Lecture 9

Programming Shared Memory I

Why Threads?

IKC-MH.57 *Introduction to High Performance and Parallel Computing* at December 15, 2023

Dr. Cem Özdoğan
Engineering Sciences Department
İzmir Kâtip Çelebi University

Programming Shared Memory
- What is a Thread?
- Threads Model
- Why Threads?
- Thread Basics: Creation and Termination
  - Thread Creation
  - Thread Termination

Contents

1 Programming Shared Memory
- What is a Thread?
- Threads Model
- Why Threads?
- Thread Basics: Creation and Termination
  - Thread Creation
  - Thread Termination

What is Threads?

- Technically, a *thread* is defined as an **independent stream of instructions** that can be scheduled to run by the operating system.

  • Suppose that a main program contains a number of procedures (functions, subroutines, ...).
  • Then suppose all of these procedures being able to be scheduled to run simultaneously and/or independently.
  • That would describe a "**multi-threaded**" program.

- Before understanding a ***thread***, one first needs to understand a UNIX ***process***.
- Processes contain information about program resources and program execution state.

  • Threads use and exist within these process resources,
  • To be scheduled by the OS,
  • Run as independent entities.
  • A thread has its own independent flow of control as long as its parent process exists (dies if the parent dies!).
  • A thread duplicates only the essential resources it needs.

- A thread is "lightweight" because most of the overhead has already been accomplished through the creation of its process.

Programming Shared Memory
What is a Thread?
Threads Model
Why Threads?
Thread Basics: Creation and Termination
Thread Creation
Thread Termination

Threads Model I

- In shared memory multiprocessor architectures, such as SMPs, *threads can be used to implement parallelism*.
- In the threads model of parallel programming, a single process can have
  - **multiple concurrent**,
  - **execution paths**.
- Most simple analogy for threads is the concept of a single program that includes a number of subroutines:

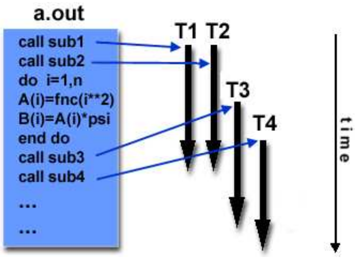

Figure: Threads model.

- Main program loads and acquires all of the necessary system and user resources to run.
- *Main program* performs some serial work,
- and then creates a number of tasks (threads) that can be scheduled and run by the OS concurrently.

Programming Shared Memory
What is a Thread?
Threads Model
Why Threads?
Thread Basics: Creation and Termination
Thread Creation
Thread Termination

Threads Model II

- Each thread has local data, but also, shares the entire resources of *main program*.

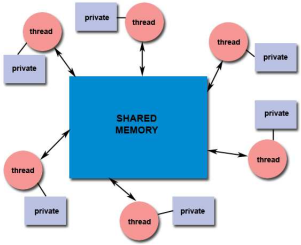

Figure: Thread shared memory model.

- This saves the overhead associated with replicating a program's resources for each thread.
- Each thread also benefits from a global memory view because it shares the memory space of program.
- Any thread can execute any subroutine at the same time as other threads.

Programming Shared Memory
What is a Thread?
Threads Model
Why Threads?
Thread Basics: Creation and Termination
Thread Creation
Thread Termination

Threads Model III

- Threads communicate with each other through global memory (updating address locations).
- Changes made by one thread to shared system resources (such as closing a file) will be seen by all other threads.
- This requires **synchronization constructs** to insure that more than one thread is not updating the same global address at any time.

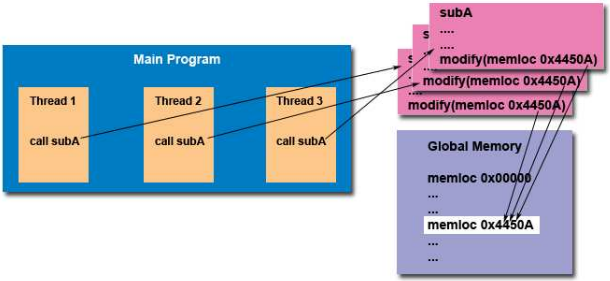

Figure: Threads **Unsafe!** Pointers having the same value point to the same data.

Programming Shared Memory
What is a Thread?
Threads Model
Why Threads?
Thread Basics: Creation and Termination
Thread Creation
Thread Termination

Why Threads? I

- The primary motivation for using threads is to realize potential program performance gains.
- When compared to the cost of creating and managing a process, a thread can be created with *much less OS overhead*.
- Managing threads requires fewer system resources than managing processes.
- Threaded programming models offer significant advantages over message-passing programming models along with some disadvantages as well.
- **Software Portability;**

  • Threaded applications can be developed on serial machines and run on parallel machines without any changes.
  • This ability to migrate programs between diverse architectural platforms is a very significant advantage of threaded APIs.

Programming Shared Memory
What is a Thread?
Threads Model
Why Threads?
Thread Basics: Creation and Termination
Thread Creation
Thread Termination

Why Threads? II

- **Latency Hiding;**
  - One of the major overheads in programs (both serial and parallel) is the access latency for memory access, I/O, and communication.
  - By allowing multiple threads to execute on the same processor, threaded APIs enable this latency to be hidden.
  - In effect, while one thread is waiting for a communication operation, other threads can utilize the CPU, thus *masking associated overhead*.
- **Scheduling and Load Balancing;**
  - While in many *structured* applications the task of allocating equal work to processors is easily accomplished,
  - In *unstructured* and *dynamic* applications (such as game playing and discrete optimization) this task is more difficult.
  - Threaded APIs allow the programmer
    - to specify a large number of concurrent tasks
    - and support system-level dynamic mapping of tasks to processors with a view to minimizing idling overheads.

Programming Shared Memory
What is a Thread?
Threads Model
Why Threads?
Thread Basics: Creation and Termination
Thread Creation
Thread Termination

Why Threads? III

- **Ease of Programming, Widespread Use**
  - Due to the mentioned advantages, threaded programs are significantly easier to write (!) than corresponding programs using message passing APIs.
  - With widespread acceptance of the POSIX thread API, development tools for POSIX threads are more widely available and stable.
- **Overlapping CPU work with I/O:** For example, a program may have sections where it is performing a long I/O operation. While one thread is waiting for an I/O system call to complete, CPU intensive work can be performed by other threads.
- **Priority/real-time scheduling:** tasks which are more important can be scheduled to supersede or interrupt lower priority tasks.
- **Asynchronous event handling:** tasks which service events of indeterminate frequency and duration can be interleaved. For example, a web server can both transfer data from previous requests and manage the arrival of new requests.

Programming Shared Memory
What is a Thread?
Threads Model
Why Threads?
Thread Basics: Creation and Termination
Thread Creation
Thread Termination

Why Threads? IV

- A number of vendors provide vendor-specific thread APIs. Standardization efforts have resulted in two very different implementations of threads.
- Microsoft has its own implementation for threads, which is not related to the UNIX POSIX standard or OpenMP.

1. **POSIX Threads**. *Library based; requires parallel coding.*
   - C Language only. Very explicit parallelism; requires significant programmer attention to detail.
   - Commonly referred to as ***Pthreads***.
   - POSIX has emerged as the standard threads API, supported by most vendors.
2. **OpenMP**. *Compiler directive based; can use serial code.*
   - Jointly defined by a group of major computer hardware and software vendors.
   - The OpenMP C/C++ API was released in late 1998.
   - Portable / multi-platform, including Unix and Windows platforms
   - Can be very easy and simple to use - provides for “incremental parallelism“.

Programming Shared Memory
What is a Thread?
Threads Model
Why Threads?
Thread Basics: Creation and Termination
Thread Creation
Thread Termination

Why Threads? V

- MPI $\Longrightarrow$ on-node communications,
  - MPI libraries usually implement on-node task communication **via shared memory**, which involves at least one memory copy operation (process to process).
- Threads $\Longrightarrow$ on-node data transfer.

  • For *Pthreads* there is **no intermediate memory copy** required because threads share the same address space within a single process.
  • There is **no data transfer**.
  • It becomes more of a cache-to-CPU or memory-to-CPU bandwidth (worst case) situation.
  • These speeds are much higher.

Programming Shared Memory
What is a Thread?
Threads Model
Why Threads?
Thread Basics: Creation and Termination
Thread Creation
Thread Termination

Thread Basics: Creation and Termination I

Programming Shared Memory
What is a Thread?
Threads Model
Why Threads?
Thread Basics: Creation and Termination
Thread Creation
Thread Termination

- The *Pthreads* API subroutines can be informally grouped into **four major groups**:
  1. **Thread management:** Routines that work directly on threads - creating, detaching, joining, set/query thread attributes (joinable, scheduling etc.), etc.
  2. **Mutexes:** Routines that deal with synchronization. Mutex functions provide for creating, destroying, locking and unlocking mutexes, setting or modifying attributes associated with mutexes.
  3. **Condition variables:** Routines that address communications between threads that share a mutex. Functions to create, destroy, wait and signal based upon specified variable values, set/query condition variable attributes.
  4. **Synchronization:** Routines that manage read/write locks and barriers.

Thread Basics: Creation and Termination II

Creating Threads:

- Initially, main program contains a single, default thread.
- **pthread_create** creates a new thread and makes it executable.

```
#include <pthread.h>
int
pthread_create ( pthread_t    *thread_handle ,
    const pthread_attr_t    *attribute ,
    void *    (*thread_function)(void *) ,
    void    *arg);
```

- Creates a single thread that corresponds to the invocation of the function ***thread_function*** (and any other functions called by *thread_function*).
- Once created, threads are peers, and may create other threads.
- On successful creation of a thread, a unique identifier is associated with the thread and assigned to the location pointed to by ***thread_handle***.
- On successful creation of a thread, **pthread_create** returns 0; else it returns an error code.

Programming Shared Memory
What is a Thread?
Threads Model
Why Threads?
Thread Basics: Creation and Termination
Thread Creation
Thread Termination

Thread Basics: Creation and Termination III

- The thread has the attributes described by the ***attribute*** argument.
- The ***arg*** field specifies a pointer to the argument to function *thread_function*.
- This argument is typically used to pass the workspace and other *thread-specific* data to a thread.
- There is **no implied hierarchy** or dependency between threads.
- Unless you are using the *Pthreads* scheduling mechanism, it is up to the implementation and/or OS to decide where and when threads will execute.
- Robust programs should not depend upon threads executing in a specific order.

Programming Shared Memory
What is a Thread?
Threads Model
Why Threads?
Thread Basics: Creation and Termination
Thread Creation
Thread Termination

Thread Basics: Creation and Termination IV

Terminating Threads.

- There are several ways in which a *Pthread* may be terminated:

a The thread returns from its starting routine (the main routine for the initial thread).

b The thread makes a call to the **pthread_exit** subroutine.

c The thread is cancelled by another thread via the **pthread_cancel** routine.

d The entire process is terminated due to a call to either the *exec* or *exit* subroutines.

  - If *main* finishes <u>before</u> the threads and exits with **pthread_exit()**, the other threads will continue to execute (join function!).
  - If *main* finishes <u>after</u> the threads and exits, the threads will be automatically terminated.

Programming Shared Memory
What is a Thread?
Threads Model
Why Threads?
Thread Basics: Creation and Termination
Thread Creation
Thread Termination

Thread Basics: Creation and Termination V

Example Code:

- This example code creates 5 threads with the **pthread_create()** routine.
- Each thread prints a 'Hello World!' message, and then terminates with a call to **pthread_exit()**.

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

#define NUM_THREADS 5

void *PrintHello(void *threadid)
{
   sleep(10);
   long tid;
   tid = (long)threadid;
   printf("Hello World! It's me, thread #%ld!\n", tid);
   pthread_exit(NULL);
}
```


Programming Shared Memory
What is a Thread?
Threads Model
Why Threads?
Thread Basics: Creation and Termination
Thread Creation
Thread Termination

Thread Basics: Creation and Termination VI

```c
int main(int argc, char *argv[])
{
   pthread_t threads[NUM_THREADS];
   int rc;
   long t;
   for(t=0;t<NUM_THREADS;t++){
      printf("In main: creating thread %ld\n", t);
      rc = pthread_create(&threads[t], NULL, PrintHello, (void *)t);
      if (rc){
         printf("ERROR; return code from pthread_create() is %d\n", rc);
         exit(-1);
         }
      }

   /* Last thing that main() should do */
   pthread_exit(NULL);
}
```


Programming Shared Memory
What is a Thread?
Threads Model
Why Threads?
Thread Basics: Creation and Termination
Thread Creation
Thread Termination