CS 444 Operating Systems

POSIX Threads

J. Holly DeBlois

August 20, 2024

POSIX Threads

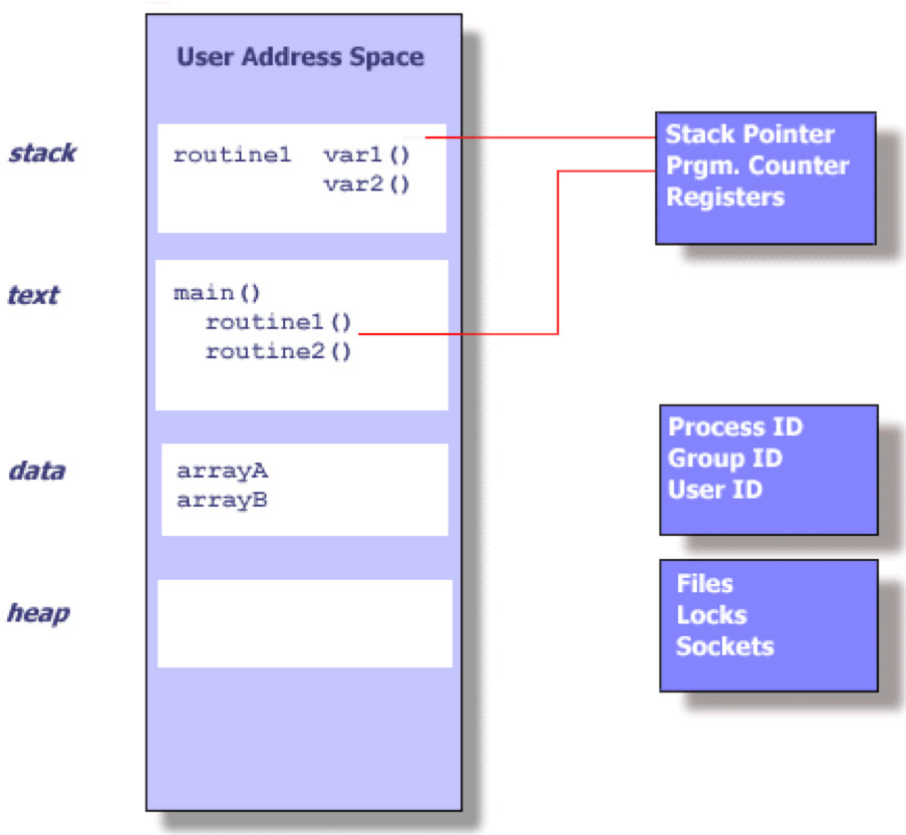
- The POSIX thread tutorial at Lawrence Livermore National Laboratory
- `https://hpc-tutorials.llnl.gov/posix/`

Process and Threads

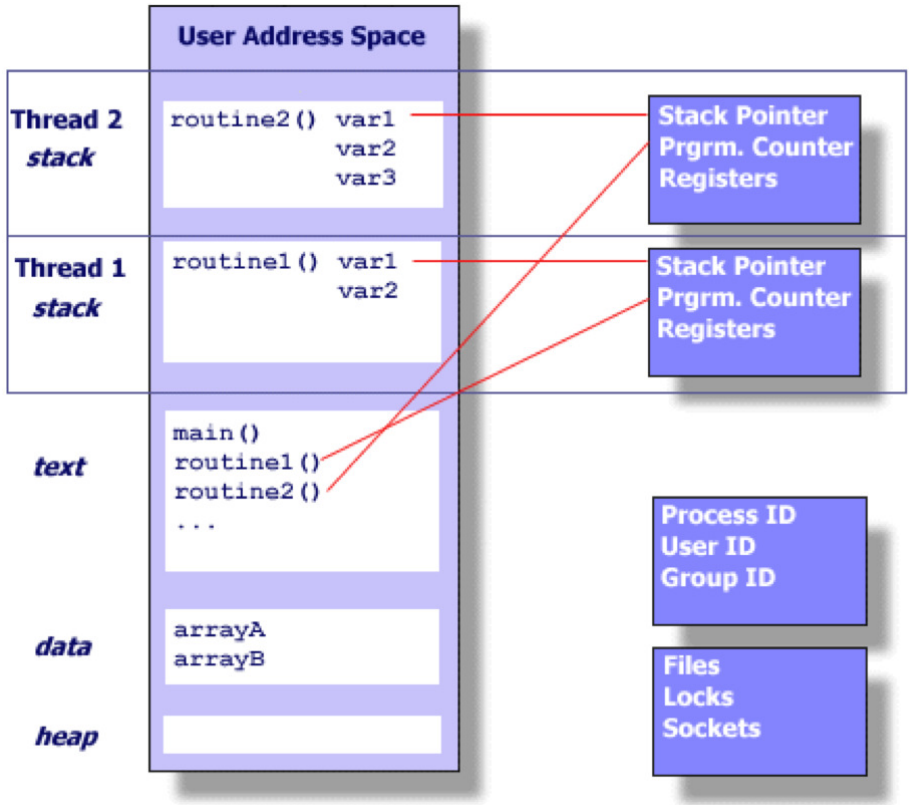
- A main program (process) contains several procedures (threads)
- These procedures can be scheduled to run simultaneously and independently by the OS
- A thread has its own independent control flow

Process and Thread Call Stacks

Process



Process and two threads



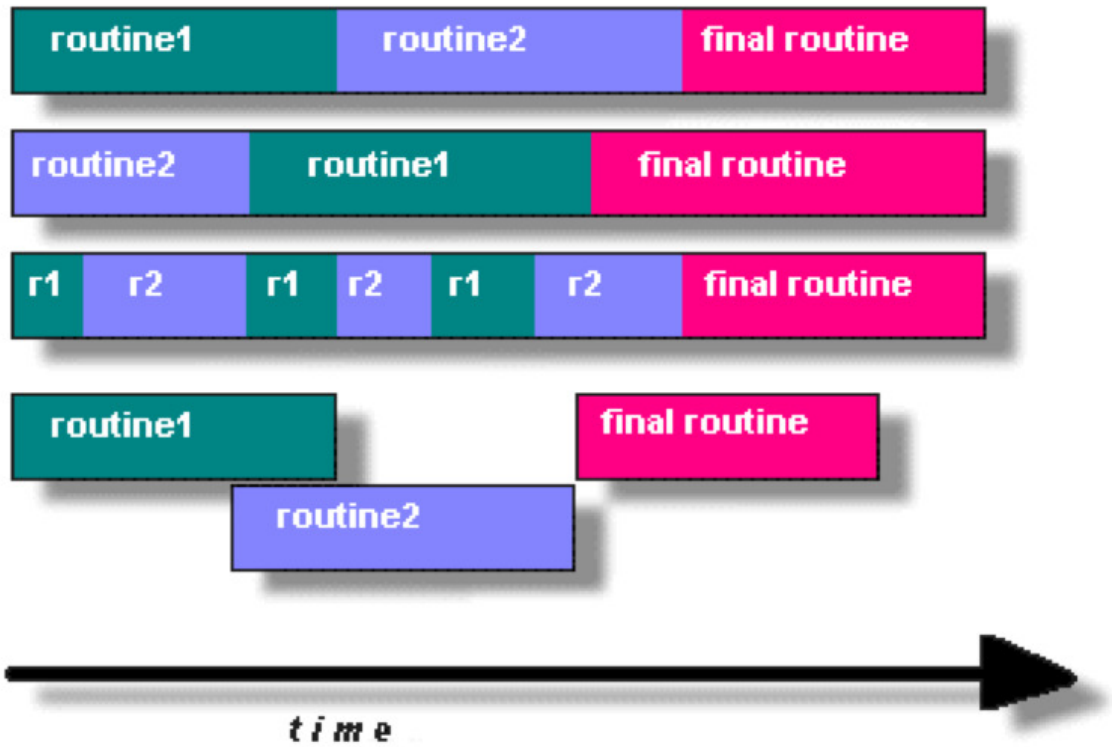
A Thread Maintains Its Own Data

- Stack pointer
- Registers
- Scheduling parameters
- Set of pending and blocked signals
- Thread specific data
- Light weight — see the table comparing timing results of `fork()` and `pthread_create()`

Advantages

- In addition to being light weight, threads have the following advantages
- Overlap CPU work with I/O work
- Better scheduling, priority or real-time
- Asynchronous event handling
- On an SMP machine
  - MPI communication involves memory copy, process to process
  - Pthread uses cache-to-CPU or RAM-to-CPU
  - See the table comparing MPI and Pthread memory bandwidths

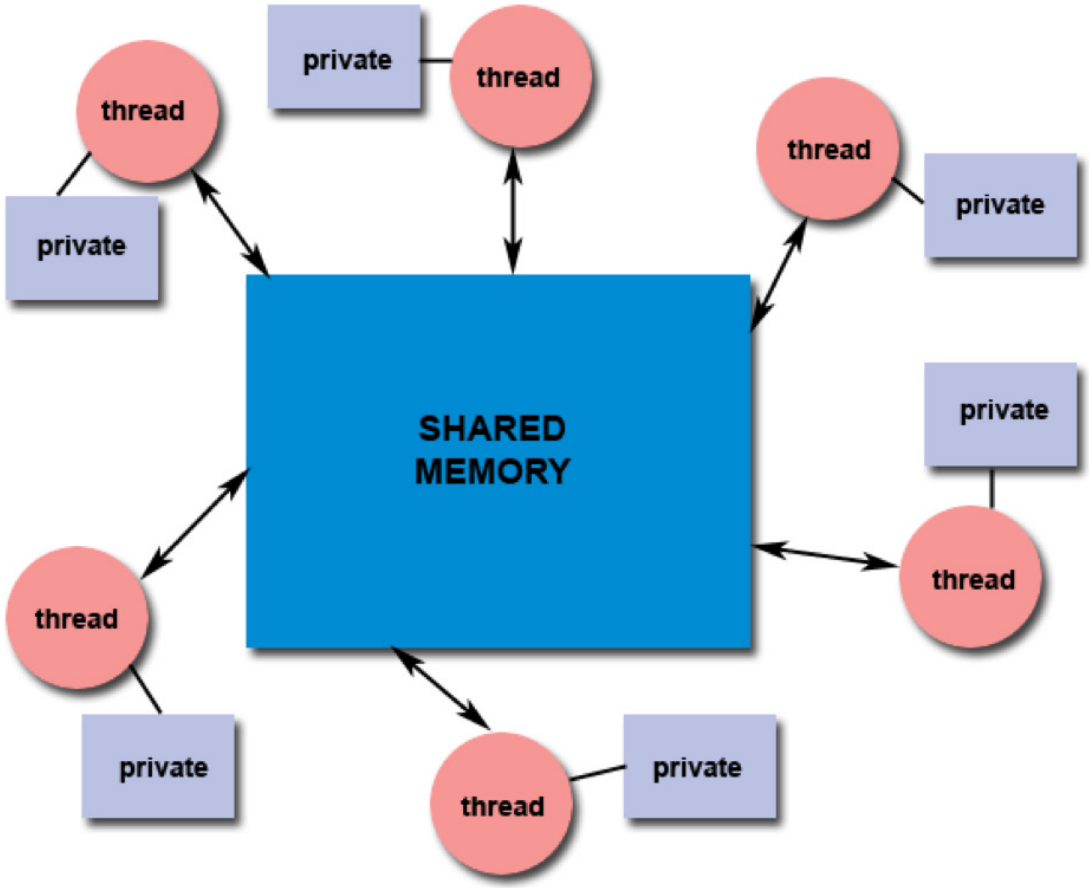
Some Tasks Can Be Overlapped



Typical Models of Multithread Programs

- Manager/worker
  - The manager thread assigns work to worker threads
  - The manager handles all I/O
  - The worker threads can be static or dynamic
- Pipeline
- Peer

Shared Memory Model



Thread Safety

- A library function is thread-safe if it can be called by multiple threads simultaneously
- `drand48()` is not thread-safe
- `drand48_r()` is thread-safe

Pthread API

- `gcc -pthread file.c`, or `g++ -pthread file.cpp`
- `#include <pthread.h>`
-
- Thread management
- Mutex
- Condition variable

Thread Management

- `pthread_create()`
- `pthread_exit()`
-
- `pthread_attr_init()`
- `pthread_attr_setdetachstate()`
- `pthread_attr_destroy()`
-
- `pthread_join()`
-
- Stack management

Mutex

- `pthread_mutex_t myMutex;`
-
- `pthread_mutex_init()`
- `pthread_mutex_destroy()`
-
- `pthread_mutex_lock()`
- `pthread_mutex_unlock()`
-
- Example: `innerProduct.c`

You Must Acquire Mutexes in a Fixed Sequence

```
#define N 100
typedef int semaphore;
semaphore mutex = 1;
semaphore empty = N;
semaphore full = 0;
void producer(void){
    int item;
    while (1) {
        item = produceItem();
        down(&empty);
        down(&mutex);
        insertItem(item);
        up(&mutex);
        up(&full);
    }
}
```

```
void consumer(void){
    int item;
    while (1) {
        down(&full);
        down(&mutex);
        item = removeItem();
        up(&mutex);
        up(&empty);
        item = consumeItem();
    }
}
```

Condition Variable

Thread A

```
pthread_mutex_lock(&mtx);
...
if (condition)
  pthread_cond_signal(&cv);
//pthread_cond_broadcast()
...
pthread_mutex_unlock(&mtx);
```

Thread B

```
pthread_mutex_lock(&mtx);
...
if (!condition)
  pthread_cond_wait(&cv,
&mtx);
...
pthread_mutex_unlock(&mtx);
```