Trabalho 2 - Uso de Locks e Variáveis de Condição

DEL/Poli/UFRJ

2021/1

1 Entrega e Pontuação

A data de entrega dos trabalhos, bem como sua pontuação, está especificada no arquivo das regras da disciplina. **Não haverá adiamento da entrega**. O código deve ser entregue em um arquivo .zip ou .rar pelo e-mail **rodsouzacouto@poli.ufrj.br**. O assunto do e-mail enviado deverá iniciar com o prefixo [EEL770][T2]. **ATENÇÃO: E-mails com assunto que não se iniciem com o prefixo podem ser ignorados. Não use esse prefixo para tirar dúvidas sobre o trabalho.**

Coloque seu nome e sobrenome no arquivo .zip ou .rar, sem caracteres de espaço. Ao zipar um diretório, certifique-se que tal diretório possui seu nome e sobrenome. Ou seja, **não** zipe um diretório com um nome genérico (p.ex., "Trabalho 2") para depois alterar o nome do arquivo zip. **ATENÇÃO: Não envie executáveis, pois serão rejeitados pelo gmail. Enviem apenas o código fonte, com o Makefile.**

2 Objetivo do Trabalho

O livro disponível em `http://greenteapress.com/semaphores/LittleBookOfSemaphores.pdf` possui diversos problemas de sincronização, apresentando sua solução com semáforos.

O objetivo deste trabalho é escolher um problema (veja Seção 3) e implementá-lo com locks e variáveis de condição.

3 Escolha e implementação do problema

Você deverá escolher um dos seguintes problemas e implementar sua solução:

- The FIFO barbershop (Seção 5.3)
- Baboon crossing problem (Seção 6.3)

O livro apresenta soluções e dicas. Você poderá usar a solução do autor do livro, mas adaptada para variáveis de condição, como também poderá utilizar sua própria solução. A ideia desse trabalho é ver um pouco, na prática, o uso dessas primitivas de sincronização.

A maior parte dos problemas são analogias com o mundo real, então você não conseguirá implementar de fato a função. Por exemplo, no problema do barbeiro, você não vai fazer uma função que corte o cabelo :) . Você deve apenas imprimir algo do tipo "Cortando o Cabelo".

As entidades do problema que competem por recursos devem ser threads. Por exemplo: cada produtor/consumidor é uma thread ; cada filósofo é uma thread, etc. Os recursos podem ser considerados como variáveis do programa. Por exemplo, no jantar dos filósofos, você pode ter cinco variáveis inteiras (isto é hashi1,hashi2,hashi3,hashi4,hashi5) que assumem valor 1 caso estejam em uso por um filósofo e 0 senão.

Quando um problema não possuir um número fixo de threads, seu programa deverá ser facilmente parametrizável para ser testado com um número arbitrário de threads. Por exemplo, no problema do jantar dos filósofos, o número de threads é sempre cinco. Entretanto, no problema do Dinning Hall, o seu

código deverá funcionar para um número arbitrário de estudantes (threads). A parametrização do número de threads pode ser feita por solicitação no início do programa ou por macro. Nesse último caso, deixe explícito no código quais são as macros de parametrização.

4 Regras

- O programa deve ser escrito em C ou C++;
- Utilize a API pthreads para threads, locks e variáveis de condição. **Não use outra API de threads**. Excepcionalmente, caso use C++, você poderá usar as threads do padrão C++11 (isto é, as classes std::thread, std::mutex, std::unique_lock, std::condition_variable, etc). Entretanto, o suporte do professor a essas classes é reduzido, já que no curso apenas a pthreads foi abordada. É importante notar que a API pthreads é também suportada em C++;
- **Não** use semáforos;
- O código deve vir acompanhado de Makefile. Códigos que não forem entregues em condições de serem compilados não serão corrigidos;
- Lembrem-se de escolher nomes adequados para as variáveis e identar;
- Muito importante: O código deve ser comentado para permitir o entendimento do raciocínio utilizado por vocês para escrever o programa e para solucionar o problema escolhido;
- Deixe claro no código qual foi o problema escolhido, especificando o número de seção (p.ex., 7.3 para o The room party problem);
- Caso o número de threads do problema não seja fixo, seu código deve ser facilmente parametrizável para um número arbitrário de threads (veja Seção 3);
- Seu programa deverá executar por várias iterações. Escolha um critério de parada para evitar que o código execute indefinidamente. Por exemplo, no jantar dos filósofos, o critério de parada pode ser definir que o programa finaliza quando 100 ações de comer forem executadas. Outro exemplo, no problema de produtores/consumidores, é definir que o programa finalizada quando houver, pelo menos, 100 ações de produção e 50 ações de consumo. O critério de parada deve ser facilmente parametrizável, por solicitação no início do programa ou por macro;
- Seu código deve ser livre de *deadlocks*. Uma dica é fazer um script de teste que chame milhares de vezes seu código em sequência. Essa é uma forma rudimentar de testar a existência de deadlocks, mas não garante a sua ausência. Lembre-se que deadlocks são difíceis de detectar, tente desenvolver o código pensando em eliminá-los. ;
- O trabalho deve ser realizado de forma individual;
- **Atentem-se para as regras de cópia de trabalho desta disciplina. Não haverá exceções! Não use código de alunos de períodos anteriores, pois isso será verificado na etapa automática de correção!**

5 Plataforma

O código poderá ser desenvolvido para o Linux, para o MacOS, ou para qualquer outro sistema UNIX. Informe no início do código qual sistema operacional você usou para fazer o trabalho. No caso do Linux, informe também a distribuição (Ubuntu, Debian, VM fornecida na disciplina, etc.).

O seguinte link possui uma VM com Linux para uso no Virtualbox. Seu uso não é obrigatório, mas pode ajudar quem não tiver muita memória RAM para máquina virtual:
`https://drive.google.com/file/d/1-R9i3QB3aqnLMBQhUlz03K2lM8yvxp7P/view?usp=sharing`. A senha da VM, e o nome do usuário, é eel770.