

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

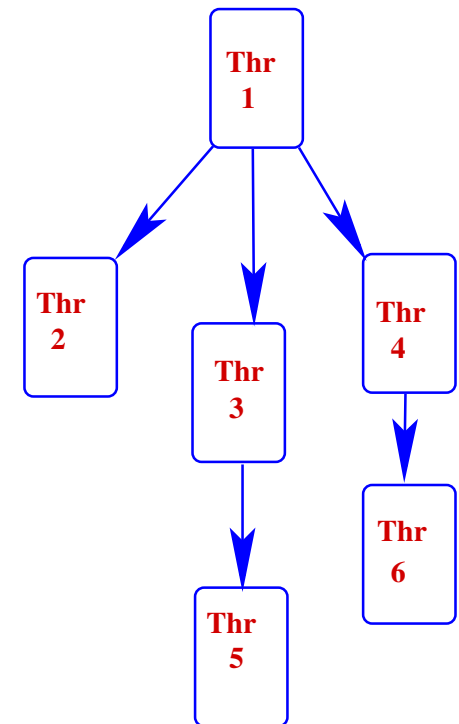
The basic commands

See <https://computing.llnl.gov/tutorials/pthreads/>

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.

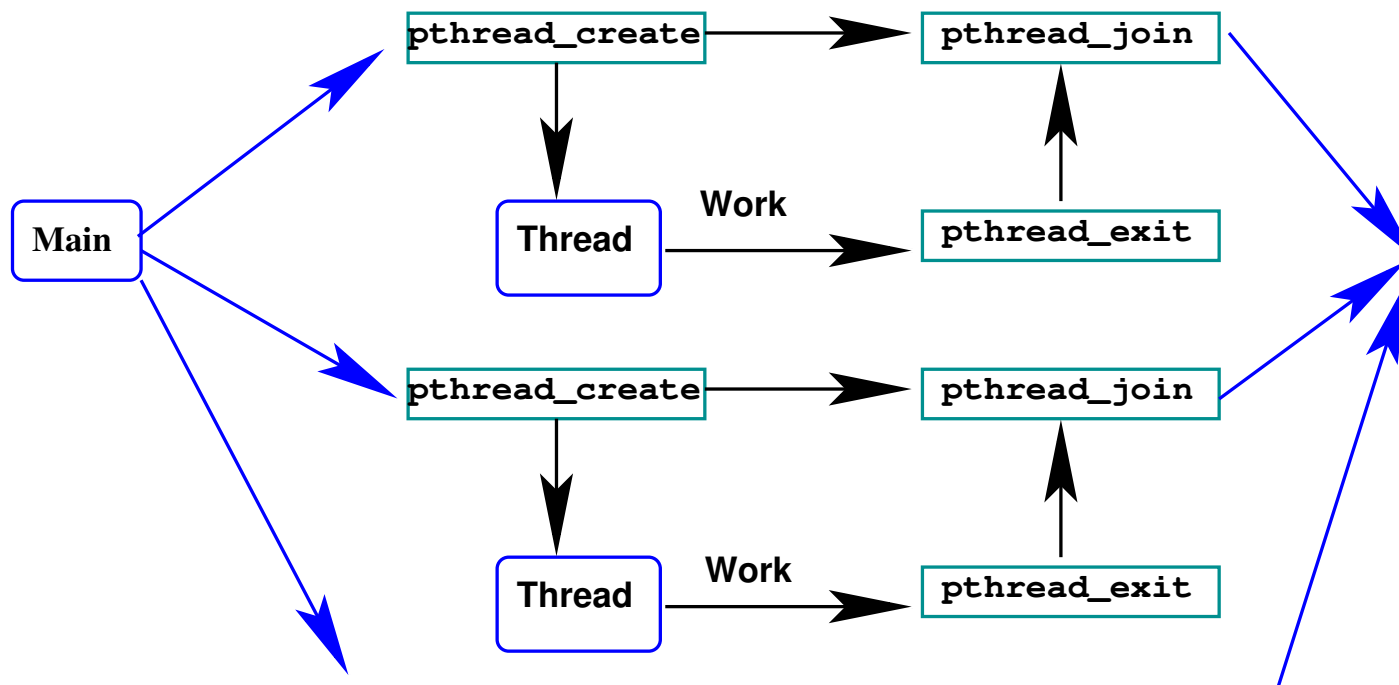
arg is of type **void*. This argument is passed to the function *thread_fun*.
If more than one argument use a struct

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**`.



Use the attribute for declaring thread as *joinable*

```
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
```

A common mistake

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

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

```

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 thrd[NUM_THREADS];
    int rc, t;
    for(t=0;t<NUM_THREADS;t++){
        rc=pthread_create(&thrd[t], 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

 Try Running this code.. What happens?

➤ Try again several times. Can you explain what happens?

 Also: run the driver without the last `pthread_exit`. Can you explain?

- A thread can finish on its own if parent thread (e.g., main) does not need it to join at completion.
- In this case: declare as 'detached'

```
1  /*----- declare attribute */
2  pthread_attr_t attr;
3  /*----- initialize it */
4  pthread_attr_init(&attr);
5  /*----- set as detached */
6  pthread_attr_setdetachstate(&attr,
7                             PTHREAD_CREATE_DETACHED);
8
9  /*----- create thread with attribute */
10 pthread_create(&threadId, attr, xfun, (void *)arg);
11 .....
12 /*----- instead of join issue detach */
13 pthread_detach(threadId);
14 /*----- do not forget: free attribute */
15 pthread_attr_destroy(&attr);
```

Other functions:

pthread_attr_getdetachstate (attr,detachstate)

pthread_attr_setdetachstate (attr,detachstate)

Shared variables and mutual exclusion: Mutexes

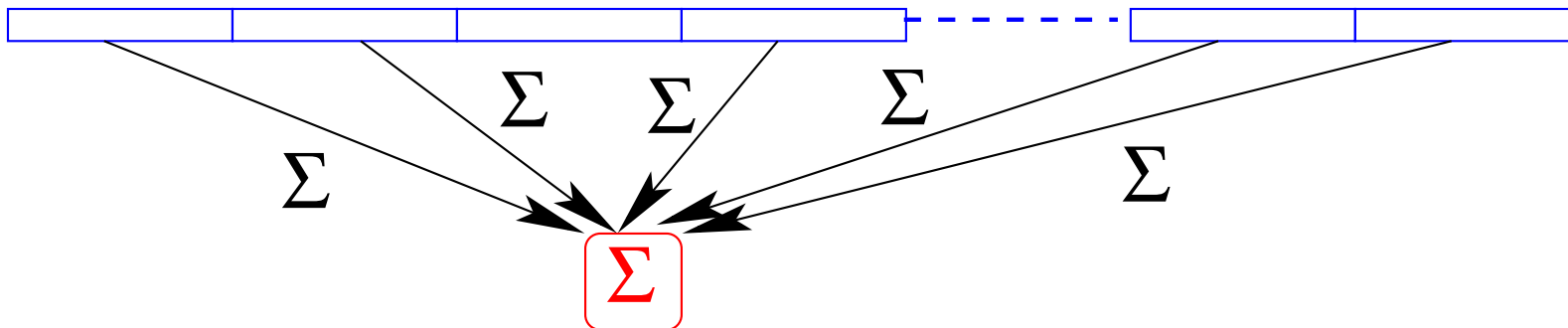
- Accessing shared variables requires careful control - if data is altered by a thread: If several threads modify a shared variable, we need to make sure only one thread accesses it at a time
- Mechanism: Mutual Exclusion or Mutex.

```
1 // Declare as
2 pthread_mutex_t mutex1 ;
3 // then set as
4 pthread_mutex_init(mutex1, attr) ;
5 // or with static initialization
6 pthread_mutex_t mutex1 = PTHREAD_MUTEX_INITIALIZER;
7
8 //lock this critical section:
9 pthread_mutex_lock (mutex1);
10 // do the work needed in this section
11 // Nobody else can modify variables in this section
12 // then unlock
13 pthread_mutex_unlock (mutex1)
14 // at completion free:
15 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.



```

1 /*
2 ! Main program generates data (a vector) of length n -- then
3 ! generates threads that call the sum_mtx function to
4 ! compute partial sums and sum them. Upon completion the
5 ! result is printed. Main thread will wait for all threads
6 ! to complete. This code also illustrates thread
7 ! *attributes*. It sets the threads to be *joinable*
8 ! (allows the main thread to join with the threads it
9 ! creates). -----*/

```

Declarations

```

1 /*-----
2 * illustrates the use of mutex variables
3 * in a threads program to sum n numbers
4 -----*/
5 #include <pthread.h>
6 #include <stdio.h>
7 #include <stdlib.h>
8 /*-----Data is passed to
9 threads through the following struct */
10 typedef struct {
11     double    *a;
12     double    sum;
13     int       loclen;
14     int       totlen;
15 } sumstr, *SumPtr;
16 sumstr SUMST;

```

Function sum_dat

```

1 #define NUMTHRDS 8
2 #define VECLEN 78
3 pthread_t callThd[NUMTHRDS];
4 pthread_mutex_t mutexsum;
5 /*----- function sum -- activated when
6                      thread is created. */
7 void *sum_mtx(void *arg){
8 /*----- local variables */
9     int i, start, end, *blkNum, len ;
10    double mysum, *x;
11    blkNum = (int*)arg;
12    len = SUMST.loclen;
13    start = (*blkNum)*len;
14    /* end = start+len > SUMST.totlen ? SUMST.totlen :
15       start+len;*/
16    end = (*blkNum)<NUMTHRDS? start+len:SUMST.totlen;
17    x = SUMST.a;
18 /*----- sum */
19    mysum = 0;
20    for (i=start; i<end ; i++)
21        mysum += x[i];
22 /*----- Lock a mutex before updating
23                      the value in the shared struct */
24    pthread_mutex_lock (&mutexsum);
25    SUMST.sum += mysum;
26 /*----- unlock it now that update is
27                      done*/
28    pthread_mutex_unlock (&mutexsum);
29    printf(" -- Local sum in thread %d is %10.2f total
30          : %10.2f\n ",
31          *blkNum,mysum,SUMST.sum);
32    printf("      len %d %d %d %5.2f \n",len,start,end,x
33          [start]) ;
34 /*----- done with this thread */
35    pthread_exit((void*) 0);
36 }

```

Main

```
1 int main (int argc, char *argv[]){
2     int i, n = VECLen;
3     int *thrNum, *status;
4     double *a;
5     pthread_attr_t attr;
6     /*----- alloc storage + initialize values */
7     a = (double*) malloc (NUMTHRDS*VECLen*sizeof(double));
8     thrNum = (int*) malloc (NUMTHRDS*sizeof(int));
9     for (i=0; i<n; i++) a[i]=(double)i;
10    /*----- have thread number in array */
11    for (i=0; i<NUMTHRDS; i++)
12        thrNum[i]= i;
13    SUMST.totlen = n;
14    SUMST.locLen = 1+(int) ((n-1)/NUMTHRDS);
15    SUMST.a = a;
16    SUMST.sum=0;
17    /*----- initialize mutex */
18    pthread_mutex_init(&mutexsum, NULL);
19    pthread_attr_init(&attr);
20    pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_JOINABLE);
21    for(i=0;i<NUMTHRDS;i++) {
22        /*----- Each thread works on a different subset */
23        pthread_create(&callThd[i],&attr,sum_mtx,(void *)&thrNum[i]);
24    }
25    pthread_attr_destroy(&attr);
26    /*----- Join to wait for the other threads */
27    for(i=0;i<NUMTHRDS;i++) {
28        status = &thrNum[i];
29        pthread_join( callThd[i], (void **)status);
30        printf(" join number %d -- status %d \n",i,*status);
31    }
32    /*----- Now print out the sum and cleanup */
33    printf ("Total Sum in main thread = %10.2f \n", SUMST.sum);
34    free (a); free (thrNum);
35    pthread_mutex_destroy(&mutexsum);
36    pthread_exit(NULL);
37 }
```