Bài 6b

LẬP TRÌNH ĐA TUYẾN (tt)

5. Đồng bộ tuyến với semaphore

a. Semaphore là gì ?

Semaphore thực sự là một cờ hiệu tương tự như mutex, nếu một tuyến cần sử dụng tài nguyên nó thông báo với semaphore.

Muốn sử dụng đối tượng semaphore, bạn cần gọi hàm **sem_init ()** để khởi tạo biến semaphore có kiểu cấu trúc **sem_t** như sau:

```
#include <semaphore>
int sem_init (sem_t* sem, int pshareed, unsined int value)
```

Để yêu cầu sử dụng tài nguyên, tuyến thực hiện gọi hàm **sem_wait ().** Sau khi sử dụng xong tài nguyên tuyến cần gọi hàm **sem_post** để trả về giá trị của semaphore.

```
#include <semaphore.h>
int sem_wait (sem_t sem);
int sem_post (sem_t sem);
```

Cả hai hàm này đều yêu cầu đối số là đối tượng sem đã được hàm **sem_init ()** tạo ra trước đó.

b. Ứng dụng của semaphore

Bài toán nổi tiếng về tranh chấp tài nguyên dễ hiểu nhất đó là bài toán "sản xuất – tiêu thụ". Bạn hình dung có hai tuyến song song. Một tuyến chịu trách nhiệm sản xuất ra sản phẩm (producer). Một tuyến có nhiệm vụ lấy sản phẩm để tiêu thụ (comsumer). Nếu sản xuất và tiêu thụ cùng nhịp với nhau thì không có vấn đề gì xảy ra. Tuy nhiên cung, cầu ít bao giờ cần đối và gặp nhau tại một điểm.

Ví dụ dưới dây giúp giải quyết vấn đề này.

pro_consumer.c

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <pthread.h>
#include <semaphore.h>

int product_val = 2; /*Sản phẩm ban đầu trong kho*/
sem_t semaphore; /*Khai báo đối tượng semaphore*/

/*Hàm thực thi tuyến*/
void * do_thread (void* data);

int main ()
{
      int res, i;
      pthread_t a_thread;
      void* thread_result;

      /*Khởi tạo đối tượng semaphore - Ở đây ta đặt giá trị cho semaphore là 2*/
      res = sem_init (&semaphore, 0, 2);
      if (res != 0)
      {
```

```
            perror ("Semaphore init error");
            exit (EXIT_FAILURE);
      }

      /*Khởi tạo tuyến đóng vai trò người tiêu thụ - consumer*/
      res = pthread_create (&a_thread, NULL, do_thread, NULL);

      if (res != 0)
      {
            perror ("Thread create error");
            exit (EXIT_FAILURE);
      }

      /*Tuyến chính đóng vai trò người sản xuất*/
      for (i = 0; i < 5; i++)
      {
            product_val++;
            printf ("Producer product_val = %d \n\n", product_val);
            /*Tăng giá trị semaphore - thông báo sản phẩm đã được đưa thêm vào kho*/
            sem_post (&semaphore);
            sleep (2);
      }

      printf ("All done\n");
      exit (EXIT_SUCCESS);
}

/*Cài đặt hàm thực thi tuyến*/
void* do_thread (void* data)
{
      printf ("Consumer thread function is running ...\n");
      while (1)
      {
            /*Yêu cầu semaphore cho biết có được phép lấy sản phẩm khỏi kho hay không*/
            sem_wait (&semaphore);
            product_val--;
            printf ("Consumer product_val = %d \n", product_val);
            sleep (1);
      }
      pthread_exit(NULL);
}
```

Biên dịch chương trình sẽ thu được kết quả như sau:

```
$./prod_consumer
Producer product_val = 3

Cosumer thread function is running ...
Consumer product_val = 2
Consumer product_val = 1
Producer product_val = 2

Consumer product_val = 1
Consumer product_val = 0
Producer product_val = 1

Consumer product_val = 0
```

```
Producer product_val = 1

Consumer product_val = 0
Producer product_val = 1

Consumer product_val = 0
```

6. Hủy bỏ và chấm dứt tuyến

Đôi khi chúng ta muốn một tuyến có thể yêu cầu tuyến khác chấm dứt khi đang thực thi. Bạn dùng hàm **pthread_cancel ()** để gởi tín hiệu đến tuyến cần hủy.

```
#include <phread.h>
int pthread_cancel (pthread_t thread);
```

- Thiết lập trạng thái chấm dứt của tuyến bằng hàm **pthread_setcancelstate ()**

```
#include <pthread.h>
int pthread_setcancelstate (int state, int *oldstate);
```

- Thiết lập kiểu chấm dứt bằng hàm **pthread_setcanceltype ()**

```
#include <pthread.h>
int pthread_setcanceltype (int state, int *oldstate);
```

thread_cancel.c

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <pthread.h>

/*Hàm thực thi tuyến*/
void * do_thread (void* data);

int main ()
{
      int res, i;
      pthread_t a_thread;
      void* thread_result;

      /*Tạo tuyến với giá trị thiết lập mặc định*/
      res = pthread_create (&a_thread, NULL, do_thread, NULL);
      if (res != 0)
      {
            perrror ("Thread create error");
            exit (EXIT_FAILURE);
      }
      sleep (3);

      /*Gởi tín hiệu yêu cầu chấm dứt tuyến a_thread*/
      printf ("Try to cancel thread ...\n");
      res = pthread_cancel (a_thread);

      if (res != 0)
      {
            perror ("Thread cancel error");
            exit (EXIT_FAILURE);
```

```
	}

	/*
	Do mặc định tuyến tạo ra với trạng thái PTHREAD_CANCEL_DEFERRED nên
	tuyến chỉ thực sự chấm dứt khi bạn gọi hàm pthread_join ()
	*/
	printf ("Waiting for thread to finish ...\n");
	res = pthread_join (a_thread, &thread_result);
	if (res != 0)
	{
		perror ("Thread waiting error");
		exit (EXIT_FAILURE);
	}

	printf ("All done \n");
	exit (EXIT_SUCCESS);
}

/*Cài đặt hàm thực thi tuyến*/
void* do_thread (void* data)
{
	int i, res;
	res = pthread_setcancelstate (PTHREAD_CANCEL_ENABLE, NULL);
	if (res != 0)
	{
		perror ("Thread set cancel state fail");
		exit (EXIT_FAILURE);
	}

	res = pthread_setcanceltype (PTHREAD_CANCEL_DEFERRED, NULL);
	if (res != 0)
	{
		perror ("Thread set cancel type fail");
		exit (EXIT_FAILURE);
	}

	printf ("Thread function is running ... \n");
	for (i = 0; i < 10; i++)
	{
		printf ("Thread is still running (%d) ...\n", i);
		sleep (1);
	}
	pthread_exit (NULL);
}
```

Biên dịch chương trình từ dòng lệnh

```
$gcc thread_cancel.c - o thread_cancel -lpthread
```

Chạy chương trình với kết xuất như sau

```
$./thread_cancel
Thread function is running ...
Thread is still running (0) ...
Thread is still running (1) ...
Thread is still running (2) ...
Try to cancel thread ...
Waiting for thread to finish ...
Thread is still running (3) ...
```

All done