Implementação de *lottery scheduling* para o Nanvix

Gustavo F. Guimarães, Gustavo Zambonin, Marcello Klingelfus*
Sistemas Operacionais I (UFSC – INE5412)

1 Descrição do algoritmo

Criado com o objetivo de prover garantia de utilização da CPU para cada processo, a estratégia de escalonamento escolhida para substituir a alternância circular presente na versão educacional do Nanvix é chamada de *lottery scheduling*. Este método foi apresentado na literatura em 1994, e consiste na determinação de uma certa "moeda", chamada de *tickets*, cuja função é representar a porcentagem relativa de execução do processo em relação a todos os outros. Após a associação de cada processo com uma certa quantidade de *tickets*, um sorteio é realizado de modo a selecionar o próximo processo a ser escalonado.

Esta estratégia é abstrata o suficiente para ser implementada independente de detalhes técnicos relativos ao hardware, e justa de acordo com certa probabilidade, por conta da quantidade de eventos pseudo-aleatórios que ocorrem a cada escalonamento individual. Entretanto, é possível perceber um aumento na acurácia da escolha dos processos com um número crescente de sorteios, como visto em [1, seção 2.2]. Nota-se também que, como a alocação de *tickets* e sorteio de processos são tecnicamente simples, o algoritmo permite a configuração de valores de *quantum* mais baixos ao longo da execução do sistema operacional.

2 Características da implementação

A simplicidade do *kernel* do Nanvix torna a implementação deste algoritmo bastante concisa; o código inicial não modifica a estrutura de um processo, e todas as computações relativas aos *tickets* e sorteio são feitas na função `yield()`, em `src/kernel/pm/sched.c`. A quantidade de *tickets* relativa à cada processo é fortemente baseada em sua prioridade atual, e também leva em conta a chamada de sistema `nice()`. A semente do gerador de números aleatórios é obtida a partir do tempo do sistema.

Porém, observou-se um grande número de cálculos realizados pelos laços internos à função (cerca de $2n$ execuções de macroinstruções, não otimizadas pelo compilador, onde n é o tamanho da tabela de processos). Assim, uma solução otimizada foi elaborada: cada estrutura de processo tem um membro responsável por guardar seu número de *tickets*, e o gerenciador de processos (`src/kernel/pm/pm.c`) guarda um contador total de *tickets*. Estas variáveis são modificadas pelas funções internas às chamadas de sistema de criação e destruição de processos (`fork()` e `bury()`, respectivamente). Dessa maneira, a sobrecarga do cálculo de macroinstruções é diminuída pela metade, e existe uma pequena melhoria no desempenho do teste focado no problema da inversão de prioridades.

Algumas estratégias de gerenciamento de recursos modulares apresentadas em [1, cap. 3] não foram implementadas por conta dos seguintes fatores: no caso da transferência de *tickets* entre processos, o caso de uso que mais se beneficiaria com este recurso, conversação entre clientes e servidores, não se aplica no sistema operacional pois este não implementa comunicação entre máquinas; como a

*{gustavo.g,gustavo.zambonin,marcello.klingelfus}@grad.ufsc.br

src/

distribuição de processos em um sistema operacional de propósito geral é heterogênea, implementar a inflação de *tickets* seria arriscado, pois um processo pode tomar conta do processador por um período elevado ao dar a si mesmo mais *tickets* por não ''confiar'' em outros; a criação de uma ''submoeda'' para que cada usuário classifique seus processos à sua maneira está fora do escopo, visto que o Nanvix admite apenas um usuário; por fim, a remoção da lógica envolvendo o membro da estrutura de processo `counter`, que lida com o caso de um processo não utilizar seu *quantum* completamente, faz a compensação de *tickets* não se mostrar necessária neste caso.

Referências

[1] Carl A. Waldspurger and William E. Weihl. Lottery scheduling: Flexible proportional-share resource management. In *Proceedings of the 1st USENIX Conference on Operating Systems Design and Implementation*, OSDI '94, Berkeley, CA, USA, 1994. USENIX Association.