Implementação e Análise do Processo de Escalonamento no Estilo Loteria(Lottery Scheduling) no XV6

Adriel Schmitz J. de Paula[1], **Maicon Brandão**[2]

[1]Universidade Federal da Fronteira Sul (UFFS)

[2]Curso de Ciência da Computação – Chapecó, SC – Brasil

adrielschmitz10@gmail.com, maincon.brandao@gmail.com

Abstract. *This meta-article aims to explain the implementation of a lottery type scheduler in xv6. After a brief explanation of xv6 and how the scheduler policy works, we use function fragments that have been changed and / or created. Examples and measurement metrics are also part of the methods used for a better understanding of the subject. The article is all formatted according to the norms of the SBC and the research was done, the great part, using the files available in the references of xv6.*

Resumo. *Este meta-artigo tem por objetivo explanar a implementação de um escalonador do tipo loteria no xv6. Após uma breve explicação sobre o xv6 e como funciona a política do escalonador, utiliza-se de fragmentos de códigos das funções que foram alteradas e/ou criadas. Exemplos e méttricas de medidas também fazem parte dos métodos usados para um melhor entendimento acerca do assunto. O artigo é todo formatado segundo as normas da SBC e a pesquisa foi feita, a grande parte, utilizando os arquivos disponibilizados nas referências do xv6.*

1. XV6

O xv6 é uma versão moderna do Sixth Edition Unix para sistemas com multiprocessadores, baseado na plataforma Intel x86, desenvolvido pelo curso de Engenharia de Sistemas Operacionais do MIT, no ano de 2006. Pode-se dizer que o xv6 é filho do v6, este todo feito em assembly, já o xv6 foi recriado em C ANSI, deixando seu aprendizado mais didático. Aliás, fins pedagógicos é o principal objetivo do xv6. **one**-page texts.

2. Escalonamento de Processos Padrão do XV6

Por padrão, o xv6 usa um escalonador no formato Round-robin convencional, ou seja, o escalonador tem por objetivo percorrer uma tabela de processos, de forma circular, a procura de um processo pronto para a execussão(i.e estado RUNNABLE). Após encontrar um processo pronto, é feita a seleção para execussão com uma fatia de tempo(Quantum) padrão para todos. Como o XV6 trabalha com múltiplas CPU's e cada CPU tem seu próprio escalonador, quando um escalonador está mexendo na tabela de processos, a tabela fica bloqueada para que não ocorra situações de impasse, garantindo a integridade de alterações na tabela. A seguir será apresentado a propósta do escalonar que foi implementado.

2.1. Loterry Scheduling

No escalonador por loteria quando um processo é criado ele recebe uma quantidade de bilhetes, após o sorteio de um número aleatório, o processo que detêm o determinado bilhete é executado na vez. Caso queira dar prioridade a um processo, atribui a ele mais bilhetes, com isso, a chance desse processo ser sorteado é, obrigatóriamente, maior. Esse tipo de escalonador é uma boa tecnica pois previne processos de caírem em inanição, mesmo que ele tenha poucos bilhetes, em algum instante ele será sorteado.

3. Estrutura dos Dados

Para ter sucesso na implementação do escalonador, foi preciso a alteração dos seguintes arquivos no código fonte do xv6: usys.s, sysproc.c, syscall.h, syscall.c, user.h, proc.h, proc.c, defs.h e por fim no arquivo MakeFile. Para os testes criou-se o arquivo chamado teste.c e teste.h. Ambos serão explanados a seguir.

3.1. Alterações

A seguir as alteraoes realizadas no arquivo **proc.c**

```
int fork(){
  return forkT(MedioTickets);
}

int forkMinima(){
  return forkT(MinTickets);
}

int forkBaixa(){
  return forkT(BaixoTickets);
}

int forkMedia(){
  return forkT(MedioTickets);
}

int forkAlta(){
  return forkT(AltoTickets);
}

int forkCritica(){
  return forkT(CritioTickets);
}
```

A criação das fork'k variados foi necessário para se ter o controle de distribuição dos tickets para cada processo. Dentro da função fork é feito a chamada da função forkT, essa função recebe um parâmetro(inteiro) que é o número de tickets a ser atribuído ao processo. Para que essa funções funcionasse, foi necessário declará-las nos arquivos syscall.h, sysproc.c, user.h e no defs.h.

Como o xv6 não tem a função rand() por padrão, criou-se uma função para exercer essa tarefa

```
int aux = 1;
int random(){
    aux = aux * 1564425 + 1013909223;
    if(aux < 0)
      return (aux * -1);

    return aux;
}
```

3.2. Função scheduler

```
void scheduler(void){
  struct proc *p;
  struct cpu *c = mycpu();
  c->proc = 0;
  int NTickets, premiado;
  for(;;){
    // Enable interrupts on this processor.
    sti();

    // Loop over process table looking for process to run.
    acquire(&ptable.lock);
    NTickets = 0;
    for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){//conta o numero
      if(p->state == RUNNABLE)
        NTickets += p->tickets;
    }

    if(NTickets > 0){
      premiado = random() % NTickets;
      for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){//procura na t
        if(p->state == RUNNABLE){
          premiado = premiado - p->tickets;
          if(premiado < p->tickets){
            break;
          }
        }
      }
      c->proc = p;
      switchuvm(p);
      p->state = RUNNING;
      swtch(&(c->scheduler), p->context);
      switchkvm();
      // Process is done running for now.
      // It should have changed its p->state before coming back.
```

```
      c->proc = 0;
    }
    release(&ptable.lock);
  }
}
```

A função scheduler é onde funciona o escalonador no xv6, nela ocorreu uma alteração significativa. Criou-se doi inteiros para se ter o crontole do total de tickes e o ticket premiado.No segundo for é feita uma varredura na tabela de processos buscando processos que estão no estado RUNNABLE, caso encontre é somado o número de tickests que ele detém na variável NTickets.Após isso faz-se o sorteio de um bilhete e então é feita a procura do processo que possui aquele bilhete. Uma vez encontrado, ele é executado.
Foi preciso alterar também na estrutura proc, do arquivo proc.h. Nela foi incrementada uma variável(inteiro) chamada tickets, que terá a quantidade de tickets que cada processo criado possui.

4. Testes de Avaliação

Para testar e avaliar o novo escalonador, criou-se um arquivo chamado teste.c. Sua função é realizar diversos testes, verificando a integridade e funcionamento do novo escalonador.

```
#include "teste.h"

#define MAX 1123456789

int test(int prioridade);
void trabalho();

int main(void){
    for(;;){
        test(1); // prioridade Minima
        test(2); // prioridade Baixa
        test(3); // prioridade Média
        test(4); // prioridade Alta
        test(5); // prioridade Extrema
    }
    exit();
}

void trabalho(){
    int i, x=0;

    // TRABALHO QUALQUER PARA UM PROCESSO
    for(i=0; i<MAX; i++)
        x++;
    for(i=0; i<MAX; i++)
        x--;
    for(i=0; i<MAX; i++)
```

```
        x++;
    for(i=0; i<MAX; i++)
        x--;
}
int test(int prioridade){
    int pid;
    switch(prioridade){
        case 1:
            pid = forkMinima();
            break;
        case 2:
            pid = forkBaixa();
            break;
        case 3:
            pid = forkMedia();
            break;
        case 4:
            pid = forkAlta();
            break;
        case 5:
            pid = forkCritica();
            break;
        default:
            pid = -1;
            break;
    }
    if(pid == 0){
        trabalho();
        exit();
    }
    return 0;
}
```

A main desse arquivo possui um for que nele chama-se cinco vezes a função test. Nela é passado como parâmetro números de 1 a 5. A função test recebe esse parâmetro e o trata no swith case. Por exemplo: Caso seja passado 1 como parâmetro, entrará no primeiro case e, consequentemente o pid receberá o retorno da função forkMinima e, criará processos com o mínimo de tickets, e assim sucessivamente. Dependendo da combinação da main, é possível criar processos com diferentes prioridades. A função trabalho tem como único objetivo dar algum trabalho qualquer aos processos criados.

5. Resultados Obtidos

```
Processo sorteado: Id: 4 Ticketis: 10
Processo sorteado: Id: 4 Ticketis: 10
Processo sorteado: Id: 6 Ticketis: 40
Processo sorteado: Id: 7 Ticketis: 55
```

```
Processo sorteado: Id: 5 Ticketis: 25
Processo sorteado: Id: 6 Ticketis: 40
Processo sorteado: Id: 6 Ticketis: 40
Processo sorteado: Id: 5 Ticketis: 25
Processo sorteado: Id: 5 Ticketis: 25
Processo sorteado: Id: 6 Ticketis: 40
Processo sorteado: Id: 8 Ticketis: 70
Processo sorteado: Id: 6 Ticketis: 40
Processo sorteado: Id: 4 Ticketis: 10
Processo sorteado: Id: 7 Ticketis: 55
Processo sorteado: Id: 6 Ticketis: 40
Processo sorteado: Id: 8 Ticketis: 70
Processo sorteado: Id: 7 Ticketis: 55
Processo sorteado: Id: 6 Ticketis: 40
Processo sorteado: Id: 4 Ticketis: 10
Processo sorteado: Id: 6 Ticketis: 40
Processo sorteado: Id: 7 Ticketis: 55
Processo sorteado: Id: 7 Ticketis: 55
Processo sorteado: Id: 5 Ticketis: 25
Processo sorteado: Id: 6 Ticketis: 40 ...
```

A tabela acima mostra como os processos que foram escalonados. Nesse teste em questão foram criados 5 processos. Nota-se que o processo 4 com 10 bilhetes foi sorteado e em seguida o processo 6, que detém 40 bilhetes. Em seguida é a vez do processo 7, que possui 55 bilhetes e assim sucessivamente, o que demonstra a execussão correta sobre a politica de sorteio, por parte do escalonador.

6. Conclusão

Após as devidas alterações e a implementação do escalonador por loteria no xv6, é possível dizer que a maior dificuldade no trabalho foi entender como funciona o xv6. Houve a necessidade de um estudo cauteloso no código fonte para se saber onde e quais alterações deveriam ser feitas. Depois de alguns testes e análises, é válido concluir que a maior vantagem do método de Lottery Scheduling é poder atribuir prioridade aos processos, que no escalonador padrão do xv6 não se tem essa opção. No padrão do xv6 os processos são selecionados de forma sequencial, dando a entender que todos os processos tem a mesma prioridade mediante a CPU, porém, não é esse cenário que se tem no dia-a-dia.

7. Referências

Cox, R., K. F. and Morris, R. (2012a). Source code: Xv6 a simple, unix-like teaching operating system. Publishing Press.
Código do xv6 no GitHub do MIT: https://github.com/ahorn/xv6
Cartilha de Utilização do XV: https://pdos.csail.mit.edu/6.828/2012/xv6/xv6-rev7.pdf