8.9 Memória

Uma das maneiras mais simples de verificar o status de memória de seu sistema como um todo é executar o comando `free` ou visualizar */proc/meminfo* e ver a quantidade de memória real que está sendo usada para caches e buffers. Como acabamos de mencionar, problemas de desempenho podem surgir por falta de memória. Se não houver muita memória para cache/buffer sendo usada (e o restante da memória real estiver ocupada), você poderá precisar de mais memória. Contudo é fácil demais culpar a falta de memória por todos os problemas de desempenho de seu computador.

8.9.1 Como a memória funciona

Lembre-se de que, de acordo com o capítulo 1, a CPU tem uma MMU (Memory Management Unit, ou Unidade de gerenciamento de memória) que traduz os endereços de memória virtual usados pelos processos em endereços reais. O kernel dá assistência ao MMU dividindo a memória usada pelos processos em partes menores chamadas *páginas*. O kernel mantém uma estrutura de dados chamada *tabela de páginas* (page table) que contém um mapeamento dos endereços de páginas virtuais de um processo para endereços de páginas reais na memória. À medida que um processo acessa a memória, a MMU traduz os endereços virtuais usados por esse processo para endereços reais, de acordo com a tabela de páginas do kernel.

Um processo de usuário não precisa que todas as suas páginas estejam imediatamente disponíveis para poder executar. O kernel geralmente carrega e aloca as páginas à medida que um processo precisar delas; esse sistema é conhecido como *paginação por demanda* (on-demand paging ou simplesmente demand paging). Para ver como isso funciona, considere o modo como um programa inicia e executa como um novo processo:

1. O kernel carrega o início do código com as instruções do programa em páginas da memória.
2. O kernel pode alocar algumas páginas de memória de trabalho para o novo processo.
3. À medida que executar, o processo poderá alcançar um ponto em que a próxima instrução de seu código não estará em nenhuma das páginas carregadas inicialmente pelo kernel. Nesse ponto, o kernel assume o controle, carrega as páginas necessárias para a memória e, em seguida, permite que o programa retome a execução.

4. De modo semelhante, se o programa exigir mais memória de trabalho do que foi inicialmente alocada, o kernel cuidará disso ao encontrar páginas livres (ou criando espaço para elas) e atribuindo-as ao processo.

8.9.2 Page faults

Se uma página de memória não estiver pronta quando um processo quiser usá-la, o processo irá disparar um *page fault* (falha de página). No caso de um page fault, o kernel assumirá o controle da CPU para preparar a página. Há dois tipos de page faults: minor (secundário) e major (principal).

Page faults minor

Um page fault minor ocorre quando a página desejada está na memória principal, porém a MMU não sabe em que local ela está. Isso pode ocorrer quando o processo solicita mais memória ou quando a MMU não tem espaço suficiente para armazenar todos as localizações de páginas para um processo. Nesse caso, o kernel informa à MMU sobre a página e permite que o processo continue. Os page faults minors não são um problema sério, e muitos ocorrem à medida que um processo executa. A menos que você precise de um desempenho máximo para algum programa que faça uso intensivo de memória, é provável que você não vá precisar se preocupar com eles.

Page faults major

Um page fault major ocorre quando a página de memória desejada não está na memória principal, o que significa que o kernel deve carregá-la do disco ou de outro sistema de armazenamento lento. Muitos page faults major deixarão o sistema lento, pois o kernel deverá realizar uma quantidade substancial de trabalho para disponibilizar as páginas, tirando a chance de os processos normais executarem.

Alguns page faults major não podem ser evitados, por exemplo, aqueles que ocorrem quando o código é carregado do disco ao executar um programa pela primeira vez. Os problemas mais sérios ocorrem quando você começa a ficar sem memória e o kernel passa a fazer *swap* das páginas da memória de trabalho para o disco a fim de criar espaço para novas páginas.

Observando os page faults

Você pode explorar os page faults de processos individuais usando os comandos `ps`, `top` e `time`. O comando a seguir mostra um exemplo simples de como o comando

time exibe os page faults. (A saída do comando cal não é importante, portanto estamos descartando-a ao redirecioná-la para */dev/null*.)

```
$ /usr/bin/time cal > /dev/null
0.00user 0.00system 0:00.06elapsed 0%CPU (0avgtext+0avgdata 3328maxresident)k
648inputs+0outputs (2major+254minor)pagefaults 0swaps
```

Como você pode ver pelo texto em negrito, quando esse programa executou, houve 2 page faults major e 254 minor. Os page faults major ocorreram quando o kernel precisou carregar o programa do disco pela primeira vez. Se você executasse o comando novamente, é provável que não haveria nenhum page fault major, pois o kernel teria carregado as páginas do disco para cache.

Se preferir ver os page faults dos processos à medida que estiverem executando, utilize top ou ps. Ao executar top, utilize f para mudar os campos exibidos e u para exibir a quantidade de page faults major. (O resultado será mostrado em uma nova coluna nFLT. Você não verá os page faults minor.)

Ao utilizar ps, um formato personalizado de saída poderá ser usado para ver os page faults de um processo em particular. A seguir, apresentamos um exemplo para o ID de processo 20365:

```
$ ps -o pid,min_flt,maj_flt 20365
  PID  MINFL  MAJFL
20365 834182     23
```

As colunas MINFL e MAJFL mostram as quantidades de page faults minor e major. É claro que você pode combinar isso com quaisquer outras opções de seleção de processos, conforme descrito na página de manual ps(1).

Visualizar page faults por processo pode ajudar você a identificar determinados componentes problemáticos. Entretanto, se estiver interessado no desempenho de seu sistema como um todo, você precisará de uma ferramenta para sintetizar as ações de CPU e de memória para todos os processos.

8.10 Monitorando o desempenho da CPU e da memória com vmstat

Entre as diversas ferramentas disponíveis para monitorar o desempenho do sistema, o comando vmstat é um dos mais antigos, com um mínimo de overhead. Você o achará útil para obter uma visão geral da frequência com que o kernel está efetuando swap de páginas, o nível de ocupação da CPU e a utilização de I/O.

O truque para usar toda a eficácia do vmstat está em entender a sua saída. Por exemplo, a seguir, apresentamos uma saída de vmstat 2, que informa estatísticas a cada dois segundos:

```
$ vmstat 2
procs -----------memory---------- ---swap-- -----io---- -system-- ----cpu----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa
 2  0 320416 3027696 198636 1072568    0    0     1     1    2    0 15  2 83  0
 2  0 320416 3027288 198636 1072564    0    0     0  1182  407  636  1  0 99  0
 1  0 320416 3026792 198640 1072572    0    0     0    58  281  537  1  0 99  0
 0  0 320416 3024932 198648 1074924    0    0     0   308  318  541  0  0 99  1
 0  0 320416 3024932 198648 1074968    0    0     0     0  208  416  0  0 99  0
 0  0 320416 3026800 198648 1072616    0    0     0     0  207  389  0  0 100 0
```

A saída se divide em categorias: procs para processos, memory para uso de memória, swap para as páginas que entram e saem da área de swap, io para uso de disco, system para o número de vezes que o kernel alterna para o código do kernel e cpu para o tempo usado pelas diferentes partes do sistema.

A saída anterior é típica de um sistema que não está executando muitas tarefas. Normalmente, você começará observando a segunda linha da saída – a primeira é uma média do tempo em que o sistema está ativo. Por exemplo, nesse caso, o sistema tem 320.416 KB de memória que sofreu swap para o disco (swpd) e aproximadamente 3.025.000 KB (3 GB) de memória real livre (free). Embora um pouco de área de swap esteja em uso, as colunas si (swap-in) e so (swap-out) com valores iguais a zero informam que o kernel não está fazendo swap de nada no momento, de e para o disco. A coluna buff indica a quantidade de memória que o kernel está usando para buffers de disco (veja a seção 4.2.5).

Na extremidade direita, embaixo do cabeçalho CPU, você pode ver a distribuição do tempo de CPU nas colunas us, sy, id e wa. Essas colunas listam (em ordem) o percentual de tempo que a CPU está gastando em tarefas de usuário, em tarefas de sistema (kernel), o tempo de idle e o tempo de espera de I/O. No exemplo anterior, não há muitos processos de usuário executando (eles estão usando um máximo de 1% de CPU); o kernel não está fazendo praticamente nada, enquanto a CPU está tranquila, não fazendo nada durante 99% do tempo.

Agora observe o que acontece quando um programa grande é iniciado algum tempo depois (as duas primeiras linhas ocorrem imediatamente antes de o programa executar):

Listagem 8.3 – Atividade da memória

```
procs -----------memory---------- ---swap-- -----io---- -system-- ----cpu----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa
 1  0 320412 2861252 198920 1106804    0    0     0     0 2477 4481 25  2 72  0❶
 1  0 320412 2861748 198924 1105624    0    0     0    40 2206 3966 26  2 72  0
 1  0 320412 2860508 199320 1106504    0    0   210    18 2201 3904 26  2 71  1
 1  1 320412 2817860 199332 1146052    0    0 19912     0 2446 4223 26  3 63  8
 2  2 320284 2791608 200612 1157752  202    0  4960   854 3371 5714 27  3 51 18❷
 1  1 320252 2772076 201076 1166656   10    0  2142  1190 4188 7537 30  3 53 14
 0  3 320244 2727632 202104 1175420   20    0  1890   216 4631 8706 36  4 46 14
```

Como você pode ver em ❶ na listagem 8.3, a CPU passa a ter um pouco de uso durante um longo período, especialmente por causa dos processos de usuário. Como há memória livre suficiente, a quantidade de espaço de cache e de buffer usada começa a aumentar à medida que o kernel passa a usar mais o disco.

Mais tarde, vemos algo interessante: observe em ❷ que o kernel traz algumas páginas para a memória que haviam sido removidas pelo swap (a coluna `si`). Isso significa que o programa que acabou de executar provavelmente acessou algumas páginas compartilhadas por outro processo. Isso é comum; muitos processos usam o código de determinadas bibliotecas compartilhadas somente quando estão inicializando.

Observe também que, de acordo com a coluna `b`, alguns processos estão *bloqueados* (impedidos de executar) enquanto esperam por páginas de memória. Em geral, a quantidade de memória livre está diminuindo, porém não está nem próxima de se esgotar. Há também uma boa dose de atividade em disco, conforme visto pelos números crescentes nas colunas `bi` (blocks in) e `bo` (blocks out).

A saída é bem diferente quando a memória se esgota. À medida que o espaço livre acaba, os tamanhos tanto do buffer quanto de cache diminuem, pois o kernel precisa cada vez mais do espaço para os processos de usuário. Quando não restar mais nada, você começará a ver atividades na coluna `so` (swapped out), à medida que o kernel passa a transferir páginas para o disco, momento em que quase todas as demais colunas da saída mudarão para refletir o volume de trabalho que o kernel estará fazendo. Você verá mais tempo de sistema, mais dados de e para o disco e mais processos bloqueados porque a memória que eles querem usar não estará disponível (sofreu swap para disco).