Implementação do Escalonamento por Loteria no Sistema Operacional Xv6

Nicholas S. Brutti[1], **Ricardo A. Müller**[1]

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

{nicholassbrutti,ricardoam0908}@gmail.com

Abstract. *This article describes the details of the implementation process of a lottery process scheduler, in a didactic operating system Unix-like (Xv6).*

Resumo. *Este artigo descreve os detalhes do processo de implementação de um escalonador de processos por loteria, em um sistema operacional didático Unix-like (Xv6).*

1. Introdução

O sistema operacional é um software responsável por gerenciar diversos recursos (i.e. processador, discos, memória principal, dispositivos de E/S) com a finalidade de apresentar um modelo computacional simples e que facilite ao programador e usuário final em suas atividades. Como o sistema operacional é a peça mais básica de software ele opera em modo núcleo, ou seja, possui acesso completo a todo o hardware. O restante opera em modo usuário, na qual o acesso é restrito a um subconjunto de instruções [Tanenbaum 2010]. As requisições de acesso a recursos privilegiados é efetuada através de chamadas de sistema (***system call***).

Como em computadores multiprogramados existem diversos processos e *threads* disputando acesso à CPU, é preciso que exista um critério de escolha para determinar qual executar. Nesse contexto, existe o escalonador. Atualmente, existem vários algoritmos de escalonamento, porém, esse estudo está voltado somente para o escalonador por loteria.

2. Escalonamento por loteria

O escalonamento por loteria consiste em atribuir a cada processo uma quantidade de bilhetes, que em outras palavras, significa a prioridade do processo, e então é realizado um sorteio. Quanto maior o número de bilhetes maior a probabilidade de ser sorteado e por consequência ser executado. Essa abordagem é dita probabilística [Tanenbaum 2010].

3. O Xv6

Trata-se de um sistema operacional escrito na linguagem C desenvolvido no MIT *(Massachusetts Institute of Technology)*. O Xv6 é uma implementação na arquitetura Intel x86 baseada no antigo Unix 6 [Cox et al. 2014].

O escalonamento no Xv6 por padrão utiliza o algoritmo *round-robin*, que basicamente, percorre circularmente um vetor de processos (*ptable.proc*) até encontrar um processo que esteja no estado *RUNNABLE* (pronto), e então esse processo é alterado para o estado *RUNNING* (executando) e ele parte para execução. A cada processo é atribuído um

intervalo de tempo (*quantum*), no qual ele é permitido executar. Caso o processo exceda seu tempo ele é bloqueado e a CPU sofrerá preempção para selecionar outro processo. É possível notar que nesse tipo de escalonamento não há como estabelecer prioridade. A próxima seção apresenta a implementação do escalonador por loteria que tem como diferencial a definição da prioridade do processo.

4. Implementação

Como no escalonamento por loteria cada processo possui uma quantidade de bilhetes, foi preciso adicionar essa informação na estrutura do processo. Então, no arquivo *proc.h* foi adicionado uma variável *tickets* para armazenar esse valor do tipo inteiro. Também foram criadas algumas macros no arquivo *param.h* para definição do número máximo (*MAXTICKET* = 4096) e mínimo (*MINTICKET* = 0) de bilhetes, e também dos níveis de prioridade: (*LOWPRIOR* = 16), (*MEDPRIOR* = 64), (*HIGHPRIOR* = 2048).

Em seguida foi preciso alterar a chamada de sistema *fork()* responsável pela criação de processos e a função *allocproc()* ambas do arquivo *proc.c*. Pois, agora no momento da criação de um processo é preciso que seja informado qual a quantidade de bilhetes que o processo filho irá possuir. Ficou definido que a função recebe um parâmetro do tipo inteiro que representa a quantidade de bilhetes. Isso resultou em algumas alterações no arquivo *sysproc.c* na função *sys_fork()*, além de todas as chamadas *fork()* que antes não recebiam parâmetro. Por padrão os processos iniciados com o sistema (i.e. *shell*, *init*) recebem prioridade média.

```
int sys_fork(void) {
  int tickets;
  if(argint (0, &tickets) < 0)
    return -1;
  return fork(tickets);
}
```

Listing 1. Função *sys_fork()* do arquivo *sysproc.c*

Então em suma, após a execução da requisição *fork(quantidade_bilhetes)*, a função *allocproc(quantidade_bilhetes)* é chamada para adicionar o valor na estrutura do processo. A partir do valor de *quantidade_bilhetes* é efetuado o tratamento caso seja preciso, por exemplo, se *quantidade_bilhetes* for maior que o limite máximo ou menor que o mínimo.

O Xv6 possui uma função *scheduler()* no arquivo *proc.c* que implementa o escalonador. Foi preciso intervir na métrica utilizada para escolha do processo, agora leva-se em consideração a quantidade bilhetes do processo. Para realização do sorteio, como o Xv6 não apresenta por padrão algumas bibliotecas do C, por exemplo, a biblioteca *time.h* que já contém uma função de geração de números randômicos, foi preciso criar a função *lotteryRand(int n)* no arquivo *proc.c* com esse propósito. Optou-se pelo algoritmo Xorshift, um gerador de números pseudo-aleatórios proposto por George Marsaglia [Wikipedia 2006].

```
//function to generate a "random" number
int lotteryRand (int num){
    if (num <= 1) return 1;
```

```
    static int z1 = 12345;
    static int z2 = 12345;
    static int z3 = 12345;
    static int z4 = 12345;

    int b = (((z1 << 6) ^ z1) >> 13);
    z1 = (((z1 & 4294967294) << 18) ^ b);

    b = (((z2 << 2) ^ z2) >> 27);
    z2 = (((z2 & 4294967288) << 2) ^ b);

    b = (((z3 << 13) ^ z2) >> 21);
    z3 = (((z3 & 4294967280) << 7) ^ b);

    b = (((z4 << 3) ^ z4) >> 12);
    z4 = (((z4 & 4294967168) << 13) ^ b);

    int rand = ((z1 ^ z2 ^ z3 ^ z4) % num);

    if(rand < 0) rand = rand * -1;

    if(rand == 0) return 1;

    return rand;
}
```

Listing 2. Detalhes da função *lotteryRand(int)* do arquivo *proc.c*

Importante ressaltar que a função *lotteryRand(int)* recebe um inteiro por parâmetro que representa o somatório de bilhetes distribuídos entre todos os processos, que estão em estado *RUNNABLE*. Essa variável global é chamada de *numTickets*. Ela é incrementada toda a vez que é solicitada uma chamada de sistema do tipo *fork(int)* e na chamada *wakeup1(void *)*, que é quando o processo é acordado. É decrementada quando o processo *p* que está em estado *RUNNING* termina tanto de forma natural *exit()* quanto de forma abrupta *kill()*, ou quando um processo é colocado no estado *SLEEP* (bloqueado). Também o valor do *numTickets* é incrementado quando o processo *userinit* é instanciado, caso contrário ele não sera nunca executado.

Então o escalonador após sortear um número, percorre a lista de processos e busca um processo que esteja em estado *RUNNABLE* e que tenha o bilhete correspondente ao número randômico gerado.

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

    // Raffle a ticket number
    random = lotteryRand(numTickets);

    // Aux to decide to which process the ticket belong
    auxTickets = 0;
```

```
    acquire(&ptable.lock);

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

      // Find the process that have the ticket that correspond
         to the random number
      if((p->tickets + auxTickets) < random) {
        auxTickets += p->tickets;
        continue;
      }

      // Switch to chosen process.
      ...
}
```

Listing 3. Detalhes da função *scheduler()* do arquivo *proc.c*

5. Análise:

Para observar o desempenho do algoritmo a fase final foi dedicada a criação de arquivos de teste. Para inserir esses arquivos no Xv6 foi preciso adicioná-los no *Makefile* do projeto. Dessa forma, eles são unidos no momento da compilação.

Para melhor exibição das informações foram realizadas duas alterações no código do Xv6. A primeira foi a implementação da chamada de sistema *gettickets()* que dado um processo retorna a quantidade de tickets do mesmo. Outra alteração foi na função *procdump()* do arquivo *proc.c*, agora ao pressionar (CTRL+P) também é possível visualizar a quantidade de *tickets* dos processos.

6. Conclusão

Pode-se concluir através da análise que o algoritmo de escalonamento por loteria cumpriu com seu objetivo, garantiu que os processos com maior quantidade de bilhetes tenham prioridade de acesso à CPU. Porém, todos em algum momento serão sorteados. Essa condição é importante porque elimina a ocorrência de morte por inanição (*starvation*).

Referências

Cox, R., Kaashoek, F., and Morris, R. (2014). xv6 a simple, unix-like teaching operating system.

Tanenbaum, A. S. (2010). *Modern operating systems*. Pearson Prentice Hall, 3th edition.

Wikipedia (2006). Xorshift.