Laboratório 1
Introdução à programação concorrente em C (com threads)

Programação Concorrente (ICP-361) - 2024.2
Profa. Silvana Rossetto

[1]Instituto de Computação/CCMN/UFRJ

Introdução

O objetivo deste Laboratório é apresentar os conceitos mais básicos de programação concorrente (com threads), criando e avaliando programas concorrentes em C. As seguintes funções serão usadas (em C):

- int **pthread_create** (pthread_t *thread, const pthread_attr_t *attr, void *(*start_routine) (void *), void *arg); (retorna 0 no caso de sucesso)
- void **pthread_exit** (void *retval);
- int **pthread_join** (pthread_t thread, void **retval); (retorna 0 no caso de sucesso)

Para cada atividade, leia com atenção tudo que está sendo pedido/explicado, siga o roteiro proposto e responda às questões colocadas.

Considera-se que os programas serão executados em um terminal Linux. Quem estiver usando Windows, provavelmente precisará configurar o ambiente de programação C para incluir a biblioteca pthread.h.

Atividade 1

Objetivo: Mostrar como criar um programa concorrente em C, usando a biblioteca `pthread`.

Roteiro:

1. Abra o arquivo **hello.c**, leia o código do programa e tente entender o que ele faz. O que será impressso na tela quando esse programa for executado? Acompanhe a explanação da professora.
2. Compile o programa *hello.c* na linha de comando fazendo: `gcc -o hello hello.c -Wall`).
3. Execute o programa **várias vezes** (ex.: `./hello`), **mantendo e alterando** a quantidade de threads. Observe o log impresso na tela. Ele muda de uma execução para outra?
4. Os resultados estão de acordo com o esperado? Acompanhe a explanação da professora.

Atividade 2

Objetivo: Mostrar como passar um argumento para uma thread.

Roteiro:

1. Abra o arquivo **helloArg.c**, leia o código do programa e tente entender o que ele faz.
2. Compile o programa na linha de comando fazendo: `gcc -o arghello helloArg.c -Wall`).
3. Execute o programa **várias vezes** e observe os resultados impressos na tela. **Observe a diferença em relação ao programa anterior.**
4. Por que foi necessário usar **long int** na variável iteradora? Acompanhe a explanação da professora.

Atividade 3

Objetivo: Mostrar como passar mais de um argumento para uma thread.

Roteiro:

1. Abra o arquivo **helloArgs.c**, leia o código do programa e tente entender o que ele faz.
2. Compile o programa na linha de comando fazendo: `gcc -o argshello helloArgs.c -Wall`).
3. Execute o programa **várias vezes** e observe os resultados impressos na tela. **Verifique se o programa funcionou como esperado.**
4. Por que foi necessário criar uma estrutura de dados nova? Acompanhe a explanação da professora.

Atividade 4

Objetivo: Mostrar como fazer a thread principal (*main*) aguardar as outras threads terminarem.

Roteiro:

1. Abra o arquivo **helloJoin.c**, leia o código do programa e tente entender o que ele faz.
2. Compile o programa na linha de comando fazendo: `gcc -o joinhello helloJoin.c -Wall`).
3. Execute o programa **várias vezes** e observe os resultados impressos na tela.
4. O que aconteceu de diferente em relação às versões/execuções anteriores? Por que? Acompanhe a explanação da professora.

Atividade 5

Objetivo: Implementar o seu primeiro programa concorrente!

Escreva um programa concorrente com **M** *threads*, para **somar 1** a cada elemento de um vetor de **N** elementos do tipo **inteiro** (Para cada elemento a_i do vetor, calcular o novo valor $(a_i + 1)$ e escrever o resultado na mesma posição do elemento.)

A tarefa completa deverá ser dividida entre as M threads de forma mais balanceada possível (as threads devem receber a mesma carga de trabalho). Os valores de M e N devem ser informados na chamada do programa.

Roteiro:

1. Comece pensando em como dividir a tarefa completa entre M threads. Importante: todas as threads deverão executar a mesma função. Qual(is) argumento(s) deverá(ão) ser passado(s) para a função executada pelas threads?
2. Implemente funções separadas para inicializar o vetor de entrada e verificar se o resultado está correto no final.
3. Na função *main*, chame a função de inicialização do vetor; crie as threads; aguarde o término da execução das threads criadas e chame a função para verificar se os valores finais do vetor estão corretos.
4. Execute o programa várias vezes e verifique se ele está funcionando corretamente.

Entrega do laboratório: Disponibilize o código implementado na **Atividade 5** em um ambiente de acesso remoto (GitHub ou GitLab). Use o formulário de entrega desse laboratório para enviar o link do repositório do código implementado e as respostas das questões propostas.