

Gerenciamento de Processos no Linux

Marcelo Toledo <<http://www.marcelotoledo.org>>

26 de abril de 2005

Resumo

O modelo de gerenciamento de processos do Linux teve uma evolução notável desde o seu início, auxiliado pelo modelo Bazaar¹ de desenvolvimento, esses algoritmos foram enumeras vezes criticados e melhorados por eruditos ao redor do mundo, isso faz com que o Linux tenha hoje um dos melhores conjuntos de algoritmos para gerenciamento de processos já visto.

1 Introdução

Um processo pode ser descrito como parte de um programa que está aparentemente rodando. Este aparente existe somente pelo fato de que determinado processo pode entrar e sair diversas vezes do processador em um único segundo, e em um determinado momento ele pode não estar no processador e mesmo assim aparentemente estar rodando.

Como qualquer sistema de compartilhamento de tempo o Linux consegue dar a impressão de execução simultânea dos processos, separando um espaço bastante curto de tempo para cada um deles. Para ter sucesso nesta tarefa ele segue uma serie de regras que não desperdiça tempo de hardware com operações desnecessárias e consegue escolher qual processo deve ser executado naquele exato momento.

O que decide essa escolha no kernel é o escalonador de processos, que em grande parte é responsável pela produtividade e eficiência do sistema. Mais do que um simples mecanismo de divisão de tempo, ele é responsável por uma política de tratamento dos processos que permite os melhores resultados possíveis.

2 Primeiros Processos

Durante a fase de inicialização do Linux a função “start_kernel” é responsável por criar um thread, este é o processo de número zero, o primeiro e o ascendente de todos os outros processos. Após inicializar toda a estrutura de dados para este processo esta mesma função é responsável por chamar a função “init” que por sua vez utiliza a chamada de sistema “execve” para rodar o executável init, que será o processo número 1, mais conhecido como init.

Podemos chamar de Deus e pai de todos os outros processos, é o segundo processo a ser criado e um dos últimos a morrer. Seus filhos, vivem como nós seres humanos, eles nascem, se desenvolvem, tem uma vida mais ou menos produtiva, podem ter inúmeros filhos em poucos segundos e eventualmente morrem.

Existem alguns outros processos que também são criados pelo kernel durante a fase de inicialização e destruídos assim que o sistema desliga. Outros são criados sob demanda, assim que surge a necessidade deles são carregados.

¹The Cathedral and the Bazaar escrito por Eric Steven Raymond fala sobre dois modelos de desenvolvimento, bazaar que é o modelo aberto e cathedral que é o modelo comercial.

3 Estados

Uma das coisas que o escalonador precisa ter ciência é em qual estado está cada processo, na estrutura que armazena os dados de cada processo temos um array de possíveis estados onde apenas uma das opções abaixo estará ativa.

TASK_RUNNING Em execução ou aguardando para ser executado.

TASK_INTERRUPTIBLE O processo está suspenso até que determinada condição se torne verdadeira.

TASK_UNINTERRUPTIBLE Como o estado anterior, exceto pelo fato de que o seu estado não será modificado quando receber um sinal. É importante para os processos que necessitam executar determinada tarefa sem ser interrompido.

TASK_STOPPED Execução do processo foi parada.

TASK_ZOMBIE O processo está terminado, mas o processo pai ainda não executou uma chamada de sistema para retornar informações sobre o processo morto - “wait”, as informações não são descartadas pois ainda podem ser utilizadas.

4 Limite de Recursos

Por padrão o Linux limita os recursos que cada processo pode ter. Isto é, quanto de recursos do sistema ele pode utilizar. Isso é uma proteção para que caso o usuário faça algo errado, não prejudique a estabilidade do sistema. Esses limites são:

RLIMIT_AS O tamanho máximo que um processo pode ter em bytes. O kernel checka esse valor quando um processo utiliza a chamada de sistema “malloc” ou similar.

RLIMIT_CORE Quando um processo é abortado, o kernel pode gerar um arquivo core contendo as informações desse aborto. Este valor é utilizado para limitar o tamanho desse arquivo. Caso o valor seja zero 0, o arquivo não é criado.

RLIMIT_CPU O tempo máximo em segundos que um processo pode ser executado. Caso esse limite seja ultrapassado o kernel envia um sinal SIGXCPU para tentar pacificamente finalizar sua execução, se isso não acontecer ele envia um SIGKILL e mata o processo.

RLIMIT_DATA O tamanho máximo do heap ou memória de dados em bytes. O kernel checka esse valor antes de expandir o heap de um processo.

RLIMIT_FSIZE O tamanho máximo em bytes permitido para um arquivo. Se o processo tentar aumentar o tamanho de um arquivo que ultrapasse esse valor, o kernel envia um SIGXFSZ.

RLIMIT_LOCKS O número máximo de arquivos que um processo pode dar lock. Toda vez que o usuário tenta dar lock em um arquivo o kernel checka esse valor.

RLIMIT_MEMLOCK O tamanho máximo em bytes de memória que não permite swap. O kernel checka esse valor toda vez são utilizadas as chamadas de sistema “mlock” ou “mlockall”.

RLIMIT_NOFILE O número máximo de descritores de arquivos abertos. Toda vez que um descritor for aberto ou duplicado o kernel checka este valor.

RLIMIT_NPROC O número máximo de processos que um usuário pode ter.

RLIMIT_RSS A quantidade máxima de memória física que um processo pode ter.

RLIMIT_STACK O tamanho máximo em bytes da stack. O kernel checa este valor antes de expandi-la.

5 Preemptivo

Os processos do Linux são preemptivos, isso significa que quando um processo entra no estado `TASK_RUNNING` o kernel vai checar se existe alguma prioridade maior do que o processo corrente. Caso exista, o processo corrente é interrompido e o que tem prioridade maior começa a rodar.

Imagine a seguinte situação onde o Sr. Gênio dos Teclados utiliza apenas dois processos. O editor de textos Emacs para escrever o seu programa e ao mesmo tempo o gcc para compilar a versão mais recente do programa Pogobol Light. Por ser um programa interativo o editor de textos tem uma prioridade maior do que o compilador, mesmo assim ele ainda é suspenso diversas vezes para que o compilador possa rodar também. O Sr. Gênio dos Teclados digita algumas palavras e imediatamente após cada tecla digitada o kernel suspende a execução do compilador para poder processar o Emacs e assim o texto digitado aparecer na tela, isso acontece tão rápido que torna-se imperceptível, então o Emacs é suspenso novamente para que o gcc possa voltar a executar. Neste caso, quando dizemos que um processo foi suspenso, significa que ele continua com o estado `TASK_RUNNING`, porém não mais utilizando o CPU.

6 Política de Escalonamento

A prioridade de um processo no Linux está em constante alteração, o escalonador se mantém informado sobre o que os processos estão fazendo e assim torna-se possível ajustar a prioridade. Dessa maneira os processos que ficaram proibidos de utilizar o CPU por um longo intervalo de tempo, tem sua prioridade incrementada automaticamente, contrariamente os processos que passaram um longo período dentro do CPU são penalizados tendo sua prioridade decrementada.

Quando tocamos no assunto escalonamento é importante notar que os processos são classificados como da Forma E/S e Forma CPU. O primeiro faz uso extenso dos recursos de entrada e saída, isso significa que grande parte do seu tempo é utilizado no aguardo da conclusão das operações de Entrada e Saída, já o segundo são processos que necessitam de um longo tempo dentro do CPU.

Uma classificação alternativa define três classes de processos:

Processos Interativos Os processos que estão em constante interação com o usuário, sendo assim eles perdem grande parte do tempo esperando uma atividade do usuário, como um clique do mouse ou o aperto de uma tecla. Quando qualquer dessas ações for recebida o sistema precisa responder de forma ágil suspendendo qualquer processo que estiver rodando e colocar o processo interativo no CPU para processar a requisição solicitada pelo usuário, caso contrário ele pode pensar que o sistema não está respondendo. Um bom tempo de resposta é em torno de 50 e 150 ms.

Processos em Lote Não necessitam de nenhuma interação do usuário e por isso muitas vezes rodam em background. Como são processos de baixa prioridade, são frequentemente penalizados pelo escalonador. Programas de processamento em lote comuns são editores de texto, compiladores e programas gráficos.

Processos em Tempo Real Esses processos nunca devem ser bloqueados por processos de baixa prioridade, precisam de um tempo de resposta super rápido e com uma variação bastante baixa. Alguns aplicativos que fazem uso do tempo real são som, vídeo, controladores de robôs e equipamentos que envolvem a segurança humana.

7 Algoritmo de Escalonamento

O algoritmo de escalonamento do Linux funciona dividindo o tempo do CPU em fatias. Em uma única fatia cada processo tem um tempo específico de duração que é computada assim que essa fatia inicia. Geralmente processos diferentes tem tempos de execução diferentes. Quando o tempo de execução de um processo termina ele é retirado do CPU e outro processo que está rodando é colocado no seu lugar. Uma fatia termina quando todos os processos esgotaram seu tempo reservado de execução, assim o escalonador é responsável por calcular o tempo de execução para todos os processos e uma nova fatia inicia.

Para o Linux escolher qual processo deve rodar ele precisa escolher qual tem a prioridade maior, existem dois tipos de prioridade:

Prioridade Estática Definido pelo usuário para processos que necessitam de tempo real, os valores variam de 1 até 99 que nunca são modificados pelo escalonador.

Prioridade Dinâmica Esta é aplicada para os processos convencionais. A prioridade dinâmica dos processos convencionais é sempre inferior aos processos com prioridade estática.

Se existir um processo com prioridade estática no estado `TASK_RUNNING` nenhum processo dinâmico irá rodar até que ele pare de executar.

Referências

- [1] Daniel P. Bovet and Marco Cesati. *Understanding the Linux Kernel*. O'Reilly, second edition, December 2002.
- [2] John Levon. Scheduling in unix and linux. <http://www.kernelnewbies.org/documents/schedule/>, 2002.
- [3] Robert Love. *Linux Kernel Development*. Novell Press, second edition, January 2005.