20. Concurrency and Threads

a single thread program : a single PC / multi-thread : multi PCs
"multiple execution points"
* PC : where instructions are being fetched from & executed

thread : like a separate process / but,
they share the same address space => thus can access the same data
=> no need to switch which page table we are using

Each thread has its **own private set of registers** it uses for computation
=> when switching, a context switch must take place.
the **register state** must be saved/restored

With processes, we saved state to a process control block (PCB)
=> with threads : one or more thread control blocks (TCBs)

in a multi-threaded process,
each thread runs independently & may call into various routines.

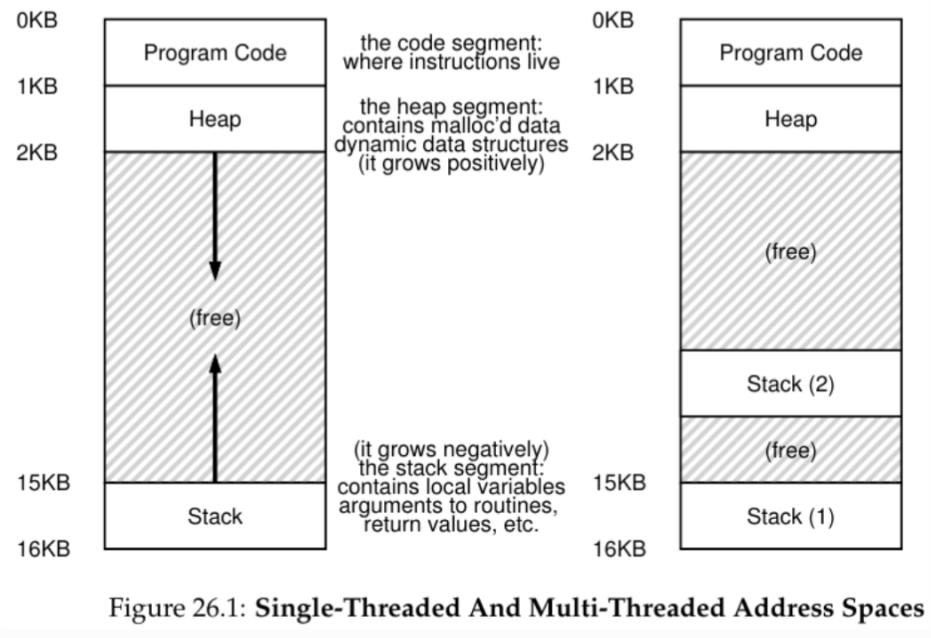

Figure 26.1: **Single-Threaded And Multi-Threaded Address Spaces**

stack per a thread == thread-local storage
=> the space is seperated.
but usually OK, since stacks do not generally have to be very large
(except heavy use of recursion)

Why Use Threads?

parallelism

if you have multiple processors, you have the potential of speeding up this process considerably by using the processors to each perform a portion of the work

parallelization
task of transforming single-threaded program => work on multiple CPU
= using a thread per CPU

avoid blocking program progress due to slow I/O

instead of waiting, you can utilize the CPU to perform something
ex. while one thread waits, the CPU scheduler can switch to other threads, which are ready to run something useful
=> Threading enables overlap of I/O with other activities within a single program, much like multiprogramming did for processes across programs
=> many modern server-based applications make use of threads in their implementations

- thread : natural choice when share data (address space are shared)
- process : for logically seperate tasks where little sharing of data structures in memory

Thread Creation

```
1  #include <stdio.h>
2  #include <assert.h>
3  #include <pthread.h>
4  #include "common.h"
5  #include "common_threads.h"
6
7  void *mythread(void *arg) {
8      printf("%s\n", (char *) arg);
9      return NULL;
10 }
11
12 int
13 main(int argc, char *argv[]) {
14     pthread_t p1, p2;
15     int rc;
16     printf("main: begin\n");
17     Pthread_create(&p1, NULL, mythread, "A");
18     Pthread_create(&p2, NULL, mythread, "B");
19     // join waits for the threads to finish
20     Pthread_join(p1, NULL);
21     Pthread_join(p2, NULL);
22     printf("main: end\n");
23     return 0;
24 }
```

Figure 26.2: **Simple Thread Creation Code (t0.c)**

pthread_join() : waits for a particular thread to complete
there are 3 threads in code : main, t1, p2
what runs next is determined by the OS scheduler

Know and Use your Tools

Disassembler

if we wish to understand the low-level code to update a counter => we run objdump (Linux) / objdump -d main

debugger like **gdb**, memory profilers like **valgrind** or **purify**

Each thread when running has its own private registers

=> the registers are virtualized by teh context-switch code that saves & restores them

Critical Section

piece of code that accesses a shared resource

Race condition (data race)

if multiple threads of execution enter the critical section at roughly the same time => both attempt to update the shared data structure, leading to strange outcome

Indeterminate program

: consists of one or more race conditions
=> the outcome is not deterministic

to avoid these problems, threads should use some kind of
mutual exclusion primitives
doing so guarantees that only a single thread ever enters a critical section
=> avoiding race => resulting in deterministic program outputs

Use Atomic Operations : all or nothing

transaction : grouping of many actions into a single atomic action
to make atomic => synchronization primitives

how to build support for synchronization primitives

- to support atomicity
- for mechanisms to support sleeping/waking interaction that is common in multi-threaded programs
  * where one thread must wait for another to complete some action before it continues

The OS was the first concurrent program