Condition Variables

and thread cancellation

Condition Variables

Condition variables provide yet another way for threads to synchronize.

While mutexes implement synchronization by controlling **thread access to data**, condition variables allow threads to synchronize **based upon the actual value of data**.

Condition Variables

Without condition variables, the programmer would need to have threads continually polling (possibly in a critical section), to check if the condition is met. This can be very resource consuming since the thread would be continuously busy in this activity. A condition variable is a way to achieve the same goal without polling.

A condition variable is always used in conjunction with a mutex lock.

Condition Variables

Thread A

- Do work up to the point where a certain condition must occur (such as "count" must reach a specified value)
- Lock associated mutex and check value of a global variable
- Call **pthread_cond_wait()** to perform a blocking wait for signal from Thread-B. Note that a call to **pthread_cond_wait()** automatically and atomically unlocks the associated mutex variable so that it can be used by Thread-B.
- When signalled, wake up. Mutex is automatically and atomically locked.
- Explicitly unlock mutex
- Continue

Thread B

- Do work
- Lock associated mutex
- Change the value of the global variable that Thread-A is waiting upon.
- Check value of the global Thread-A wait variable. If it fulfills the desired condition, signal Thread-A.
- Unlock mutex.
- Continue

Condition Variables

Condition variables must be declared with type **pthread_cond_t**, and must be initialized before they can be used. There are two ways to initialize a condition variable:

1. Statically, when it is declared. For example:
   pthread_cond_t myconvar = PTHREAD_COND_INITIALIZER;
2. Dynamically, with the **pthread_cond_init()** routine.

Waiting and Signaling on Condition Variables

pthread_cond_wait() blocks the calling thread until the specified *condition* is signalled.

This routine should be called while *mutex* is locked, and it will automatically release the mutex while it waits.

After signal is received and thread is awakened, *mutex* will be automatically locked for use by the thread.

The programmer is then responsible for unlocking *mutex* when the thread is finished with it.

Example

condition_var.c

Example

Condition Variables and Prime numbers

find_prime_number_seq_0.c

Exercise 4c

Create parallel version of program for finding prime number under certain position. Use at least **two threads**.

template: find_prime_number_seq_4.c

Exercise 4d

Create a program that starts N threads. Each thread does some job - gets and increments current value from global counter and processes it (just “sleep” for now). Create separate thread that monitors how fast worker threads do the job. Use a **condition variable** for monitoring thread synchronization.

Template: pthread_control_1.c

Thread cancellation

#include <pthread.h>

int pthread_cancel (pthread_t thread);

0 - if successful, otherwise the parameter **thread** is incorrect

Cancellation policy

The cancellation state of the thread (enabled by default)

#include <pthread.h>

int pthread_setcancelstate (int state, int *oldstate);

state: **PTHREAD_CANCEL_ENABLE** или **PTHREAD_CANCEL_DISABLE**

Type of cancelation

- Asynchronous:

At any time

- Deferred:

Only at certain moments

Change the type of cancellation

#include <pthread.h>

int pthread_setcanceltype (int type, int *oldtype);

type: **PTHREAD_CANCEL_ASYNCHRONOUS** или **PTHREAD_CANCEL_DEFERRED**

Check point for deferred thread cancellation

```
#include <pthread.h>

void pthread_testcancel(void);
```

Release resources before completion

pthread_cleanup_push(void *func, void *arg)

Run when:

- Thread cancel
- Exit via pthread_exit
- pthread_cleanup_pop

Example

Delayed thread stop

pthread2.c

Exercise 4e

Add ability to change the current number of working threads. User can add or remove some number of threads in real-time.

Question:

- Check the calculation speed based on the threads number

Template: pthread_control_2.c

Exercise 4f

Change program from abstract calculation task (“sleep”) to prime number calculation task.

Template: pthread_control_2.c (last version)