Prova de Programação: Threads e mecanismos de sincronização POSIX

11 de Maio de 2011

Duração: 120 + 30 minutos

Consulta: documentação instalada no computador

1 Preparação

Antes de avançar faça a descompactação do arquivo fornecido. Esta acção deverá criar um directório de nome **pp** o qual contém, entre outros ficheiros, alguns ficheiros com o esqueleto das soluções a desenvolver. Desenvolva a sua solução no directório **pp** assim criado.

2 Problema

Pretende-se que desenvolva em C e para o sistema Unix um programa *multithreaded* que use 2 *threads* para inicializar um vector de inteiros partilhado.

O programa deverá tomar como argumentos 2 inteiros, que especificam o nº de milhares elementos que cada *thread* deverá inicializar, sendo a sua soma o tamanho do vector a inicializar. Para inicializar um elemento do vector, um *thread* deverá atribuir-lhe o seu nº identificação (0 ou 1). Por ex., quando invocado como se segue:

```
./pp 6000 4000
```

o programa deverá criar 2 *threads*, o primeiro dos quais deverá inicializar 6000 milhares de elementos com o valor 0 e o segundo 4000 milhares de elementos com o valor 1.

A inicialização deverá ser feita por forma a permitir a máxima concorrência no acesso ao vector partilhado: uma variável, também partilhada, actuando como índice, deverá percorrer o vector e os *threads* irão competir por inicializar o elemento por ela apontado numa dada altura.

Para evidenciar a informação partilhada pelos *threads* em competição, ela será colocada numa estrutura de dados do seguinte tipo:

O programa deverá ser desenvolvido em múltiplos passos, cada um dos quais corresponde a uma alínea que será avaliada. Assim, na realidade, deverá escrever e entregar múltiplos programas, um por alínea, devendo o código fonte de cada um estar num ficheiro de nome pp<alinea>.c, onde <alinea> é a letra da alínea correspondente.

Os esqueletos básicos dos códigos-fonte de cada alínea já estão preparados e estão disponíveis em **ppa.c**, **ppb.c**,

•••

Um conjunto de rotinas de apoio e, especialmente, de verificação da correcção do s programas que desenvolver, está contido no ficheiro **util.c**. Deverá incluir sempre este ficheiro no comando de compilação do seu código como ilustrado a seguir para gerar o executável correspondente ao programa pedido na alínea b:

```
cc -Wall -D_REENTRANT ppb.c util.c -lpthread -o ppb
```

Nota Em princípio, não deverá precisar de estudar o código contido em **util.c**. A informação relativa às interfaces é fornecida em **util.h**. Assim, se sentir necessidade de o fazer, por favor peça-nos esclarecimentos.

- a) (10%) Escreva um programa que processe os argumentos da linha de comando, inicialize a estrutura de tipo vector_t a usar e, finalmente, mostre o estado do vector partilhado por via da invocação da função print_vector() (incluída em util.c).
- **b)** (10%) Desenvolva um novo programa que além de fazer o que é pedido em **a)** deverá criar os 2 *threads* de inicialização. Cada um destes *threads* deverá invocar a função printf() do seguinte modo:

```
printf ("Thread %d running\n", pthread_self());
```

O thread principal deverá esperar que cada um dos threads de inicialização termine, após o que deverá imprimir:

```
printf ("Main thread exiting\n");
```

Nota: Nesta alínea não precisa de passar qualquer argumento aos *threads* de inicialização.

c) (15%) Desenvolva um novo programa que além de fazer o que é pedido nas alíneas anteriores deverá fazer com que cada um dos *threads* de inicialização invoque a função printf() do seguinte modo:

```
printf ("Thread %d: cnt = %d, vp = %p\n", X, Y, Z);
```

onde X representa o identificador do *thread* (ou 0 ou 1), Y representa o nº de elementos que o *thread* deverá inicializar, e Z o endereço da estrutura de dados do tipo vector_t partilhada pelos *threads* de inicialização. Antes de terminar cada *thread* deverá ainda invocar a função print_vector() já referida.

Nota: Cada *thread* deverá invocar esta instrução com um valor X diferente.

d) (15%) Desenvolva um novo programa que, para além de fazer o que é pedido nas alíneas anteriores, faça com que os 2 *threads* de inicialização inicializem **mesmo** o vector partilhado, mas sem tentarem evitar erros de competição (*race conditions*). A impressão da informação já pedida na alínea anterior irá permitir ver o trabalho que foi efectuado.

Nota: aqui já deve ser possível observar que há erros de competição, mesmo para um vector partilhado com um nº de elementos relativamente pequeno! É óbvio, pois não houve esforço de sincronização no acesso ao vector!

e) (10%) Desenvolva um novo programa que, além de fazer o que é pedido nas alíneas anteriores, cria um thread de verificação, i.e. que executa a função checker () declarada em util.h. O argumento desta função deve ser um apontador para uma variável do tipo chkr_arg_t definida também em util.h, e aqui reproduzida:

O *thread* de verificação deve ser criado antes dos *threads* de inicialização e esperado pelo *thread* principal apenas depois daqueles *threads* terem terminado.

Nota: aqui também ainda devem ser patentes os erros de competição, mesmo para um vector partilhado com um nº de elementos relativamente pequeno, pois ainda pois não houve esforço de sincronização no acesso ao vector!

Nota sobre a função checker() A função checker () faz a validação dos argumentos que lhe são passados e em seguida verifica periodicamente (cada 100 ms) o estado do vector partilhado. À semelhança da função com o mesmo nome usada nas aulas laboratoriais, esta função termina se não detectar qualquer alteração no estado do vector partilhado durante 5 segundos. Isto pode significar que o vector já foi completamente inicializado, ou que ocorreu um *deadlock*.

Nota sobre a função checker() Porque a função checker() sincroniza de forma apropriada no acesso às variáveis partilhadas, pode ficar bloqueada indefinidamente se os restantes *threads* não sincronizarem correctamente. Assim, se o seu programa não gerar qualquer saída após cerca de 30 segundos, quando invocado com 2 argumentos cujo valor é 5000, pode concluir que com bastante confiança que tem alguns problemas.

f) (25%) Desenvolva um novo programa, corrijindo a versão da alínea **e**) por forma a que, garantidamente, não haja *race conditions*, mesmo para um vector partilhado com um elevado nº de elementos!

g) (15%) Desenvolva um novo programa que garanta que, em qualquer instante da inicialização do vector partilhado, a diferença máxima (em milhares) entre o nº de elementos inicializado por cada *thread* não excede um dado valor. Isto permitirá um maior grau de "entremeação" - e competição - na execução dos *threads*. Tal diferença máxima de elementos inicializados será apresentada como um parâmetro adicional na linha de comando. Por ex., com uma invocação na forma :

```
./ppg 5000 5000 1
```

o seu programa deverá criar 2 *threads*, o primeiro dos quais deverá inicializar 5000 milhares de elementos com o valor 0 e o segundo com 5000 milhares com o valor 1, sendo que em qualquer instante o nº de elementos inicializado por cada *thread* não pode diferir em mais de 1000.

O seu programa deverá usar semáforos ou variáveis de condição para sincronizar correctamente os 2 *threads* de inicialização sem *busy waiting*.

Nota: Assuma que o programa é invocado sempre com argumentos cujos valores são tais que o valor absoluto da diferença dos 2 primeiros argumentos não excede o valor do 3º argumento.

3 Submissão da Prova

Para terminar a prova deverá:

- verificar se os ficheiros esperados se encontram no directório corrente dando o comando ./check.sh num terminal, no directório em que desenvolveu a prova. .
- comprimir num ficheiro . zip único o directório que criou no início da prova e que deverá ter sua solução, i.e. ficheiros com o código em C.
- submeter esse ficheiro através da interface *web* do sistema de gestão de exames (SIGEX), http://sigex.fe.up.pt/, para o que será necessário fazer a autenticação com o nome e a senha do FEUPSIG e utilizar o código público desta prova, que depende da sala onde a realizou:

B208 EHK3306

B213 QNG1627