Today's topic

- Pthread
  - Some materials and figures are obtained from the POSIX threads Programming tutorial at https://computing.llnl.gov/tutorials/pthreads

What is a Thread?

- OS view: A thread is an independent stream of instructions that can be scheduled to run by the OS.
- Software developer view: a thread can be considered as a “procedure” that runs independently from the main program.
  - Sequential program: a single stream of instructions in a program.
  - Multi-threaded program: a program with multiple streams
    - Multiple threads are needed to use multiple cores/CPUs

Example multithread programs?

- Computer games
  - each thread controls the movement of an object.
- Scientific simulations
  - Hurricane movement simulation: each thread simulates the hurricane in a small domain.
  - Molecular dynamic: each thread simulates a subset of particulars.
  - ……
- Web server
  - Each thread handles a connection.
- ……

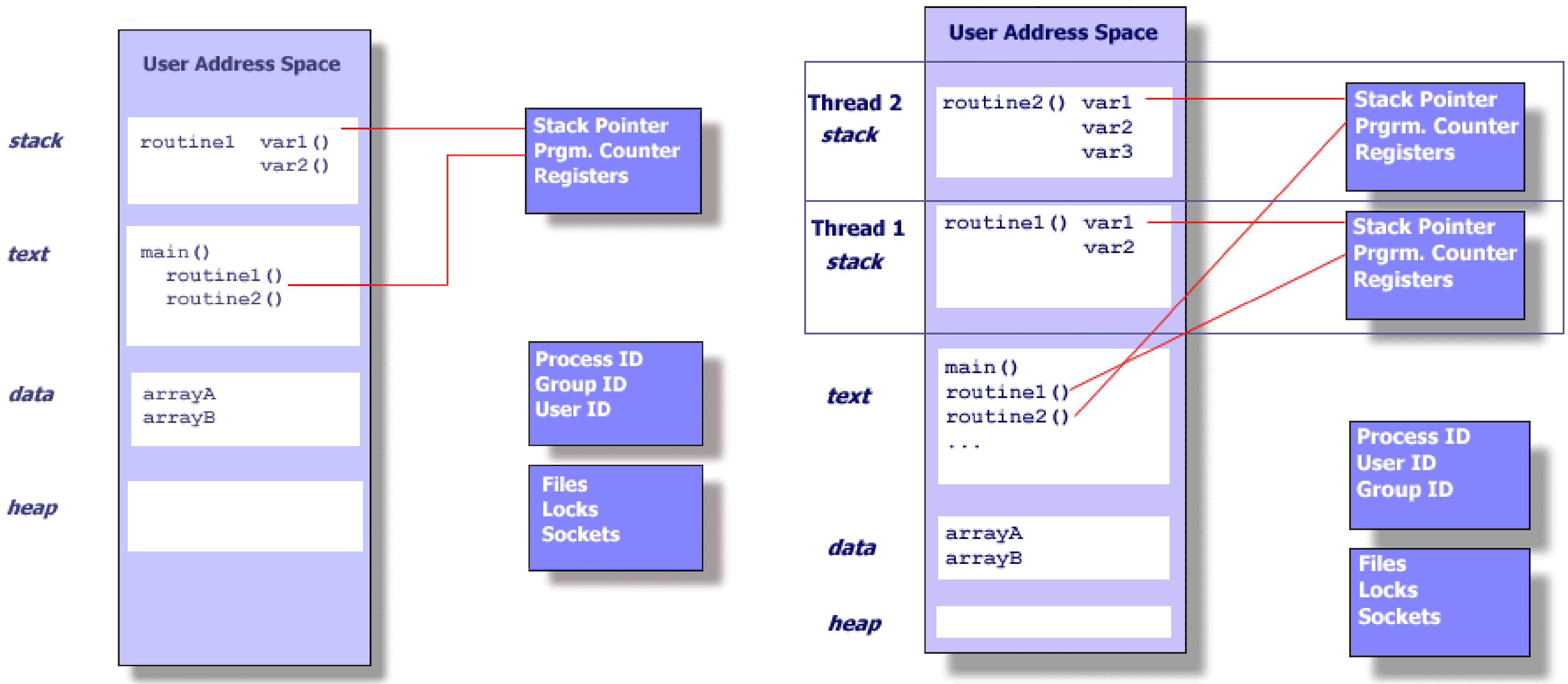
Process and Thread

- **Process context**
  - Process ID, process group ID, user ID, and group ID
  - Environment
  - Working directory.
  - Program instructions
  - Registers (including PC)
  - Stack
  - Heap
  - File descriptors
  - Signal actions
  - Shared libraries
  - Inter-process communication tools
- Two parts in the context: self-contained domain (protection) and execution of instructions.

Process and Thread

- **What are absolutely needed to support a stream of instructions, given the process context?**
  - Process ID, process group ID, user ID, and group ID
  - Environment
  - Working directory.
  - Program instructions
  - Registers (including PC)
  - Stack
  - Heap
  - File descriptors
  - Signal actions
  - Shared libraries
  - Inter-process communication tools

Process and Thread



Threads…

- Exist within processes
- Die if the process dies
- Use process resources
- Duplicate only the essential resources for OS to schedule them independently
- Each thread maintains
  - Stack
  - Registers
  - Scheduling properties (e.g. priority)
  - Set of pending and blocked signals (to allow different react differently to signals)
  - Thread specific data

Pthreads…

- Hardware vendors used to implement proprietary versions of threads
  - Thread programs are not portable
- Pthreads = POSIX threads, specified in IEEE POSIX 1003.1c (1995)

Advantages of Threads

- **Light-weight**
  - **Lower overhead for thread creation**
  - **Lower Context Switching Overhead**
    - **Fewer OS resources**

Platform	fork()			pthread_create()		
	real	user	sys	real	user	sys
AMD 2.4 GHz Opteron (8cpus/node)	41.1	60.1	9.0	0.7	0.2	0.4
IBM 1.9 GHz POWER5 p5-575 (8cpus/node)	64.2	30.8	27.7	1.8	0.7	1.1
IBM 1.5 GHz POWER4 (8cpus/node)	104.1	48.6	47.2	2.0	1.0	1.5
INTEL 2.4 GHz Xeon (2 cpus/node)	55.0	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	1.3	0.7

Advantages of Threads

- Shared State
  - Don't need IPC-like mechanism to communicate between threads of same process

Disadvantages of Threads

- Shared State!
  - Global variables are shared between threads. Accidental changes can be fatal.
- Many library functions are not thread-safe
  - Library Functions that return pointers to static internal memory. E.g. gethostbyname()
- Lack of robustness
  - Crash in one thread will crash the entire process.

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.
  - Why not anything related to inter-thread communication?
- 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
  - Can we still use exit?
- Wait (parent/child synchronization): pthread_join
- Pthread header file <pthread.h>
- Compiling pthread programs: gcc –lpthread aaa.c

Creation

- Thread equivalent of fork()

- int pthread_create(
  ```
  pthread_t  * thread,
  pthread_attr_t  *  attr,
  void * (*start_routine)(void *),
  void * arg
  );
  ```
- Returns 0 if OK, and non-zero (> 0) if error.
- Parameters for the routines are passed through void * arg.
  - What if we want to pass a structure?

Termination

Thread Termination

- `void pthread_exit(void * status)`

Process Termination

- `exit()`
- `main()`

Waiting for child thread

- int pthread_join(pthread_t tid, void **status)
- Equivalent of `waitpid()` for processes

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())
```

Some multi-thread program examples

- A multi-thread program example: Example1.c
- Making multiple producers: example2.c
  - What is going on in this program?

Matrix multiply and threaded matrix multiply

- Matrix multiply: C = A × B

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

$$\begin{pmatrix} C[0,0], & C[0,1], \ldots\ldots, & C[0,N-1] \\ C[1,0], & \boxed{C[1,1]}, \ldots\ldots, & C[1,N-1] \\ \multicolumn{3}{l}{\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots} \\ C[N-1,0], & C[N-1,1], \ldots\ldots, & C[N-1,N-1] \end{pmatrix} = \begin{pmatrix} A[0,0], & A[0,1], \ldots\ldots, & A[0,N-1] \\ A[1,0], & A[1,1], \ldots\ldots, & A[1,N-1] \\ \multicolumn{3}{l}{\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots} \\ A[N-1,0], & A[N-1,1], \ldots\ldots, & A[N-1,N-1] \end{pmatrix} \times \begin{pmatrix} B[0,0], & B[0,1], \ldots\ldots, & B[0,N-1] \\ B[1,0], & B[1,1], \ldots\ldots, & B[1,N-1] \\ \multicolumn{3}{l}{\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots} \\ B[N-1,0], & B[N-1,1], \ldots\ldots, & B[N-1,N-1] \end{pmatrix}$$

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] * A[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

$$\begin{pmatrix} C[0,0], & C[0,1], \ldots\ldots, & C[0,N-1] \\ C[1,0], & C[1,1], \ldots\ldots, & C[1,N-1] \\ \ldots\ldots\ldots\ldots & \ldots\ldots\ldots\ldots & \ldots\ldots \\ C[N-1,0], & C[N-1,1], \ldots\ldots, & C[N-1,N-1] \end{pmatrix}$$

Matrix multiply and threaded matrix multiply

- Threaded code program
  - From N, nthreads, myid
    - I am responsiable for sub-array
      C[0..N-1][N/Nthreads*myrank ..N/Nthreads*(myrank+1))
  - The calculation of c[I,j] does not depend on other C term. Mm_pthread.c.

$$
\begin{pmatrix}
C[0,0], & C[0,1], \ldots\ldots, & C[0, N-1] \\
C[1,0], & C[1,1], \ldots\ldots, & C[1, N-1] \\
\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots & & \\
C[N-1,0], & C[N-1,1], \ldots\ldots, & C[N-1, N-1]
\end{pmatrix}
$$

PI calculation

$$PI = \lim_{n \to \infty}\left(\frac{1}{n}\sum_{i=1}^{n}\frac{4.0}{1.0+\left(\frac{i-0.5}{n}\right)^2}\right)$$

- Sequential code: pi.c
- Multi-threaded version: pi_pthread.c
  - Again domain partition based on N, nthreads, and myid.