BUCKNELL UNIVERSITY

Computer Science

CSCI 315 Operating Systems Design

Thread Fundamentals

Notice: This set of slides is based on the notes by Professor Perrone of Bucknell and the textbook authors Silberschatz, Galvin, and Gagne, as well as the tutorial by Blaise Barney from Lawrence Livermore National Lab

A Different Model for Process Communication

- We discussed two forms of IPC
  - Shared memory and message passing
- In message passing, the communicating processes are running in different context, thus passing information is slower;
- In shared memory, IPC is faster. However, we need to set up the shared memory.
- In this segment, we explore a different model for processes to communicate with shared memory, that is, using *threads.*

What Is A Thread?

A *thread* is a light-weight process.

- Shared code
- Shared data
- Shared heap

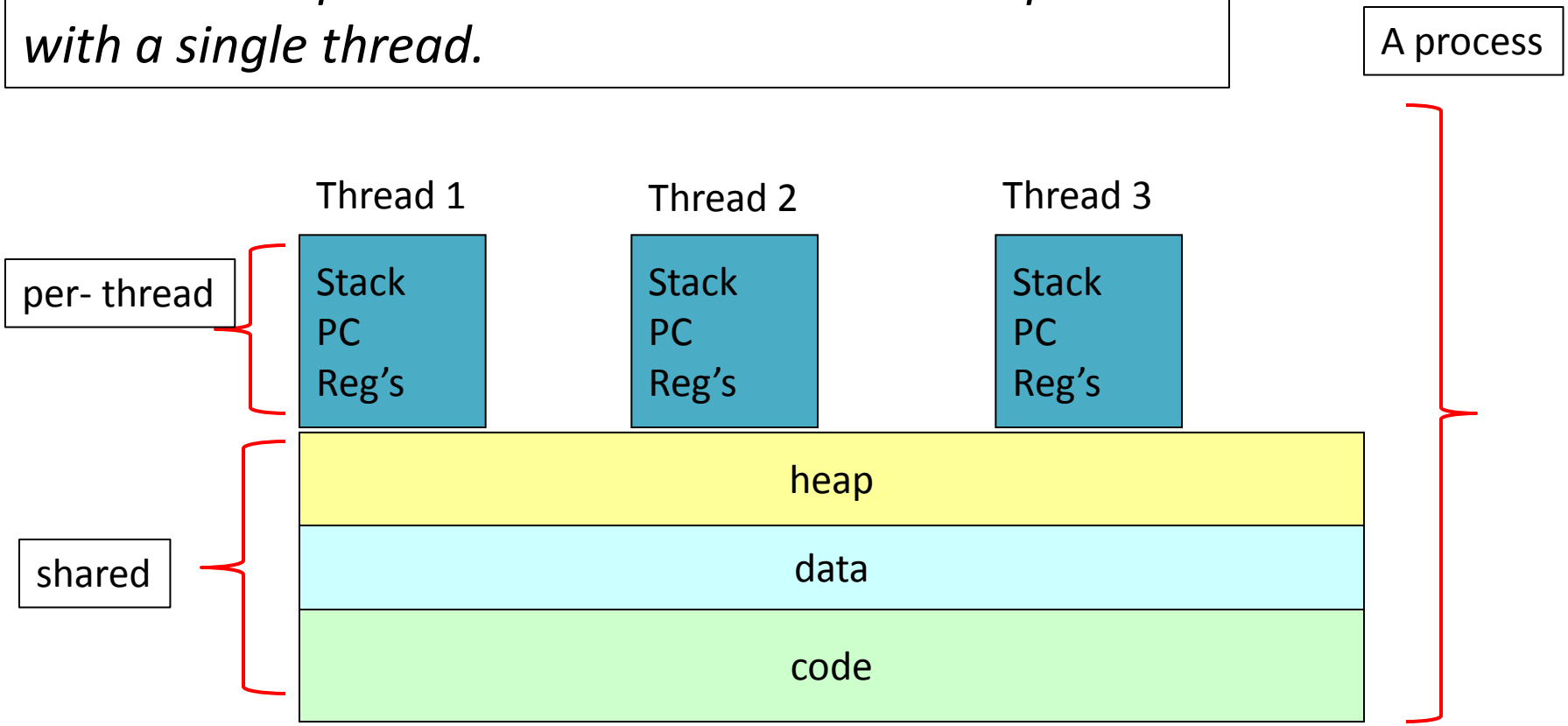
In memory, just like processes

- Independent PC
- Independent registers

Compared to: a *process* is a program in execution.

Process and Thread

Example: *A process that contains three threads. A traditional process can be considered as a process with a single thread.*



Why Threads?

- **Responsiveness:** multiple threads can be executed in parallel, reducing the completion time needed for a problem
- **Resource sharing:** multiple threads have access to the same data, sharing made easier
- **Economy:** creating process (allocating memory and other resources) is costly. For the same number of execution units, threads are less expensive
- **Scalability:** thread model can be easily scaled up

POSIX Threads

- While threads can be implemented in many different ways, the POSIX thread is a popular and effective implementation of threads on UNIX-like system
- POSIX: Potable Operating Systems Interface

A Simple, Complete Thread Example

```
/* gcc thisfile.c -lpthread */
#include <stdio.h>
#include <pthread.h>

#define NUM_THREADS 5
#define SLEEP_TIME  3
void *sleeping(void *);   /* thread routine */

int  main(int argc, char *argv[]) {

  int i;
  pthread_t tid[NUM_THREADS];      /* array of thread IDs */

  for ( i = 0; i < NUM_THREADS; i++)
    pthread_create(&tid[i], NULL, sleeping,(void *)SLEEP_TIME);

  for ( i = 0; i < NUM_THREADS; i++)
    pthread_join(tid[i], NULL);
  printf("main() reporting that all %d threads have terminated\n", i);
  return (0);
}  /* main */
```

http://www.eg.bucknell.edu/~cs315/Fall13/code/thread/trd-sleep.c

The Thread Work: *sleeping()*

```
void * sleeping(void *arg)   {

  int sleep_time = (int)arg;
  printf("thread %ld sleeping %d seconds ...\n", pthread_self(),
     sleep_time);
  sleep(sleep_time);
  printf("\nthread %ld awakening\n", pthread_self());
  return (NULL);
}
```

Compile and Execute the Program

```
[xmeng@linuxremote]$ gcc -o thread-sleep trd-sleep.c –lpthread
[xmeng@linuxremote]$ ./thread-sleep
thread 140550497642240 sleeping 3 seconds ...
thread 140550518621952 sleeping 3 seconds ...
thread 140550508132096 sleeping 3 seconds ...
thread 140550476662528 sleeping 3 seconds ...
thread 140550487152384 sleeping 3 seconds ...
thread 140550497642240 awakening
thread 140550518621952 awakening
thread 140550508132096 awakening
thread 140550487152384 awakening
thread 140550476662528 awakening
main() reporting that all 5 threads have terminated
[xmeng@linuxremote]$
```

Creating Threads

```
#include <pthread.h>
```

Including the pthread library headers

```
pthread_create(&tid[i],     NULL, sleeping,
               (void *)SLEEP_TIME);
```

Creating threads

Thread ID

Thread attributes, NULL for now

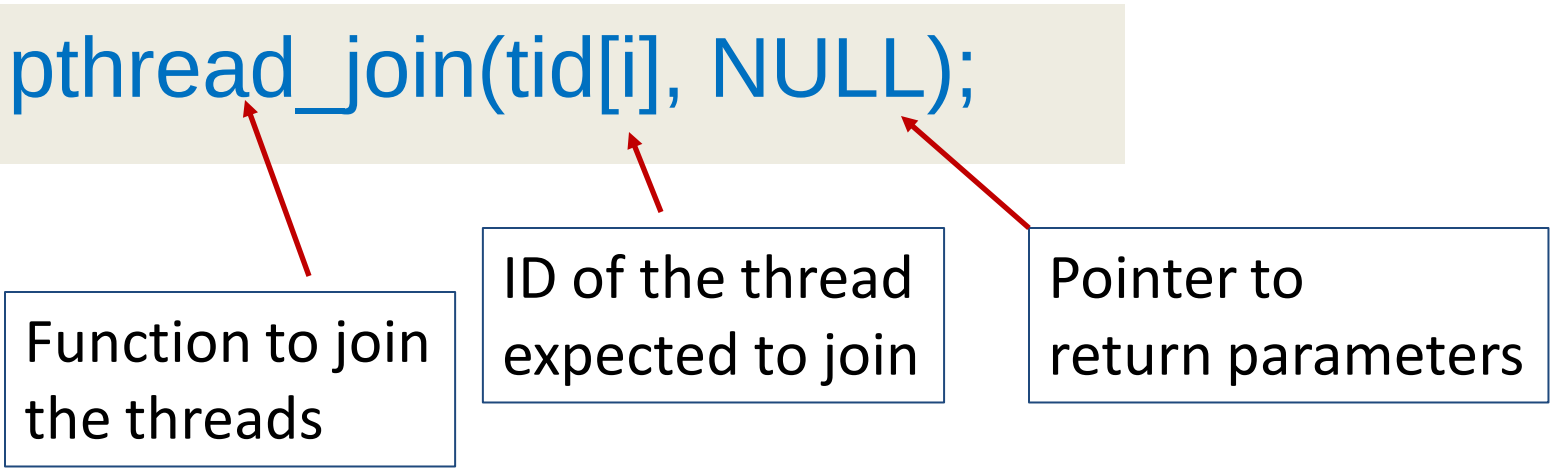
Pointer to the parameter block

Name of thread worker function

As soon as threads are created, they start to execute the *worker* function

Joining Threads When Finishing

```
pthread_join(tid[i], NULL);
```



The second parameter is of the type void **ptr, which is an address to a pointer (pointer to a pointer). If it is used, usually it returns the exit status of the thread.

Global Variables Among Threads

- Global variables and data structures among threads are shared
- They are in the “data” segment of the memory
- In our example, the following are “global” that every thread sees and can access

```
#define NUM_THREADS 5
#define SLEEP_TIME  3
void *sleeping(void *);   /* thread routine */
```

An Example of Shared Data

```
#include <stdio.h>
#include <pthread.h>
#define NUM_THREADS  5
void *work(void *);   /* thread routine */
int v = 0;  /* global variable, shared */
int  main(int argc, char *argv[]) {
      int i;
      pthread_t tid[NUM_THREADS];      /* array of thread IDs */
      for ( i = 0; i < NUM_THREADS; i++)
            pthread_create(&tid[i], NULL, work, NULL);
      for ( i = 0; i < NUM_THREADS; i++)
            pthread_join(tid[i], NULL);
      printf("main() reporting that all %d threads have terminated\n", i);
      printf("v should be %d, it is %d\n", NUM_THREADS, v);
      return (0);
}  /* main */
```

The Worker Function and Result

```
void * work(void *arg)   {
    v ++;    // 'v' is a global variable
    return (NULL);
}
```

```
[xmeng@polaris thread]$ ./trd-share
main() reporting that all 5 threads have terminated
v should be 5, it is 5
[xmeng@polaris thread]$
```

Everything seems working fine. However if one increases the number of threads to a larger value, e.g., 5000, we may see something incorrect.

http://www.eg.bucknell.edu/~cs315/Fall13/code/thread/trd-share.c

There May Be A Problem …

```
#define  NUM_THREADS  5000   // everything else is the same
```

```
[xmeng@polaris thread]$ ./trd-share
main() reporting that all 5000 threads have terminated
v should be 5000, it is 4998
[xmeng@polaris thread]$
```

Who stole the two counts from me?!!