POSIX Threads

CMPUT 379

Winter 2013

Arnamoy Bhattacharyya *

*Material for the slides are taken from https://computing.llnl.gov/tutorials/pthreads/

What is a Thread?

An independent stream of instructions that can be scheduled to run as such by the operating system.

To the software developer - a "procedure" that runs independently from its main program

Processes contain information about program resources and program execution state, including:

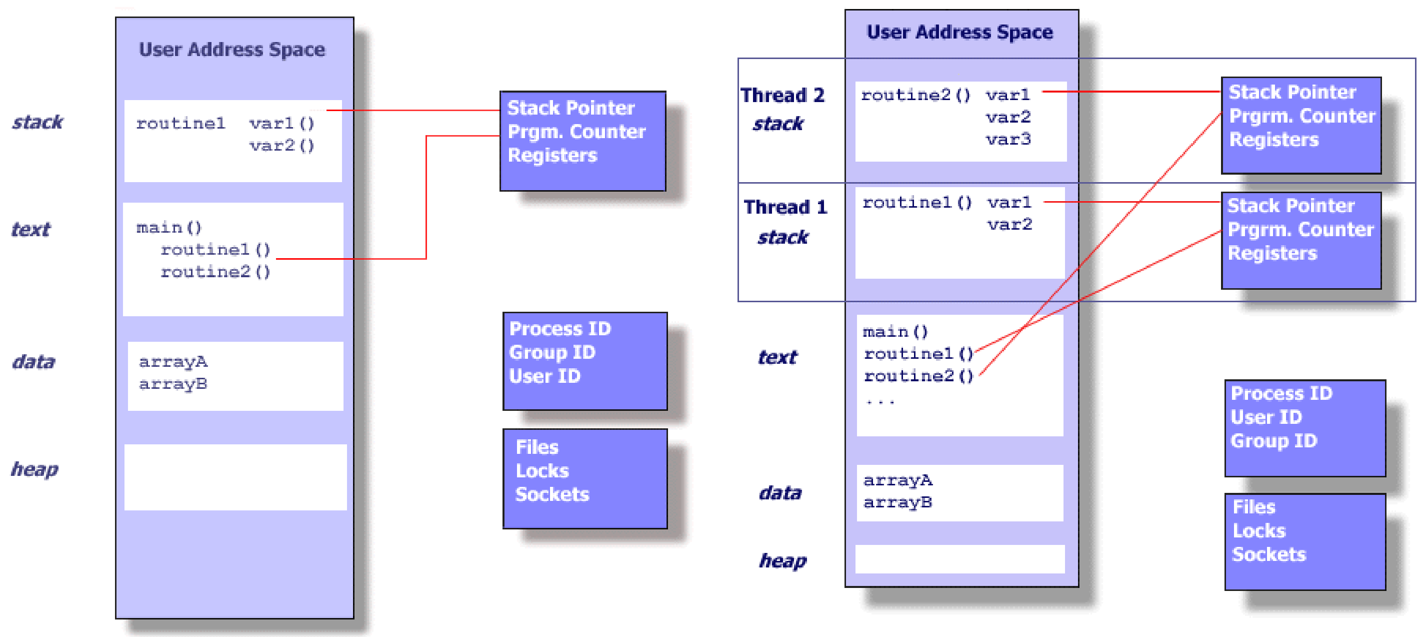
Process ID, process group ID, user ID, and group ID
Environment
Working directory.
Registers, Stack, Heap etc

Threads use and exist within these process resources

Are able to be scheduled by the operating system and run as independent entities

A thread has it’s own:

Stack pointer, Registers, Scheduling properties (such as policy or priority), Set of pending and blocked signals, Thread specific data.



UNIX PROCESS

THREADS WITHIN A UNIX PROCESS

Because threads within the same process share resources:

Changes made by one thread to shared system resources (such as closing a file) will be seen by all other threads.

Two pointers having the same value point to the same data.

Reading and writing to the same memory locations is possible, and therefore requires explicit synchronization by the programmer.

What are Pthreads?

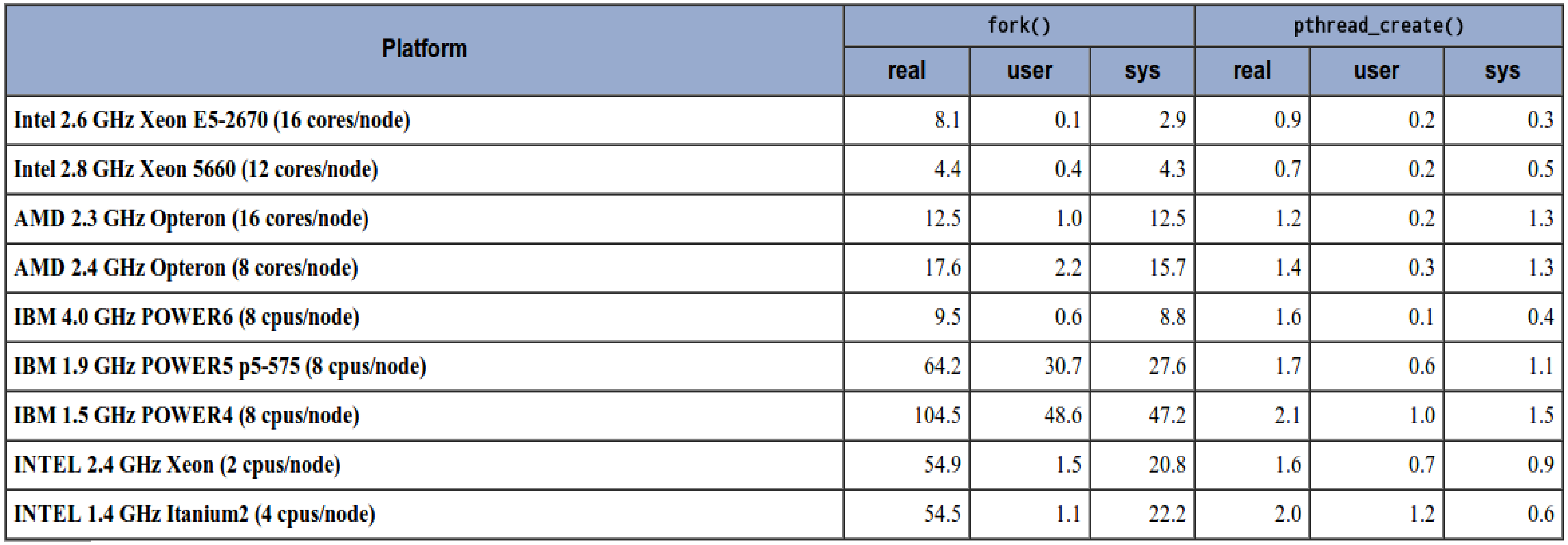
Historically, hardware vendors have implemented their own proprietary versions of threads.

In order to take full advantage of the capabilities provided by threads, a standardized programming interface was required

For UNIX systems, this interface has been specified by the IEEE POSIX 1003.1c standard (1995).

Why Pthreads?

a thread can be created with much less operating system overhead



Platform	fork()			pthread_create()		
	real	user	sys	real	user	sys
Intel 2.6 GHz Xeon E5-2670 (16 cores/node)	8.1	0.1	2.9	0.9	0.2	0.3
Intel 2.8 GHz Xeon 5660 (12 cores/node)	4.4	0.4	4.3	0.7	0.2	0.5
AMD 2.3 GHz Opteron (16 cores/node)	12.5	1.0	12.5	1.2	0.2	1.3
AMD 2.4 GHz Opteron (8 cores/node)	17.6	2.2	15.7	1.4	0.3	1.3
IBM 4.0 GHz POWER6 (8 cpus/node)	9.5	0.6	8.8	1.6	0.1	0.4
IBM 1.9 GHz POWER5 p5-575 (8 cpus/node)	64.2	30.7	27.6	1.7	0.6	1.1
IBM 1.5 GHz POWER4 (8 cpus/node)	104.5	48.6	47.2	2.1	1.0	1.5
INTEL 2.4 GHz Xeon (2 cpus/node)	54.9	1.5	20.8	1.6	0.7	0.9
INTEL 1.4 GHz Itanium2 (4 cpus/node)	54.5	1.1	22.2	2.0	1.2	0.6

Why Pthreads?

Inter-thread communication is more efficient and in many cases, easier to use than inter-process communication

Overlapping CPU work with I/O

Priority/real-time scheduling

Asynchronous event handling ,

For example, a web server can both transfer data from previous requests and manage the arrival of new requests.

Parallel Programming:

Some issues to be considered -

Problem partitioning

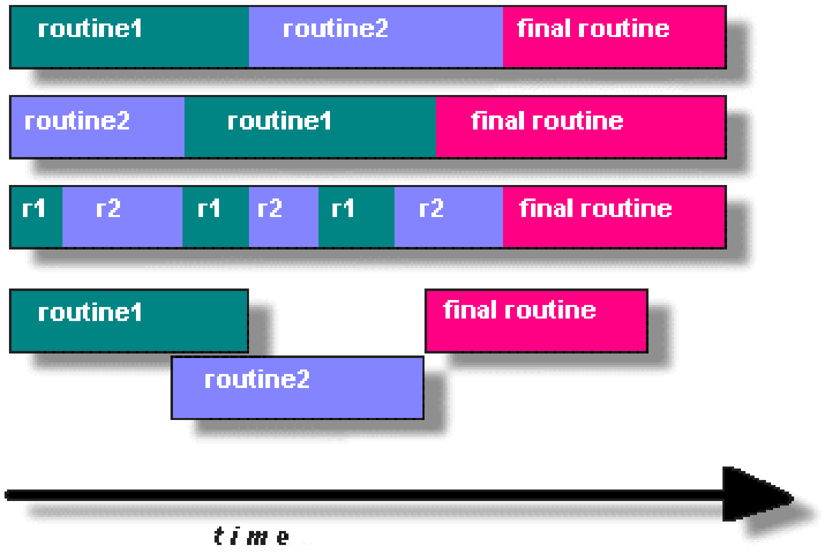
Load balancing

Communications

Data dependencies

Synchronization and race conditions

if routine1 and routine2 can be interchanged, interleaved and/or overlapped in real time, they are candidates for threading.



Common models for threaded programs

Manager/worker: a single thread, the manager assigns work to other threads, the workers.

Pipeline: a task is broken into a series of suboperations, each of which is handled in series, but concurrently, by a different thread.

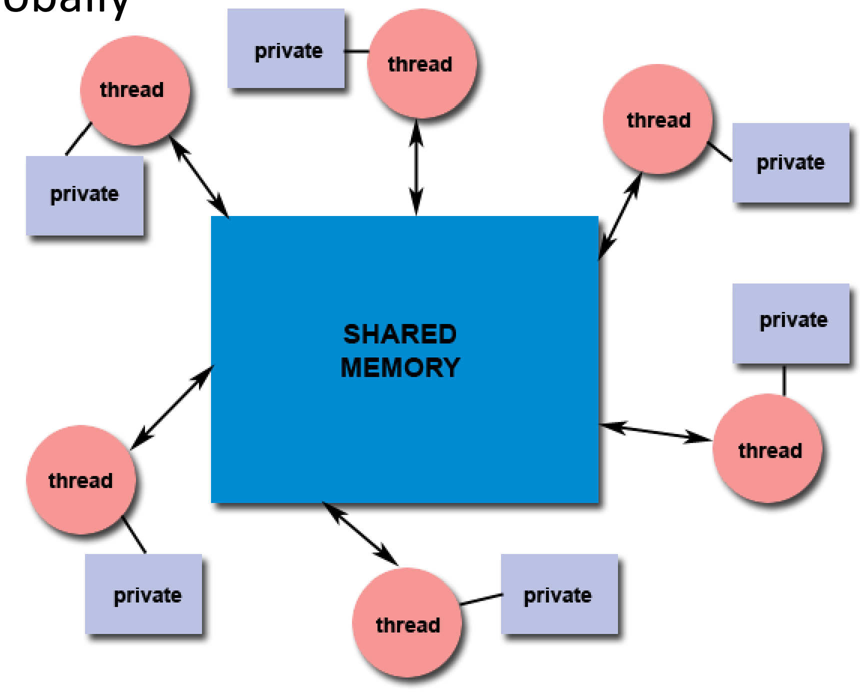
Peer: similar to the manager/worker model, but after the main thread creates other threads, it participates in the work.

Shared Memory Model:

All threads have access to the same global, shared memory

Threads also have their own private data

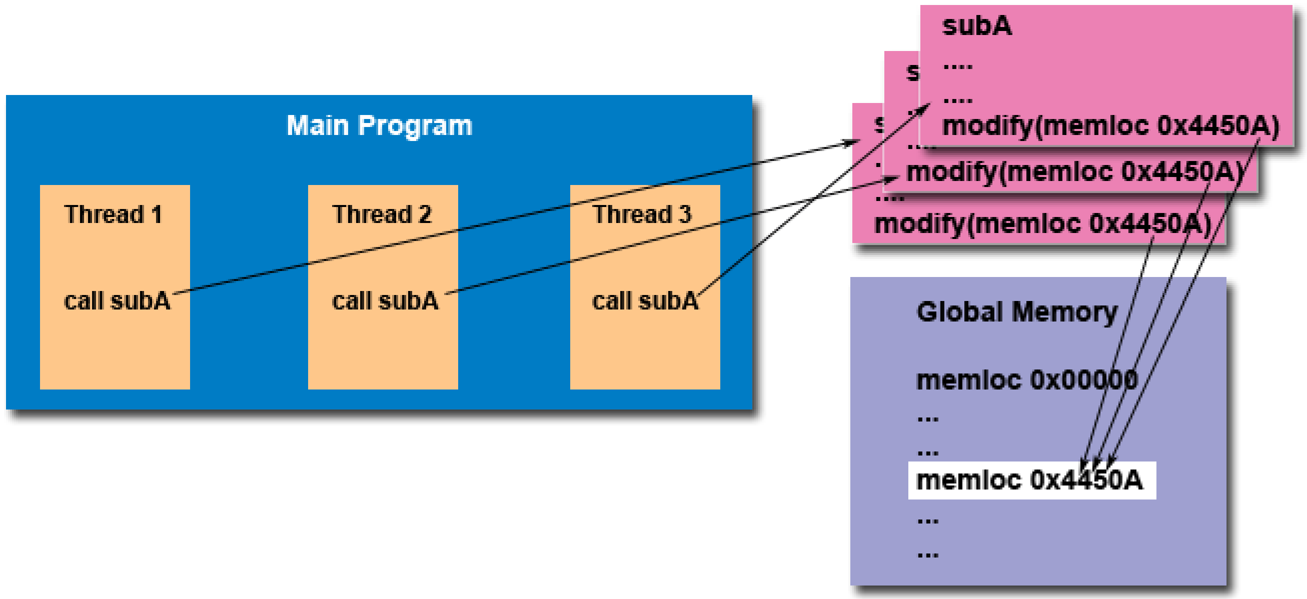
Programmers are responsible for synchronizing access (protecting) globally shared data.



Thread-safeness:

If the routine does not employ some sort of synchronization constructs to prevent data corruption, then it is not thread-safe

Be careful if your application uses libraries or other objects that don't explicitly guarantee thread-safeness.



The Pthreads API

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

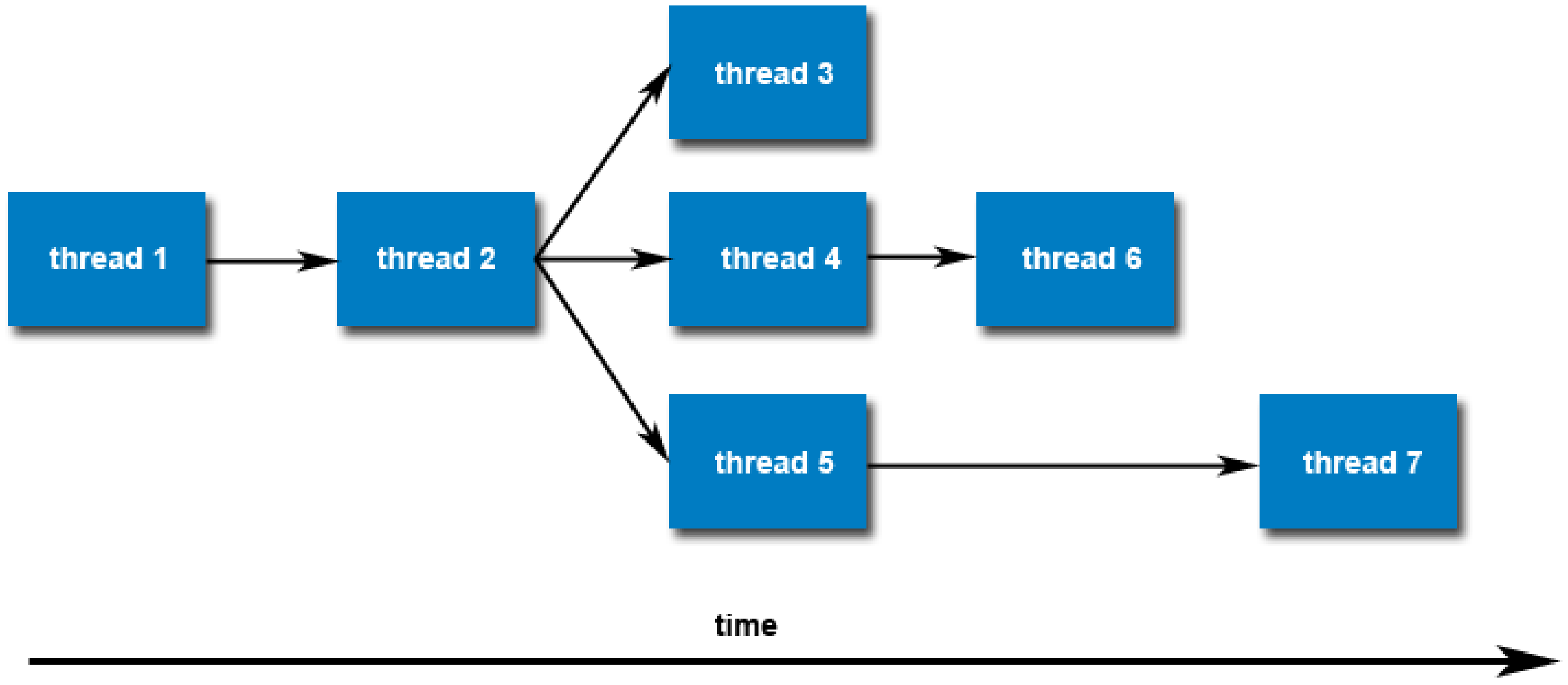
thread: An opaque, unique identifier for the new thread returned by the subroutine.

attr: An opaque attribute object that may be used to set thread attributes. NULL for the default values.

start_routine: the C routine that the thread will execute once it is created.

arg: A single argument that may be passed to start_routine. It must be passed by reference as a pointer cast of type void. NULL may be used if no argument is to be passed.

Once created, threads are peers, and may create other threads. There is no implied hierarchy or dependency between threads.



Terminating Threads & pthread_exit():

Problem if main() finishes before the threads it spawned and pthread_exit() is not called explicitly.

All of the threads it created will terminate because main() is done and no longer exists to support the threads.

By having main() explicitly call pthread_exit() as the last thing it does, main() will block and be kept alive to support the threads it created until they are done.

Demo – pthread create and destruction

Passing Arguments to Threads

create a structure and cast to a (void *) before passing to pthread_create()

Demo Example

Joining:

"Joining" is one way to accomplish synchronization between threads.

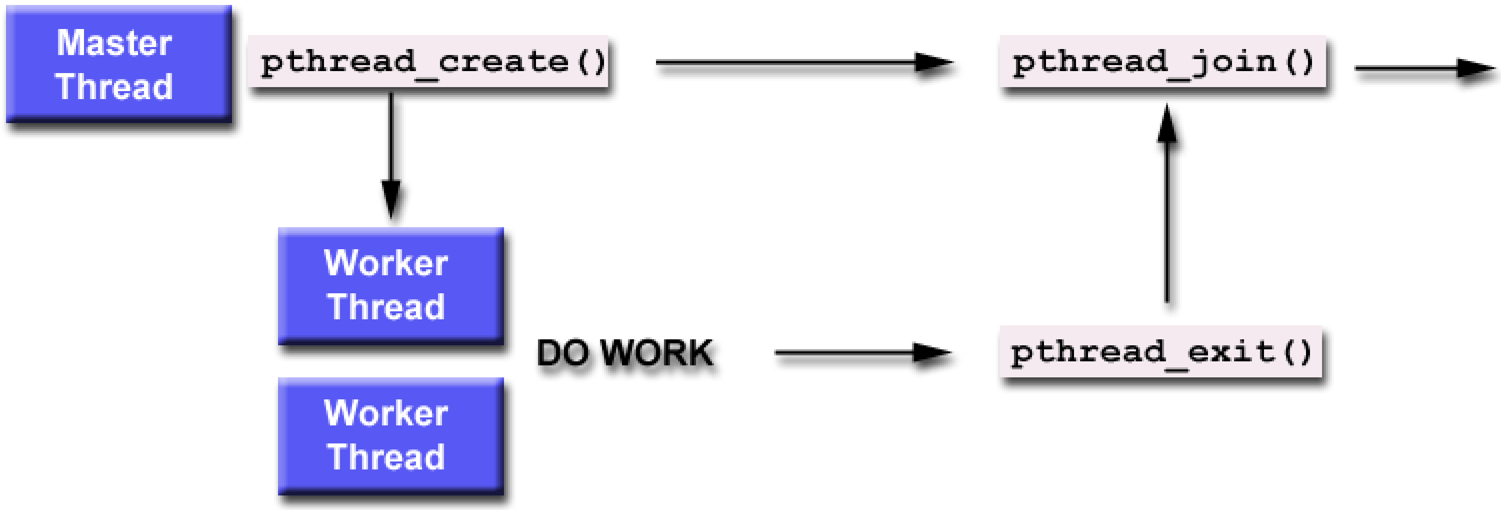
pthread_join() blocks the calling thread until the specified threadid thread terminates

The programmer is able to catchthe target thread's termination return status.

A joining thread can match one pthread_join() call.

** If a thread requires joining, consider explicitly creating it as joinable. This provides portability as not all implementations may create threads as joinable by default.

**If you know in advance that a thread will never need to join with another thread, consider creating it in a detached state. Some system resources may be able to be freed.



Demo of pthread_join()