MC514 – Sistemas Operacionais
Lab02 – Campeonato com Futex

Grupo:
David Kurka - RA070589
Felipe Eltermann - RA070803
Anderson Birocchi – RA 072787

Neste laboratório, implementamos uma função de sincronização de threads que se baseia nas chamadas de sistema *futex_wait* e *futex_wake,* ao invés de espera ocupada.

- **Descrição do algoritmo**

O programa cria N threads, que realizam sua tarefa de acesso a uma posição de memória compartilhada (seta a variável com o valor do seu id e imprime uma linha com seu id e o valor atual da variável, que idealmente deve ser o mesmo).

Para isso, primeiro encontramos P como a menor potência de 2, maior que N. Então, as threads competem entre si com um sistema de campeonato composto por fases e disputas. Na primeira fase ocorrem P/2 disputas, correspondendo a “duelos” 1 a 1 entre as threads. Disputas entre threads vazias (devido à diferença P-N) não produzem efeito, sendo desconsideradas seus resultados.

Após a primeira fase, P/2 threads passam para a próxima fase, deixando as threads anteriores “dormindo”. Essas P/2 threads passam por mais uma fase de disputa, restando em N/4 campeãs. Isso ocorre progressivamente, em log2(P) fases, até chegar na ultima fase, onde há somente uma disputa e o vencedor consegue acessar a região crítica. A estrutura fica semelhante à de uma arvore binária ou à uma tabela de chaves de campeonatos.

Após uma thread conseguir acessar a região crítica, ela desmarca seu interesse em todas as fases do campeonato que passou e “acorda” as threads com que fez disputas desde a primeira fase. Isso faz com que as threads que estavam em fases mais avançadas continuem em vantagem e disputando o acesso a região crítica.

Competição entre duas threads: cada thread inicia seu procedimento indicando que está interessada (atribuindo 1 à sua célula correspondente no *vetor de interesse*). Depois disso, ela modifica uma variável chamada *último* (compartilhada entre as duas threads, somente). Como as duas threads fazem essa segunda atribuição, uma fará primeiro que a outra. Se apenas uma thread está interessada (possivelmente por que a outra não conseguiu chegar na mesma fase ao mesmo tempo), a thread interessada primeiro passa para a próxima fase, rumo à região crítica, enquanto a thread “perdedora” fica dormindo (pela chamada futex_wait) até sua rival acabar todas suas disputas. Caso as duas tenham interesse simultaneamente, as duas tentam dormir, usando a mesma variavel (no caso a

variavel “ultimo”) e apenas a thread que tiver ultimo igual a seu id consegue dormir, fazendo com que sua rival passe para a próxima fase.

- **Implementação**

Foram alocados dois apontadores para vetores, sendo um para os vetores referentes aos interesses de cada thread em cada competição; e outro para os vetores de últimos. O número de vetores alocados (cada um com seu comprimento) é função de N.

Cada turno tem um vetor de *últimos* com X elementos e um vetor de interesse com 2*X elementos. A primeira fase possui P elementos no vetor de interesse e P/2 elementos no vetor últimos. A cada turno esses números são divididos por 2, até chegar no penúltimo turno (semi-final) que tem 2 últimos e 4 interesses; e no último turno que tem 1 último (vencedor) e 2 interesses.

Com essa estrutura de dados, posso criar futex especificos para cada disputa entre as threads e ter um bom controle.

- **Problema da ordem da atribuição de zeros:**

Segundo o que foi proposto no enunciado desse lab, a ordem com que a thread ganhadora atribui zeros para seu interesse nas disputas pelas quais passou podem interferir no resultado final. Para isso, no arquivo camperro.c também em anexo, na função desinteressa(), foi invertida a atribuição de zeros do vetor interesses e colocado um sleep para aumentar a probabilidade dos problemas acontecerem. Basta roda-lo para ver o resultado.