An introduction to the Posix Thread API

- General introduction
- Creation and termination
- Mutex locks
- An example: parallel sum or inner product

Threads

➤ Mode of programming for shared memory [shared address space or symmetric multi-processing (SMP)]
➤ Very common – supported by all vendors. Part of unix standard.
➤ Low-level
➤ Helps understand issues with racing, synchronization, etc.
➤ Here: we will provide basic overview + cover an example.

Pros: simple approach –
Cons: Limited to SMPs. Gets complicated for longer codes

The basic commands

See [https://computing.llnl.gov/tutorials/pthreads/](https://computing.llnl.gov/tutorials/pthreads/) among many resources for details.

“a thread is defined as an independent stream of instructions that can be scheduled to run as such by the operating system.” (source: above)

➤ Initially, main() comprises a single thread
➤ Programmer can instruct the program to start threads that execute independently.
➤ *However*: you are responsible for coordinating the concurrent accesses/ modifications of memory variables by different threads
➤ Once initiated a thread can itself create other threads

Thread creation

`pthread_create (thread, attr, thread_fun, arg)`

thread is of type `pthread_t = (unique) identifier of thread`
attr is of type `pthread_attr_t = may be used to set thread attributes`. thread_fun is a function pointer. The thread will execute this function after creation.
arg is of type `*void`. This argument is passed to the function `thread_fun`. If more than one argument use a struct
**pthread_join**: blocks calling thread until specified thread ends.

Allows to synchronize

`pthread_join(threadid, status)`

`threadid` is of type `pthread_t` = identifier of thread

`status` is of type `void**`.

### A common mistake

Function to be called by each thread. It will print a message that contains the thread id.

```c
void *P_hello(void *arg){
    int * thrId = (int *) arg;
    printf("\n--> Hello from thread number: %d \n",* thrId);
    pthread_exit(NULL);
}
```

Use the attribute for declaring thread as **joinable**

```c
pthread_attr_t attr; // declare
pthread_attr_init(& attr); // initialize
pthread_attr_setdetachstate(& attr, PTHREAD_CREATE_JOINABLE); // set
/* at end */
pthread_attr_destroy(& attr); // Free attr
```

```c
int main(int argc, char *argv[]){
    /* Adapted from llnl online tutorial. A basic "hello world" Pthreads program showing thread creation + termination ------------------------*/
    #include <pthread.h>
    #include <stdio.h>
    #include <stdlib.h>
    #define NUM_THREADS 8
    pthread_t thrdNUM_THREADS[];
    int rc, t;
    for(t=0;t<NUM_THREADS;t++){
        rc= pthread_create(& thrdt ,NULL , P_hello ,&t);
        if (rc){
            printf("ERROR: return code: %d\n",rc);
            exit(-1);
        }
    }
    pthread_exit(NULL);
}
```

Pay attention to argument passed to `P_hello` – last arg. of `pthread_create`
**Discussion**

1. Try Running this code. What happens?
   - Try again several times. Can you explain what happens?
2. Also: run the drive without the last `pthread_exit`. Can you explain?

---

**Shared variables and mutual exclusion: Mutexes**

- Accessing shared variables requires careful control - if data is altered by a thread: If several threads modify a shared variable, we need to make sure only one thread accesses it at a time.

- **Mechanism**: Mutual Exclusion or Mutex.

```c
// Declare as
pthread_mutex_t mutex1;
// then set as
pthread_mutex_init(mutex1, attr);
// or with static initialization
pthread_mutex_t mutex1 = PTHREAD_MUTEX_INITIALIZER;

// lock this critical section:
pthread_mutex_lock(mutex1);
// do the work needed in this section
// Nobody else can modify variables in this section
// then unlock
pthread_mutex_unlock(mutex1);
// at completion free:
pthread_mutex_destroy(mutex1)
```

---

**Example: parallel sum of n numbers**

- We want to sum the n numbers of an array `a[0:n-1]` by dividing the sums into `p` subsums which are added in a common location in memory. Shared variable `SUM` will contain the final sum.

  ... Each thread computes its subsum
  ... locks the code section that updates `SUM`
  ... adds subsum to `SUM`
  ... Unlocks critical section
  ... and exit thread.
Main
1 int main ( int argc , char * argv [] ) {
2 /* --------------------main ------------------------*/
3 int i, n = ... . sum );
32 free (a); free ( thrNum );
33 pthread_mutex_destroy (& mutexsum );
34 pthread_exit ( NULL );
35 }
4-15

Function
sum_dat
1 /* --- Define global variables and a mutex */
2 #define NUMTHRDS 8
3 #define VECLEN 50
4 pthread_t callTid[NUMTHRDS];
5 thread_mutex_t mutexsum;
6 */----------- function sum -- activated when thread is created. */
7 void * sum_dat(void * arg){
8 /*----------- local variables */
9 int i, start, end , * blkNum , len ;
10 double mysum , *x;
11 blkNum = (int*)arg;
12 len = SUMST. veclen;
13 start = (*blkNum)*len;
14 end = start + len;
15 x = SUMST.a;
16 } sumstr SUMST ;
17 /* ---------------- Lock a mutex before updating value in the shared struct */
18 pthread_mutex_unlock (&mutexsum);
19 SUMST. sum += mysum;
20 }--------- update done, unlock*/
21 pthread_mutex_unlock (&mutexsum);
22 printf("-- Local sum in thread %d is %10.2 f total %10.2 f\n",...
23 +blkNum,mysum,SUMST. sum);
24 printf(" len %d %d %d %5.2 f 
",len , start ,end ,x
25 * blkNum , mysum , SUMST . sum );
26 /*------------------- sum */
27 mysum = 0;
28 for (i=start; i<end ; i++)
29 mysum += x[i];
30 }--------- done with this thread */
31 pthread_exit((void*) 0);
32 }
3-15

Declarations
1 /* Main program generates data (a vector) of length
2 NUMTHRDS*VECLLEN -- then generates threads that
3 call the sum_dat function. Upon completion the
4 result is printed. Main thread will wait for all
5 threads to complete. This code also illustrates
6 thread *attributes*. It sets the threads to be
7 *joinable* (allows the main thread to join with
8 the threads it creates).
9 */
10
11 * illustrates the use of mutex variables
12 * in a threads program to sum n numbers
13 * ----------------------------------------
14 */
15 #include <pthread.h>
16 #include <stdio.h>
17 #include <stdlib.h>
18 /* Data is passed to threads through the
19 following struct. */
20 typedef struct {
21 double *a;
22 double sum;
23 int veclen;
24 } sumstr , *SumPtr ;
25
26 /* --- Define global variables and a mutex */
27 int i, n = VECLEN *NUMTHRDS , * thrNum , *status ;
28 double *a ;
29 pthread_attr_t attr ;
30 #define NUMTHRDS 8
31 #define VECLEN 50
32 pthread_t callTid[NUMTHRDS];
33 thread_mutex_t mutexsum;
34 */----------- function sum -- activated when
35 thread is created. */
36 void * sum_dat(void * arg){
37 /*----------- local variables */
38 int i, start, end , * blkNum , len ;
39 double mysum , *x;
40 blkNum = (int*)arg;
41 len = SUMST. veclen;
42 start = (*blkNum)*len;
43 end = start + len;
44 x = SUMST.a;
45 } sumstr SUMST ;
46 /* ---------------- Lock a mutex before
47 updating value in the shared struct */
48 pthread_mutex_unlock (&mutexsum);
49 SUMST. sum += mysum;
50 }--------- update done, unlock*/
51 pthread_mutex_unlock (&mutexsum);
52 printf("-- Local sum in thread %d is %10.2 f total %10.2 f\n",...
53 +blkNum,mysum,SUMST. sum);
54 printf(" len %d %d %d %5.2 f 
",len , start ,end ,x
55 * blkNum , mysum , SUMST . sum );
56 /*------------------- sum */
57 mysum = 0;
58 for (i=start; i<end ; i++)
59 mysum += x[i];
60 }--------- done with this thread */
61 pthread_exit((void*) 0);
62 }
6-15