Trabalho 1 - Escalonamento de Processos por Loterias no Nanvix

INE5412 - Sistemas Operacionais I

Pedro José de Souza e Pedro H. Aquino Silva

Abril 2021

Este relatório trata da implementação do algoritmo de *Lottery Scheduling* no sistema operacional Nanvix como trabalho 1 da disciplina INE5412. O grupo procurou minimizar as alterações no código fonte do Nanvix, restringindo-as aos arquivos `sched.c`, `pm.c` e `pm.h`.

O algoritmo implementado requer a geração de um número aleatório para decidir o *ticket* sorteado. O gerador de números aleatório utilizado na solução foi o disponibilizado no arquivo `klib.h` do próprio kernel, com as funções `ksrand()` para inicialização do PRNG com o valor arbitrário de 123456 e `krand()` para buscar o próximo número da sequência de números aleatórios.

Os *tickets* que cada processo recebe seguem a ordem de prioridade previamente utilizada no Nanvix, sendo que as prioridades são definidas, por exemplo, pelas chamadas da rotina `sleep()`, ou na criação do processo. Processos que precisam ser selecionados rapidamente, como os que estão esperando para inicializar operações de IO, recebem um número maior de tickets. O número de tickets que um processo pode receber antes da compensação segue a Tabela 1.

Prioridade	**Tickets**
`PRIO_IO`	80
`PRIO_BUFFER`	70
`PRIO_INODE`	60
`PRIO_SUPERBLOCK`	50
`PRIO_REGION`	40
`PRIO_TTY`	30
`PRIO_SIG`	20
`PRIO_USER`	10

Tabela 1: Quantidade de tickets para cada tipo de prioridade no Nanvix

O número total de tickets, utilizado para saber o intervalo dentro do qual o número sorteado deve estar, é calculado dinamicamente no mesmo laço de repetição já existente anteriormente no escalonador *round-robin* do Nanvix para os alarmes, e contabilizando somente os processos no estado pronto (`PROC_READY`). Desta forma, não é necessário manter uma variável global atualizada, que, quando testado, causou piora no desempenho do escalonador.

Também não foi adicionado nenhum outro campo no `struct` de processo, e portanto todas as demais partes de gerenciamento de processos são idênticas. Quando um processo é criado, ele recebe os tickets de acordo com a prioridade do seu processo pai, como feito anteriormente, e quando é terminado, sua última prioridade não é contabilizada no número total de tickets.

A implementação do sorteio em si segue a ideia apresentada no artigo base dado no enunciado, utilizado um sorteio baseado em listas. Utilizamos o mesmo laço de repetição do es-

calonador anterior e uma variável para acumular os tickets de cada processo. Caso esse valor seja maior que o ticket sorteado, escolhemos este processo e saímos do laço de repetição. Caso nenhum processo esteja no estado pronto, escalonamos `IDLE`. Quando um processo é escolhido, sua prioridade, isto é, sua quantidade de tickets, é decrementada em uma unidade, até que o valor chege na prioridade mínima (`PRIO_USER`), caso em que ele se mantém nessa prioridade. Isso é feito para garantir que os tickes não utilizados sejam mantidos pelo processo para ser escalonado novamente.

Destacamos que, com o escalonamento realizado desta forma, nenhuma estrutura de dados auxiliar é necessária, mesmo que seu uso resultasse em uma melhora de desempenho do escalonador, por exemplo, como em um escalonador baseado em árvores. A escolha de realizar esta implementação mais simples e pouco otimizada se deu principalmente por restrições de tempo dos integrantes do grupo, utilizando o máximo possível do que o código fonte do Nanvix oferece.

O tamanho de *quantum* escolhido foi de 10 *ticks* de clock. Este valor foi escolhido de forma parcialmente arbitrária, porque um *quantum* muito menor faria com que a troca de contexto ficasse demasiado custosa, e um maior reduziria a preemptividade do algoritmo e afetaria o suporte multitarefa do Nanvix, mas pouca diferença foi percebida entre um *quantum* de 10 ticks e outro de 8, por exemplo.

A ideia de *compensation tickets* foi implementada na função de escalonamento `sched()`. Como o atributo *counter* do processo representa o quanto o processo ainda tem de seu *quantum*, calculamos a fração $1/f$ descrita no artigo base, fazendo a operação $quantum \div (quantum - counter)$. Com esta fração calculada, podemos inflacionar o os tickets do processo, multiplicando o valor atual por $1/f$.

Infelizmente, o grupo não conseguiu resolver um *bug* no código implementado. O grupo buscou ainda, em última instância, a ajuda do professor, mas não foi respondido a tempo da entrega do trabalho. O *bug* em questão foi encontrado ao teste de escalonamento 2 (`sched_test2()`), que aponta uma falha de segmentação. Na Figura 1, o erro não ocorreu. Contudo, ao rodar o teste de escalonamento repetidas vezes, temos um *core dumped*. Certamente, o erro se refere a alguma lógica incorreta na própria implementação do algoritmo de escalonamento, e provavelmente na seleção do proximo processo a ser executado.

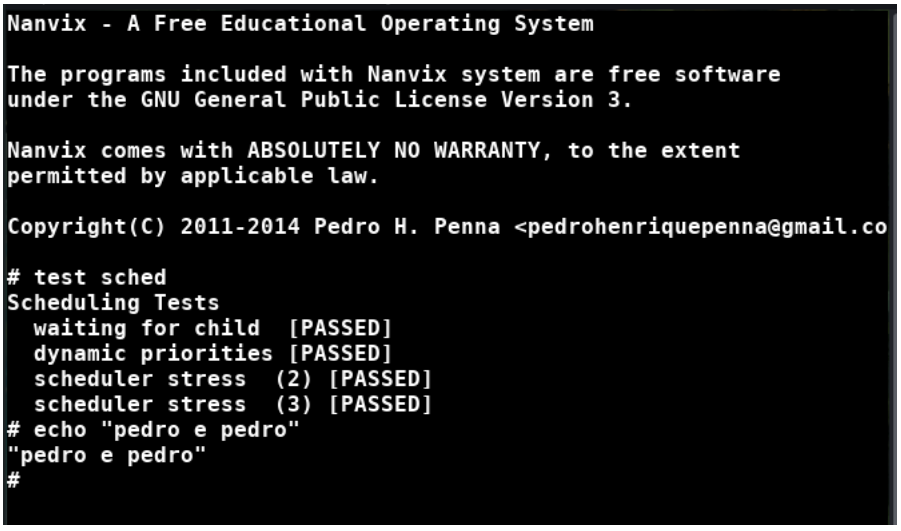

Figura 1: Captura de tela do Nanvix executando com o escalonador implementado.