Trabalho Prático 2: Threads/Race Condition/ Região Crítica

Aluno em Graduação da Universidade Federal de Ouro Preto do curso Ciência da Computação:

Halliday Gauss Costa dos Santos.

Matrícula: 18.1.4093.

Área: Sistemas Operacionais.

Introdução:

No TP1 foi mostrado que o valor de ‘a’ pode ser diferente de 0, mas isso não era para acontecer, pois o número de vezes em que a variável ‘a’ é incrementada ela é decrementada. Esse problema está associado ao problema de Race Condition e o acesso simultâneo de várias threads em uma mesma região da memória (Região Crítica. Este documento apresenta a análise da implementação da resolução de problemas usando threads em **C.**

Desenvolvimento:

No trabalho prático, incialmente uma variável inteira e global ‘a’ foi criada e inicializada com 0. Além disso foi criado duas funções chamadas ProcUp() e ProcDown(), e ambas recebem como parâmetro o número de vezes em que serão executadas. A função ProcUp() incrementa em 1 o valor de ‘a’ e a função ProcDown decrementa a variável ‘a’ em 1. A quantidade ‘n’ vezes em que as funções serão executadas é igual a primeira letra do nome do aluno em ASCII concatenada com a segunda letra também em ASCII.

Em seguida duas threads distintas são criadas, a thread1 irá executar a função ProcUp() ‘n’ vezes, e a thread2 irá executar a função ProcDown() ‘n’ vezes.

Uma variável inteira e global chamada ‘turn’ também é criada e inicializada em 0. Enquanto ‘turn’ for igual a 0 a região crítica de ProcUp() é executada, caso ‘turn’ estiver com o valor igual a 1, o programa executará a região crítica de ProcDown() . Cada função é responsável por alterar o valor de ‘turn’ assim que sua região crítica for executada, portanto, passará a “vez” para a outra função executar a sua região crítica.

Uma região crítica é uma área de código de um algoritmo que acessa um recurso compartilhado que não pode ser acedido concorrentemente por mais de uma linha de execução. Ou seja, a região crítica, no TP, é o incremento e decremento do valor de ‘a’, portanto conclui-se que a região crítica não precisa ser a

mesma em cada thread, mas sim uma instrução qualquer sobre a mesma posição de memória (o mesmo serve para a região não crítica). Caso uma região crítica compartilhe sua área com mais de uma thread poderá ocorrer o problema de Race Condition (condição de corrida). Race Condition é uma falha num sistema ou processo em que o resultado do processo é inesperadamente dependente da sequência ou sincronia doutros eventos.

Existem diversas instruções dentro do incremento ou decremento da variável ‘a’. Essas instruções devem ser executadas em determinada ordem. Com o uso de threads não é possível garantir que essas instruções executem na ordem correta, gerando assim um valor diferente do esperado para variável ‘a’ (Race Condition).

Porém, o TP2 resolve o Problema de Race Conditon utilizando a variável ‘turn’, pois a região crítica é executada por uma thread para depois ceder o acesso a região crítica de outra thread gerando uma certa “atomicidade”. No entanto, esse problema não é resolvido de maneira eficiente pois uma thread pode bloquear uma outra thread e também porque uma thread espera uma outra thread executar a sua região crítica para depois poder executar também.

A função ‘join’ ajuda parcialmente em evitar a Race Condition, é uma função necessária, mas não suficiente, pois, o seu uso impede que os comandos seguintes sejam executados até que as threads terminem sua execução, podendo piorar a situação sem seu uso(haveria um grande acesso indevido a região crítica). No entanto, o uso de ‘join’ não impede que as regiões críticas sejam acessadas concorrentemente pelas threads.

A primeira e segunda lei que garante eficiência no uso de threads estão sendo cumpridas, já que a região crítica das threads não são executadas simultaneamente e o usuário não tem acesso ao número de CPUs. Já a terceira e quarta lei não são cumpridas, pois, uma thread bloqueia a outra usando a variável ‘turn’, portanto, uma thread espera a outra sair da região crítica para executar a sua área crítica, descumprindo assim a quarta lei.

Executando o programa, o sistema irá solicitar a primeira e a segunda letra do nome do nome do usuário e fará o cálculo do número de vezes em que as funções supracitadas serão executadas. Em seguida o programa criará duas threads e

atribuirão tarefas a elas. Ao executar o programa digitando qualquer caractere o valor de ‘a’ será sempre 0.

Conclusão:

A variável ‘a’ termina com o valor zero, pois, o mesmo número de vezes em que a variável ‘a’ é incrementada ela é decrementada, as threads estão rodando de maneira sincronizada e a implementação garante uma “certa” atomicidade.

A realização desse trabalho foi de suma importância para o melhor entendimento do funcionamento de threads e os problemas ao utilizá-los e como resolver esses problemas em uma linguagem de programação.