Real-Time Programming
Barriers

Rola El Osta

CCNE Department
Lebanese University

Barriers in pthreads

- pthreads allows us to create and manage threads in a program. When multiple threads are working together, it might be required that the threads wait for each other at a certain event or point in the program before proceeding ahead.

  Let us say we have four threads, each of which is going to initialize a global variable. The 4 variables in turn might be used by all the four threads. Thus it would be feasible that all the threads wait for each other to finish the initialization of the variables before proceeding.

  Such operations can be implemented by adding a barrier in the thread. A barrier is a point where the thread is going to wait for other threads and will proceed further only when predefined number of threads reach the same barrier in their respective programs.

- To use a barrier we need to use variables of the type pthread_barrier_t. A barrier can be initialized using the function pthread_barrier_init, whose syntax is

- int pthread_barrier_init(pthread_barrier_t *restrict barrier,const pthread_barrierattr_t *restrict attr, unsigned count);

- Arguments:

barrier: The variable used for the barrier

attr: Attributes for the barrier, which can be set to NULL to use default attributes

count: Number of threads which must wait call pthread_barrier_wait on this barrier before the threads can proceed further.

- Once a barrier is initialized, a thread can be made to wait on the barrier for other threads using the function pthread_barrier_wait whose syntax is

```
int pthread_barrier_wait(pthread_barrier_t *barrier);
```

Where the barrier is the same variable initialized using pthread_barrier_init

- A thread will keep waiting till the "count" number of threads passed during init do not call the wait function on the same barrier.

- In the example below, we create 4 threads, and each thread assigns a value to one of he global variables and then later all the threads print the value of all the four variables. Thus to be able to print the value of all the variables the threads have to wait until all threads have not finished assigning the values. Thus we can make use of barriers to make the threads wait.
- The init function :
  pthread_barrier_init(&our_barrier,NULL,4);

  initializes the barrier "our_barrier" to wait for four threads.

. Thus till all the four threads have not called the pthread_barrier_wait the other threads will continue to wait.

. Once pthread_barrier_wait has been called by 4 threads, all the threads will continue their execution of program.

- A barrier can be destroyed using the function pthread_barrier_destroy, whose syntax is :
  int pthread_barrier_destroy(pthread_barrier_t *barrier);
- But note that the barrier should be destroyed only when no thread is executing a wait on the barrier.

Example:

- See pthread_barrier.c

- Save the program as pthread_barrier.c and compile it :

- $gcc -lpthread -o pthread_barrier pthread_barrier.c
- $ ./pthread_barrier
- Enter value for t1: 10
- Enter value for t2: 20
- Enter value for t3: 30
- Enter value for t4: 40

- values entered by the threads are 10 20 30 40

- values entered by the threads are 10 20 30 40

- values entered by the threads are 10 20 30 40

- values entered by the threads are 10 20 30 40

- To stress on usefulness of the barrier, we can change barrier initialization in the above code to wait for only 3 threads i.e. pthread_barrier_init(&our_barrier,NULL,3);
- And remove the pthread_barrier_wait from the the 4th thread.

After the above changes, if we compile and execute the code the output would be

Created threads

Enter value for t1: 10

Enter value for t2: 20

Enter value for t3: 30

values entered by the threads are 10 20 30 0

values entered by the threads are 10 20 30 0

values entered by the threads are 10 20 30 0

Enter value for t4: 40

values entered by the threads are 10 20 30 40

The first three threads could not get the value entered by the fourth thread as they moved over the barrier as soon as three threads executed the wait function.