Documentação de Trabalho Prático:
Sistemas Operacionais - Trabalho Prático 2

Victor Vieira de Melo
Departamento de Ciência da Computação – Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte – MG – Brazil
victorvmelo@ufmg.br

1. Escalonamento

a. Qual a política de escalonamento atual no xv6?

O xv6 atualmente utiliza do Round Robin como política de escalonamento de processos.

b. Quais processos essa política seleciona pra rodar?

O round robin seleciona processos com estado em *RUNNABLE*, a partir da ordem de chegada do processo.

c. O que acontece quando um processo retorna de uma tarefa de I/O?

Sempre que um processo volta de um I/O, ele chama o método *wakeup*, basicamente uma chamada de sistema que "acorda" o processo que estava dormindo.

d. O que acontece quando um processo é criado e o quão frequente o escalonamento acontece?

A cada tick um processo é preemptivo, o próximo processo a ser executado é avaliado pelo escalonador a partir da estrutura ptable.lock, sempre o próximo processo estará em estado de *RUNNABLE.*

2. Modificações

a. Processo de preempção

Dada a especificação do trabalho, foi mudada a forma como o escalonador realizava a preempção dos processos, de forma em que o processo de preempção ocorresse a cada 5 unidades de tempo da cpu, originalmente era realizado para cada unidade de tempo, para implementar isso bastou criar uma constante com o valor 5 e usá-la como um multiplicador na na função *lapicw*.

b. Alterando a prioridade de um processo

Como especificado na documentação do trabalho, foi criada uma função de nome *set_prio* que recebe um processo e um inteiro como parâmetro, esse inteiro varia entre 0 e 2, a função então valida a prioridade dada na entrada e altera a fila de prioridade do processo para a sua nova prioridade.

c. Política de escalonamento de processos

A política de escalonamento foi modificada para atender as especificações do TP, passamos a utilizar a técnica de escalonamento de filas multinível. Para tal, foram criadas 3 filas, cada uma delas responsáveis por enfileirar os processos em cada nível de prioridade, a fila 2, por exemplo, é responsável por armazenar os processos que têm como nível de prioridade 2, a fila 1 organiza os processos de nível 1 de prioridade e, por fim, a fila 0 guarda os processos de menor prioridade e de valor 0. Dentro da função *allocproc* é definido então a prioridade inicial de um processo como 2 e o processo é enfileirado na fila 2, com a chamada de sistema *set_prio* podemos redefinir a prioridade desse processo e sua fila de prioridade também é alterada. Por fim, a função *scheduler* agora executa os processos respeitando, respectivamente, a ordem das filas 2, 1 e 0.

d. Aging

A fim de evitar a inanição com a nova política de escalonamento, foi criado o método *aging*. Para tal, foram criadas 2 constantes *ZERO_TO_ONE* e *ONE_TO_TWO*, que armazenam a informação de quantos ticks um processo tolera ficar ocioso em uma fila de baixa prioridade. Então, os atributos *ctime*, *retime*, *rutime*, e *stime* foram declaradas no TAD *proc*, tal como a especificação do trabalho expôs. A implementação em si da função *aging* consiste de uma checagem da atual prioridade do processo, caso seja 0 e seu *retime* for maior ou igual a *ZERO_TO_ONE*, o processo terá sua prioridade reajustada para o nível 1 de prioridade, o mesmo ocorre para processos dentro da fila 1 de prioridade, caso seu *retime* seja maior ou igual ao *ONE_TO_TWO*, a nova prioridade do processo agora é dois.

3. Testes

a. Levantamento de dados

A fim de levantar dados para comparar resultados obtidos a partir dessa nova política de escalonamento, foi criada a função *wait2* que lê toda a tabela de processos até encontrar um filho que se encontre em um estado de *ZOMBIE*, quando encontrado, as informações de execução do filho são armazenados e então, a função *wait2* encerra sua execução.

b. Programa de testes

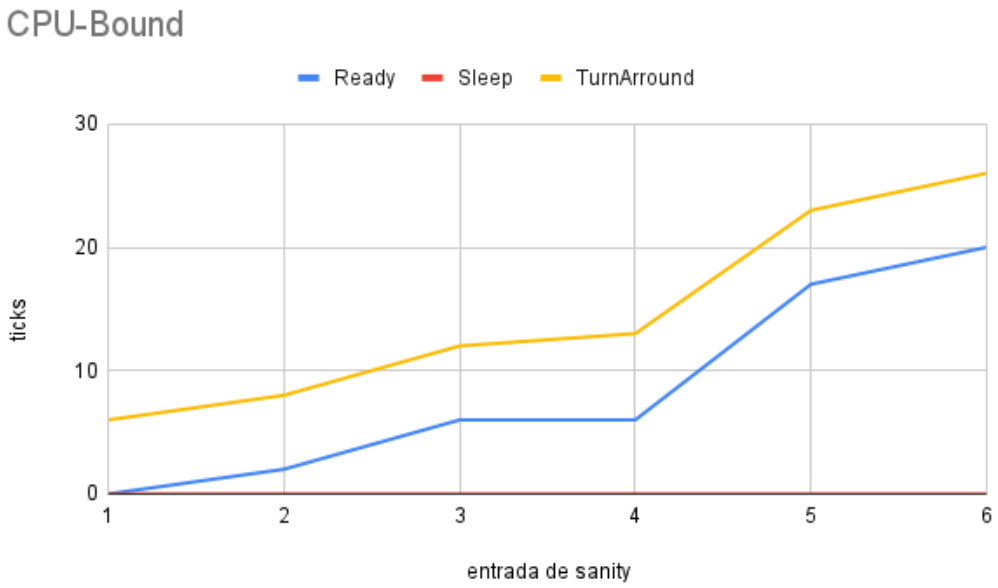
Foi criado o programa sanity, que recebe como parâmetro um inteiro n e cria 5n processos de 3 tipos diferentes:

- *CPU-Bound*: processos que buscam exaurir a cpu com computação (nessa implementação, utilizamos a instrução nop para extressar a cpu.
- *S-Bound*: processos que são entregues a CPU via yield.
- *IO-Bound*: processos que têm como gargalo alguma operação de Entrada/Saída (foi realizada a chamada de *sleep* para emular a E/S).

O programa então, imprime todos os processos em estado de *ZOMBIE* e imprime via saída padrão a média dos *rutime*, *retime* e *stime*.

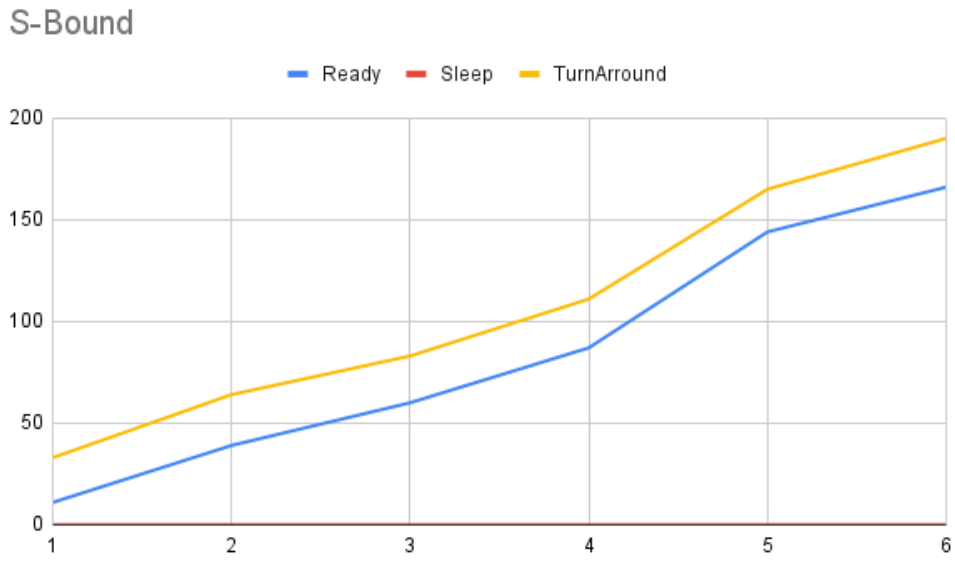
c. Resultados

i. CPU-Bound



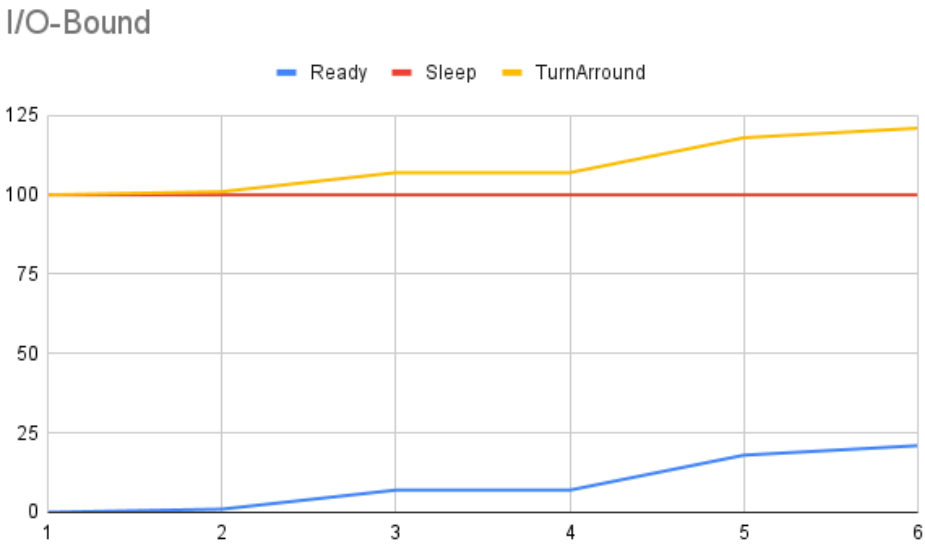
É perceptível um crescimento bastante linear para entradas baixas o suficiente

ii. S-Bound



Como esperado, o resultado para processos s-bound se comportam de maneira similar aos CPU-Bound, mas com valores mais elevados.

iii. I/O-Bound



Os processos I/O-Bound tem uma caracteristica de pouco impacto em relação à política de escalonamento, isso devido ao seu grande gargalo de espera.

4. Bibliografia

- Xv6 - a simple, Unix-like teaching operating system https://pdos.csail.mit.edu/6.828/2014/xv6/book-rev8.pdf acessado no dia 27/06/2022.
- http://pdos.csail.mit.edu/6.828/2014/xv6.html acessado no dia 27/06/2022.
- https://github.com/mit-pdos/xv6-public acessado no dia 27/06/2022.