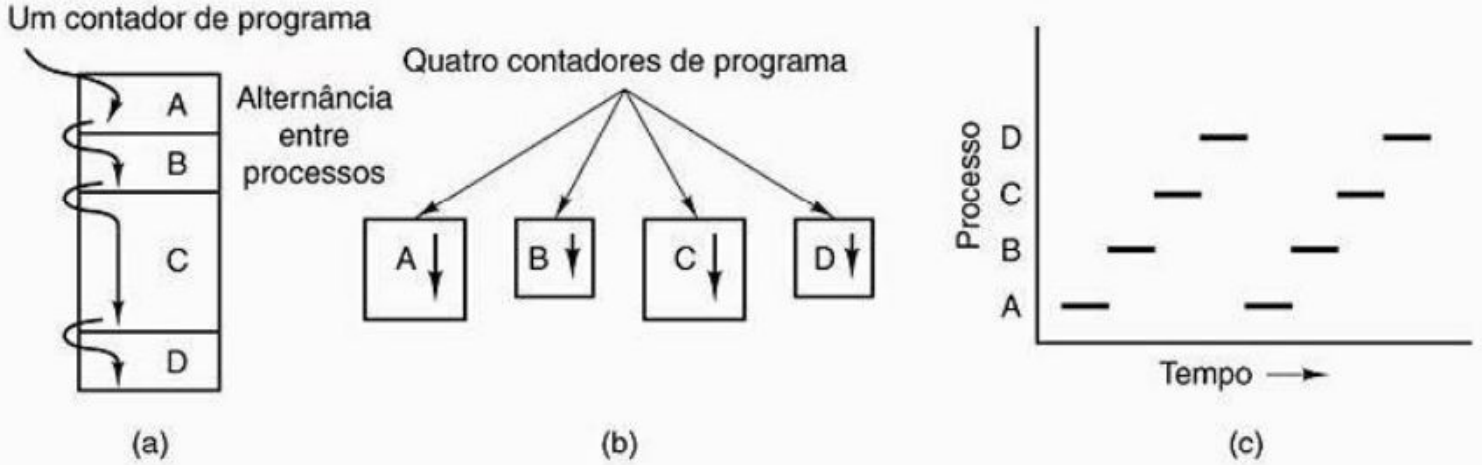


Figura 2.1 (a) Multiprogramação de quatro programas. (b) Modelo conceitual de quatro processos sequenciais independentes. (c) Somente um programa está ativo a cada momento.

Processo - Linux

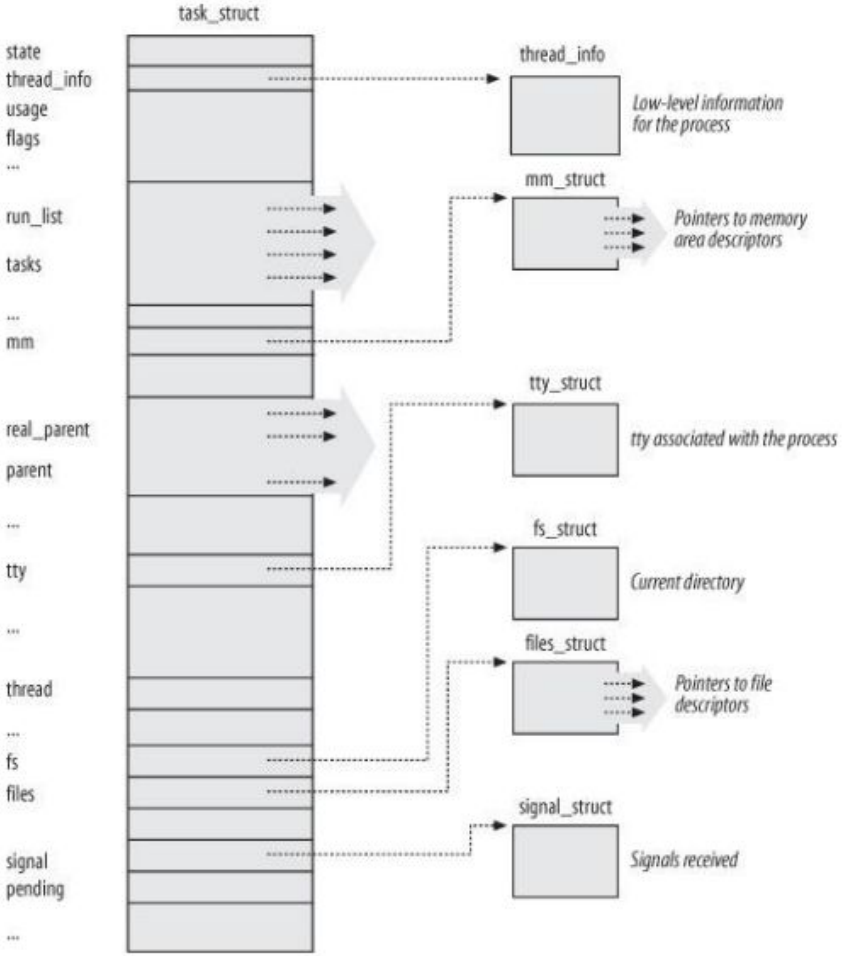
- O kernel deve ter uma imagem clara do que cada processo está fazendo

- Quais informações precisamos saber ?
  - Prioridade do processo,
  - Estado do processo
  - Espaço de endereço
  - entre outros

Processo - Linux

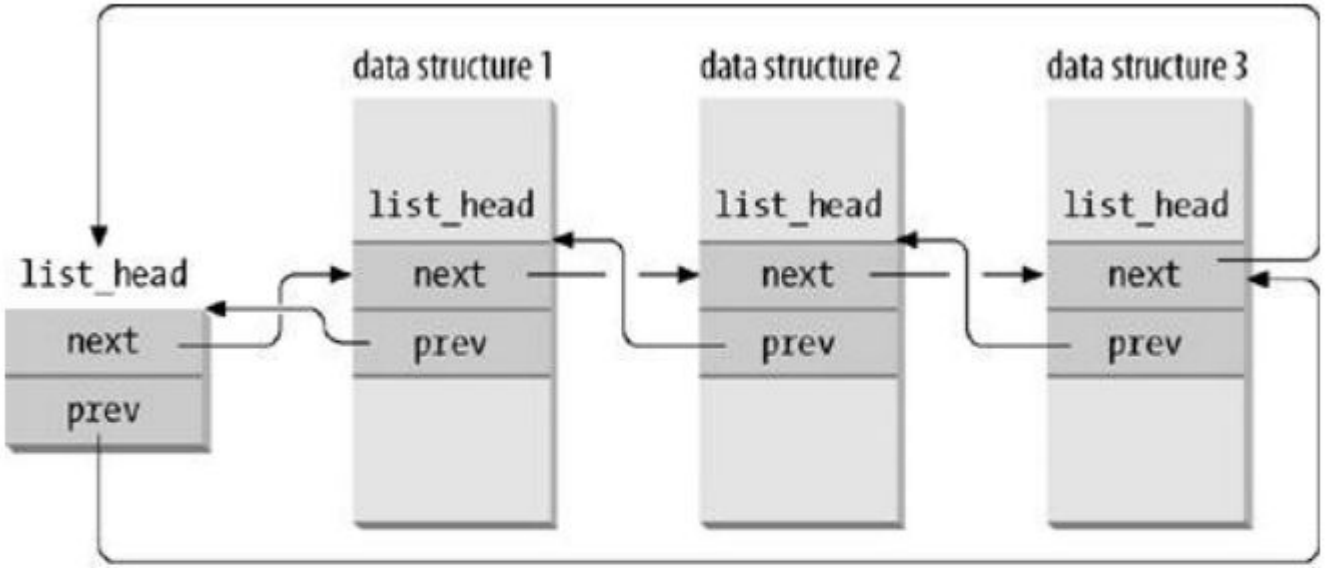
- Descritor de processo : struct task_struct
- Definir estado do processo
  - p->state = TASK_RUNNING;
  - TASK_RUNNING
  - TASK_INTERRUPTIBLE
  - EXIT_ZOMBIE
  - entre outros

Figure 3-1. The Linux process descriptor



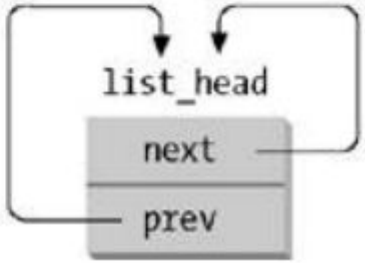
Processo - PID

- Cada processo possui um identificador
- init => PID = 1
- Qual é o PID do escalonador de processos do Linux?



(a) a doubly linked listed with three elements

(b) an empty doubly linked list



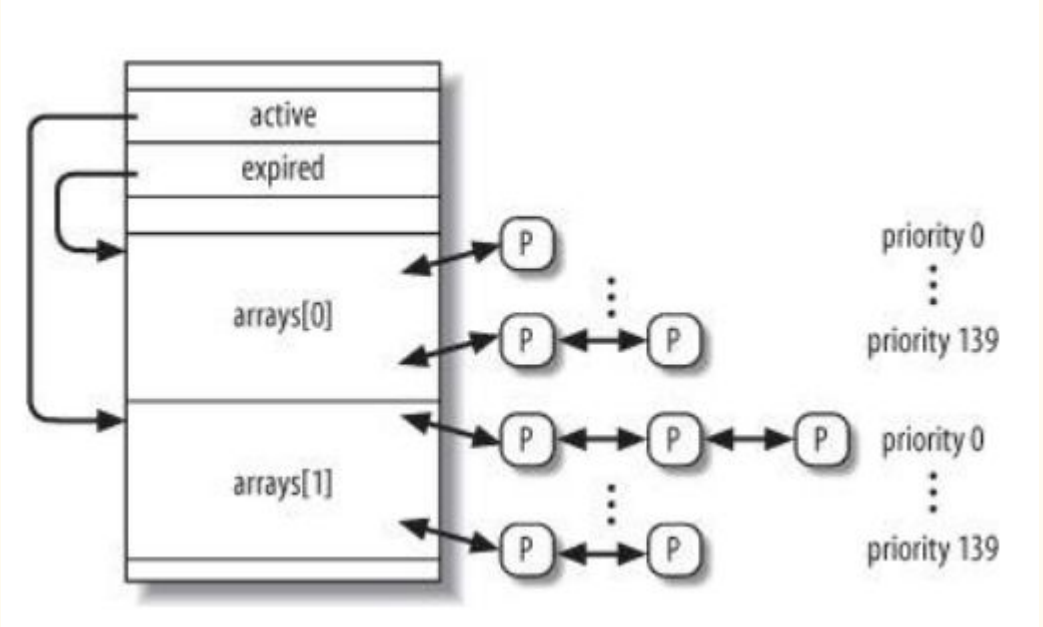
Processo - Lista duplamente ligada

- Utilizada para armazenar os descritores de processos
- Versões anteriores do Linux colocavam todos os processos executáveis na mesma lista(fila de execução)
- Atualmente existem várias filas de execução, uma lista por prioridade do processo
- Cada processador possui sua própria fila de execução, isto é, seu próprio conjunto de listas de processos

Escalonador - Estrutura de dados

Ativo: processos executáveis que ainda não esgotaram seu tempo de CPU

Expirado: processos executáveis que esgotaram seu tempo e são, portanto, proibidos de executar até que todos os processos ativos expire



Política de Escalonamento

- O algoritmo de escalonamento tradicional do Linux busca atender vários objetivos conflitantes, como:
    - Tempo de resposta do processo;
    - Taxa de transferência;
    - Starvation;
    - Conciliar processos com alta e baixa prioridades;

- Conjunto de regras que determina quando um processo deve ser executado.

Política de Escalonamento

- O Escalonador do Linux é baseado em técnicas compartilhamento de tempo e concorrência.
- A política de escalonamento também considera um ranking de prioridades dos processos.
- No Linux a prioridade de um processo é dinâmica.

- Processos são tradicionalmente classificados entre “CPU-bound” ou “I/O-bound”, além dos outros três tipos de classificação de processos que já vimos.

Política de Escalonamento

- Processos de Tempo Real são explicitamente reconhecidos pelo Linux.
- Não há como distinguir entre processos Batch e Interativos.
- Politicamente, o Linux sempre favorece os processos “I/O-bound” sobre os “CPU-bound”.

- É possível mudar a prioridade de um processo no Linux com chamadas ao sistemas.

Chamadas de Sistema ao Escalonador

- Existem diversas chamadas ao escalonador do Linux:
  - Usuários sempre podem diminuir a prioridade dos seus processos. Mas…

- A chamada de sistema *Agradável*
  - **nice( ) System Call:** É utilizada para aumentar ou diminuir a prioridade de um processo que será executado. Recebe um valor inteiro como parâmetro que pode ir de -20 até 19.
  - Valores positivos significam decréscimo da prioridade, valores negativos são acréscimos.
  - **renice()** define prioridades para processos em execução.

- Chamadas de sistema getpriority( ) e setpriority( )
  - São chamadas de sistema feitas sobre grupos de processos.
  - Parâmetros: PRIO_PROCESS, PRIO_PGRP, PRIO_USER.

Chamadas ao sistema sobre escalonamento

System Call	Description
`nice( )`	Change the priority of a conventional process.
`getpriority( )`	Get the maximum priority of a group of conventional processes.
`setpriority( )`	Set the priority of a group of conventional processes.
`sched_getscheduler( )`	Get the scheduling policy of a process.
`sched_setscheduler( )`	Set the scheduling policy and priority of a process.
`sched_getparam( )`	Get the scheduling priority of a process.
`sched_setparam( )`	Set the priority of a process.
`sched_yield( )`	Relinquish the processor voluntarily without blocking.
`sched_get_ priority_min( )`	Get the minimum priority value for a policy.
`sched_get_ priority_max( )`	Get the maximum priority value for a policy.
`sched_rr_get_interval( )`	Get the time quantum value for the Round Robin policy.

Preempção de processo

- O Linux trabalha com preempção de processos.
  - Quando ela pode ocorrer?
- flag need_resched

- **Exemplo de dois processos rodando:** Um processo de editor de texto, classificado como interativo e “I/O-bound”, e outro processo de um compilador no início de sua execução, classificado como Batch e “CPU-bound”. O que acontece durante a execução desses dois processos?

- Porém, o Kernel Linux é não preemptivo.

Quanto tempo um Quantum pode durar?

- A duração do quantum é crítica para a performance do sistema.

- **Imagine:** Um quatum demora 10 milisegundos, porém a troca de tarefas também leva 10 milisegundos.
  - Nesse caso 50% do tempo do processador é gasto em trocas de tarefas!!

- Por outro lado, um quantum muito grande pode passar a noção ao usuário de que o sistema operacional não responde.

- O Linux busca uma duração de quantum o mais longa possível, desde que o SO mantenha um bom tempo de resposta.

Cálculo Quantum

$$\text{base time quantum (in milliseconds)} = \begin{cases} (140 - \text{static priority}) \times 20 & \text{if static priority} < 120 \\ (140 - \text{static priority}) \times 5 & \text{if static priority} \geq 120 \end{cases}$$

- Prioridade estática
  - 100 = Alta prioridade
  - 139 = Baixa prioridade
- O escalonador olha a propriedade dinâmica para selecionar

Propriedade dinâmica

- dynamic priority= max (100, min(static priority- bônus + 5, 139))

- Bônus é um valor que varia de 0 a 10; um valor menor que 5 representa uma penalidade(diminui propriedade dinâmica), enquanto que um valor superior a 5 aumenta a prioridade

- O valor do bônus depende do tempo de espera

Algoritmo de Escalonamento

- O algoritmo de escalonamento do Linux divide o tempo em épocas.
- No início de cada época, todo processo tem um Quantum definido.

- Um processo pode ser selecionado para executar mais de uma vez durante uma mesma época, se o seu Quantum não tiver terminado.

- A época termina quando o Quantum de todos os processos for consumido.

Algoritmo de Escalonamento

- No Linux o Quantum do escalonador de processos é definido nas tarefas de inicialização do sistema, pela função DEF_PRIORITY:

  - #define DEF_PRIORITY (20*HZ/100)
    - HZ é a frequência de interrupções por tempo;
    - Nos computadores IBM é definido como 100;
    - Então temos um DEF_PRIORITY de 20 ticks do Clock para o escalonador, que é um quantum de aproximadamente 210ms.

Algoritmo de Escalonamento

- Como vimos anteriormente, processos possuem uma descrição que inclui diversos campos sobre escalonamento:
  - **need_resched:** Flag para chamar o escalonador voluntariamente.
  - **policy:** Podendo ser SCHED_FIFO, SCHED_RR, SCHED_OTHER ou SCHED_YIELD.
  - **rt_priority:** Prioridade estatica de processos Tempo Real.
  - **priority:** Base do calculo de Quantum.
  - **counter:** Contador do programa.

- E quando um programa faz fork de um processo?
  - current->counter >>= 1;
  - p->counter = current->counter;

Função schedule( )

- O escalonador pode ser chamado diretamente ou de forma adiada através de diversas rotinas do Kernel.
  - A chamada direta do escalonador ocorre quando um processo precisa de um recurso que não está disponível. A rotina executa os seguintes procedimentos:
  - 1. Inserir o processo atual na fila de espera apropriada.
  - 2. Muda o status do processo atual tanto para TASK_INTERRUPTIBLE quanto para TASK_UNINTERRUPTIBLE.
  - 3. Invoca o schedule( ).
  - 4. Checa se o recurso está disponível, se não, vai para o passo 2.
  - 5. Uma vez que o recurso está disponível, remove o processo da fila de espera.

Função schedule( )

- A chamada ao escalonador ocorre de forma adiada quando:
  - A flag **need_resched** é apontada igual a 1;
  - O Quantum do processo termina (**update_process_times( )**);
  - Quando um processo “acorda” e sua prioridade é maior do que a que está executando (**reschedule_idle( )**);
  - Quando é feita uma chamada de sistema as funções **sched_setscheduler( )** ou **sched_ yield( ).**

- O escalonador é chamado para definir as filas das rotinas do SO;
  - run_task_queue(&tq_scheduler);

O escalonador SMP do Linux

- Symmetric multiProcessor (SMP).
- O escalonador do Linux pode ser modificado para trabalhar com arquiteturas de multiprocessamento simétrico.
    - **Problema:** Temos um processo rodando na CPU 1 quando chega um segundo processo, de alta prioridade sobre o primeiro e que estava rodando na CPU 2. Nesse momento o Kernel precisa decidir sobre um dilema: Devo colocar o segundo processo para roda na CPU 1 imediatamente?

- O Linux adota uma solução empírica.

Função schedule( ) do SMP Linux

Algumas diferenças em relação a execução usual do escalonador:

- Na invocação do método **goodness()** o ultimo processo executado nessa CPU tem um “bonus” na prioridade.
- Guarda informações do tempo médio da execução anterior do processo.
- Na preempção de processos que estavam inativos, considera os valores do quanto o processo possui guardado em cash e o seu tempo médio na execução anterior.

Performance do escalonador do Linux

- O escalonador do Linux é considerado fácil de entender e modificar, porém a modificação de poucos parâmetros chave gera resultados muito diferentes em relação a performance.
- O algoritmo é considerado apropriado para as necessidade atuais do Linux.

- Porém ele não escalona muito bem...
  - Processos iterativos com alto tempo de resposta.
  - Raramente beneficia o escalonamento de processos IO-bound.

Performance do escalonador do Linux

- A predeterminação do Quantum é muito grande para o carregamento de sistemas grandes.
  - Que é a média do número de processos aptos à executar...
- A estratégia de priorizar processos IO-bound não é opcional.
  - Alguns programas Batch são IO-bound...
- Tem pouco suporte a aplicações em Tempo Real.
  - Processos rodando em modo Kernel podem tomar alguns milissegundos...

Referência

- Sistemas operacionais modernos - Andrew Stuart Tanenbaum - Cap. 2
- Understanding the Linux Kernel, Third Edition - Cap 3
-