Escalonador em passos largos (Stride Scheduler) no sistema operacional educacional Xv6

Jovani de Souza[1], **Davi R. Pegoraro**[1]

[1]Ciência Da Computação – Universidade Federal da Fronteira Sul (UFFS)
Chapecó – SC – Brazil

jovanidesouza@hotmail.com, davi.pegoraro@hotmail.com

Abstract. *This paper will present a implementation of the stride scheduling method, using the unix-like educational Xv6 as plataform. In this article will be cover all the steps of the Stride Scheduler implementation, highlighting the implementation key phases and showing to the reader by means of tests the principals benefits e disadvantages of the scheduler. So will be demostrated the main problems finding during the implatation fase of this kind of scheduler.*

Resumo. *Este artigo apresentará a implementação de um sistema de escalonamento em passos largos(Stride scheduler), usando como plataforma o sistema Unix-like educacional Xv6. Nesse artigo será abordado todos os processos da implementação do escalonador em passos largos, assim ressaltando as etapas chaves do desenvolvimento e mostrando ao leitor por meio de testes as principais vantagens e desvantagens do escalonador. Também será apontado os principais problemas encontrados no desenvolvimento deste tipo de escalonador de processos.*

1. Introdução

Em um sistema operacional com múltipla programação, é de alta importância a boa implementação de um escalonador de processos. Um escalonador que é também chamado de agendador de tarefas, é responsável por decidir qual processo concorrente será executado pelo sistema operacional.

Com isso, será implementado um escalonador de processos em passos largos (Stride Scheduler) para o sistema operacional educacional Xv6, desenvolvido pelo Instituto de tecnologia de Massachusetts. O objetivo é mostrar o escalonamento de múltiplos processos em um sistema operacional interativo, por meio de uma abordagem determinística para escolher entre os processos que disputam os recursos do CPU.

Para realizar todo o projeto, várias partes do código base do sistema precisam ser alteradas. Todo processo criado passará a contar com um número de tickets, além disso, o escalonador calculará a passada de cada processo, usando uma incrementação determinística para selecionar o processo a ser executado. Após concluir a implementação do escalonador, pretende-se demonstrar a sua eficácia por meio de testes de criação e execução de processos.

2. Sistema Operacional

De modo geral, um sistema operacional atua como um gerenciador de recursos em um sistema computacional (Sistema embarcado, Computador, etc). Ele serve como um intermediário entre hardware, software e interação do usuário. Um sistema operacional é

responsável pelo sistema de arquivos de um computador, pelo gerenciamento da memória e também pela execução de comandos e softwares do usuário.

Também é tarefa do sistema operacional, definir mecanismos de otimização de tarefas da CPU, buscando sempre maneiras de melhor aproveitar o tempo de trabalho de um processador. O bom desempenho de um sistema computacional depende de uma das principais aplicações encontradas em um sistema operacional interativo, o escalonador de tarefas.

2.1. Xv6

O Xv6 é um sistema Unix-Like, versão 6, com foco educacional. Trata-se de uma implementação de um sistema Unix para multiprocessadores x86, desenvolvido pelo Instituto de tecnologia de Massachusetts.[MIT]

Esse sistema, tona-se uma boa escolha para demonstrar a construção de aplicações para um sistema computacional, pois sua estrutura básica é bem simples, fazendo com que a quantidade de código do sistema em questão seja bem menor do que outros sistemas mais robustos.

Por ser destinado ao ensino de sistemas operacionais, ele permite que todas as suas funcionalidades existentes sejam modificadas e ainda que outras sejam facilmente adicionadas. Fatos que tornam o Xv6 ideal para demonstrar a construção de um escalonador de processos.

3. Escalonador

O escalonador de processos de um sistema operacional, é uma das aplicações mais importantes, tratando-se de sistemas multitarefas. Cada núcleo de CPU de um processador é capaz de executar apenas um processo por vez, o escalonador então, faz a escolha do processo que vai ser executado e ainda decide quais serão os próximos da fila de execução por meio de um algoritmo de escalonamento.

A escolha do processo que vai ter prioridade para ser executado depende do sistema operacional em questão. Sistemas menores com processos de igual prioridade, podem simplesmente usar uma fila, onde o primeiro processo a ficar pronto, ganha o uso da CPU. Porém, em sistemas operacionais mais sofisticados, essa prática pode ser prejudicial ao sistema, pois os processos tendem a precisar de níveis de prioridades diferentes.

Para sistemas computacionais em geral, existem diversos algoritmos de escalonamento aplicáveis, e a escolha do escalonamento depende muito da aplicação do sistema operacional e da complexidade dos processos que estão sendo executados.

3.1. Escalonador de passos largos

O escalonador em passos largos é um tipo de escalonador de processos que usa uma abordagem determinística. Para isso, todo processo criado ganha um numero de bilhetes (tickets) que é usado para definir a prioridade de execução do processo. De modo geral, um processo precisa ter um número de tickets maior que zero e também os processos com um número maior de tickets tem maior prioridade de execução. [Carl A. Waldspurger 1995]

O escalonador de passos largos utiliza de alguns recursos que precisam ser adicionados ao projeto. Além de um número de tickets, um processo também vai ter um valor

de ”passo”e uma variável ”passada”que será incrementada toda vez que o processo for selecionado para a execução.

Para calcular o valor do ”passo”do processo será utilizado o resultado da divisão de uma constante com valor alto, pelo número de tickets do processo. Já a ”passada”do processo inicialmente é zero, e vai sendo incrementada sempre que o processo for executado, fazendo com que o número de passada do processo aumente a cada execução.

O escalonador vai escolher sempre o processo com o menor valor de passada, a incrementação faz com que o processo que acabou de ser executado ”perca”parte ou toda a vantagem que ele teve quando foi escolhido, como inicialmente como todos os valores de passada dos processos escalonáveis são zero, o critério de desempate é o processo com o maior valor de PID (identificação única do processo).

4. Implementação

O projeto iniciou com muita teoria. A leitura principal foi baseada no manual do Xv6 disponibilizado pelo MIT, nele é possível encontrar a descrição do sistema operacional como um todo, desde sua estrutura básica até seu escalonador padrão. Inicialmente foi realizado o clone do repositório do sistema para iniciar os testes iniciais e conhecer o funcionamento básico do Xv6. Para instalar o sistema foi utilizado o emulador QEMU, com a finalidade de evitar danos ao hardware e também facilmente reiniciar as configurações e o próprio sistema operacional.

No primeiro contato com o código do sistema, foi necessário realizar uma pequena configuração inicial. No arquivo de configurações ”Makefile”, onde foi alterado o número lógico de CPUs do sistema, que por padrão é 2, e para a implementação dos testes foi optado por usar apenas 1 CPU.

```
208 # QEMU's gdb stub command line changed in 0.11
209 QEMUGDB = $(shell if $(QEMU) -help | grep -q '^-gdb'; \
210     then echo "-gdb tcp::$(GDBPORT)"; \
211     else echo "-s -p $(GDBPORT)"; fi)
212 ifndef CPUS
213 //CPUS := 2
214 CPUS := 1
215 endif
216 QEMUOPTS = -drive file=fs.img,index=1,media=disk,format=
217
218 qemu: fs.img xv6.img
219     $(QEMU) -serial mon:stdio $(QEMUOPTS)
220
```

Arquivo: Makefile. Linha: 214

4.1. Estrutura

O seguinte passo foi alterar a estrutura base dos processos. O Xv6 tem uma estrutura única definida no arquivo proc.h onde todos os processos criados são baseados. A alteração é necessária pois originalmente um processo possui apenas o básico para existir.

Então com a alteração, todo processo criado passa a ter: um número de ”tickets”, para que se possa definir a prioridade de execução que o processo vai ter; uma variável ”passo”que vai armazenar o tamanho do stride do processo; uma variável ”passada”que será incrementada com o valor do ”passo”toda vez que o processo ganhar a CPU; e também uma variável ”executado”que será um contador usado para armazenar a quantidade de vezes que o processo foi executado.

```
57   struct inode *cwd;           // Current directory
58   char name[16];               // Process name (debug
59
60   int tickets;                 //Número de tickets do
61   int passo;                   //variavel passo (stri
62   int passada;                 //variavel passada que
63   int executado;               //variavel que guarda
64 };
65
66 // Process memory is laid out contiguously, low addre
67 //   text
```

Arquivo: proc.h

4.2. userinit() e fork()

Na função ”userinit”do arquivo ”proc.c”, que é responsável por criar o primeiro processo do SO e vai ser o pai de todos os outros processos, foi adicionado um valor de tickets padrão baseado em uma variável global definida no arquivo ”proc.h”.

```
131   inituvm(p->pgdir, _binary_initcode_start,
132   p->sz = PGSIZE;
133   memset(p->tf, 0, sizeof(*p->tf));
134   p->tf->cs = (SEG_UCODE << 3) | DPL_USER;
135   p->tf->ds = (SEG_UDATA << 3) | DPL_USER;
136   p->tf->es = p->tf->ds;
137   p->tf->ss = p->tf->ds;
138   p->tf->eflags = FL_IF;
139   p->tf->esp = PGSIZE;
140   p->tf->eip = 0;  // beginning of initcode.
141   p->tickets = TICKET_DEFAUT;              //
142
```

Arquivo: proc.c

Depois dessa pequena alteração na função ”userinit”, as principais alterações na criação dos processos são feitas na função fork(). Nessa função foi alterado o parâmetro que antes era vazio e agora passa a receber um valor inteiro que será atribuído ao número de tickets do processo, essa atribuição permite que a aplicação do usuário atribua uma quantidade específica de tickets para um determinado processo.

Também na função fork, foi adicionado um pequeno código que garante a atribuição de tickets para todo processo criado, esse código é usado para evitar problemas na definição dos tickets, uma vez que o escalonador usará esse número para escalonar o processo. Caso não seja passado um valor de tickets para o processo, ele receberá o valor padrão, caso o valor passado por parâmetro seja inferior ao mínimo ou superior ao máximo permitido, o código vai atribuir o valor mínimo ou máximo respectivamente.

```
205   ////////////////////DEFINIÇÃO DA QUANTIDADE DE TICKETS DE CADA PROCESSO//
206
207   if(tickets == 0){
208     np->tickets = TICKET_DEFAUT;
209   }
210   else if(tickets < 0 || tickets < TICKET_MIN){
211     np->tickets = TICKET_MIN;
212   }
213   else if(tickets > TICKET_MAX){
214     np->tickets = TICKET_MAX;
215   }
216   else{
217     np->tickets = tickets;
218
219   }
220
221   ////////////////////////////////
222   np->passo = CONS_PASSO / np->tickets;
223   np->passada = 0;
224   np->executado = 0;
225   ////////////////////////////////
226
227   cprintf("O processo %d foi criado com o valor de tickets: %d e passo: %d\n",
228
```

Alterações na função fork()

Para fins de 'debugging' um Printf foi adicionado na função fork(), com a finalidade de informar na tela toda vez que um processo é criado, informando também o "pid"do processo e o número de tickets com que o mesmo foi criado.

A última alteração na função fork() foi as atribuições das variáveis criadas em proc.h, por meio de uma contante definida com o valor de 10000, é calculado o valor do passo do processo, o valor do passo do processo é o resultado da divisão da constante pelo número de tickets do mesmo, é importante ressaltar que o número de stride do processo vai ser inversamente proporcional ao número de tickets que ele possui, ou seja, quanto mais tickets o processo ganhar, menor vai ser o seu passo.

Por padrão a "passada"de todos os processos vai ser 0 e então vão sendo incrementadas quando o processo for executado, da mesma forma a flag "excutado"inicializa com 0 e depois é incrementada.

5. Scheduler()

O escalonador de passos largos funciona de uma maneira determinística, ou seja, ele deve apresentar o mesmo resultado para todas as vezes que um conjunto de processos for escalonado da mesma forma, portanto ele usa uma abordagem que determina de maneira constante os processos que vão ganhar o CPU.

Ele faz uma varredura linear na tabela de processos do SO, buscando primeiramente pelos processos prontos para serem executados, em seguida seleciona entre eles o processo com o menor valor de "passada", caso ele encontre processos com o mesmo valor de "passada", o desempate é feito pelo "pid", dando prioridade para o processo com o maior "pid". Ao finalizar a varredura na tabela o escalonador já possui o processo vencedor, e então o libera para execução alterando o estado do processo para "running".[Carl A. Waldspurger 1995]

A primeira alteração significativa no escalonador antigo foi feito no loop que faz a varredura pela tabela de processos. O escalonador original do Xv6, faz a varredura a partir do início da tabela e vai mandando pra execução o primeiro processo encontrado que esteja em estado de "runnable".

O novo escalonador faz a varredura na tabela e então, ao final do loop, o processo com menor número de passada é escolhido e mandado para execução. Isso faz com que o escalonador precise sempre verificar toda a tabela de processos antes de decidir qual processo vai ganhar a CPU, sendo assim determinístico na sua escolha.

Dentro do laço de varredura, o escalonador ignora todos os processos que não estão prontos para executar. Também no loop o escalonador sempre atualiza uma variável chamada "menor"com o menor valor de passada do processo, e caso encontre um processo com um menor valor de passada esse processo é copiado para uma variável auxiliar chamada "escolhido". o escalonador por fim trata os empates, caso o processo atual do loop tenho o mesmo número de passada do auxiliar "escolhido"então o desempate é feito pelo maior "pid".

Saindo do loop, o escalonador já determinou qual foi o processo "escolhido"para ser executado, então ele faz a alteração no valor de "passada", somando a ele o valor de "passo"do processo e atualiza a flag "executado"somando 1, para indicar que o processo ganhou a CPU mais uma vez, por fim libera a CPU para que outro processo seja

escalonado.

```
362   for(;;){
363     // Enable interrupts on this processor.
364     sti();
365
366     // Loop over process table looking for process to run.
367     acquire(&ptable.lock);
368
369     menor = -1;//variavel de referencia para primeiro caso
370     escolhido = ptable.proc;
371
372     for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){//varredura na tabela
373       if(p->state != RUNNABLE){//se o processo não for runnable então contin
374         continue;
375       }
376
377       if(menor == -1 || p->passada < menor){//verifica a passada do processo
378         menor = p->passada;//se for menor. atualiza a referencia
379         escolhido = p;
380       }
381
382       if(p->passada == menor){//caso de desempate
383         if(p->pid > escolhido->pid){//se der empate entre as passadas, então
384           escolhido=p;
385         }
386       }
387
388     }
389       // Switch to chosen process.  It is the process's job
390       // to release ptable.lock and then reacquire it
391       // before jumping back to us.
392       c->proc = escolhido;
393       switchuvm(escolhido);
394       escolhido->passada = escolhido->passada + escolhido->passo;//atualiza
395       escolhido->executado = escolhido->executado + 1;//atualiza o numero de
396       escolhido->state = RUNNING;//muda o estado do processo para running
397
398       swtch(&(c->scheduler), escolhido->context);
399       switchkvm();
400       c->proc = 0;
401       // Process is done running for now.
402       // It should have changed its p->state before coming back.
403
404     release(&ptable.lock);
405
406   }
407 }
```

Alterações na função Scheduler()

6. Teste

Como forma de testar o escalonado, foi desenvolvido um aplicativo simples que cria 3 processos com diferentes números de tickets. Então escalona os 3 e analisa o padrão de execução em diversas vezes, como o algoritmo é determinístico, todas as vezes que o aplicativo rodar, ele deve apresentar o mesmo resultado.

7. Considerações Finais

O escalonador é uma das principais aplicações do sistema operacional Multiprogramado, ele se torna indispensável para aproveitar melhor os recursos do CPU, dando ao usuário a ilusão de paralelismo ao alternar entre os processos que requisitam o uso do CPU.

Em um sistema interativo o escalonador pode ser implementado com diversos algoritmos, o que foi adotado aqui, por loteria, mostrou-se eficaz ao introduzir uma nova forma de escalonar ao sistema Xv6, que originalmente usa uma fila circular para agendar a execução dos seus processos. Usando uma abordagem determinística para definir qual processo vai vencer a disputa pela CPU, o escalonador por passos largos mostrou-se de grande utilidade quando se busca escalonar processos com diferentes níveis de prioridade

Conclui-se que o algoritmo de escalonamento por passos largos é eficaz mas não deve ser usado como regra para todo sistema, pois cada sistema tende a precisar de um

escalonamento dedicado as suas necessidades. No entanto o escalonador, consegue suprir com qualidade dois pontos interessantes para um sistema operacional, executar múltiplos processos e criar níveis de prioridade.

Referências

Carl A. Waldspurger, W. E. W. (1995). *Stride Scheduling: Deterministic Proportional-Share Resource Management*. MIT Laboratory for Computer Science.

MIT. *xv6a simple, Unix-like teaching operating system*.