Implementação do Algoritmo de Escalonamento por Passos Largos (stride scheduling) no XV6

Douglas Davy Breyer

[1]Ciência da Computação – Universidade Federal da Fronteira Sul (UFFS)
Caixa Postal 181 – Chapecó – SC – Brazil

douglasbreyer@gmail.com

Resumo.

Este trabalho tem como objetivo apresentar, demonstrar, explicar e implementar de forma fácil e breve o algoritmo de ”Passos Largos”para escalonadores de processos. No nosso caso utilizaremos o XV6 como base para tudo. Também, por final analisaremos pontos positivos e negativos se existentes entre o algoritmo padrão do XV6 para o stride scheduling.

1. Introdução

O Xv6 é um simples sistema operacional UNIX para fins didáticos. Este foi desenvolvido em 2006 pelo Massachusetts Institute of Technology - MIT. Foi desenvolvido para rodar em multiprocessadores baseados na arquitetura x86 da Intel. Por ser muito mais simples que um sistema completo como por exemplo o Ubuntu, ele é utilizado para ensino na maioria das instituições de ensino superior do mundo pelo fato de ser um sistema de código aberto e didático.

A princípio, o XV6 possui um vetor circular de 64 posições que nada mais são do que os processos. Esse vetor é varrido por um laço na intenção de encontrar um processo no estado de pronto. No momento em que é encontrado este é executado. Sendo assim, não existe separação (entre processos de usuário e processos de sistema por exemplo) nem diferenciação entre processos (alguns com maior prioridade que outros) o que mostra como o algoritmo default é simples e pouco utilizável na prática. Para contornar essa situação existem várias opções de algoritmos de escalonamento que se comportariam melhor dependendo do ambiente de utilização do sistema, como por exemplo o algoritmo de múltiplas fitas, SRT, FIFO entre outros. Sendo assim, este trabalho tem como principal finalidade implementar o escalonamento passos largos. O escalonamento por passos largos funciona buscando o processo com passada menor. Quanto maior o numero de tickets menor será a passada do processo e antes ele será processado. O algoritmo será abordado de forma mais aprofundada a seguir

2. O que é o Escalonamento

O escalonamento é a forma em que os processos são dispostos e como serão executados. Em computadores, para que um sistema seja fluido, eficaz e que possa retirar o máximo que o hardware pode oferecer se torna necessário que várias coisas aconteçam ao mesmo tempo, ou seja, vários processos executem ao mesmo tempo. Entretanto em vários momentos diversos processos estarão competido pela CPU. É nesse momento em que o escalonador se torna indispensável para escolher qual processo deve ganhar a CPU de forma coesa que gere ganho no desempenho e o não desperdício de hardware.

3. Implementação do Algoritmo de Passos Largos

Neste trabalho usaremos como base para implementação do algoritmo o sistema XV6. Ele pode ser baixado do repositório oficial no Github pelo link: https://github.com/mit-pdos/xv6-public. Entretanto nesse trabalho será utilizado o XV6 já modificado de acordo com as especificações do trabalho 1 da disciplina de Sistemas Operacionais que foi o escalonamento por loteria. Recomenda-se o uso de alguma distro Linux para o manuseio, uso e trabalho com o mesmo.

Os arquivos que receberam as maiores alterações, assim como no primeiro trabalho foram os arquivos proc.c e proc.h. Podemos ver abaixo a estrutura dos processos. Nesta vamos adicionar mais dois inteiros para conseguirmos fazer o controle das passadas e para tornar possível a implementação do escalonador.

```
// Per-process state
struct proc {
  uint sz;                     // Size of process memory (bytes)
  pde_t* pgdir;                // Page table
  char *kstack;                // Bottom of kernel stack for this process
  enum procstate state;        // Process state
  int pid;                     // Process ID
  struct proc *parent;         // Parent process
  struct trapframe *tf;        // Trap frame for current syscall
  struct context *context;     // swtch() here to run process
  void *chan;                  // If non-zero, sleeping on chan
  int killed;                  // If non-zero, have been killed
  struct file *ofile[NOFILE];  // Open files
  struct inode *cwd;           // Current directory
  char name[16];               // Process name (debugging)
  int passo;
  int passada;
};
```

No mesmo arquivo ainda é utilizado algumas constantes para controle dos tickets.

```
#define MAX_TICKETS 2000
#define MIN_TICKETS 25
#define DEFAULT_TICKETS 50
```

São definidos um valor máximo e mínimo de tickets. Também é definido um número de tickets default em q todos os processos iniciarão contendo. Os valores podem variar de acordo com o teste desejado.

No proc.c é onde ficam as funções de escalonamento e é nele que foram feitas as principais alterações. No momento da alocação dos processos está sendo definida o valor do passo e das passadas. É nessa função que é garantido que os processos receberam o passo condizente com o numero de tickets que eles possuirão. A variável passo do processo recebe um valor que é resultado da divisão de 10000 (numero aleatório que funcionou bem nos testes) pela numero de tickets. Ou seja, quanto maior o numero de tickets menor o passo.

```
  p->passo = 10000/stride;     // n qualquer que produz passo menor para valores maiores de ticktes
  p->passada = 0;
```

A partir do init são criados os outros processos. Para que eles sejam criados é necessário invocar a função fork(). Entretanto, essa função é void. Para que seja possível passar por parâmetro o numero de tickets essa função será alterada para receber como parâmetro um inteiro.

```
int fork(int stride){
  int i, pid;
  struct proc *np;
  if(!stride){
      stride = DEFAULT_TICKETS;
  } else if (stride<MIN_TICKETS)
      stride = MIN_TICKETS;
  else if (stride>MAX_TICKETS)
      stride = MAX_TICKETS;

  // Allocate process.
  if((np = allocproc(stride)) == 0)
    return -1;

  // Copy process state from p.
  if((np->pgdir = copyuvm(proc->pgdir, proc->sz)) == 0){
    kfree(np->kstack);
    np->kstack = 0;
    np->state = UNUSED;
    return -1;
  }
  np->sz = proc->sz;
  np->parent = proc;
  *np->tf = *proc->tf;

  // Clear %eax so that fork returns 0 in the child.
  np->tf->eax = 0;

  for(i = 0; i < NOFILE; i++)
    if(proc->ofile[i])
      np->ofile[i] = filedup(proc->ofile[i]);
  np->cwd = idup(proc->cwd);

  safestrcpy(np->name, proc->name, sizeof(proc->name));

  pid = np->pid;
  printaProcessos(np, 1);

  // lock to force the compiler to emit the np->state write last.
  acquire(&ptable.lock);
  np->state = RUNNABLE;
  //WARNING: if use a variable to control tickets for all process, maybe need add here
  release(&ptable.lock);

  return pid;
}
```

Nesse momento é necessário a alteração em todos os demais arquivos que utilizam essa função. Pode-se utilizar de um editor de código como o sublime ou Atom para buscar dentro da pasta do projeto.

Dentro da mesma função é adicionado um controle básico dos tickets. Caso não seja passado como parâmetro o número de tickets, o processo recebe o número default. Se o processo recebeu um número menor do que o mínimo de tickets ele recebe o valor mínimo. Se o processo receber um valor acima do máximo de tickets definido nas constantes, o processo recebe o valor máximo. Também foi adicionada uma chamada para uma função implementada apenas para exibir a situação dos processos. Ela pode ser vista abaixo. O restante da função continua inalterada.

```
void printaProcessos(struct proc *p, int estado){
  if (estado){
    cprintf("Iniciando Id: %d Passo calculado %d\n", p->pid, p->passo);
```

```
  }else{
    cprintf("Finalizado - ID: %d Passo atual %d\n", p->pid,p->passada);
  }
}
```

Na função scheduler() também é necessário alterações. Como o próprio nome já diz, é nela onde é encontrado o algoritmo do escalonador. Para o escalonamento por passos é necessário pegar o processo com o menor passo, ou seja, o processo que tem maior numero de tickets. Para isso é utilizado um for passando pela tabela de processos do XV6 e pegando a menor passada. Após esse processo ser escolhido ele é executado A função scheduler() modificada pode ser vista abaixo:

```
void scheduler(void){
  struct proc *p, *min_proc;
  int min; //menor passada encontrada
  for(;;){
    sti();
    acquire(&ptable.lock);
    for(p = ptable.proc,min = 0; p < &ptable.proc[NPROC]; p++){
      if(p->state == RUNNABLE && (p->passada <= min || min == 0)){
          min = p->passada;
          min_proc = p;
      }
    }

    if(min_proc){
        min_proc->passada += min_proc->passo;
      proc = min_proc;
      switchuvm(min_proc);
      min_proc->state = RUNNING;
      swtch(&cpu->scheduler, proc->context);
      switchkvm();

      proc = 0;
    }
    release(&ptable.lock);
  }

}
```

O algoritmo já implementado é parecido com o original do XV6, entretanto agora são verificados os processos que estão prontos para serem executados na tabela de processos, contanto que o processo tenha a menor passada dentre os processos. Se essa condição for satisfeita, o processo ganhará um tempo na CPU.

4. Resultados

Para verificar e analisar os resultados foi criado um arquivo chamado teste.c. Neste, o sistema cria N processos, valor este definido em um define no topo do código. Esses processos são exatamente iguais, compostos por um for().

Os resultados obtidos foram os esperados. Pode facilmente ser visto que os processos que possuem mais tickets, ou seja, os que possuem menor passada são os que geralmente acabam primeiro. Segue abaixo a saída de um teste com 6 processos. Sendo 1000 o valor no for da chamada fork no arquivo de teste.

```
Iniciando Id: 4 Passo calculado 10
Iniciando Id: 5 Passo calculado 20
```

```
Iniciando Id: 6 Passo calculado 30
Iniciando Id: 7 Passo calculado 40
Iniciando Id: 8 Passo calculado 50
Iniciando Id: 9 Passo calculado 60
Finalizado - ID: 4 Passo atual 2450
Finalizado - ID: 5 Passo atual 5440
Finalizado - ID: 6 Passo atual 8100
Finalizado - ID: 7 Passo atual 10640
Finalizado - ID: 8 Passo atual 12900
Finalizado - ID: 9 Passo atual 15060
```

5. Conclusão

Na conclusão deste trabalho se mostrou que o escalonador é determinístico, os processos com mais tickets terminam primeiro, pois suas passadas são as primeiras escolhidas dentre as outras já que essas são as menores.