

Concurrent Programming

Session 7: POSIX Thread Programming

Computer Engineering Department
Iran University of Science and Technology
Tehran, Iran

Instructor: Hadi Salimi
Distributed Systems Lab.
Computer Engineering Department,
Iran University of Science and Technology,
hsalimi@iust.ac.ir



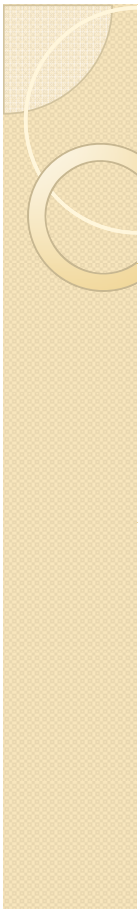
What's POSIX

- **POSIX** stands for **P**ortable **O**perating **S**ystem **I**nterfaces for **uniX**.
- Is a family of related standards specified by IEEE to define the application programming interface (API).
- The term POSIX was suggested by Richard Stallman in response to an IEEE request for a memorable name.



POSIX Threads

- POSIX standard for thread programming
- Available for **Linux** and **Unix** OS family
- Available for **Windows**:
 - Open Source
- C Language Interface
 - Programming types and method calls
 - Implemented as standalone library or another library like libc.



Pthread API

- **Thread Management**: creation, detaching, joining, etc.
- **Mutexes**: deal with synchronization
- **Condition Variables**: communication between threads and sharing a variable.

pthread_create()

```
int pthread_create(tid , attr , func , arg)
```

- pthread_t *tid
 - Handle of created thread
- const pthread_attr_t *attr
 - Attributes of the created thread
- void* (*function) (void*)
 - Function to be mapped to thread
- void* arg
 - Single argument to thread

pthread_create() explained

- Spawn a thread running the function
- Thread **handle** returned via the first parameter.
 - Specify **NULL** for default parameters.
- Single argument to function
 - If NO arguments, send NULL.
- Check error codes.

Example

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

void *hello (void * arg) {
    printf("Hello Thread\n");
}

main() {
    pthread_t tid;

    pthread_create(&tid, NULL, hello, NULL);
}
```

What Happens?

Waiting for a thread

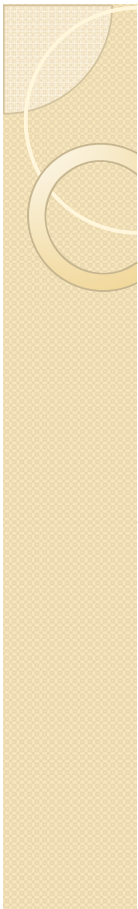
```
int pthread_join(tid , val_ptr)
```

- pthread_t tid
 - Handle for joinable thread
- void** val_ptr
 - Exit value returned by joined thread.



pthread_join Explained

- Calling thread **waits** for thread with handle **tid** to terminate
 - Only one thread can be joined
 - Thread must be joinable
- Exit value is returned from joined thread
 - Type returned is (void *)
 - Use **NULL** if no return value expected



Thread States

- Pthreads threads have two states
 - **joinable** and **detached**
- Threads are joinable by default
 - Resources are kept until pthread_join
 - Can be reset with attributes or API call
- Detached threads cannot be joined
 - Resources can be reclaimed at termination
 - Cannot reset to be joinable

An Example

```
#include <stdio.h>
#include <pthread.h>
#define NUM_THREADS 4

void *hello (void *arg) {
    printf("Hello Thread\n");
}

main() {
    pthread_t tid[NUM_THREADS];
    for (int i = 0; i < NUM_THREADS; i++)
        pthread_create(&tid[i], NULL, hello, NULL);

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

Q: What's Wrong?

```
void *threadFunc(void *pArg) {
    int* p = (int*)pArg;
    int myNum = *p;
    printf( "Thread number %d\n", myNum);
}

. . .
// from main():
for (int i = 0; i < numThreads; i++) {
    pthread_create(&tid[i], NULL, threadFunc, &i);
}
```

Solution: Local Storage

```
void *threadFunc(void *pArg)
{
    int myNum = *((int*)pArg);
    printf( "Thread number %d\n", myNum);
}
. . .

// from main():
for (int i = 0; i < numThreads; i++) {
    tNum[i] = i;
    pthread_create(&tid[i], NULL, threadFunc, &tNum[i]);
}
```

Pthread mutex variables

- Enables correct programming structures for avoiding **race conditions**
- New types
 - pthread_mutex_t
 - the mutex variable
 - pthread_mutexattr_t
 - mutex attributes
- Before use, mutex must be initialized

pthread_mutex_init

```
int pthread_mutex_init( mutex, attr );
```

- pthread_mutex_t *mutex
 - mutex to be initialized
- const pthread_mutexattr_t *attr
 - attributes to be given to mutex

Programmer must always pay attention to mutex scope

pthread_mutex_lock

```
int pthread_mutex_lock( mutex );
```

- pthread_mutex_t *mutex
 - Mutex to attempt to lock
- Attempts to lock mutex
 - If mutex is locked by another thread, calling thread is **blocked**
- Mutex is held by calling thread until unlocked
 - Mutex lock/unlock must be paired or deadlock occurs

pthread_mutex_unlock

```
int pthread_mutex_lock( mutex );
```

- pthread_mutex_t *mutex
 - Mutex to attempt to lock

Example

```
#define NUMTHREADS 4
pthread_mutex_t gMutex; // why does this have to be global?
int g_sum = 0;

void *threadFunc(void *arg)
{
    int mySum = bigComputation();
    pthread_mutex_lock( &gMutex );
    g_sum += mySum;           // threads access one at a time
    pthread_mutex_unlock( &gMutex );
}

main() {
    pthread_t hThread[NUMTHREADS];

    pthread_mutex_init( &gMutex, NULL );
    for (int i = 0; i < NUMTHREADS; i++)
        pthread_create( &hThread[i], NULL, threadFunc, NULL );

    for (int i = 0; i < NUMTHREADS; i++)
        pthread_join(hThread[i]);
    printf ( "Global sum = %f\n", g_sum );
}
```