Introduction to Computer Systems
15-213/18-243 Spring 2009
April 28, 2009

Threading and Thread Safety

Updated version of Fall 2002 recitation slides

Overview

- **News**
- **Threading**
  - Basics
  - Thread Lifecycle
- **Thread Safety**
  - Race Conditions
  - Synchronization Techniques
- **Proxy Lab**

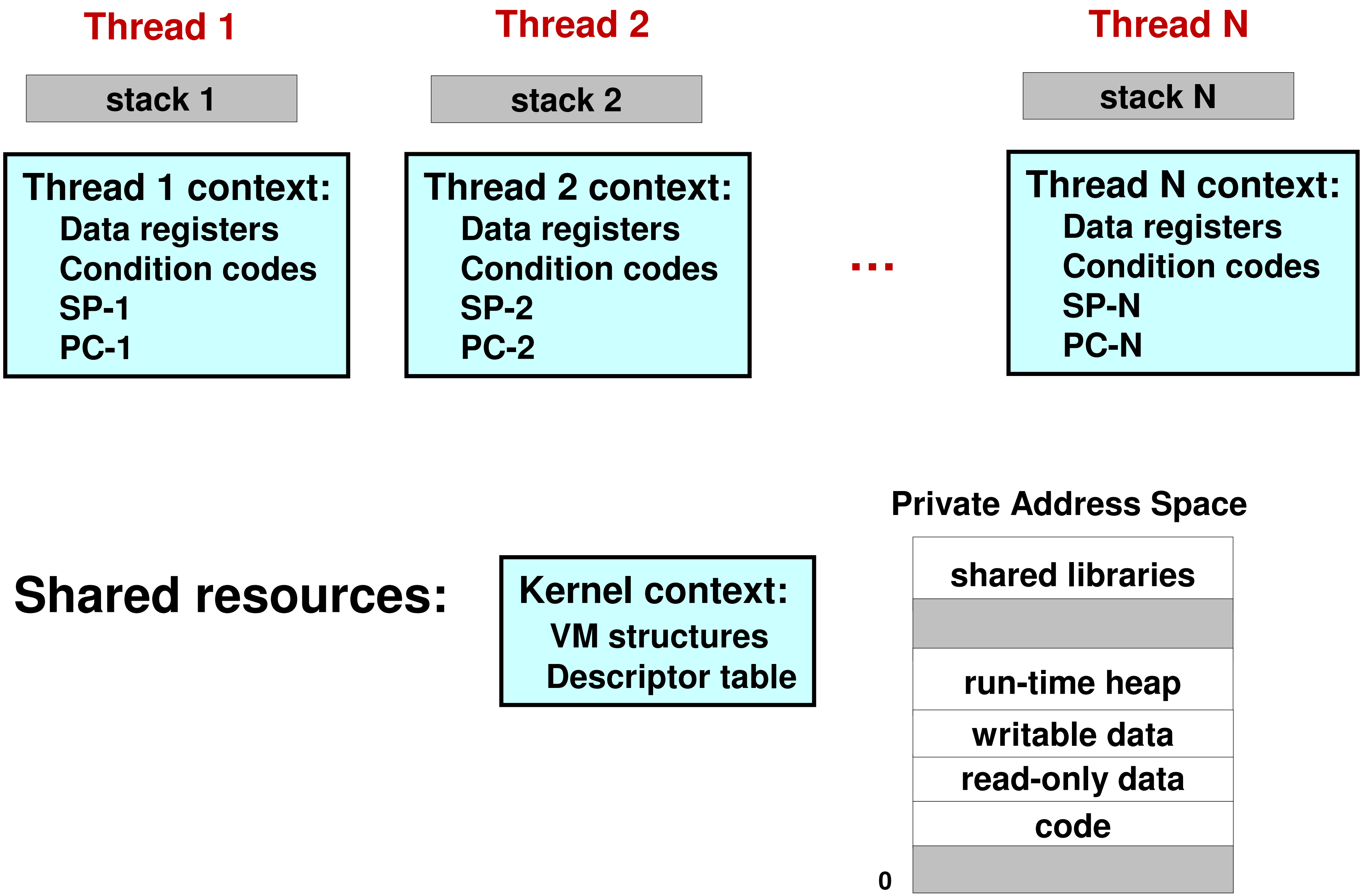
News

- **Proxy due Friday at 11:59pm**
  - **NO LATE DAYS**

- **Final exam: Tue May 12, at 8:30am**

Threading

Multi-Threaded process



Thread 1

stack 1

Thread 1 context:
Data registers
Condition codes
SP-1
PC-1

Thread 2

stack 2

Thread 2 context:
Data registers
Condition codes
SP-2
PC-2

...

Thread N

stack N

Thread N context:
Data registers
Condition codes
SP-N
PC-N

Shared resources:

Kernel context:
VM structures
Descriptor table

Private Address Space

shared libraries

run-time heap

writable data

read-only data

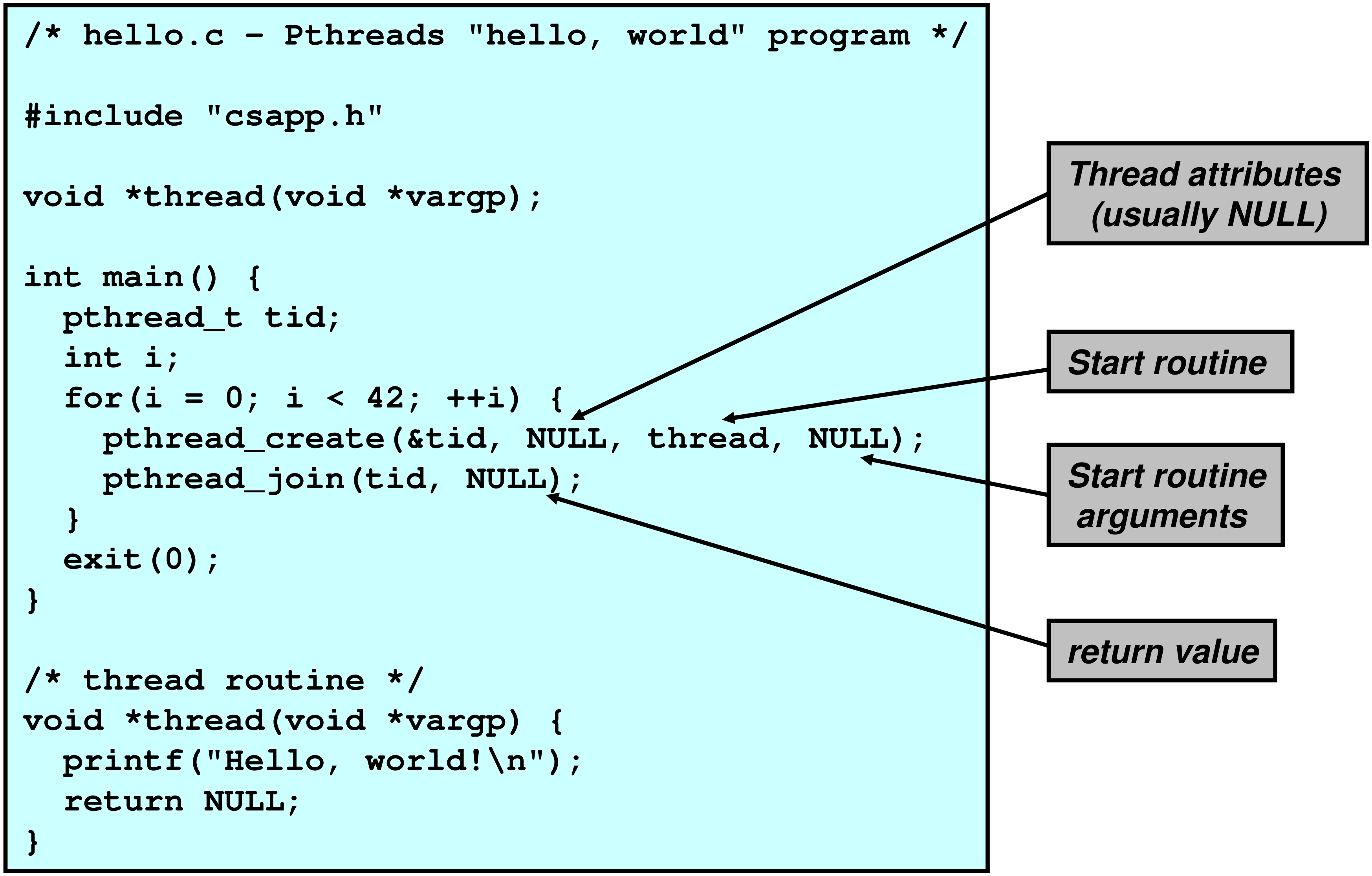
code

0

Posix Threads (Pthreads) Interface

- **Standard interface for ~60 functions**
  - Creating and reaping threads.
    - pthread_create
    - pthread_join
    - pthread_detach
  - Determining your thread ID
    - pthread_self
  - Terminating threads
    - pthread_cancel
    - pthread_exit
  - Synchronizing access to shared variables
    - sem_init
    - sem_wait
    - sem_post
    - pthread_rwlock_init
    - pthread_rwlock_[wr]rdlock

Multi-threaded Hello World



```
/* hello.c - Pthreads "hello, world" program */

#include "csapp.h"

void *thread(void *vargp);

int main() {
  pthread_t tid;
  int i;
  for(i = 0; i < 42; ++i) {
    pthread_create(&tid, NULL, thread, NULL);
    pthread_join(tid, NULL);
  }
  exit(0);
}

/* thread routine */
void *thread(void *vargp) {
  printf("Hello, world!\n");
  return NULL;
}
```

Thread attributes (usually NULL)

Start routine

Start routine arguments

return value

Exiting a process and thread

- pthread_exit() only terminates the current thread, NOT the process

- exit() terminates ALL the threads in the process, i.e., the process itself

Joinable & Detached Threads

- **Joinable thread can be reaped and killed by other threads**
  - must be reaped (with pthread_join) to free memory resources.

- **Detached thread cannot be reaped or killed by other threads**
  - resources are automatically reaped on termination.

- **Default state is joinable**
  - use pthread_detach(pthread_self()) to make detached.

Thread Safety

Race condition

- **A race occurs when the correctness of a program depends on one thread reaching point x in its control flow before another thread reaches point y.**
  - Access to shared variables and data structures
  - Threads dependent on a condition

- **Use synchronization to avoid race conditions**

- **Ways to do synchronization**
  - Semaphores
  - Mutex
  - Read-write locks

Synchronization

- **Semaphore**
  - Restricts the number of threads that can access a shared resource
- **Mutex**
  - Special case of semaphore that restricts access to one thread
- **Read-write locks**
  - Multiple readers allowed
  - Single writer allowed
  - No readers allowed when writer is present

Semaphore

- **Classic solution: Dijkstra's P and V operations on semaphores.**

- **Semaphore: non-negative integer synchronization variable.**
  - P(s): [while (s == 0) wait(); s--;]
  - V(s): [s++;]
  - OS guarantees that operations between brackets [ ] are executed indivisibly.
  - Only one P or V operation at a time can modify s.
  - Semaphore invariant: (s >= 0)
  - Initialize s to the number of simultaneous threads allowed

Posix synchronization functions

- **Semaphores**
  - sem_init
  - sem_wait
  - sem_post

- **Read-write locks**
  - pthread_rwlock_init
  - pthread_rwlock_rdlock
  - Pthread_rwlock_wrlock

Proxy Lab

- **Graceful error handling**
- **Document design decisions**
- **Code organization**
  - Break proxy into multiple functions
- **Complete lab in three stages**
  - Basic sequential proxy
  - Handling concurrent requests
  - Caching
- **Understand what is robust about the rio package**
  - Behavior of network sockets

Exam Review

Questions?