Substituição de páginas com WSClock no Nanvix

Trabalho 2 de Sistemas Operacionais I (INE5412)

Pedro José de Souza e Pedro H. Aquino Silva

Maio de 2021

Este relatório trata da implementação do algoritmo WSClock no gerenciador de memória do sistema operacional Nanvix como segundo trabalho da disciplina INE5412/UFSC. O algoritmo de substituição de páginas *WSClock* utiliza um *working set* e um *clock* locais ao processo. Seu funcionamento é descrito em [1], e a Figura 1 traz um fluxograma detalhando-o.

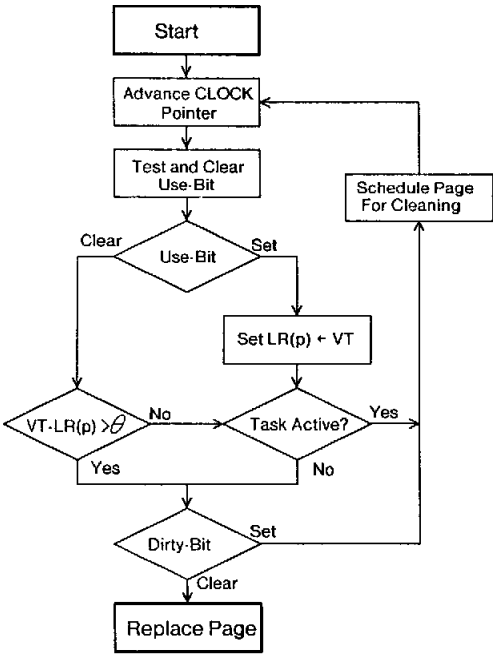

Figura 1: Flow Diagram do algoritmo WSClock

Algumas decisões importantes foram tomadas em relação a implementação do WSClock no Nanvix. Primeiramente, uma vez que as escritas do Nanvix são síncronas, sabemos que (1) todas as escritas são executadas em ordem, e (2) o algoritmo de subsituição de páginas só é retomado ao fim de uma escrita. Por conta deste fato de implementação do Nanvix, podemos garantir que a primeira escrita realizada é a que será finalizada primeiro. Deste modo, considerou-se realizar a escrita somente da primeira página suja encontrada, de acordo com uma ideia tirada do livro texto da disciplina [2]:

> In principle, all pages might be scheduled for disk I/O on one cycle around the clock. To reduce disk traffic, a limit might be set, allowing a maximum of n pages to be written back. Once this limit has been reached, no new writes would be scheduled.

Assim evitamos realizar alterações nos procedimentos `swap_in()` e `swap_out()`, uma vez que na implementação padrão do Nanvix, uma página não pode estar em memória e em *swap* simultaneamente, e precisaríamos de formas de fazer um *write back* para *swap*. Como otimização,

podemos armazenar em uma variável o índice da primeira página escalonada para escrita, e usá-la diretamente ao invés de realizar uma segunda volta no relógio. Esta informação é armazenada na variável `sched`.

Outra informação útil que podemos tomar logo na primeira volta é o índice da primeira página limpa. Desta forma, não precisamos executar outras voltas no relógio para encontrá-la.Armazenamos este dado na variável `clean_pg`. Por fim, assim como indicado em sala de aula, armazenamos o índice da página mais antiga em memória na variável `oldest`. Como o dado `oldest` só é útil caso não haja nenhuma página limpa em memória, então sempre que ele for usado, sabemos que seu bit *dirty* será 1.

Neste último caso, ressaltamos que o WSClock como apresentado em aula difere levemente do livro. Em aula, o professor afirma que quando não se encontra nenhuma página que não faz parte do *working set* e não temos escritas escalonadas, devemos então substituir a página mais velha, enquanto o livro diz que a página escolhida deve ser a atualmente apontada pelo relógio. Escolhemos manter a implementação de acordo com o apresentado em aula, e neste caso, a página substituída é `oldest`, mas o ponteiro continua apontando para a página que apontava antes da substituição.

Tomando estas variáveis, e considerando a escrita síncrona do Nanvix, não precisamos efetivamente realizar mais que uma volta no relógio no Nanvix, sendo suficiente atualizarmos o ponteiro do processo, caso necessário. Cabe agora explicar como é simulado o comportamento de relógio nesta implementação.

O primeiro aspecto é onde é armazenada a posição atual para qual o ponteiro de cada processo aponta. Incluímos um atributo `clock_hand` no `struct process (pm.h)`. Na criação de um processo, este valor é inicializado em -1, de modo que após o incremento no iníciio do WSClock, a primeira posição apontada é a primeira moldura de página em `frames`. Quando começamos o procedimento `allocf()`, este valor é restaurado em `i`, o ponteiro local para o relógio. Utilizamos este valor para iterar sobre as molduras de página até encontrar todos as páginas necessárias para tomarmos as decisões: `sched`, `clean_pg`, `oldest`, e a página apontada para substituição na primeira volta, o `tbr`. O ponteiro local `i` é incrementado de forma modular, seguindo a regra `i = (i+1) % NR_FRAMES`.

Se não encontrarmos um `tbr`, então avaliamos se alguma escrita foi escalonada. Caso positivo, realizamos um `swap_out()` e selecionamos esta página para ser substituída. Se não, o *write back* escalonado não é executado; e avaliamos se há alguma página limpa, ou seja `clean_pg` ≥ 0, e caso sim, esta é a página a ser substituída. Por fim, se não escolhemos uma página na primeira volta, não há páginas para serem escritas, e não há página limpa, então, escolhemos a página mais velha (`oldest`) para ser substituída. Nesse caso, precisamos ver se a versão em memória é limpa, e caso contrário essa página precisa ser escrita em *swap*, novamente com o procedimento `swap_out()`.

Quanto à restauração do ponteiro do relógio do processo, `clock_hand` é atualizado seguindo a regra abaixo, se não encontramos um `tbr` na primeira volta (caso em que o ponteiro salvo é o próprio `tbr`):

Para testes, reduzimos o número de molduras de página para 128, e utilizando o `gdb`, determinou-se que o N mínimo em `swap_test()` para forçar a substituição de página é 195.

Infelizmente, o `swap_test()` não foi finalizado com êxito, ocorrendo um PANIC que indica que algumas páginas estão sendo liberadas mais de uma vez ("*freeing user page twice*"). Ocasionalmente, ocorre a situação apresentada na Figura 2, em que o teste apresenta "PASSED", mas ainda ocorre o PANIC. Nesta captura de tela, o teste executado é com NR_FRAMES = 128 e N = 195 no procedimento `swap_test()`.

Contudo, o *bug* não foi encontrado pelos membros do grupo, e portanto não conseguimos decidir outros aspectos importantes do WSClock, como o `t`, envelhecimento mínimo para uma página ser considerada para substituição.

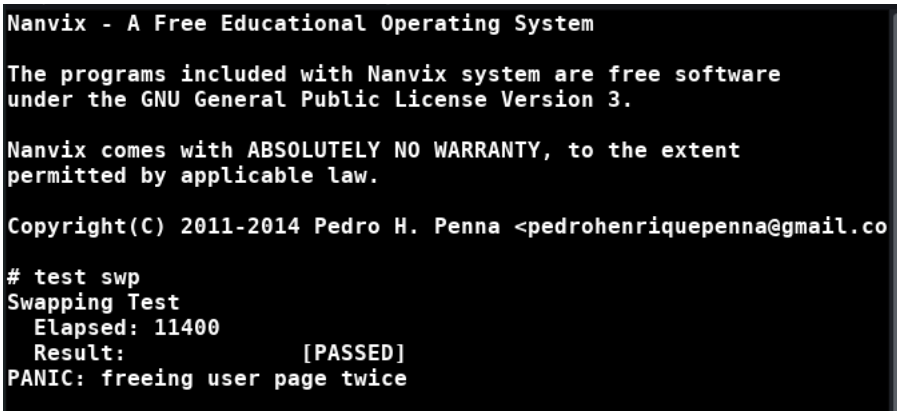

Figura 2: Captura de tela do Nanvix executando com o algoritmo implementado.

Referências

[1] Richard Carr and J. Hennessy. WSCLOCK—a simple and effective algorithm for virtual memory management. *Proceedings of the eighth ACM symposium on Operating systems principles*, 1981.

[2] Andrew S. Tanenbaum and Herbert Bos. *Modern Operating Systems*. Prentice Hall Press, USA, 4th edition, 2014.