Escalonador do tipo loteria 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 article describes the experience of implementing a lottery scheduler for the Xv6 educational operating system. In a system dedicated to education like Xv6 it becomes simple to implement new functionalities, such as lottery scheduling, which basically assigns tickets to processes competing with CPU resources, then through a lottery, determines the process that will earn the requested resources. In this article, we intend to present the steps of the development of the scheduler by lottery, as well as the advantages found in this type of scheduler.*

Resumo. *Este artigo descreve a experiência de implementar um escalonador em loteria para o sistema operacional educacional Xv6. Em um sistema dedicado à educação como o Xv6 torna-se simples a implementação de novas funcionalidades, como o escalonador de processos por loteria (lottery scheduling) que, basicamente atribui bilhetes para os processos concorrentes aos recursos do CPU, então atrvés de um sorteio, determina o processo que ganhará os recursos requisitados. Neste artigo, pretende-se apresentar os passos do desenvolvimento do escalonador por loteria, bem como as vantagens encontradas nesse tipo de escalonador.*

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.

A patir disso, será implementado um escalonador de processos em loteria 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 sorteio de tickets 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 fará sorteio do ticket vencedor, usando uma função randômica para gerar valores e por fim o escalonador vai destinar o CPU para o processo que possui o ticket sorteado. Após concluir a implementação do escalonador, pretende-se demonstrar a sua eficácia por meio de testes de criação de processos e sorteio para decidir qual será executado primeiro.

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 por loteria

O escalonador do tipo loteria (lottery scheduler) é o tipo de escalonador de processos que será implementado no sistema Xv6 para gerenciar a execução dos processos do sistema. Esse escalonador implementa um algoritmo de loteria, que realiza sorteios para

decidir qual processo será executado pela CPU. Os processos ao serem criados recebem um número de tickets definidos pelo usuário, caso não seja atribuído, o processo recebe um valor padrão.

No momento do escalonamento o escalonador realiza o sorteio por meio de uma função de geração de números randômicos entre os processos que estão em estado de pronto para serem executados. Após o sorteio, o escalonador faz uma varredura na tabela de processos para encontrar o processo com o ticket vencedor e então executá-lo.[David Petrou 1999]

Implementado de maneira correta, esse escalonar oferece uma solução justa para a execução de múltiplos processos. Com o uso de tickets, é possível criar níveis de prioridade entre os processos, dando um número maior de tickets para um processo com maior prioridade. Um exemplo disso é: dois processos A e B estão prontos para serem executados e o escalonador oferece um total de 100 tickets, o processo A tem maior prioridade e então recebe 75 tickets, o processo B fica com os outros 25 tickets, ao realizar o sorteio, o processo A (com maior prioridade) tem 75 por cento de chance de ser executado antes que o processo B.

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

O seguinte passo foi alterar a estrutura base dos processos. O Xv6 tem uma estrutura única 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.

Em outras palavras, é preciso definir um espaço na estrutura, para que o processo criado possa ter tickets e consequentemente possa ”participar do sorteio”.

```
38 struct proc {
39   uint sz;                     // Size of process memory (bytes)
40   pde_t* pgdir;                // Page table
41   char *kstack;                // Bottom of kernel stack for this process
42   enum procstate state;        // Process state
43   int pid;                     // Process ID
44   struct proc *parent;         // Parent process
45   struct trapframe *tf;        // Trap frame for current syscall
46   struct context *context;     // swtch() here to run process
47   void *chan;                  // If non-zero, sleeping on chan
48   int killed;                  // If non-zero, have been killed
49   struct file *ofile[NOFILE];  // Open files
50   struct inode *cwd;           // Current directory
51   char name[16];               // Process name (debugging)
52    int tickets;                //Número de tickets do processo
53 };
```

Arquivo: proc.h

Depois de deixar a estrutura do processo pronta, foi necessário fazer com que as funções que criam os processos fossem modificadas, para que passassem a criar processos com um valor de tickets.

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”.

```
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.S
141   p->tickets = TICKET_DEFAUT;
142
```

Arquivo: proc.c

Sendo assim, o primeiro processo vai ser criado e inicializado com o padrão de 10 tickets. A partir do processo pai, a função ”fork()”do arquivo ”proc.c”, é responsável por criar os processos filhos baseados no processo pai. Nessa função é preciso fazer modificações para que agora ela crie os processos com tickets.

```
int
fork(int tickets)
{
  int i, pid;
  struct proc *np;
  struct proc *curproc = m
```

Parâmetro da função fork() alterado

Essa foi a mudança no parâmetro da função, que antes era vazio e agora recebe um número inteiro que é atribuído ao número de tickets do processo. Também foi preciso criar uma proteção para que essa atribuição não seja violada, para isso, foram criadas mais duas variáveis globais com o número mínimo e máximo de tickets que um processo pode ter, além da variável que já foi criada para número padrão de tickets”.

```
207   ////////////////////////DEFINIÇÃO DA QUANTIDADE DE TICKETS
208   if(tickets == 0){
209     np->tickets = TICKET_DEFAUT;
210   }
211   else if(tickets < 0 || tickets < TICKET_MIN){
212     np->tickets = TICKET_MIN;
213   }
214   else if(tickets > TICKET_MAX){
215     np->tickets = TICKET_MAX;
216   }
217   else{
218     np->tickets = tickets;
219   }
```

Proteção de atribuição de tickets

Nesse código, o processo recebe o número padrão de tickets caso ele receba 0 na sua criação; Recebe o valor mínimo 1 caso receba um número negativo na sua criação; Recebe o valor máximo 100 caso receba um valor maior que o máximo na sua criação e por fim recebe o próprio valor de criação, caso esse valor esteja dentro da faixa de valores de 1 até 100.

```
220
221   cprintf("O processo %d foi criado com o valor de tickets: %d\n", np->pid, np->tickets);
222
```

Print na tela para cada processo criado

Ainda na função fork(), foi adicionado um printf, para que toda vez que um processo é criado a função também imprima na tela do sistema o id(identificador) do processo e a quantidade de tickets com que ele foi criado.

Terminando as alterações na função fork(), ainda foi necessário criar algumas funções antes de alterar o escalonador. Basicamente o escalonador por loteria vai precisar do número de tickets que vão participar do sorteio e também de um número randômico para representar o ticket vencedor.

Para gerar um número randômico foi utilizado a função rand() disponibilizada pelo próprio sistema Xv6, essa função foi encontrada no arquivo ”usertests.c”e foi levemente modificada para que retornasse um valor aleatório para o escalonador.

```
337 ////////////////////////////////////////////////////////////////
338 //Função para gerar numero randomico
339 //Função encontrada no próprio xv6 em usertests.c
340
341 unsigned long randstate = 1;
342 unsigned int rand(int TICKET_TOTAL){///////////////função rand
343   randstate = (randstate * 1664525 + 1013904223)%TICKET_TOTAL;
344   return randstate;
345 }
```

Função rand utilizada para gerar um número aleatório

Outra função adicionada foi a função ticketcount(), que realiza uma busca na tabela de processos, e para cada processo encontrado que esteja no estado RUNNABLE ela soma o seu número de tickets à uma variável. No final a função retorna essa variável para o escalonador que passará a ter o número de tickets que vão participar do sorteio.

```
347 //Função contador para contar o numero total de tickets que vão participar
348 ////////////////////////////////////////////////////////////////////////////
349
350 int ticketcount(void) {
351   struct proc *p;
352   int TICKET_TOTAL = 0;
353   for (p = ptable.proc; p < &ptable.proc[NPROC]; p++) {
354     if (p->state == RUNNABLE) { //VERIFICA SE O PROCESSO É RUNNABLE
355       TICKET_TOTAL += p->tickets; //SOMA TODOS OS TICKETS EM TICKET_COUNT
356   }
357   return TICKET_TOTAL; //RETORNA O VALOR FINAL
358 }
```

Utilizada para contar e retornar o número de tickets para o sorteio

Por fim, a alteração principal foi feita na função scheduler() do sistema Xv6. o escalonador do Xv6 originalmente realiza uma busca circular pela tabela de processos e executa o primeiro processo em estado RUNNABLE encontrado. A modificação na

função faz com que o escalonador passe a ser do tipo loteria, ele usa o número de tickets dos processos prontos para fazer a busca dado por ticketcount() e o número randômico gerado pela função rand().

```
void
scheduler(void)
{
  struct proc *p;
  struct cpu *c = mycpu();
  c->proc = 0;

  int vencedor=1;//variavel que recerá o número aleatório da função rand
  int numtickets;//variavel que recerá o número de tickets da função ticketcount

  for(;;){
    // Enable interrupts on this processor.
    sti();

    // Loop over process table looking for process to run.
    acquire(&ptable.lock);

////////////////////////////////////////////////////////////////////////////////////////////
    numtickets = ticketcount();//passa o numero de tickets do sorteio para o escalonador
    if(numtickets > 0){
      vencedor = rand(numtickets);//chama a função rand usando o número de tickets RUNNABLE como semente

      if(vencedor < 0){//re o rand retornar negativo, multiplica por -1
        vencedor = vencedor * -1;
      }
      if(numtickets < vencedor){//se o numero do sorteado for maior que o numero de tickets, feito outra
      valor
        vencedor = vencedor % numtickets;
      }
  //escalonador
      for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){//semelhante ao original
          if (p->state == RUNNABLE) {//verifica se o processo é runnable
            vencedor = vencedor - p->tickets;// se encoutrou um processo, então desconta o valor de ticke
          }
          if(p->state != RUNNABLE || vencedor >= 0) {//encontra o ticket vencedor quando o valor de vence
            continue;
          }
```

Escalonador por loteria

O novo escalonador chama as funções criadas e com base no número de tickets disponíveis para o sorteio e o número randômico, realiza uma busca linear na tabela de processos verificando os processos em estado RUNNABLE, e então desconta do número sorteado de tickets o valor de tickets do processo atual. Isso faz com que quando o número do vencedor chegar em 0, a busca encontrou o processo vencedor. A partir dai, a busca é quebrada e o escalonador muda o estado do processo para RUNNING fazendo com que o vencedor seja executado.

5. Teste

Para realizar os testes com o escalonador, foi anexado junto ao sistema, pelo arquivo ”Makefile”, um programa que testa a criação de um simples processo e um programa que cria 10 processos, atribui valores diferentes de tickets para eles, e então o escalonador faz o sorteio até que todos sejam executados. Com o teste é possível notar que processos com

um número muito baixo de tickets tendem a serem executados no final, porém isso não é regra. Fazendo com que se crie prioridades de execução, mas se mantenha a característica principal do escalonador que é o sorteio.

6. 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 a matemática para gerar o fator sorte para o processo e ainda conseguindo criar uma forma de implementar prioridades nos processos, de modo que ao se criar um processo com maior prioridade, basta fornecer ao processo um maior número de tickets, e assim ele terá maior probabilidade de ser sorteado.

Conclui-se que o algoritmo de escalonamento por loteria nem sempre é o melhor para todos os casos, pois cada sistema tende a precisar de um escalonamento dedicado as suas necessidades. Porém, o escalonador por loteria, consegue suprir com qualidade dois pontos interessantes para um sistema operacional, executar múltiplos processos e criar níveis de prioridade.

Referências

David Petrou, John W. Milford, G. A. G. (1999). *Implementing Lottery Scheduling: Matching the Specializations in Traditional Schedulers*. USENIX.

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