Iniciado em	terça, 29 mar 2022, 16:22
Estado	Finalizada
Concluída em	terça, 29 mar 2022, 19:33
Tempo empregado	3 horas 11 minutos
Avaliar	Ainda não avaliado

Questão 1

Completo

Vale 1,00 ponto(s).

Compile o main-race.c

Examine o código em um editor e encontre a condição de corrida nele.

Em seguida rode o helgrind, com o comando valgrind --tool=helgrind main-race, e veja como ele mostra a condição de corrida.

Ele mostrou as linhas corretas? Que outras informações ele mostrou?

A condição de corrida é a variável balance. Mostrou as linhas corretas 8 e 15, a criação dos threads e os endereços que os threads estavam modificando.

Questão 2

Completo

Vale 1,00 ponto(s).

O que acontece quando você remove uma das linhas que gera a corrida? Adicione travas em torno primeiro de uma das atualizações da variável compartilhada, e em sequencia da segunda, rodando o helgrind cada vez. O que ele mostra em cada cenário?

Se remover a linha 15 ele não mostra nenhum erro. Se colocar travas somente na linha 15 acontece o mesmo erro na Questão 1, o mesmo erro se colocar trava nas duas linhas 8 e 15.

O helgrind vê que tem locks, porem ainda não acontece a corrida.

Questão 3

Completo

Vale 1,00 ponto(s).

Agora examine o código em main-deadlock.c. Este código tem um problema chamado deadlock. Você consegue visualizar qual o problema?

Sim, o m2 é criado primeiro do que m1 e o m1 está travado dentro do m2. No entanto, o que causa o deadlock é que o programa tenta destravar m1 primeiro mas ele esta em m2.

Questão 4

Completo

Vale 1,00 ponto(s).

Agora compile main-deadlock.c e rode o helgrind nele. O que ele mostra?

Um erro, o programa tentou acessar um endereço travado.

Questão 5

Completo

Vale 1,00 ponto(s).

Agora rode o helgrind no main-deadlock-global.c.

Examine o código. Ele tem o mesmo problema do main-deadlock.c? O helgrind deveria mostrar o mesmo erro? O que isto te diz sobre ferramentas como o helgrind?

Sim, o helgrind mostra o mesmo erro. Porém, tem endereços travados acessados antes de ser destravado, isso mostra que o helgrind não consegue detectar todos os erros.

Questão 6

Completo

Vale 1,00 ponto(s).

Examine o código main-signal.c.

Esse código usa uma variável (done) para avisar que uma thread filha acabou e que o pai pode continuar.

Por que esse código é ineficiente? O que o pai gasta o tempo fazendo, especialmente se o filho demorar?

Porque ele espera o filho terminar de escrever o que causa a corrida, o pai fica comparando variavel enqunato espera o filho.

Questão 7

Completo

Vale 1,00 ponto(s).

Agora rode o helgrind em main-signal.c

O que ele reporta? O código está correto?

23 erros, não.

Questão 8

Completo

Vale 1,00 ponto(s).

Agora examine main-signal-cv.c, uma versão modificada do anterior usando variáveis de condição.

Por que este código é melhor que o anterior? Corretude, performance, ou ambos?

Por que ele é melhor em corretude, também é melhor em performance, pois a main espera a thread acabar antes de verificar a condição, portanto melhor em ambos.

Questão 9

Completo

Vale 1,00 ponto(s).

Rode o helgrind no main-signal-cv.

Ele mostra algum erro?

Alguns erros.