Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Programação Concorrente 2019/2

Relatório de Entrega de Atividades

Aluno(s): Gabriel Sylar
Matrícula: 123456789
Atividade: Aula Prática 03 – Espera Ocupada

1.1.2. Qual o problema do código anterior? Se há algum problema, ele acontece sempre? Por quê?

Resposta: O valor mostrado na tela esperado é 1000, mas em algumas execuções o valor apresentado é um pouco menor. Não acontece sempre, mas acontecem com baixa frequência dado um grande número de execuções. Acontece quando as threads acessam a variável global ao mesmo tempo, ocasionando apenas uma das atribuições com incremento seja feita.

1.1.3. De que forma seria possível resolver o problema do código, utilizando os conhecimentos já apresentados na disciplina?

Resposta: Uma solução seria esperar o termino da execução de cada thread. Isso evitaria que uma variável global fosse acessada por mais de uma thread ao mesmo tempo. Como todas as threads estão executando a mesma operação (soma com atribuição), essa solução não expressaria diferença considerável no tempo de execução.

1.2. Quais são as quatro condições para se evitar condições de corrida?

Resposta:

a) Dois ou mais processos não podem estar simultaneamente dentro de suas regiões críticas;

b) Nenhuma consideração pode ser feita a respeito da velocidade relativa dos processos, ou a respeito dos processadores disponíveis;

c) Nenhum processo que esteja fora de sua região crítica pode bloquear a execução de outro processo.

d) Nenhum processo pode ser a obrigado a esperar indefinidamente para entrar em sua região crítica.

1.2.2. O que acontece, com o funcionamento do algoritmo, como um todo, se a thread responsável pelo depósito terminar sua execução?

Resposta: Da forma como foi implementado, se colocarmos a condição de parada após um valor arbitrário de iterações na função de depósito, os laços continuam funcionando, porém como a chave de alternância para de receber troca de valores, a condição para as próximas operações de retirada nunca são satisfeitas. Assim o programa é finalizado, embora sua execução continue (por causa do laço) inutilmente. Um código (122.c) foi feito para exemplificar este item.

1.2.3. Altere o algoritmo anterior para, após 10000 iterações, a thread que executa a função deposito terminar a sua execução.

Observação: O código 123.c foi feito como resposta para este item. Mas é interessante notar que, visando um termino de execução intencional para a operação de deposito, o código foi escrito de modo que a operação de retirada continue funcionando (em contraposição ao item anterior) até o limite (quando a conta chega a um valor igual à 0), finalizando todas as threads, incluindo a principal, após isto.

3.1.2. Lendo o conjunto de instruções do binário gerado, em qual instrução função __sync_fetch_and_add foi convertida?

Resposta:

```
com contador++:
<       movl   contador(%rip), %eax
<       addl   $1, %eax
<       movl   %eax, contador(%rip)
com __sync:
>       lock addl    $1, contador(%rip)
```