Threads

Instructor: Linyi Li
Slides adapted from Dr. B. Fraser

Topics

- What is thread?
- How can two “threads of execution” share the same memory space?
- How can we start and work with threads?

What's a Thread

- What is a Thread?
  - .. A thread is a unit of execution.
  - Similar to a process but it's lighter weight.
  - Sometimes called.. a "light-weight process".
- Main Thread
  - A process always has at least one thread, called main thread: from main()

Details

- Can find more info in OSTEP book
  (more depth than we require)

  – Chapter 26 Concurrency: An Introduction
  https://pages.cs.wisc.edu/~remzi/OSTEP/threads-intro.pdf

  – Chapter 27 Interlude: Thread API
  https://pages.cs.wisc.edu/~remzi/OSTEP/threads-api.pdf

Threads vs Processes

Thread vs Process

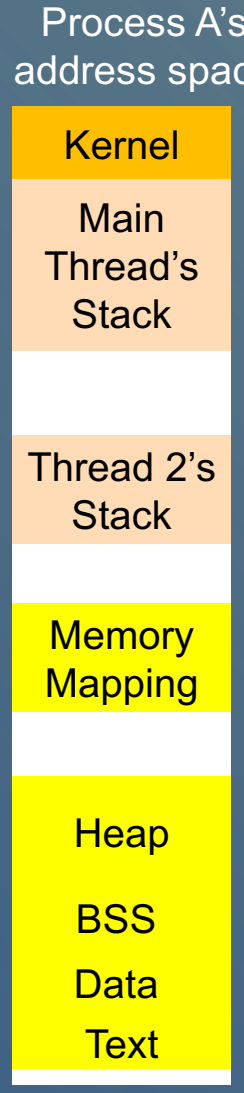
- If threads and processes execute in parallel on different cores, then what are the difference?
  - .. Processes have separate (virtual) address spaces.

fork() creates a child process with its own address space.

  - .. Threads share the same address space:

text, data, bss, and heap segments.

- Each thread gets its own:
  - stack
  - registers
  - program counter (running different function).
  - errno



Benefits on Threads & Processes

- Benefit of a thread
  - Threads in a process share the same addresses
  - .. hence data sharing is easy.
  - E.g., any thread can read from or write to a global variable.

  Can pass pointers between them.
  - .. Threads faster to create (lightweight)
- Benefit of a process
  - .. Memory isolation

POSIX Threads

man pthreads

```
pthreads(7)              Miscellaneous Information Manual              pthreads(7)

NAME
       pthreads - POSIX threads

DESCRIPTION
       POSIX.1 specifies a set of interfaces (functions, header files) for
       threaded programming commonly known as POSIX threads, or Pthreads.  A sin-
       gle process can contain multiple threads, all of which are executing the
       same program.  These threads share the same global memory (data and heap
       segments), but each thread has its own stack (automatic variables).
```

- man pthreads
  Review sections:
  - Description: What's shared & not
  - Return value & errno
  - Thread ID
  - Thread safe Functions

Common Functions

- pthread_create()
  - Check man page
  - pthread_t: this is the type used for thread IDs.
  - .. Function pointer to a thread function (that will run as a thread).
  - void *arg, passed in.
    - void* can be cast to any pointer. Use a struct to pass multiple arguments.
  - pthread_attr_t specifies various attributes of the new thread.
- pthread_exit() terminates the calling thread.
  - Done implicitly when returning from thread function

```
return 0;       // Leaving thread function!
```

Common Functions

- pthread_self()
  - Returns the caller's id
  - .. Use gettid() to be able to print Linux thread ID as a number.
- pthread_join()
  - .. Waits until that thread terminates.
  - Thread return value with `void **retval`.
- pthread_detach()
  - Lets the calling thread just run.
  - You can use this when you don't need to return anything.

ABCD: PThread

- Each thread gets its own...

(a) **Stack**

(b) **Heap**

(c) **Text / Code**

(d) **stdout**

ABCD: pthread_create()

- Which of the following is true about pthread_create()?

(a) It creates a new process running the provided thread start function.

(b) It passes nothing to the function (void).

(c) It waits until the spawned thread finishes.

(d) It stores the thread_id for later user.

Pthread Activity

- [15 min] Write a program where:
  - Main thread will:
    - create another thread.
    - wait until thread terminates,
    - print out the return value.
  - New thread accept a string as its argument,
    - print out the argument and its own ID (use gettid()),
    - return the length (using strlen(char*)) of the received string.

Start simple! Make a thread and print “hi”!

Thread function can return a number:

```
return (void*) 42;
```

main() can get the number:

```
void* ret_val = 0;
pthread_join(...);
printf(“%ul”, (uint64_t) ret_val);
```

  - Compile with `-pthread` compiler option:

e.g., `clang -pthread example.c`.

Data Race

Data Race Activity

- [10 min] Activity:
  write a program that has two additional threads.
  - Create global variable:

```
int cnt = 0;
```

  - Each new thread adds 1 to `cnt` 10 million times.
  - Main thread waits for new threads, and prints `cnt`.
- Run multiple times; see output!

Deterministic

- Deterministic:
  - .. Where the program output is the same every time.

Usually, this is what we want!

- Note that the "behaviour" might not be the same each time: the order that threads get scheduled will be different each time.
- However, non-deterministic behaviour does not lead to non-deterministic output unless you have a race case.

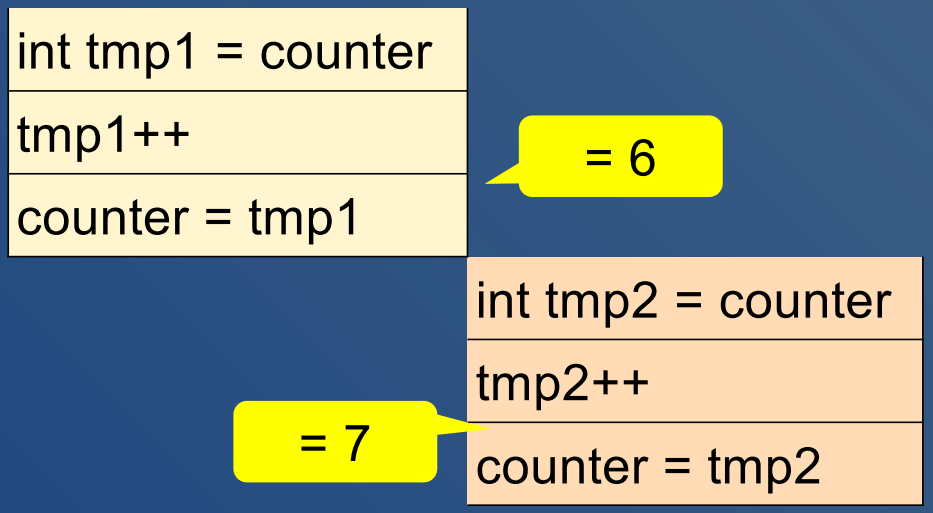
Usually, what we want to avoid!

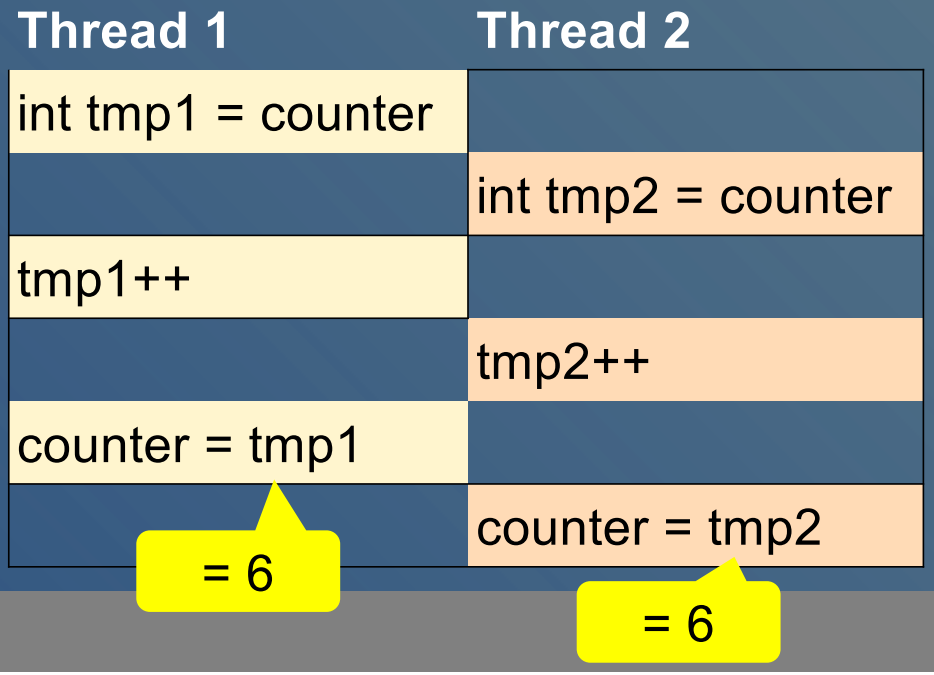
Data Race Problem

- Consider the statement counter++
  - It seems like `counter++` is one operation.
  - .. In reality, it's more like:

```
int tmpRegister = counter;          // Load from memory
tmpRegister++;                      // Charge value
counter = tmpRegister;              // Store value to memory
```

- What happen if this runs on 2 threads? (assume counter = 5)





Race Condition

- Data Race
  - This is called the data race problem:
    - where different threads *race* to update data and overwrite each other's result.
- Race Condition
  - More generally, a race condition is a condition in which
    - the correctness of a program depends on the timing and/or order of operations.
- Difference between a Data Race and a Race Condition
  - Very similar and related ideas.
  - We'll not get into details. For more info: https://blog.regehr.org/archives/490

Thread & Linux Scheduling

- Does Linux schedule at the process or thread level?

  – Linux schedules at the thread level.

  – The Linux scheduler makes no fundamental distinction between a process and a thread; both are "tasks" from the kernel's perspective.

- Does thread implementation in Linux require system calls?

  – Yes. Both process & thread use clone system calls to create, but with different sharing resources.

Summary

- Threads
  - Lightweight processes that share a memory space.
  - Always have main thread
- PThread
  - POSIX library / API for threads
  - pthread_create(), pthread_join(), ...
- Data Race
  - When two threads may access the same data at the same time.