Threads in C
The Pthreads library

EPL222 – Lab3

Process vs. Thread

- process:
  - an address space with 1 or more threads executing within that address space, and the required system resources for those threads
  - a program that is running

- thread:
  - a sequence of control within a process
  - shares the resources in that process

Threads overview

Advantages

- The overhead for creating a thread is significantly less than that for creating a process
- Multitasking, i.e., one process serves multiple clients
- Switching between threads requires the OS to do much less work than switching between processes

Disadvantages

- Not as widely available as longer established features
- Writing multithreaded programs requires more careful thought
- More difficult to debug than single threaded programs
- For single processor machines, creating several threads in a program may not necessarily produce an increase in performance

Thread Programming with Shared Memory

- Each thread has a set of private variables, e.g., local stack variables
- Also a set of shared variables, e.g., static variables, shared common blocks, or global heap
- Threads communicate implicitly by writing and reading shared variables
- Threads coordinate by synchronizing on shared variables

The Pthreads API

- Three types of routines:
  - Thread management: create, terminate, join, and detach
  - Mutexes: mutual exclusion, creating, destroying, locking, and unlocking mutexes
  - Condition variables: event driven synchronizaiton.
  - Mutexes and condition variables are concerned about synchronization.

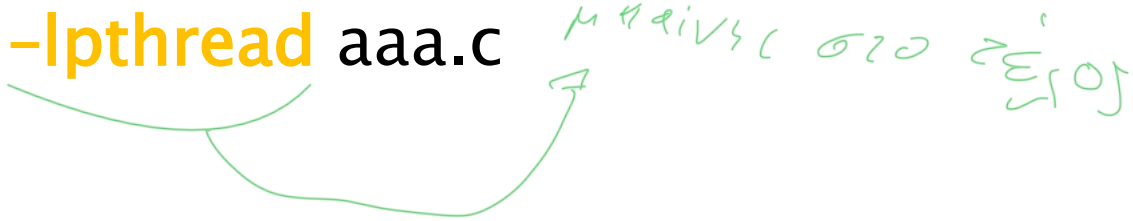
- The concept of opaque objects pervades the design of the API.

The Pthreads API naming convention

Routine Prefix	Function
Pthread_	General pthread
Pthread_attr_	Thread attributes
Pthread_mutex_	mutex
Pthread_mutexattr	Mutex attributes
Pthread_cond_	Condition variables
Pthread_condaddr	Conditional variable attributes
Pthread_key_	Thread specific data keys

Thread management routines

- Creation: pthread_create
- Termination:
  - return
  - pthread_exit
- Wait (parent/child synchronization):
  - pthread_join
- Pthread header file <pthread.h>
- Compiling pthread programs: gcc -lpthread aaa.c



Main thread

- Initial thread created when main() is invoked by the process loader
- Once in the main(), the application can create daughter threads
- If the main thread returns, the process terminates even if there are running threads in that process, unless special precautions are taken
- To explicitly avoid terminating the entire process, use pthread_exit()

Στη C αν κάνει το main thread, αρχίνουν όλα αυτά στην Java συνήθως.

Create thread

```
int pthread_create(pthread_t *thread,
    pthread_attr_t *attr,
    void *(*thread_function)(void *),
    void *arg );
```

- Thread equivalent of fork()
  - 1st arg – pointer to the identifier of the created thread
  - 2nd arg – thread attributes. If null, then the thread is created with default attributes
  - 3rd arg – pointer to the function the thread will execute
  - 4th arg – parameters of the executed function
  - returns 0 for success
- After this function gets executed, a new thread has been created and is executing the function indicated by *thread_function*

Function started by pthread_create

- Prototype:

```
void* thread_function (void* args_p);
```

- void* can be cast to any pointer type in C.
- So args_p can point to a list containing one or more values needed by *thread_function*
- Similarly, the return value of *thread_function* can point to a list of one or more values

Waiting threads

```
int pthread_join(pthread_t thread,
     void **thread_return)
```

- Equivalent of waitpid() for processes
  - 1st arg – the thread to wait for
  - 2nd arg – pointer to a pointer to the return value from the thread
  - returns 0 for success

- main thread will wait for daughter thread *thread* to finish

Thread termination

- pthreads exist in user space and are seen by the kernel as a single process
  - if one issues and **exit()** system call, all the threads are terminated by the OS
  - if the **main()** function exits, all other threads are terminated

- Thread Termination
  - Return from initial thread function
  - `void pthread_exit(void * status)`
    status: the exit status of the thread – passed to the ***status*** variable in the pthread_join() function of a thread waiting for this one

Detaching a thread

- The detached thread can act as daemon thread
- The parent thread doesn't need to wait

```
int pthread_detach(pthread_t tid)
```

- Detaching self :

```
pthread_detach(pthread_self())
```

Thread creation example

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

void worker(void *a)  {
    int *cnt = (int *)a;
    printf("This is thread %d\n", *cnt);
    pthread_exit(0);
}

int main(int argc, char **argv){
    pthread_t t1;
    int thread_id = 1;

    if (pthread_create(&t1, NULL, (void *) &worker, (void *) &thread_id) != 0) {
        printf("Error creating thread\n");
        exit(1);
    }
    pthread_join(t1, NULL);
    return 0;
}
```

Hello world with threads

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

void *PrintHello(void * id){
   printf("Thread%d: Hello World!\n", id);
}

void main (){
   pthread_t t0, t1;
   pthread_create(&t0, NULL, PrintHello, (void*) 0);
   pthread_create(&t1, NULL, PrintHello, (void*) 1);
   pthread_join(t0, NULL);
   pthread_join(t1, NULL);
}
```

Some More Pthread Functions

- `pthread_yield();`
  - Informs the scheduler that the thread is willing to yield
- ```
  pthread_t me;
  me = pthread_self();
  ```
  - Allows a pthread to obtain its own identifier pthread_t *thread*


  ```

Some multi-thread program examples

- A multi-thread program example: example1.c
- Making multiple producers, give each an ID: example2.c
  - What is going on in this program?
- Open both c files
  - The files are on moodle
  - study, compile and run them

Matrix multiply and threaded matrix multiply

- Matrix multiply: C = A × B
  - mm.c is single threaded
  - mm_pthread.c is multithreaded

$$C[i,j] = \sum_{k=1}^{N} A[i,k] \times B[k,j]$$

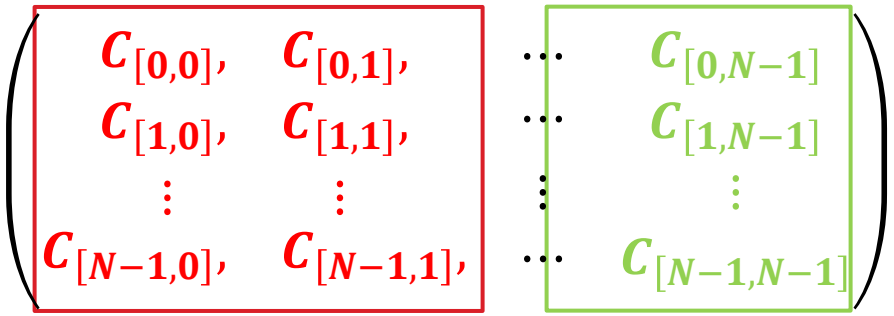
$$\begin{pmatrix} C_{[0,0]}, & C_{[0,1]}, & \cdots & C_{[0,N-1]} \\ C_{[1,0]}, & \boldsymbol{C_{[1,1]}}, & \cdots & C_{[1,N-1]} \\ \vdots & \vdots & \vdots & \vdots \\ C_{[N-1,0]}, & C_{[N-1,1]}, & \cdots & C_{[N-1,N-1]} \end{pmatrix} = \begin{pmatrix} A_{[0,0]}, & A_{[0,1]}, & \cdots & A_{[0,N-1]} \\ \boldsymbol{A_{[1,0]}}, & \boldsymbol{A_{[1,1]}}, & \boldsymbol{\cdots} & \boldsymbol{A_{[1,N-1]}} \\ \vdots & \vdots & \vdots & \vdots \\ A_{[N-1,0]}, & A_{[N-1,1]}, & \cdots & A_{[N-1,N-1]} \end{pmatrix} \times \begin{pmatrix} B_{[0,0]}, & \boldsymbol{B_{[0,1]}}, & \cdots & B_{[0,N-1]} \\ B_{[1,0]}, & \boldsymbol{B_{[1,1]}}, & \cdots & B_{[1,N-1]} \\ \vdots & \boldsymbol{\vdots} & \vdots & \vdots \\ B_{[N-1,0]}, & \boldsymbol{B_{[N-1,1]}}, & \cdots & B_{[N-1,N-1]} \end{pmatrix}$$

Matrix multiply and threaded matrix multiply

- Sequential code:

```
for (i=0; i<N; i++)
    for (j=0; j<N; j++)
        for (k=0; k<N; k++)
            C[i,j] = C[i,j] + A[i,k] * B[k,j]
```

- Threaded code program
  - Do the same sequential operation, different threads work on different part of the C array. How to decide who does what? Need three parameters: **N, nthreads, myid**



Matrix multiply and threaded matrix multiply

- Threaded code program
  - Each thread sees variables **N, nthreads** and has its **own myid** variable
  - Each thread is responsible for a sub-array of C
    - from column *N/Nthreads * myid*
    - to column *N/Nthreads * (myid+1) -1*
  - The calculation of $C_{[i,j]}$ does not depend on any other C term

$$\begin{pmatrix} C_{[0,0]}, & C_{[0,1]}, & \cdots & C_{[0,N-1]} \\ C_{[1,0]}, & C_{[1,1]}, & \cdots & C_{[1,N-1]} \\ \vdots & \vdots & \vdots & \vdots \\ C_{[N-1,0]}, & C_{[N-1,1]}, & \cdots & C_{[N-1,N-1]} \end{pmatrix}$$