POSIX Condition Variables ADT

- *One actual Attributes*
  * 1. CV-list = list ids of thread waiting on CV
  - Domain: empty-list, list w/ 1 tid, ...
- *Two logical Attributes*
  * 1. Boolean Condition C
  - Condition variable used to wait for C becoming true
  - C can be a complex condition
  * 2. CV-mutex = a mutex associated with CV
  - Domain: occupied, unoccupied
- *Logical Operations*
  * t.wait(CV, mutex) -
  - Add thread t to CV-list, unlock the mutex
  - Typically while associated condition C is false
  - Condition C is tested explicitly!
  * t.signal(CV) -
  - Wake up a thread from CV-list
  - Woken up thread will test associated condition C
  - and may t.wait(CV, mutex) if C is false

Condition Variables Operations- Syntax

- *Synopsis: pp. 382*

- *CV.wait*

```
int pthread_cond_wait(pthread_cond-t *cond,
                      pthread_mutex_t *mutex);
```

- *CV.signal*

```
int pthread_cond_signal(pthread_cond-t *cond);
```

- *Initialization*

  * Static Initialization - macro

    ```
    pthread_cond_t v = PTHREAD_COND_INITIALIZER;
    ```

  * Run-time Initialization - system call

    ```
    int pthread_cond_init(pthread_cond_t *cond,
            const pthread_condattr_t *attr);
    ```

- *Recycling*

```
int pthread_cond_destroy(pthread_cond-t *cond);
```

Condition Variables (CV) : Purpose/Usage

- *Main purpose*
  * wait on a complex condition
  * Ex. C1 = (buffer is not full)
  * Ex. C2 = (producer is done) and (buffer is empty)
- *Example: Program 10.6 (pp. 384-5)*
  * See function producer()
  - code-fragment before/after put_item()
  * See function consumer()
  - code-fragment before/after get_item()

Condition Variables (CV) : Protocol

- *Note: Protocol of usage for (CV + mutex)*
  * Steps on pp. 379
  * Rules - bullets on pp. 383
- *Rules:*
  * (1) Get mutex lock M before testing predicate
  * (2) Retest predicate after returning from cond_wait()
  - while (not predicate) cond_wait(&V, &M),
  * (3) Get mutex before changing variables affecting condition
  * (4) Get mutex before calling cond_signal(), cond_broadcast()
  * (5) Hold mutex for only a short time
  - Release mutex via mutex_unlock() or cond_wait()

Exercise on Condition Variables

- *Consider Code fragment for thread 1.*

```
lock_mutex(&m); /*A */
while (x !=y) /* B */
cond_wait(&v, &m); /* C */
/* do some stuff related to x and y */ /* D */
unlock_mutex(&m); /* E */
```

- *Code fragment for Thread 2 code*

```
lock_mutex(&m); /* F */
x++; /* G */
cond_signal(&v); /* H */
unlock_mutex(&m); /* I */
```

- *Ex. Suppose x = 0 and y = 2 initially.*
  - Q? What happens after interleaving A, B, C, F, G, H, I ?
  - Q? Which statement does thread 1 execute next?
- *Q? Are the following interleaving possible?*
  - (i) A B C F G H B C I
  - (ii) A B F G C H

Condition Variables (CV) : Exercises

- *Compare Programs 10.6 (pp. 384-5) and 10.4 (pp. 376)*
  - * How can CVs simulate semaphore operations?
  - * Which condition does producer wait on in each program?
  - * Which condition does consumer wait on in each program?
  - * How consumers are allowed in each program?
  - * How producers are allowed in each program?
- *Compare CVs with semaphores:*

- Let C1 = (buffer is not full)
- and C2 = (producer is done) and (buffer is empty)
  - * Can condition C1 be monitored by a Semaphore?
  - * Can condition C2 be monitored by a Semaphore?
  - * Does semaphore.wait() test for associated condition, e.g. C1?
  - * Does CV.wait() test for associated condition, e.g. C1?
  - * Does program using semaphore always need mutexes?

Condition Variables (CV) vs. Semaphores

- *Why sempahores do not monitor complex conditions?*
  - * Two semaphores to wait for:
  - - Buffer empty, Producer is done
  - * Recipe for indefinite wait
  - - since the events are not ordered!
- *How do CVs differ from semaphores?*
  - * Semaphores monitor simple conditions, e.g. C1
  - * Semaphore.wait() implicitly tests condition (count==0)
  - - and block the thread
  - * CV.wait() only blocks the thread
  - - condition testing is explicit in code
  - * CV is used with mutex
- *Q? What is the associated mutex used for?*
  - * Protect two critical sections
  - * (a) wait(CV, mutex); acquire resource
  - * (b) release resource; signal(CV);

Departing Note on CVs

- *Honor system*
  * Each thread must follow protocol
- *Complex protocol*
  * Use simpler mechanisms (e.g. mutex, semphore) if possible
  * Hide shared data-structures and
  - associated condition variables inside an ADT
- *Note- CV is often generated by compiler*
  * monitors in high level language constructus
  * Java synchronized classes, methods = critical sections

10.4 Threads and Rest of POSIX

- *Threads interact with everything!*
  * There are many issues
  * Let us review a few representative ones!
- *Threads and Processes*
  * 1. Is a system calls at process level or thread level?
  - exit, sleep, thread_exit, wait, thread_join, ...
  * 2. fork() in a mutli-threaded program
  - How many threads are in the child process?

10.4 Threads and Rest of POSIX

- *Threads and Files*
  * 1. Is a system calls at process level or thread level?
  - open, read, write, ioctl, close
  * 2. Threads in a process shared files, file descriptors, FDTs
  - Avoid conflicts in access to shared resources
  - via synchronization (Ch. 10) or careful division (Ch. 9)
- *Threads and Signals*
  * 1. Is a system calls at process level or thread level?
  - kill, sigprocmask, sigaction, sigsuspend, pause, ...
  * 2. Can each thread have diferent mask?
  * 3. Can each thread have diferent handlers for a signal?
  * 4. Which thread receives a signal to the process?
  * 5. How threads affects signal handlers?

10.4 Signal Handling and Threads

- *1. Is a system calls at process level or thread level?*

  * Chapter 5; system calls were at process level!
  - kill, sigprocmask, sigaction, sigsuspend, pause, ...

  * Chapter 10.4: thread level system calls were for signals
  - pthread_kill(), pthread_sigmask()
  - See pp. 386 for synopsis

- *2. Can each thread have diferent mask?*

  * Signal masks can be thread specific

  * A thread can block a signal while others can receive it!

  * Parameter 1 (how) = SIG_BLOCK / SIG_UNBLOCK / SIG_SETMASK

  * Parameter 2 = new mask

  * Parameter 3 = old mask

  * Semantics similar to sigprocmask()

  ```
  int pthread_sigmask(int how, const sigset_t *set,
                      sigset_t *oset)
  ```

10.4 Signal Handling and Threads

- *3. Can each thread have diferent handlers for a signal?*
  * NO, signal handlers are process wide
- *4. Which thread receives a signal ?*
  * Three cases (Table 10.1, pp. 386)
  * Synchronous signal (SIGFPE) : the thread causing it
  * Asynchronous signal (SIGINT) : Any thread not blocking it
  * Designated thread if signal generated by
    int pthread_kill(pthread_t thread, int sig)
- *Signal handler for asynchronous signals - common designs*
  * 1. Block the signal in its handler via sigaction()
  * 2. Designate a thread to handle asynchronous signals
  - Other threads will block asynchronous signals
  - Ex. Program 10.8 (pp 392-4)
  - sigusr1_thread() handles all signals

10.4 Signal Handlers and Threads

- *5. How threads affects signal handlers?*
  * Consider Signal S caught by thread T1
  * Signal S is blocked in its handler H for T1
  * However S may not be blocked for threads T2, T3
  * Thread T2 may enter the handler H as well
- *Handler H should be reentrant function!*
  * Use only reentrant system calls, libraries
  * Either avoid use of global variables
  - Or use synchronization (critical section)
  - Example: Program 10.7 (pp. 388-390)
  - See catch_sigusr1() on pp. 388