Variáveis Condicionais – API e exemplo

Este ficheiro destina-se a ser projetado e, portanto, tem um formato e tamanho de letra diferente do habitual.

Tópicos

1. Compilação de programas com uso de variáveis condicionais
2. Ficheiros header usados
3. Conceito de variáveis condicionais
4. API
5. Exemplo

1. Compilação de programas com variáveis condicionais:

As variáveis condicionais são usadas no contexto de threads e o API está incluído no das threads.

gcc *etcetc.c* –pthread

2. Ficheiros header diretamente envolvidos

pthread.h

3. Conceito de Variáveis condicionais

- Permitem eliminar um teste de espera activa em situações onde uma thread aguarda um dado acontecimento.

    - O ciclo de espera e sinalização já implicaria, normalmente, um mutex para a variável de controlo (para sinalização) e eventuais outros dados partilhados associados

Variáveis condicionais permitem transformaro padrão de código:

Thread que **aguarda**

```
while (1) {
   pthread_mutex_lock(Mutex_A);
   // … dentro de secção crítica
   if (flag_partilhada == 1)
      break;
   pthread_mutex_unlock(Mutex_A);
}  // break->precisa sair da sec.c
pthread_mutex_unlock(Mutex_A);
```

Thread que **assinala**

```
//…
pthread_mutex_lock(Mutex_A);
// … dentro de secção crítica
flag_partilhada = 1;
// … dentro de secção crítica
pthread_mutex_unlock(Mutex_A);
// …
```

(-> Espera activa na thread que aguarda –> ocupa o processador inutilmente)

Em

Thread que **aguarda**

```
// …
pthread_mutex_lock(Mutex_A);
// dentro de secção crítica
pthread_cond_wait(Cond_Var);
// secção crítica é recuperada
pthread_mutex_unlock(Mutex_A);
// …
```

Thread que **assinala**

```
//…
pthread_mutex_lock(Mutex_A);
// … dentro de secção crítica
pthread_cond_signal(Cond_Var);
// … secção crítica é recuperada
pthread_mutex_unlock(Mutex_A);
// …
```

- (A função pthread_cond_wait e pthread_signal_var são apresentadas de forma esquemática -> os parâmetros reais são ligeiramente diferentes)
- O uso de variável condicional é feito em associação com um mutex, que, neste caso, já existia no código inicial

-> Espera activa removida. As duas threads competem pela secção crítica cuja semântica é preservada

O ciclo de espera ativa na thread que aguarda pode ter outra aparência

Exemplo:

```
// thread que aguarda
pthread_mutex_lock (& Mutex_A);
while ( 1 ) {
    // liberta mutex para que outra thread possa
    // mudar a variável sinalizadora
    pthread_mutex_unlock(& Mutex_A);
    // faz algo / espera explicitamente
    // (tenta) volta a adquirir o mutex
    pthread_mutex_lock(& Mutex_A);
    // testa variável e sai se for caso disso
    if (flag_partilhada == 1) // "1", ou outro valor
       break;
}
// fim de secção crítica - liberta mutex
pthread_mutex_unlock(& Mutex_A);
```

Independentemente da aparência, nota-se a existência de um ciclo fechado em teste exaustivo, ocupando o processador (espera ativa)

A variável condicional permite transformar o código anterior no seguinte

Na thread que aguarda

```
// Thread que aguarda
pthread_mutex_lock (&Mutex_A);

// liberta implicitamente o mutex e aguarda
pthread_cond_wait (&Cond_Var, &Mutex_A);
// readquire o mutex automaticamente

// Resto da secção crítica e posterior libertação
// do mutex
pthread_mutex_unlock (&Mutex_A);
```

E na thread que assinala o acontecimento

```
// o mutex terá ficado livre pelo uso de pthread_cond_wait
// na thread que aguarda permitindo a esta avançar
pthread_mutex_lock (& Mutex_A);

// assinala a variável condicional permitindo à thread
// que aguarda avançar ("acorda-a / acorda-as")
pthread_cond_signal (&Cond_Var);
// o mutex é libertdo e reqdquirido automaticamente
// as duas threads competem pelo mutex. Uma delas apanha-o
// primeiro e avança, liberta o mutex permitindo à
// outra avançar também
pthread_mutex_unlock (& Mutex_A);
```

- Nestes dois últimos excertos de código as funções **pthread_cond_wait** e **pthread_cond_signal** já são apresentadas usando os parâmetros reais

4. API – Variáveis condicionais

Tipo de dados para variável condicional: pthread_cond_t

Declaração de uma variável condicional "MyCondVar"

```
pthread_cond_t MyCondVar;
```

Inicialização de variável condicional

Usar a função seguinte

```
pthread_cond_init (pthread_cond_t *)
```

ou então usar a simples atribuição

```
pthread_cond_t MyCondVar = PTHREAD_COND_INITIALIZER;
```

Destruição de variável condicional ("des-inicialização")

```
pthread_cond_destroy (pthread_cond_t *)
```

Esperar numa variável condicional

```
pthread_cond_wait (pthread_cond_t *,pthread_mutex_t *)
```

- Liberta mutex associado e espera na variável (até ficar "sinalizada")
- Quando a variável é assinalada, o mutex é readquirido automaticamente em competição com a thread que assinalou a variável -> a semântica de secção crítica é preservada

```
pthread_cond_timedwait (pthread_cond_t *,
          pthread_mutex_t *, const struct timespec *)
```

- Como a anterior, mas com um timeout de tempo máximo de espera

Sinalização da variável

(=dar permissão a outra(s) thread(s) que aguardava(m) para avançar)

```
pthread_cond_signal (pthread_cond_t * )
```

- Acorda uma das das threads que aguardava nesta variável condicional. O mutex é libertado e volta a ser adquirido automaticamente, em competição coma thread que acordou
  - A semântica de secção crítica é preservada

```
pthread_cond_broadcast
```

- Acorda todas as threads que aguardam nesta variável condicional

5. Exemplo

Este exemplo ilustra o uso de variáveis condicionais

-> Isto implica também o uso de threads e mutexes

Cenário do exemplo

- São criadas várias threads que incrementam um contador **comum** a todo o programa e threads (mas não é uma variável global). O valor do contador é analisado pela thread que executa a função main.
- Quando a valor atinge um determinado valor, a thread que executa a função main avança e faz terminar o programa.

Situações de espera/sinalização

- A função que executa a função main é a função que aguarda um acontecimento.
- Esse acontecimento é o incremento do contador.
- A thread da função main aguarda por esse acontecimento através de uma variável condicional.
- Sempre que uma das restantes threads incrementa esse contador, sinaliza esse acontecimento à thread da função main usando a variável condicional.

Recursos comuns

- O acesso ao contador constitui uma secção crítica e é guardado por um mutex.
- A variável condicional e o mutex estão assim interligados.
- A variável condicional e o mutex são, tal como o contador, comuns a todo o programa (partilhados entre as threads).

Neste exemplo não se verificam situações de erro nas chamadas sistema efetuadas para manter o código mais claro e focado no que se pretende transmitir.

-> É evidente que em código real é obrigatório analisar todos os valores de retorno e possíveis situações de erro (exemplo "código real": o trab. prático)

Para compilar o programa do exemplo: *linkar* com pthread

Assumindo que o ficheiro de código fonte se chama varcond.c

```
gcc -pthread varcond.c -o varcond
```

```
// Ficheiros header usados
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>

// Número de threads a usar. O valor 7 é apenas um exemplo
const int NUMTHREADS = 7;
```

Para evitar que as threads teNham um comportamento exatamente igual umas às outras, o número de loops vai ser baseado num "valor central" com variação aleatória de mais ou menos um

```
// Número de loops a usar (valor central). O número 4 é apenas um exemplo.
const int NUMLOOPS = 4;
```

Deve evitar-se a todo o custo o recurso a variáveis globais. As informações necessárias às threads são-lhes dadas a conhecer através do ponteiro que é passado por parâmetro às respetivas funções. Esse ponteiro apontará para uma variável estruturada que conterá os vários itens de informação de interesse à thread

```
// Estrutura de dados para conter os dados que as threads precisam
// incluindo acesso à variável partilhada "prontas", o mutex e a variável condicional
typedef struct {
    int * ptrProntas;
    pthread_t tid;    // thread ID
    int myID;
    int myNumLoops;
    pthread_mutex_t * ptrMutex;
    pthread_cond_t * ptrCondV;
} ThreadDados;

// Função da thread
//  - O ponteiro info apontará para a estrutura ThreadDados que o necessário a cada thread
void * funcaoThread(void* info) {
    ThreadDados * ptrMyDados = (ThreadDados *) info;

    for (int i=0; i<ptrMyDados->myNumLoops; i++) {
        printf("Thread %d working (%d de %d)\n",
            ptrMyDados->myID, i, ptrMyDados->myNumLoops );
        sleep(1 + (rand()&1) ); // simula um trabalho qualquer demorado + 0 / 1 rand
    }
```

Ao terminar, a thread aumenta o contador e indica esse fato assinalando a var. cond.

```
    // O contador está em *(ptrMyDados->ptrProntas);
    // O contador é partilhado, logo é necessário usar o mutex
    pthread_mutex_lock(ptrMyDados->ptrMutex);

    // Aumenta a variável partilhada apontada por ptr
    (*ptrMyDados->ptrProntas)++;
    printf( "Thread %d pronta. Total prontas agora = %d\n",
                ptrMyDados->myID, *(ptrMyDados->ptrProntas) );

    // Assinalar a variável condicional -> acorda a thread "main"
    pthread_cond_signal(ptrMyDados->ptrCondV);
    pthread_mutex_unlock(ptrMyDados->ptrMutex);

    return NULL;
}
```

Função main

Tarefas:

- Lançar as threads
- Aguardar pela sinalização na variável condicional
- De cada vez que é assinalada averigua o valor do contador
    - Se atingir o valor pretendido, avança e termina o programa
    - Caso contrário, volta a aguardar

```
int main( int argc, char * argv[] ) {
    int i;

    // Contador partilhado
    //   - é usado em paralelo (simultâneo) por isso deve ser protegido por um mutex
    //   - quanto atingir o numero total de threads a thread "inicial" avança
    int prontas = 0;

    // Mutex e variável condicional
    //   - São dados a conhecer às threads por ponteiro em ThreadDados
    pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
    pthread_cond_t condv = PTHREAD_COND_INITIALIZER;

    printf("Thread Inicial a correr a função main\n");

    ThreadDados dadosthreads[NUMTHREADS];

    // Cria as threads
    for(i=0; i<NUMTHREADS; i++) {
        dadosthreads[i].myID = i+1;

        // Variação +- 1 aleatória no número de loops a cumprir por esta thread
        dadosthreads[i].myNumLoops = NUMLOOPS + (1-rand()%3);
        dadosthreads[i].ptrProntas = & prontas;
        dadosthreads[i].ptrMutex = & mutex;
        dadosthreads[i].ptrCondV = & condv;
        pthread_create( & dadosthreads[i].tid, NULL, funcaoThread,
                                                    dadosthreads+i );
    }
```

Vai-se usar um ciclo para esperar por todas as threads

- A condição do ciclo usa a variável partilhada prontas (é o contador)
- Usa-se então um mutex que é fechado antes de iniciar o ciclo
    - O mutex será libertado e readquirido dentro do ciclo pelo uso da espera na variável condicional
    - É essa libertação temporária do mutex que permitirá às outras threads usar o contador e incrementá-lo
    - Trata-se de um dos padrões exemplificados mais acima neste documento

```
    // Fechar o mutex antes de chegar à condição do ciclo
    pthread_mutex_lock( &mutex );

    // Usa um ciclo para aguardar por todas as threads
    // No entanto este ciclo não vai ter espera activa devido ao uso da variável condicional
    while(prontas < NUMTHREADS ) {

        // Aguarda sinalização da variável condicional
        //    - Bloqueia no wait e o mutex é libertado enquanto bloqueado
        //    - O mutex é automaticamente readquirido quanto o wait acorda
        pthread_cond_wait( & condv, & mutex );
        printf("Thread \"main\" acordou: prontas %d de %d\n",
            prontas, NUMTHREADS);
    }
```

No final do ciclo todas as (outras) threads já assinalaram "pronto" e, portanto, deverão ter terminado ou a terminar

```
    // Libertar o mutex (porque dentro do ciclo readquiriu o mutex
    // este unlock após o ciclo é a acção “simétrica” do lock antes do ciclo
    pthread_mutex_unlock( & mutex );
    printf("Thread \"main\": tudo pronto\n");

    return 0;
}
```

Nota: este exemplo pretende exemplificar *threads*. Não pretende ser fonte de *copy&paste* para trabalhos práticos. Adapte - não faça meros decalques