Trabalho II - Escalonamento em Passos Largos / Stride Scheduler

Eduardo Tonatto, Gabriel H. Moro

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

edtonatto@gmail.com, gabrielhmoro@gmail.com

1. Descrição do planejamento e implementação

Nesta seção do artigo/relatório será explicado como foi planejado e implementado o escalonador, bem como as modificações relevantes realizadas em cima do código-fonte original do xv6.

1.1. Planejamento do trabalho

Para a realização do trabalho foi necessário, primeiramente, para a implementação, identificar os arquivos, estruturas e funções em que se havia necessidade de modificação e quais seriam estas modificações, além das pesquisas necessárias para isto, majoritariamente usando o artigo Stride Scheduling: Deterministic Proportional- Share Resource Management[Waldspurger and Weihl. W. 1995]. Para o artigo houve o planejamento dos tópicos a serem escritos e da forma a serem estruturados.

1.2. Implementação e modificações

O escalonador em passos largos, ou em inglês, *stride scheduler*, funciona a partir da atribuição de bilhetes (*tickets*) aos processos, estes que serão usados para calcular o passo (*stride*) do processo como sendo

$$stride = Valor/N^{o}Tickets$$

, e com esta fórmula é observável que processos com maior quantidade de tickets terão menores passos e, consequentemente, maior prioridade. Quanto a quantia de tickets, cada processo deve receber no mínimo um bilhete até no máximo N bilhetes, e não deve ser permitido um processo receber zero bilhetes ou valores negativos de bilhetes, pois sendo zero ocorre uma divisão impossível e sendo negativo o passo do processo seria negativo, o que faria com que a passada diminuísse sempre que o processo fosse escalonado e, com isso, ele, eventualmente, sempre teria a menor passada e sempre seria o processo escalonado. Esta atribuição de bilhetes deve ocorrer assim que os processos iniciais são criados, e a cada chamada de fork realizada pelo sistema.

Os processos são iniciados com um valor para a quantia de tickets e a partir disso é calculado o passo do processo e a passada dele será iniciada com o valor da menor passada dentre os outros processos, ou como o valor do próprio passo, caso seja o primeiro processo. O escalonador funciona selecionando o processo com o menor valor de passada, selecionando este para ser escalonado e somando à passada o valor do passo do processo para, assim, novamente repetir o processo e selecionar o processo com a menor passada.

Segue abaixo as modificações relevantes realizadas, explicitando o nome do arquivo modificado e comentários referentes as alterações realizadas no código do xv6:

Arquivo: proc.h

```
// Definições dos números máximo e padrão para os tickets
2 - #define MAX_TICKETS 101
3 - #define DEFAULT_TICKETS 10
4 - #define VALOR_FIXO 1000

// Dentro da struct proc
56 - int tickets; // Nº de tickets do processo
57 - int chamado; // Nº de vezes que o processo
                     foi selecionado pelo escalonador
58 - int passo;
59 - int passada
```

Arquivo: proc.c

```
12 - int minimo = 112345678;    //Váriavel global contendo
                                   o valor da menor passada

//Na função scheaduler()
386 - acquire(&ptable.lock);

388 - for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
389 -   if(p->state != RUNNABLE)
390 -    continue;
391 -   if(p->passada <= minimo){
392 -       minimo = p->passada;
393 -       aux = p;
394 -   }
395 - }
396 - if(aux){
397 -   p = aux;
398 -   p->passada += p->passo;
399 -   p->chamado++;
400 -   c->proc = aux;
401 -   switchuvm(p);
402 -   p->state = RUNNING;
403 -   swtch(&(c->scheduler), p->context);
404 -   switchkvm();

406 -   c->proc = 0;
407 - }
408 - release(&ptable.lock);
```

Aqui se encontra o código do escalonador. O que se faz para se encontrar o processo com a menor passada é: percorre-se a lista de processos por completo, comparando um valor denominado ”minimo”que contém o valor da menor passada encontrada ou,

caso ainda não se tenha esse valor, [valor que tu colocou no mínimo]. Quando é encontrado um processo com a menor passada atual salva-se ele numa variável e o processo que se encontrar nesta variável ao final do laço de repetição que percorre a lista será o processo com a menor passada e, consequentemente, o processo que será escalonado. A complexidade deste código se dá por $O(n)$, pois sempre se percorre a lista de processos, que contem *n* processos, independentemente da posição do processo na tabela.

Arquivo: Makefile

```
177 - _teste\ // Inclusão do arquivo teste.c no make
250 - teste.c\ // Inclusão do arquivo teste.c no make
```

2. Testes para avaliação do funcionamento de desenvolvimento do escalonador

Para determinar o funcionamento do escalonador foram realizados testes da seguinte forma: houve a criação de um arquivo de testes, chamado de ”teste.c”, onde foi definida uma constante N representando a quantia de processos a serem criados, há um laço de repetição indo de n até N, onde n inicialmente vale zero, e dentro dessa repetição são chamados os N forks, sempre passando um valor que define a quantia de bilhetes do novo processo que será criado, esse número de bilhetes é controlado pela variável de nome ”lcg”, a variável que recebe o valor retornado pela função ”lcg_rand”que gera um valor aleatórios de bilhetes para o processo. Após gerados os N processos, é utilizado o comando Ctrl + P do QEMU, alterado dentro da função ”void procdump(void)”na linha 583, para mostrar na tela o PID, o estado e o nome do processo, bem como a sua quantia de bilhetes e quantas vezes foi chamado pelo escalonador.

O arquivo ”teste.c”utilizado para a realização dos testes de fork e escalonamento é o seguinte:

```
1 - #include "types.h"
2 - #include "stat.h"
3 - #include "user.h"
4 - #include "fs.h"

6 - #define N 10 // N é o numero de processos que serao criados

8 - unsigned int lcg = 3; // Variavel de controle da randomização
                             dos tickets

10 - unsigned int lcg_rand(unsigned int state) // Função de randomização
11 - {
12 -     state = ((unsigned int)state * 48271u) % 0x7fffffff;
13 -     return state;
14 - }

17 - void testfork(void){
18 -  int n, pid;

20 -   printf(1, "Executando testes\n");
```

```
22 -   for(n = 0; n < N; n++){
23 -     lcg = lcg_rand(lcg) % 101; // Sorteia um valor para
                                     definir a quantia
                                     de ticket do processo
                                     a ser criado
24 -     pid = fork(lcg);
25 -     if(pid < 0) break;
26 -     if(pid == 0){
27 -       for(;;);
28 -     }
29 -   }

31 -   if(n == N){
32 -     printf(1, "Fork foi chamado %d vezes!\n", N);
33 -   }

34 -   for(; n > 0; n--){
35 -     wait();
36 -   }

39 -   exit();
40 - }

42 - int main(void){
43 -   testfork();
44 -   exit();
45 - }
```

Foram realizados alguns testes durante a implementação do escalonador, estes são os resultados mais relevantes e as suas implicações no progresso da implementação:

OBS.: Os PIDs das tabelas começam em 4, pois não são considerados os dois processos de iniciação do xv6 e o processo chamado para criar os processos gerados para o teste.

Teste 1:

Primeiro minuto:

PID	BILHETES	PASSO	PASSADA	CHAMADAS
4	80	12	1428	118
5	46	21	1428	67
6	82	12	1428	118
7	32	31	1426	45
8	79	12	1428	118
9	53	18	1422	78
10	33	30	1440	47
11	72	13	1417	108
12	1	1000	2000	1
13	94	10	1420	141

Após 5 minutos:

PID	BILHETES	PASSO	PASSADA	CHAMADAS
4	80	12	57972	4830
5	46	21	57981	2760
6	82	12	57972	4830
7	32	31	57970	1869
8	79	12	57972	4830
9	53	18	57978	3220
10	33	30	57990	1932
11	72	13	57980	4459
12	1	1000	58000	57
13	94	10	57980	5797

Após 10 minutos:

PID	BILHETES	PASSO	PASSADA	CHAMADAS
4	80	12	131640	10969
5	46	21	112350	5349
6	82	12	112356	9362
7	32	31	112375	3624
8	79	12	112356	9362
9	53	18	112356	6241
10	33	30	112350	3744
11	72	13	112346	8641
12	1	1000	113000	112
13	94	10	112350	11234

Teste 2:

Primeiro minuto:

PID	BILHETES	PASSO	PASSADA	CHAMADAS
4	80	12	5316	442
5	46	21	5313	252
6	82	12	5316	442
7	32	31	5332	171
8	79	12	5316	442
9	53	18	5310	294
10	33	30	5310	176
11	72	13	5317	408
12	1	1000	6000	5
13	94	10	5310	530
14	49	20	5320	265
15	61	16	5312	331
16	78	12	5316	442
17	60	16	5312	331
18	85	11	5313	482
19	11	90	5310	58
20	24	41	5330	129
21	34	29	5336	183
22	65	15	5325	354
23	50	20	5320	265
24	54	18	5328	295
25	26	38	5320	139
26	20	50	5350	106
27	62	16	5312	331
28	71	14	5375	379
29	8	125	5324	42
30	45	22	5313	241
31	89	11	5313	482
32	84	11	5313	482
33	18	55	5335	96
34	76	13	5317	408
35	74	13	5317	408
36	88	11	5313	482
37	91	10	5320	531
38	70	14	5320	379
39	15	66	5346	80
40	97	10	5320	531
41	28	35	5320	151
42	6	166	5312	31
43	59	16	5312	331
44	92	10	5320	531
45	63	15	5325	354

PID	BILHETES	PASSO	PASSADA	CHAMADAS
46	64	15	5325	354
47	57	17	5321	312
48	5	200	5400	26
49	66	15	5325	354
50	43	23	5313	230
51	2	500	5500	10
52	87	11	5313	482
53	98	10	5320	531
54	21	47	5311	112
55	55	18	5328	295
56	19	52	5356	102
57	69	14	5320	379
58	22	45	5355	118
59	48	20	5320	265
60	68	14	5320	379
61	29	34	5338	156
62	100	10	5320	531
63	7	142	5396	37

Após 15 minutos:

PID	BILHETES	PASSO	PASSADA	CHAMADAS
4	80	12	30708	2558
5	46	21	30702	1461
6	82	12	30708	2558
7	32	31	30721	990
8	79	12	30708	2558
9	53	18	30708	1705
10	33	30	30720	1023
11	72	13	30706	2361
12	1	1000	31000	30
13	94	10	30700	3069
14	49	20	30720	1535
15	61	16	30704	1918
16	78	12	30708	2558
17	60	16	30704	1918
18	85	11	30701	2790
19	11	90	30780	341
20	24	41	30709	748
21	34	29	30711	1058
22	65	15	30705	2046
23	50	20	30720	1535
24	54	18	30708	1705
25	26	38	30704	807

PID	BILHETES	PASSO	PASSADA	CHAMADAS
26	20	50	30750	614
27	62	16	30704	1918
28	71	14	30702	2192
29	8	125	30750	245
30	45	22	30712	1395
31	89	11	30701	2790
32	84	11	30701	2790
33	18	55	30745	558
34	76	13	30706	2361
35	74	13	30706	2361
36	88	11	30701	2790
37	91	10	30710	3070
38	70	14	30702	2192
39	15	66	30756	465
40	97	10	30710	3070
41	28	35	30730	877
42	6	166	30710	184
43	59	16	30705	1918
44	92	10	30705	3070
45	63	15	30800	2046
46	64	15	30705	2046
47	57	17	30702	1805
48	5	200	30800	153
49	66	15	30705	2046
50	43	23	30705	1334
51	2	500	31000	61
52	87	11	30701	2790
53	98	10	30710	3070
54	21	47	30738	653
55	55	18	30708	1705
56	19	52	30732	590
57	69	14	30702	2192
58	22	45	30735	682
59	48	20	30720	1535
60	68	14	30702	2192
61	29	34	30702	902
62	100	10	30710	3070
63	7	142	30814	216

3. Conclusões

A partir da implementação do escalonador e dos testes realizados, chega-se a conclusão de que o escalonador em passos largos consegue realizar o escalonamento dos processos de forma ”justa”, ou seja, dando maior prioridade aos processos com maior quantidade de tickets. Isso se dá devido a forma como o escalonador funciona, escolhendo processos

não por sorteio, como o *Lottery Scheduler*, mas escolhendo aquele com a menor passada, que mais frequentemente é aquele(s) com o menor passo.

Referências

Waldspurger, C. A. and Weihl. W., E. (1995). Stride scheduling: Deterministic proportional- share resource management. Technical report.