Implementação de um escalonador passo largo (Stride Scheduler) no XV6

Adriel Schmitz Joaquim 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

Resumo. *Este meta-artigo tem por objetivo explanar a implementação de um escalonador do tipo passo largo no processador 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 métricas 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.*

Abstract. *This meta-article aims to explain the implementation of a wide-step scheduler in the XV6 processor. 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 measures 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 the 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 no Massachusetts Institute of Technology (MIT), em 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 desse processador.

2. Escalonamento no 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 execução(i.e estado RUNNABLE). Após encontrar um processo pronto, é feita a seleção para execução 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 proposta do escalonar que foi implementado.

3. Problema proposto

Implementar o escalonador de processos stride scheduling (escalonamento em passos largos) sobre a arquitetura do processador XV6.

4. Stride Scheduler

Pode-se dizer que o escalonador de ”passos largos”é semelhante ao escalonamento por loteria, nessa abordagem cada processo também recebe um número padrão de bilhetes (tickets). Porém, contudo, ao invés de utilizar um sorteio, com abordagem probabilística, calcula-se o passo (stride) de cada processo como sendo o resultado da soma de um valor constante pelo número de bilhetes do processo. Essa constante pode ser definida anteriormente ou usar a quantidade de tickets do processo como passo. Neste trabalho usou-se o resultado da divisão de um valor constante (e.g., 10.000) pelo número de bilhetes do processo para calcular o passo. Continuando com a explicação, nessa abordagem tem-se que cada processo inicia com uma determinada “passada inicial” igual a zero (0). O escalonador seleciona o processo com o menor valor de passada atual; portanto, inicialmente qualquer um dos processos, que estejam em estado RUNNABLE, podem ser selecionados (utilizou-se o menor número do PID como critério de desempate, caso aconteça). Uma vez selecionado, a passada do processo é incrementada com o valor da passada do processo. Esse tipo de escolha de processos é bem vinda pois evitam processos de caírem em estado de inanição (processos que são ”esquecidos”pelo escalonador e nunca são escolhidos), mesmo que algum processo tenha uma quantidade pequena de bilhetes, em algum momento ele será escolhido.

5. Estrutura dos Dados

Para que fosse possível a implementação do escalonador, foi preciso alterar os 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.

5.1. Alterações e Implementações

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 funções “fork’s” foi para se ter o controle de distribuição dos tickets que cada processo recebe. 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 essas funções funcionassem, foi necessário declará-las nos arquivos syscall.h, sysproc.c, user.h e no defs.h. Cada fork detêm uma certa quantidade de bilhete, é possível fazer diversas combinações ao criar os processos.
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;
}
```

5.2. Função scheduler

```
1 void scheduler(void){
2  struct proc *p;
3  struct proc *aux;
4  struct cpu *c = mycpu();
5  c->proc = 0;
6  int MINX;
7  for(;;){
8    MINX = 12345678;
9    // Enable interrupts on this processor.
10    sti();
11
12    // Loop over process table looking for process to run.
13    acquire(&ptable.lock);
14
15    for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
16        if(p->state == RUNNABLE && p->passada < MINX){
17            aux = p;
18            MINX = p->passada;
19        }
```

```
20    }
21
22    if(MINX != 12345678){
23      p = aux;
24      p->passada += p->stride;
25      c->proc = p;
26
27      switchuvm(p);
28      p->state = RUNNING;
29
30      swtch(&(c->scheduler), p->context);
31      switchkvm();
32
33      c->proc = 0;
34
35      if(p->pid > 3)
36         cprintf("Processo sorteado: Id: %d tickets: %d Stride: %d Pa
38    }
39
40    release(&ptable.lock);
41  }
42 }
```

A função scheduler, situada no arquivo proc.c é onde funciona o escalonador no xv6, nela ocorreu uma alteração significativa.
Houve também a necessidade de alteração na estrutura proc, do arquivo proc.h. Foram criadas três variáveis, tickets e stride ambas do tipo inteiro, e a variável passada, que é do tipo uint. A variável tickets terá a quantidade de bilhetes de cada processo criado, já a stride o passo de cada processo, por fim, a variável passada representa a passada (10000/N de Bilhetes) do processo.
Nessa função, criou-se uma variável chamada aux, que é do mesmo tipo do processo, na linha , tem-se a criação de uma variável do tipo inteiro (MINX), e é inicializada na linha 8 por um número, considerávelmente, grande, ela será usada para comparações futuras. Na linha 16, o if faz a comparação se o processo que está na tabela de processo está em estado RUNNABLE (pronto para ser sexecutado) e se a passada do processo é menor que o MINX. O primeiro processo sempre terá a passada menor que a variável pré setada, com isso, o MINX recebe a passada do processo da vez. Na linha 17, a variável aux, serve para guardar qual é o processo que detêm a menor passada. Após verificar e achar qual é o processo com a menor passada, faz uma verificação (if da linha 22), testando se o MINX recebeu outro valor, indicando que existe um processo com a passada menor e está pronto para ser estalonado. Na linha 23 a variável p recebe o auxiliar que detêm o processo a ser escaonado e atualiza sua passada, isso na linha 24. As linhas abaixo da 24 nao foram alteradas, e na 35 só tem um if para printar os processos que foram escalonados. E é isso que o escalonador faz.

6. 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 que 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.

7. Resultados Obtidos

7.1. Teste 1

```
Processo escolhido: Id: 4 tickets: 100 Stride: 100 Passada: 100
Processo escolhido: Id: 5 tickets: 250 Stride: 40 Passada: 40
Processo escolhido: Id: 4 tickets: 100 Stride: 100 Passada: 200
Processo escolhido: Id: 5 tickets: 250 Stride: 40 Passada: 80
Processo escolhido: Id: 4 tickets: 100 Stride: 100 Passada: 300
Processo escolhido: Id: 5 tickets: 250 Stride: 40 Passada: 120
Processo escolhido: Id: 6 tickets: 400 Stride: 25 Passada: 25
Processo escolhido: Id: 5 tickets: 250 Stride: 40 Passada: 160
Processo escolhido: Id: 6 tickets: 400 Stride: 25 Passada: 50
Processo escolhido: Id: 5 tickets: 250 Stride: 40 Passada: 200
Processo escolhido: Id: 6 tickets: 400 Stride: 25 Passada: 75
Processo escolhido: Id: 5 tickets: 250 Stride: 40 Passada: 240
Processo escolhido: Id: 6 tickets: 400 Stride: 25 Passada: 100
Processo escolhido: Id: 5 tickets: 250 Stride: 40 Passada: 280
Processo escolhido: Id: 6 tickets: 400 Stride: 25 Passada: 125
Processo escolhido: Id: 7 tickets: 550 Stride: 18 Passada: 18
Processo escolhido: Id: 6 tickets: 400 Stride: 25 Passada: 150
Processo escolhido: Id: 7 tickets: 550 Stride: 18 Passada: 36
```

```
Processo escolhido: Id: 6 tickets: 400 Stride: 25 Passada: 175
Processo escolhido: Id: 7 tickets: 550 Stride: 18 Passada: 54
Processo escolhido: Id: 6 tickets: 400 Stride: 25 Passada: 200
Processo escolhido: Id: 7 tickets: 550 Stride: 18 Passada: 72
Processo escolhido: Id: 6 tickets: 400 Stride: 25 Passada: 225
Processo escolhido: Id: 7 tickets: 550 Stride: 18 Passada: 90
Processo escolhido: Id: 6 tickets: 400 Stride: 25 Passada: 250
Processo escolhido: Id: 7 tickets: 550 Stride: 18 Passada: 108
Processo escolhido: Id: 6 tickets: 400 Stride: 25 Passada: 275
Processo escolhido: Id: 7 tickets: 550 Stride: 18 Passada: 126
Processo escolhido: Id: 7 tickets: 550 Stride: 18 Passada: 144
Processo escolhido: Id: 6 tickets: 400 Stride: 25 Passada: 300
Processo escolhido: Id: 7 tickets: 550 Stride: 18 Passada: 162
Processo escolhido: Id: 7 tickets: 550 Stride: 18 Passada: 180
Processo escolhido: Id: 5 tickets: 250 Stride: 40 Passada: 320
Processo escolhido: Id: 7 tickets: 550 Stride: 18 Passada: 198
Processo escolhido: Id: 8 tickets: 700 Stride: 14 Passada: 14
...
Processo escolhido: Id: 8 tickets: 700 Stride: 14 Passada: 84
Processo escolhido: Id: 7 tickets: 550 Stride: 18 Passada: 306
Processo escolhido: Id: 8 tickets: 700 Stride: 14 Passada: 98
Processo escolhido: Id: 4 tickets: 100 Stride: 100 Passada: 400
Processo escolhido: Id: 8 tickets: 700 Stride: 14 Passada: 112
Processo escolhido: Id: 6 tickets: 400 Stride: 25 Passada: 325
Processo escolhido: Id: 8 tickets: 700 Stride: 14 Passada: 126
Processo escolhido: Id: 7 tickets: 550 Stride: 18 Passada: 324
Processo escolhido: Id: 8 tickets: 700 Stride: 14 Passada: 140
Processo escolhido: Id: 5 tickets: 250 Stride: 40 Passada: 360
Processo escolhido: Id: 8 tickets: 700 Stride: 14 Passada: 154
Processo escolhido: Id: 7 tickets: 550 Stride: 18 Passada: 342
Processo escolhido: Id: 8 tickets: 700 Stride: 14 Passada: 168
Processo escolhido: Id: 6 tickets: 400 Stride: 25 Passada: 350
Processo escolhido: Id: 8 tickets: 700 Stride: 14 Passada: 182
Processo escolhido: Id: 7 tickets: 550 Stride: 18 Passada: 360
Processo escolhido: Id: 8 tickets: 700 Stride: 14 Passada: 196
Processo escolhido: Id: 6 tickets: 400 Stride: 25 Passada: 375
Processo escolhido: Id: 8 tickets: 700 Stride: 14 Passada: 210
Processo escolhido: Id: 5 tickets: 250 Stride: 40 Passada: 400
Processo escolhido: Id: 8 tickets: 700 Stride: 14 Passada: 224
```

A tabela acima mostra como os processos que foram escalonados. Nesse teste em questão foram criados 5 processos, cada um com um ”trabalho”infinito. Nota-se que o processo 4 com 100 bilhetes é o primeiro a ser escalonado tendo um stride de tamanho 100 e com a passada em 0, isso se da pois todos os processos iniciam com a passada em 0 e o critério de desempate é por ordem na lista de processos. Sendo executado em seguida, pela primeira vez tendo seu passo somado a sua passada que também fica em 100. Agora

a menor passada não pertence mais a ele, mas sim ao processo 5 que já foi criado com 250 tickets, tendo assim um stride de 40, que sera somado a sua passada a cada vez que ganhar a cpu. Usou a configuração, no arquivo de teste, da seguinte forma:

```
test(1); // prioridade Minima
test(2); // prioridade Baixa
test(3); // prioridade Média
test(4); // prioridade Alta
test(5); // prioridade Extrema
```

Essa configuração cria 5 processos com prioridades (numero de bilhetes) diferentes. teste(1) com 100 bilhetes, test(2) com 250, test(3) com 400, test(4) com 550 e por fim o test(5) com 700 bilhetes. É importante salientar que, caso o usuário queira alterar as prioridades (número de bilhetes) de cada processo, deve fazer a alteração no arquivo proc.h.
A partir do momento em que os 5 processos estão todos criados é que se pode observar melhor o desempenho do escalonador, como podemos ver no trecho a seguir:

```
...
Processo escolhido: Id: 8 tickets: 700 Stride: 14 Passada: 84
Processo escolhido: Id: 7 tickets: 550 Stride: 18 Passada: 306
Processo escolhido: Id: 8 tickets: 700 Stride: 14 Passada: 98
Processo escolhido: Id: 4 tickets: 100 Stride: 100 Passada: 400
Processo escolhido: Id: 8 tickets: 700 Stride: 14 Passada: 112
Processo escolhido: Id: 6 tickets: 400 Stride: 25 Passada: 325
Processo escolhido: Id: 8 tickets: 700 Stride: 14 Passada: 126
Processo escolhido: Id: 7 tickets: 550 Stride: 18 Passada: 324
Processo escolhido: Id: 8 tickets: 700 Stride: 14 Passada: 140
Processo escolhido: Id: 5 tickets: 250 Stride: 40 Passada: 360
Processo escolhido: Id: 8 tickets: 700 Stride: 14 Passada: 154
Processo escolhido: Id: 7 tickets: 550 Stride: 18 Passada: 342
Processo escolhido: Id: 8 tickets: 700 Stride: 14 Passada: 168
Processo escolhido: Id: 6 tickets: 400 Stride: 25 Passada: 350
Processo escolhido: Id: 8 tickets: 700 Stride: 14 Passada: 182
Processo escolhido: Id: 7 tickets: 550 Stride: 18 Passada: 360
Processo escolhido: Id: 8 tickets: 700 Stride: 14 Passada: 196
Processo escolhido: Id: 6 tickets: 400 Stride: 25 Passada: 375
Processo escolhido: Id: 8 tickets: 700 Stride: 14 Passada: 210
Processo escolhido: Id: 5 tickets: 250 Stride: 40 Passada: 400
Processo escolhido: Id: 8 tickets: 700 Stride: 14 Passada: 224
```

Tirando a parte de que os processos estejam com passadas iguais a 0 e estão sendo criados, o escalonador segue a abordagem conforme o esperado, escolhendo os processos com as menores passadas e incrementando seu valor ao final de sua escolha.

7.2. Teste 2

```
Processo escolhido: Id: 4 tickets: 100 Stride: 100 Passada: 100
Processo escolhido: Id: 5 tickets: 400 Stride: 25 Passada: 25
```

```
Processo escolhido: Id: 6 tickets: 700 Stride: 14 Passada: 14
Processo escolhido: Id: 5 tickets: 400 Stride: 25 Passada: 50
Processo escolhido: Id: 6 tickets: 700 Stride: 14 Passada: 28
Processo escolhido: Id: 5 tickets: 400 Stride: 25 Passada: 75
Processo escolhido: Id: 6 tickets: 700 Stride: 14 Passada: 42
Processo escolhido: Id: 5 tickets: 400 Stride: 25 Passada: 100
Processo escolhido: Id: 4 tickets: 100 Stride: 100 Passada: 200
Processo escolhido: Id: 6 tickets: 700 Stride: 14 Passada: 56
Processo escolhido: Id: 5 tickets: 400 Stride: 25 Passada: 125
Processo escolhido: Id: 6 tickets: 700 Stride: 14 Passada: 70
Processo escolhido: Id: 5 tickets: 400 Stride: 25 Passada: 150
Processo escolhido: Id: 6 tickets: 700 Stride: 14 Passada: 84
Processo escolhido: Id: 5 tickets: 400 Stride: 25 Passada: 175
Processo escolhido: Id: 6 tickets: 700 Stride: 14 Passada: 98
Processo escolhido: Id: 5 tickets: 400 Stride: 25 Passada: 200
...
```

O teste acima foi realizado criando 3 processos com prioridades distintas. O primeiro processo com prioridade mínima (100 bilhetes), o segundo com prioridade média (400 bilhetes) e o último com prioridade crítica (700 bilhetes). Nesse teste é possível ver com clareza o escalonador funcionando, como o processo 6 tem um total de tickets de 700, seu passo será o menor, consequentemente terá uma prioridade maior. Com isso, ele é chamado mais vezes pelo escalonador, exceto quando lhe alcançam. Percebe-se que o processo 4 é chamado poucas vezes, tendo em vista que sua prioridade é menor que as dos outros. Isso reitera o bom funcionamento do escalonador, mostrando que ele respeita as condições que lhe foram impostas.

8. Conclusão

Após as devidas alterações e a implementação do escalonador por loteria no xv6, é possível dizer que o maior desafio no trabalho foi entender como funciona o xv6, esmiuçar seu interior. O estudo cauteloso no código fonte para se saber onde e quais alterações deveriam ser feitas foi mais que necessário. Após alguns testes e análises, pode-se concluir que a maior vantagem do método de passos largos é poder atribuir prioridade aos processos e também o fato de impedir de caírem em inanição. No padrão do xv6 os processos são selecionados de forma sequencial, dando a entender que todos os processos têm a mesma prioridade mediante à CPU, porém, não é esse cenário que se tem no dia-a-dia. Outra grande dificuldade foi a ausência de chamadas básicas de um Sistema Operacional, tais como:(scanf, malloc e etc). Essas faltas limitaram a implementação, acarretando a não implementação de ideias a fim de deixar o escalonador mais eficiente em sua operação. Com isso, é válido salientar e, até parabenizar, os criadores do xv6, levando em conta sua enorme contribuição acadêmica, sendo uma ótima ferramenta de aprendizado em geral.

9. 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