Operating Systems: Pthreads and Synchronization Tools in C

Neerja Mhaskar

Department of Computing and Software, McMaster University, Canada

Acknowledgements: Material based on the textbook Operating Systems Concepts (Chapters 6 & 7)

Pthreads

- May be provided either as user-level or kernel-level
- Pthreads, is a POSIX standard API for thread creation and synchronization
  - Specification **not** implementation
- API specifies behavior of the thread library, implementation is up to development of the library
- Common in UNIX operating systems (Solaris, Linux, Mac OS X)
- On Linux, pthread library implements the 1:1 model
- Function names start with “pthread_”

Pthreads

- To be able to create threads in your C program you need to include the `pthread.h` header file.
- Each thread has a unique thread ID. To create thread IDs for your threads in your program you should use the `pthread_t` data type.
- Thread attributes should be created/modified using `pthread_attr_t` data structure.
- Declare and code the function in which the thread begins control. For an example see the `runner()` function on the next slide.
- To create threads, use `pthread_create()` function and pass in the necessary parameters.
- For the parent thread to output the sum after all summing threads have exited, it is important that you use the `pthread_join()` function.

Pthreads Example

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

#include <stdlib.h>

int sum; /* this data is shared by the thread(s) */
void *runner(void *param); /* threads call this function */

int main(int argc, char *argv[])
{
  pthread_t tid;  /* the thread identifier */
  pthread_attr_t attr; /* set of thread attributes */

  /* set the default attributes of the thread */
 pthread_attr_init(&attr);
  /* create the thread */
  pthread_create(&tid, &attr, runner, argv[1]);
  /* wait for the thread to exit */
  pthread_join(tid,NULL);

  printf("sum = %d\n",sum);
}
```

Pthreads Example (Cont.)

```
/* The thread will execute in this function */
void *runner(void *param)
{
  int i, upper = atoi(param);
  sum = 0;

  for (i = 1; i <= upper; i++)
    sum += i;

  pthread_exit(0);
}
```

Pthreads Code for Joining 10 Threads

```
#define NUM_THREADS 10

/* an array of threads to be joined upon */
pthread_t workers[NUM_THREADS];

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

Pthreads Synchronization

- Pthreads API is available for programmers at the user level and is not part of any particular kernel.
  - Provides mutex locks, condition variables, and read–write locks for thread synchronization.

`pthread_mutex_t` data type for mutex locks.

- mutex initialized with **`pthread_mutex_init()`** function.

```
pthread_mutex_lock();

pthread_mutex_unlock();

pthread_mutex_destroy();
```

Pthread Mutex Example

```
/*Declaring mutex*/

pthread_mutex_t mutex;

/*Initialize mutex returns 0

if mutex initialized with no errors.*/

if (pthread_mutex_init(&mutex, NULL) !=0){

      printf("Error in initializing mutex \n");

}

/*Acquire mutex lock*/

pthread_mutex_lock(&mutex);

/*Critical Section*/

/*Release mutex locks*/

pthread_mutex_unlock(&mutex);
```

Null indicates default mutex attributes passed.

Semaphores in C

- Semaphores are not part of Pthread standards and instead they belong to the POSIX SEM extension.
- To use semaphores in C on a Linux machine include ‘semaphore.h’ header file:
  - **`#include <semaphore.h>`**
  - **`sem_t`** data type for semaphores.
  - semaphores created with **`sem_init()`** function.
    - **`sem_init()`** - Takes 3 arguments:
      1. Pointer to the semaphore
      2. Flag indicating the level of sharing
      3. The semaphore’s initial value
  - **`sem_wait();`**
  - **`sem_post();`**

Semaphores in C

```
/*Declaring Semaphore*/

sem_t sem;

/*Initialize Semaphore*/

if (sem_init(&sem, 0, n) !=0){

     printf("Error in initializing empty semaphore \n");

}

/* acquire the semaphore */

sem_wait(&sem);

/* critical section */

/* release the semaphore */

Sem_post(&sem);
```

sem_init arguments
1. pointer to the semaphore
2. flag indicating the level of sharing
3. The semaphore's initial value

Returns 0 when semaphore created with no errors, otherwise returns non zero value.

POSIX Condition Variables

- **Condition variables** in Pthreads behave similarly to those described under Monitors.
- Since Pthreads is typically used in C programs—and since C does not have a monitor—we accomplish locking by associating a condition variable with a mutex lock.
- Condition variables in Pthreads use the
  - `pthread_cond_t` data type, and
  - `pthread_cond_init()` function to initialize it.
- The following code creates and initializes a condition variable as well as its associated mutex lock:

```
pthread_mutex_t mutex;

pthread_cond_t cond_var;

pthread_mutex_init(&mutex,NULL);

pthread_cond_init(&cond_var,NULL);
```

Null indicates default attributes passed

POSIX Condition Variables Contd…

- `pthread_cond_wait()` function – used to wait/block on a condition variable. It takes two parameters
  - Pointer to condition variable
  - Pointer to mutex.
    - This mutex must be locked by the calling thread before the `pthread_cond_wait()` is used, or undefined behaviour will result.
- `pthread_cond_wait()` atomically unlocks the mutex and wait on the condition variable.
- When condition variable is signaled, before returning to the calling thread, the the mutex lock is acquired on entrance to **pthread_cond_wait()**

POSIX Condition Variables Contd…

- Example: A thread waiting for the condition `a == b` to become true using a Pthread condition variable:

```
pthread_mutex_lock(&mutex);
while (a != b)
     pthread_cond_wait(&cond_var, &mutex);
pthread_mutex_unlock(&mutex);
```

POSIX Condition Variables Contd…

- `pthread_cond_signal()` function
  - Takes in a pointer to the condition variable.
  - used to signal any one thread waiting on the condition variable.
  - If no thread waiting on the condition variable nothing happens.
- Example:

```
pthread_mutex_lock(&mutex);

a = b;

pthread_cond_signal(&cond_var);

pthread_mutex_unlock(&mutex);
```