DCC605 - Sistemas Operacionais – 2021/2
Escalonador por Prioridades

Nomes:
Otávio de Meira Lima 2019054900
Gabriel Torres Bolognani 2019054579
Erick Henrique Campolina da Silva 2016120309

1 - Introdução

O objetivo deste trabalho é desenvolver um escalonador por prioridades. Para a realização do trabalho, é preciso entender o funcionamento de um escalonamento em um sistema operacional e como os processos são executados. Nessa implementação, os processos possuem prioridades e o tempo de execução dentro de uma janela definida será distribuída para os processos de forma proporcional às suas prioridades.

Para a implementação do escalonador, foi necessário compreender como os processos são criados, onde são armazenados e como podem ser priorizados.

2 - Implementação

2.1 - Processos

Os processos estão modelados por uma estrutura de dados denominada **proc**, definida no arquivo proc.h. Foi necessário criar novos atributos para que seja possível priorizar os processos em uma tabela.

Os atributos implementados foram:

- priority: inteiro que armazena prioridade do processo. Sempre que um processo é iniciado, sua prioridade é 1.
- exec_time: tempo de execução do processo.
- expected_exec_time: tempo esperado de execução do processo. Serve para armazenar o tempo alocado durante uma janela de tempo.

```
35  enum procstate { UNUSED, EMBRYO, SLEEPING, RUNNABLE, RUNNING, ZOMBIE };
36
37  // Per-process state
38  struct proc {
39    uint sz;                     // Size of process memory (bytes)
40    pde_t* pgdir;                // Page table
41    char *kstack;                // Bottom of kernel stack for this process
42    enum procstate state;        // Process state
43    int pid;                     // Process ID
44    struct proc *parent;         // Parent process
45    struct trapframe *tf;        // Trap frame for current syscall
46    struct context *context;     // swtch() here to run process
47    void *chan;                  // If non-zero, sleeping on chan
48    int killed;                  // If non-zero, have been killed
49    struct file *ofile[NOFILE];  // Open files
50    struct inode *cwd;           // Current directory
51    char name[16];               // Process name (debugging)
52    int priority;
53    int exec_time;
54    int expected_exec_time;
55  };
```

2.2 - Tabela de Processos

A tabela de processos é onde os processos ficam armazenados. Eles podem possuir os seguintes status: `UNUSED, EMBRYO, SLEEPING, RUNNABLE, RUNNING, ZOMBIE`.
A tabela se enche de processos com PID igual a 0. É feita uma filtragem para manter a tabela limpa apenas com os processos ativos, logo a priorização não será afetada por processos inativos que não precisam de alocação de tempo de execução.

Processos	Prioridade	Estado do processo	Exec esperada
1	1	2	166
2	1	2	166
3	1	2	166
4	1	4	166
5	1	3	166
6	1	4	166

Na imagem acima, todos os processos possuem a mesma prioridade. Logo, todos terão o mesmo tempo de execução esperado.

2.3 - Janela de Execução

Foi definido que todos os processos serão executados dentro de uma janela de 10 segundos. No arquivo **trap.c** definimos que a cada 1000 ticks (10 segundos) deve-se chamar a função que atualiza o tempo de execução esperado baseado na prioridade dos processos e é impresso na tela a tabela de processos com seus respectivos PID, prioridade, Estado e tempo de execução esperado, como mostrado na imagem anterior.

2.4 - Cálculo do Tempo de Execução

No arquivo **proc.c**, foi implementado uma função denominada **recalcExecutionTime**. Sua execução consiste em percorrer toda a tabela de processos, filtrando processos com PID =

0 que geram lixo na tabela, somando suas prioridades e alocando o tempo de execução esperado dentro da janela de 10 segundos para o processo. Em um primeiro laço de repetição, são contabilizadas todas as prioridades e em um segundo laço, os tempos são alocados de acordo com suas prioridades.

2.5 - Priorizador | Chamada setprio

A chamada de sistema **setprio** tem como finalidade alterar a prioridade de um certo processo presente na tabela. Ela recebe como parâmetros um ID de processo e o novo valor de prioridade para o processo cujo ID foi selecionado. Por exemplo:
setprio 1 9
A chamada anterior seleciona o processo com PID igual a 1 e altera sua prioridade para 9. Dessa forma, o processo 1 irá receber tempo de execução proporcional à sua prioridade.

```
setprio 1 9
Iniciando 4 com prioridade 1.
$ Processos     Prioridade      Estado do processo      Exec esperada
        1               9               2               900
        2               1               2               100
setprio 1 1
Iniciando 5 com prioridade 1.
$ Processos     Prioridade      Estado do processo      Exec esperada
        1               1               2               500
        2               1               2               500
```

3 - Casos de Teste

3.1 - Instruções de uso

Para executar os testes, devemos compilar o programa com os seguintes comandos utilizados em sequência:

- make
- make qemu-nox

Se ocorrer como esperado, deve-se ter acessado o terminal do sistema operacional xv6.

3.2 - Programa de teste

A chamada de sistema **time**, implementada no trabalho prático passado, foi alterada para que sua execução crie dois processos por meio de duas utilizações do comando **fork()**, criando, dessa forma, dois processos com status RUNNING que preenchem a tabela de processos.

Podemos rodar o comando:

- time realtimetest

Dessa forma, quatro processos serão criados (dois deles são processos filhos da chamada time) e se manterão na tabela até que eles acabem.

```
Iniciando 5 com prioridade 1.
Iniciando 6 com prioridade 1.
Processos       Prioridade      Estado do processo      Exec esperada
        1               1               2               166
        2               1               2               166
        3               1               2               166
        4               1               3               166
        5               1               4               166
        6               1               4               166
Processos       Prioridade      Estado do processo      Exec esperada
        1               1               2               166
        2               1               2               166
        3               1               2               166
        4               1               4               166
        5               1               3               166
        6               1               4               166
%f
Tempo de execução do processo 4: 215.030 seconds
$ %f
zombie!
%f
zombie!
Processos       Prioridade      Estado do processo      Exec esperada
        1               1               2               500
        2               1               2               500
```

É possível perceber que, quanto mais processos com a mesma prioridade 1 presentes na tabela, menor será o tempo alocado para cada um. É possível priorizar algum processo por meio da chamada setprio, alterando sua prioridade e dando mais tempo de execução esperado.

```
Iniciando 2 com prioridade 1.
$ Processos     Prioridade      Estado do processo      Exec esperada
        1               1               2               500
        2               1               2               500
setprio 1 4
Iniciando 3 com prioridade 1.
$ setprio 2 6
Iniciando 4 com prioridade 1.
$ Processos     Prioridade      Estado do processo      Exec esperada
        1               4               2               400
        2               6               2               600
```

```
setprio 1 4
Iniciando 5 com prioridade 1.
$ setprio 2 3
Iniciando 6 com prioridade 1.
$ Processos     Prioridade      Estado do processo      Exec esperada
        1               4               2               571
        2               3               2               428
```

O tempo de execução é calculado dividindo a prioridade do processo pela soma das prioridades de todos os processos presentes na tabela.

4 - Conclusão

A implementação desse projeto possibilitou ao grupo entender como implementar um escalonador em um sistema operacional. Além disso, percebemos que o tratamento das traps é um ponto muito importante ao se realizar essa tarefa. Com isso, o grupo foi capaz de entender de forma prática o funcionamento do priorizador e do escalonamento em um sistema operacional.