Implementação de Escalonador por Loteria no Sistema Operacional Didático Xv6

Felipe Chabatura Neto[1]

[1]Departamento de Ciência da Computação - Universidade Federal da Fronteira Sul (UFFS)
Chapecó – SC – Brasil

felipechabat@gmail.com

Abstract. *This article describes the planning and implementation of the lottery scheduling algorithm in the didactic operational system Xv6, reporting the process since the operational system analysis till the new scheduler test phase.*

Resumo. *Este artigo descreve o planejamento e implementação do escalonador de processos por loteria (lottery scheduling) no sistema operacional didático Xv6, relatando o processo desde a análise do sistema operacional até a fase de testes do novo escalonador.*

1. Introdução

O escalonador é o componente do sistema operacional responsável por decidir quais processos são os próximos a utilizarem o processador para serem executados. Este trabalho consiste na implementação de um escalonador por loteria no sistema operacional didático Xv6.

1.1. O Xv6

O Xv6 é um sistema operacional didático desenvolvido em 2006 no curso de sistemas operacionais do MIT (*Massachusetts Institute of Technology*). O sistema é baseado no Unix Versão 6 e foi escrito na linguagem C ANSI [2].

1.2. Escalonamento por loteria

Escalonamento por loteria é um algoritmo de escalonamento de processos probabilístico. Os processos recebem números de bilhetes em quantidade equivalente ao nível de prioridade que cada um possui. A seguir, o escalonador realiza o sorteio de um ticket, o processo que possuir o ticket sorteado é o próximo a ser executado[4].

2. Planejamento e implementação

A primeira etapa se deu pela análise do sistema operacional Xv6, mais especificamente de seu escalonador, e do planejamento dos componentes que teriam que ser alterados para a implementação do novo escalonador.

O escalonador original do xv6 possui um funcionamento muito simples: Ele executa uma varredura linear pela tabela de processos, procurando o primeiro processo disponível para ser executado, o escalona para que ele inicie e reassume quando o processo transfere o controle de volta para o escalonador, que busca o próximo processo a ser executado.

As primeiras modificações feitas no código foram a criação de níveis de prioridade baseados em quantidade de bilhetes, de acordo com o método de escalonamento por loteria. Foram definidos três níveis de prioridade: *LOW*(5 bilhetes), *MEDIUM*(10 bilhetes) e *HIGH*(20 bilhetes). A seguir, uma propriedade foi adicionada a estrutura que representa um processo (propriedade *tickets* na estrutura *struct proc* no arquivo *proc.h*), representando a quantia de bilhetes que cada processo possui para o sorteio.

O próximo passo, foi fazer com que um número de bilhetes fosse atribuído aos processos no momento de sua criação. Para isso, dois segmentos de código foram modificados: o primeiro deles é o da função responsável por criar o primeiro processo (função *userinit* no arquivo *proc.c*), o segundo, é o da função reponsável pela bifurcação dos processos (função *fork* no arquivo *proc.c*), por onde todos processos, exceto o primeiro, passam ao ser criados (todos os processos são inicialmente bifurcações de seus pais). Na função de bifurcação, além da atribuição de um número de bilhetes, um parâmetro foi adicionado, para que através da passagem de valor via parâmetro diferentes quantidades de bilhetes possam ser atribuídos a diferentes processos, possibilitando assim, diferentes níveis de prioridade. Adicionando um parâmetro a função de bifurcação, todas as chamadas desta função tiveram que ser alteradas, o valor 0 foi adicionado como parâmetro, indicando prioridade padrão. Desta forma, uma prioridade padrão foi definida aos processos de sistema: a média. Além disso, como a função de bifurcação é uma chamada do sistema, ela não aceita normalmente argumentos como parâmetro, para mudar isso, a função de chamada de sistema para bifurcação (função *sys_fork* no arquivo *sysproc.c*) foi alterada, de modo a receber o parâmetro passado e adicioná-lo na função de bifurcação.

O último passo foi alterar o código do própio escalonador (função *scheduler* no arquivo *proc.c*). A primeira coisa necessária para o sorteio de um bilhete, é saber o total de bilhetes destribuidos para os processos executáveis (com o estado *RUNNABLE*), para isso uma varredura linear na tabela de processos foi implementada, contando os bilhetes dos processos prontos para execução. O Sorteio do bilhete foi implementado da seguinte maneira: Um número pseudo-aleatório é gerado (utilizando um gerador congruente linear[3]) a partir de uma semente, então realiza-se a operação de módulo entre o número gerado e o total de bilhetes que os processos prontos possuem, gerando um número entre *0* e *NúmeroDeBilhetes - 1*. Após o sorteio, uma varredura é feita na tabela de processos, buscando o processo sorteado. Como a única informação armazenada é a quantia de bilhetes que cada processo tem, assume-se que o primeiro processo possui os bilhetes de número *0* até *qtdT(1) - 1*, o segundo de *qtdT(1)* até *qtdT(2) + qtdT(1) - 1* (onde *qtdT(i)* é a quantidade de bilhetes que o processo *i* possui) e assim sucessivamente. Desta forma, efetua-se a varredura até que o número de bilhetes do processo sendo considerado seja no máximo o valor acumulado da soma das quantidades de bilhetes dos processos testados anteriormente. Vale ressaltar que apenas os processos disponíveis para a execução são considerados nessa varredura.

```
if(usedTickets){
  //Generate a pseudo-random number
  randomNumber = (7 * randomNumber + 1337) % usedTickets;

  // Loop over process table looking for process to run.
  for(p = ptable.proc, tmp = 0; p < &ptable.proc[NPROC]; p++, tmp++){
    if(p->state != RUNNABLE)
      continue;
    if(p->tickets <= randomNumber){
      tmp += p->tickets;
      continue;
    }
```

Figura 1. Trecho do código do escalonador responsável pelo sorteio e busca do processo.

3. Testes

Após terminanda a implementação do escalanonador, um programa para testes começou a ser desenvolvido. De forma a acrescentar um novo programa no Xv6, alterações no seu arquivo de configurações de compilações tiveram de ser feitas, adicionando o novo programa (linhas 178 e 248 do arquivo *MakeFile*)[1].

O programa de testes funciona da seguinte maneira: a partir do processo do programa de testes um número determinado de processos filhos são criados, com diferentes níveis de prioridade. Os novos processos tem uma rotina extremamente básica, simplesmente percorrem um laço com o propósito de disperdiçarem o tempo necessário para se checar se os processos estão sendo escalonados de maneira coerente de acordo com as suas prioridades.

Com o fim de se obter uma métrica para o funcionamento do escalonador enquanto os processos filhos do programa de testes rodam por tempo indeterminado, foi adicionado uma nova propriedade na estrutura do processo (atributo *used*), a fim de medir quantas vezes o escalanonador escalonou aquele processo. Este atributo é incrementado a cada vez que o processo é escalonado. Para utilizar esta informação, uma alteração na função que lista informações dos processos em execução (função *procdump* no arquivo *proc.c* , a função pode ser acionada com o atalho de teclado *ctrl + p*) foi feita, adicionando informações acerca da quantidade de tickets e número de vezes que ele foi escalonado.

Desta forma, pode-se observa que a medida que os processos são escalonados, os com maior prioridade tendem a serem escalonados mais vezes, seguidos pelos de média e então baixa prioridade.

```
for(i = 0; i < NPROC_T; i++, num *= 2){
  if(num > 20) num = 5;
  printf(1, "Process %d is having a baby with priority %d!\n", getpid(), num);
  if(fork(num) == 0){
    timewaster();
    printf(1, "Process %d is over. Number of Tickets: %d\n", getpid(), num);
    exit();
  }
}
for(i = 0; i < NPROC_T; i++) wait(); //Wait for Children to exit
```

Figura 2. Trecho do código do programa de teste onde os processos filhos são criados.

4. Conclusões

A implementação do escalonamento por loteria no sistema operacional Xv6 foi bem sucedida. O escalonador se mostrou eficiente e coerente com sua proposta. Algumas otimizações ainda poderiam ter sido feitas mas foram deixadas de lado para simplificar a implementação: Manter um total de bilhetes em uso e atualiza-lo quando um novo processo é criado e encerrado extiguiria o custo linear da varredura que faz a soma dos bilhetes de processos prontos, além disso uma estrutura de árvore poderia ter sido implementada para busca do processo sorteado, diminuindo ainda mais o seu custo.

Referências

[1] ”Adding a User Program to xv6”. http://recolog.blogspot.com.br/2016/03/adding-user-program-to-xv6.html. [*fetch(); decode(); execute();*]. Acessado em Maio de 2017.

[2] ”Xv6, a simple Unix-like teaching operating system”. https://pdos.csail.mit.edu/6.828/2012/xv6.html. Acessado em Maio de 2017.

[3] Hull, T. E.; Dobell, A. R. (1962-01-01). ”Random Number Generators”

[4] Weihl E. William, Waldspurger A. Carl. Lottery Scheduling: Flexible Proportional-Share Resource Management [*The 1994 Operating Systems Design and Implementation conference (OSDI ’94). November, 1994. Monterey, California*].