9

Programação com Posix Threads

Programação multi-threaded com Pthreads

1. O que é um thread?

Um processo represente a execução pelo sistema operativo dum programa. Uma thread represente uma linha de execução das instruções deste programa. Um processo poderá conter várias threads, pelo menos uma. O comando Unix **`ps`** permite ver os processos que estão a correr e dando a opção correcta as threads que estão a executar. O comando **top** também é útil para obter uma lista ordenado dos processos.

2. O que são Pthreads?

Historicamente cada sistema operativo/hardware implementava a sua própria versão de threads. Claro que cadaim plementação varia entre si portanto era difícil para programadores de desenvolver aplicações portáveis que utilizassem threads. Para superar estas dificuldades a padrão POSIX1003.1-2001 foi desenvolvida. Esta define um *application programming interface* (API) para a escrita de aplicações *multithreaded.* Implementações de threads que aderem a este padrão são conhecidos como POSIX threads ou simplesmente pthreads.

- Pthreads são definidos como um conjunto de tipos de dados em C e um conjunto de rotinas.
- Os Pthreads API contêm mais de 60 sub-rotinas.
- Todos os identificadores na livraria começam com pthread_
- O ficheiro pthread.h tem que ser incluído em cada ficheiro de código fonte.
- A linkagem com qualquer biblioteca dinamica/estatica necessária é feita conforme o sistema.

3. Pthread Criação e Terminação

Quando um programa começa a executar terá um processo com uma thread a executar. Mais threads são depois criadas com a função pthread_create() e destruídas com a função pthread_exit().

Exemplo 9.1 Pthread Criação e Terminação

```
#include <stdio.h>       /* standard I/O routines                 */
#include <pthread.h>     /* pthread functions and data structures */

/* function to be executed by the new thread */
void *OLA(void *argumentos)    {
   printf("\nOla\n");
   pthread_exit(NULL);
}

int main ( )
{
   pthread_t thread;
   int flag, i;

   printf("A criar uma nova thread\n");
   flag = pthread_create(&thread, NULL, OLA, NULL);
   if (flag!=0) printf("Erro na criação duma nova thread\n");

   OLA(NULL);
   pthread_exit(NULL);
   return 0;   /* O programa não vai chegar aqui.         */
}
```

Notas Explicativas:
A chamada à função pthread_create() tem quatro argumentos. O primeiro é usado para guardar informação sobre a thread criada. O segundo especifica algumas propriedades da threada criar, utilizamos o valor NULL para significar os valore por defeito. O Terceiro é a função que a nova thread vai executar e o ultimo é usado pararepresentar argumentos a esta função

A chamada à função pthread_exit() provoca a terminação da thread e a libertação dos recursos que está a consumir. Aqui não há realmente necessidade para usar esta função porque quando a função da thread termine a thread seria destruída. A função é apena sútil se for necessário destruir a thread no meio da sua execução.

A função inicial a ser executada por uma nova thread tem sempre o formato

```
void * funcao ( void * argumentos ) ;
```

Exercício 9.1

Escreveo programa do exemplo 9.1. Compile o programa (cc -o criar criar.c -lpthread)e depois executá-lo. Quantos threads são criados. Quantas mensagens aparecem naecrã ?

4. Esperando pela Terminação duma Thread

As threads podem executar duma forma desunidas (*detached*) dathread que as criou ou unidas. Desta maneira usando a rotina pthread_join() uma thread pode esperar pela terminação duma thread específica.

Exemplo 9.2 Esperando uma thread

```
#include <stdio.h>
#include <pthread.h>
#define NUM_THREADS     5

void *OLA(void *argumentos)
{
   printf("\nOLA\n");
   return (NULL);
}

int main ()
{
  pthread_t threads[NUM_THREADS];
  int i;

  for(i=0;i < NUM_THREADS;i++)
       pthread_create(&threads[i], NULL, OLA, NULL);

  printf("Thread principal a esperar a terminação das threads criadas \n");

  for(i=0;i < NUM_THREADS;i++)
      pthread_join(threads[i],NULL);  /* Esperara a junção das threads */
  return 0;   /* O programa chegará aqui. */
}
```

Exercício 9.2

Escreve, Compile e Execute o programa do exemplo 9.2.

Exercício 9.3

- Neste exercício vai escrever um programa que poder ser parado para depois inspeccionar as threads criadas usando as ferramentas do sistemas (ps, activity monitor etc.). Usando como base o exemplo 9.2 escreve uma programa multithreaded onde as threads criadas pela thread principal entrarão num ciclo enquanto o valor duma variável global, x, seja igual ao valor 1. O programa deverá declarar e inicializar a variável global **x** ao valor 1. Na thread principal depois de criar as threads pede o novo valor de **x** usando por exemplo scanf.
  Ver o código em baixo!

```
void*funcao (void *args)
{
  while (1==x)
     ; /* fazer nada (spin) */
  return (NULL);
}
```

```
intx=1;
main()
  for...pthread_create( funcao )
   printf("Valor de x ..");
  scanf("%d",&x);
  for...pthread_join();
```

(i) Compile e Execute o seu programa.
(ii) Execute o seu programa mas na altura de efectuar a introdução dum novo valor de **x** parar o programa com *ctrl-z* e depois inspeccionar o processo usando o comando ***ps*** e/ou o comando ***top***.
(iii) Execute de novo o seu programa. Numa outra janela de terminal inspeccione as threads do processo e os seus estados usando os comandos **ps/top/activity monitor etc**.

Nota: Podem ser úteis as opções A,l,m,M,T do commando ps

5. Passagem de Argumentos para threads

- A rotina pthread_create() permite ao programador passar apenas um argumento a rotina de thread nova.
- Um argumento é passado por referencia (apontador) e é feito um cast para (void *).
- Para passar vários argumentos temos que empacotar os valores numa estrutura e depois passar o endereço para esta estrutura

Exemplos 9.3 –(I) Passagem dum Inteiro (II) uma String (III) Umestrutura

I.Passagem dum inteiro

```
int x=5;

pthread_create(&threads[i], NULL, funcao, (void*)&x);

void * funcao ( void * argumentos ){
    int valor =  * (int *) argumentos;
    printf("recebi um inteiro: %d \n", valor );
}
```

II.Passagem duma String

```
char mesg[ ]=”Ola”;

pthread_create(&threads[i], NULL, funcao, (void*)mesg);

void * funcao ( void * argumentos ){
    char *message = (char *) argumentos ;
    printf(" %s ", message );
}
```

III.Passagem de múltiplos parâmetro susando uma estrutura

```
typedef struct { int a; floatb; } ST;
ST v;
```

```
v.a=5;  v.b=2.5;
pthread_create(&threads[i], NULL, funcao, (void*)&v);

void * funcao ( void * argumentos ){
    ST in =  *(ST *) argumentos ;
    printf("recebi dois valores: %d %f ", in.a, in.b );
}
```

6. Condiçoes de Corrida na Passagem de Argumentos para threads

Existe uma dificuldadeadicional em relação a passagem de dados para as novas threads. Dado a sua inicialização não-deterministica e impossibilidade de controlar o escalonamento e despacho das threads pode ser difícil assegurar a passagem de dados com segurança, sem condiçoes de corrida. O exemplo seguinte ilustre esta dificuldade.

Exemplo 9.4 O ficheiro test1.c

```
#define NUM_THREADS 5

void *funcao(void *args)
{
   int x=*(int *)args;
   printf("Thread %d\n",x);
   return (NULL);
}

int main ()
{
  pthread_t threads[NUM_THREADS];
  int i;

  for(i=0;i < NUM_THREADS;i++)
       pthread_create(&threads[i], NULL, funcao, &i);

  for(i=0;i < NUM_THREADS;i++)
      pthread_join(threads[i],NULL);

  return 0;
}
```

Dois ouputs tipicos de execuçãodo programa são mostrado em baixo :

```
alunos:~/so/cprogs/pthreads crocker$ cc -o test1 test1.c -lpthread
alunos:~/so/cprogs/pthreads crocker$ ./test1
Thread 2
Thread 2
Thread 2
Thread 5
Thread 5
```

```
alunos:~/so/cprogs/pthreads crocker$
./test1
Thread 0
Thread 1
Thread 2
Thread 4
Thread 4
```

7. A Passagem Correcta dos argumentos para identifcar uma thread.

A maneira correcta é passar um endereço para uma variável que identifique unicamente a thread e usada exclusivamente para identificar uma thread .. vejam e estudam o próximo exemplo.

Exemplo 9.5 O ficheiro test2.c

```
void *funcao(void *args)
{
   int x = *(int *)args;
   printf("Thread %d\n", x);
   return (NULL);
}
int main ()
{
  pthread_t threads[NUM_THREADS];
  int i, ids[NUM_THREADS];

  for (i = 0; i < NUM_THREADS; i++) ids[i]=i;

  for(i=0;i < NUM_THREADS;i++)
       pthread_create(&threads[i], NULL, funcao, &ids[i]);

  for(i=0;i < NUM_THREADS;i++) pthread_join(threads[i],NULL);
  ...
```

Uma alternativa é usar malloc (cuidado aqui com a memória alocada e "memory leeks").

```
int *id;
for(i=0;i < NUM_THREADS;i++)  {
    id = (int *)malloc(sizeof(int);
    *id = i;
    pthread_create(&threads[i], NULL, funcao, id);
}
```

Ouputs Correctos de execução programa são agora mostrados

```
alunos:~/so/cprogs/pthreads crocker$ test2
Thread 1
Thread 0
Thread 2
Thread 3
Thread 4
```

```
alunos:~/so/cprogs/pthreads crocker$ test2
Thread 0
Thread 1
Thread 2
Thread 4
Thread 3
```

Desta maneira podemos controlaro fluxo de cada thread dentro da função usando instruções de condição (if,switch etc).

Exercício 9.4

Escreve,Compile e Execute os programas "test1" e "test2".

Exercício9.5

Escreve um Programa multithreaded para calcular o valo da funcao $y=\sin^3(x)+\sqrt{\cos(x)}$. Deverá criara duas threads novas, uma para calcularf1=sin3(x) e a segunda para calcular $f2=\sqrt{\cos(x)}$. A threadprincipal deverá depois calcular o valor final (f1+f2) depois de terminação ejunção das duas threads.

Exercício9.6

Escreve,Compile e Execute um Programa multithreaded para calcular o produto de duasmatrizes quadrados de dimensão dois. Deverá criara quatro threads novas, cadauma usada para calcular um dos quarto elementos de matriz resultante. Poderáusar as estruturas de dados em baixo !

intmatrix1[2][2]={1,2,3,4}, matrix2[2][2]={3,4,7,8}, matrix_produto[2][2].