CSCI4430 Data Communication and Computer Networks

Pthread Programming

HAN Shujie

sjhan@cse.cuhk.edu.hk

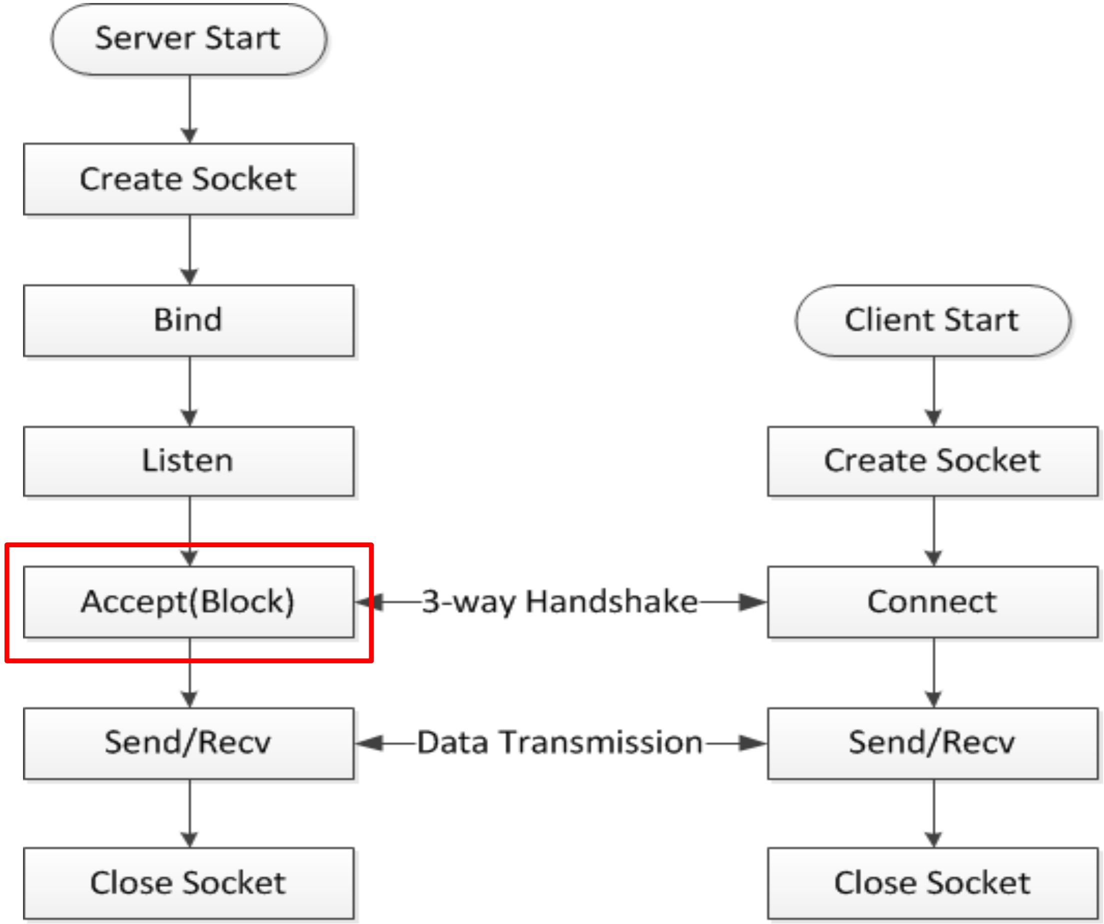
Jan. 23, 2020

Outline

- Recall
- Introduction
- What is Multi-thread Programming?
- Why to use Multi-thread Programming?
- Basic Pthread Programming
- Recommended Materials

Recall

- Network programming for TCP



Recall

- Basic Socket programming

➢ Server accepts connection requests

```
while(1){
        int client_sd = accept(sd,…);
        // Do something
}
```

➢ Exchange data

```
while(1){
        int len = recv(…);
        // Handle received messages
}
```

Introduction

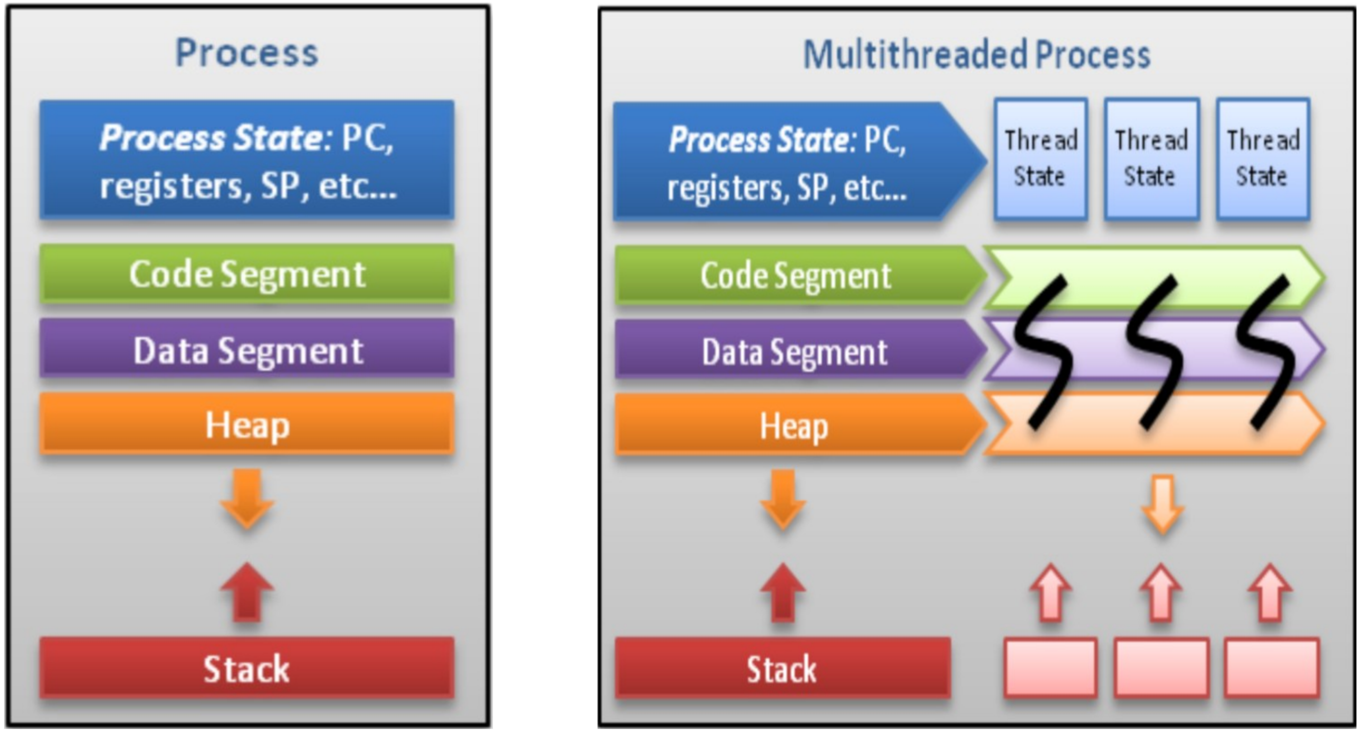
- Can we do **both** operations at the same time?
- Recall the blocking functions.
  - The whole program will be blocked waiting for incoming connection requests and data.
  - We cannot handle both with only **one** thread.

```
while(1){
        int client_sd = accept(sd..);
        …
}
```

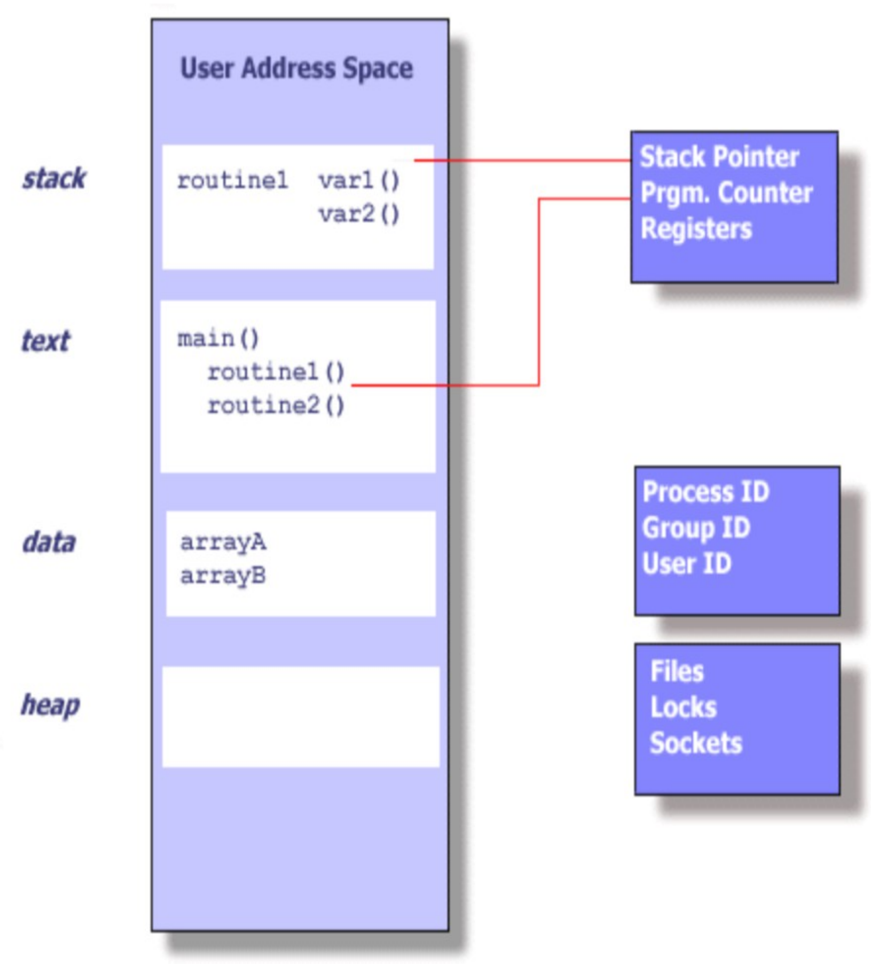
```
while(1){
        int len = recv(…);
        …
}
```

What is Multi-thread Programming

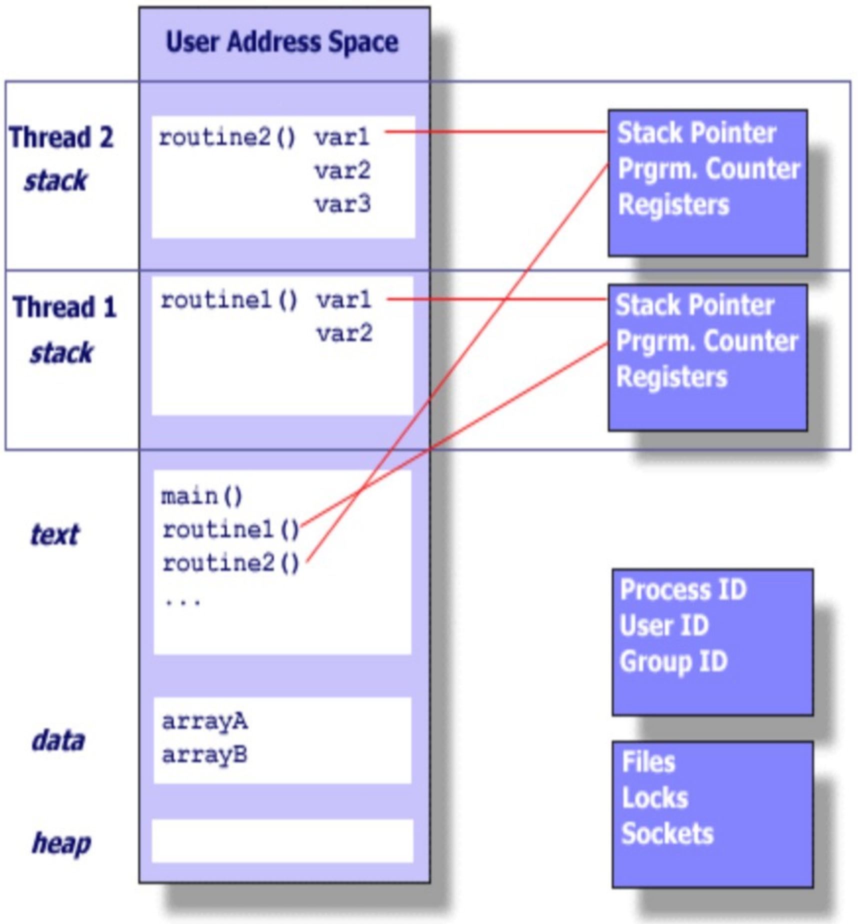
- A **thread** is a sequence of instructions within a program that can be executed independently of other code.



What is Multi-thread Programming



UNIX PROCESS



THREADS WITHIN A UNIX PROCESS

What is Multi-thread Programming

- Thread
  - Exists within one process.
  - Has independent flow of control.
    - Duplicates the essential resources only,
      - e.g., a stack, a copy of registers, program counts, etc.
    - May share the process resources
      - e.g., code, data, heap, etc.
  - Dies if the parent dies.
  - Is “lightweight”.

Why Multi-thread Programming

- Multi-thread programming
  - Shared data in one process.
  - A thread can be created with little operation system overhead.
  - Managing threads requires less system resources than managing processes.

Why Multi-thread Programming

- To accomplish the functionalities of the server within one program, we use multiple threads.
  - The blocking operations, will block one thread instead of the whole program.

Thread 1 (Parent):

```
while(1){
        int client_sd = accept(sd..);
        …
}
```

Thread 2 (Child):

```
while(1){
        int len = recv(...);
        …
}
```

Basic Pthread Programming

- Pthreads: POSIX standard threads.
  - Thread management: creating, detaching, joining, etc.
  - Mutexes: creating, destroying, locking and unlocking mutexes.
  - Condition variables: addressing communications between threads that share a mutex
  - Synchronization: managing read/write locks and barriers

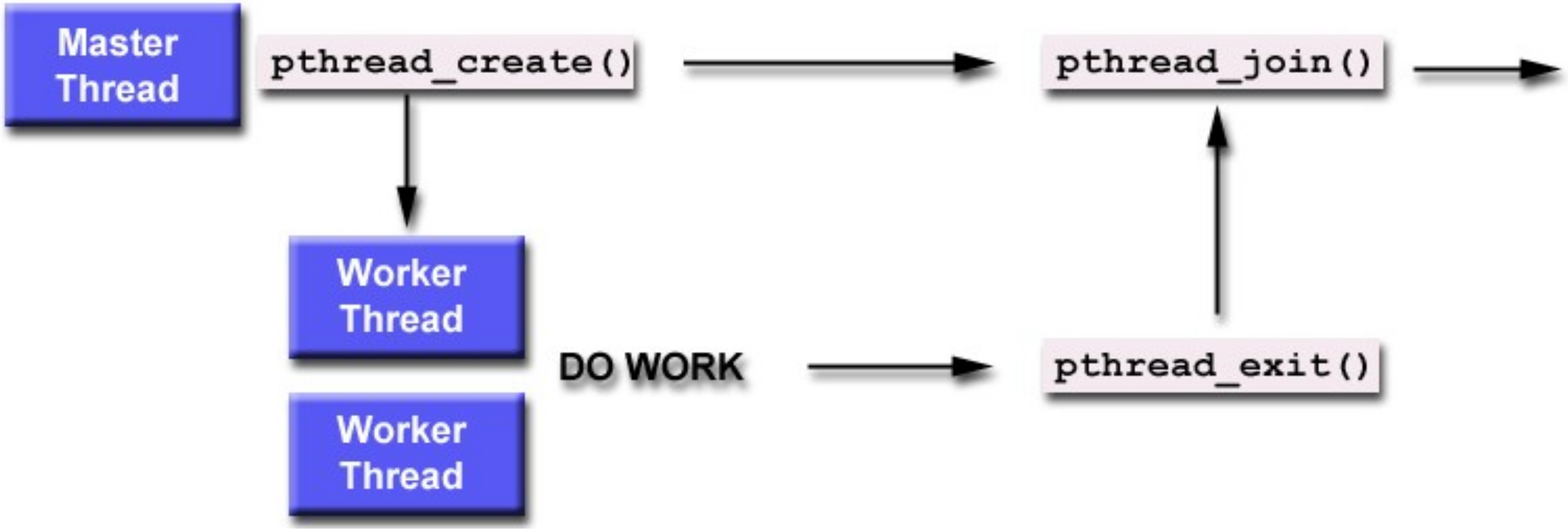
Basic Pthread Programming

- All identifiers in <pthread.h> begin with **pthread_**.
- Some examples are shown below.

Routine Prefix	Functional Group
pthread_	Threads themselves and miscellaneous subroutines
pthread_attr_	Thread attributes objects
pthread_mutex_	Mutexes
pthread_cond_	Conditional variables
pthread_rwlock_	Read/write locks

Basic Pthread Programming

- Most common model for threaded programs: Master/worker
  - A single thread, the master assigns work to other threads, the workers.
  - The master handles all input and parcels out work to the other tasks.



Basic Pthread Programming

- pthread_create()
  - Starts a new thread in the calling process.
  - Syntax
    - `int pthread_create(pthread_t * thread, const pthread_attr * attr,  void* (*start_routine)(void *), void* arg);`
  - Parameters
    - *thread*: the thread handler of the newly created thread;
    - *attr*: the attributes of the thread, in most cases set to NULL;
    - *start_routine*: the pointer pointing to the function which will run in the thread;
    - *arg*: the argument for the start_routine function NULL when there is no arguments.

Basic Pthread Programming

- pthread_create()
  - The new thread starts execution by invoking start_routine();
  - arg is passed as the sole argument of start_routine().
  - Example

```
pthread_t thread;
int rc = pthread_create(&thread, NULL, start_routine, NULL);
```

Basic Pthread Programming

- pthread_join()
  - Waiting for another thread to terminate
  - Syntax
    - `int pthread_join( thread_t* th,void ** thread_ret);`
    - *th*: waiting for the thread with the thread handler "th" to terminate
    - *thread_ret*: if the return value is not NULL, "thread_ret" will point to the place where the return value of thread "th" is stored
  - Example

```
pthread_join(thread, NULL);
```

Basic Pthread Programming

- pthread_detach()
  - detach a thread
  - Syntax
    - `int pthread_detach(pthread_t thread);`
- The resources of the detached thread can be reclaimed when that thread terminates.
  - This routine can be used to explicitly detach a thread even though it was created as joinable.
  - Detached thread can never be joined.
  - Use it carefully!

Basic Pthread Programming

- pthread_exit()
  - Termination of the calling thread
  - Syntax
    - `void pthread_exit( void * ret_value)`
    - *ret_value* is the return value of the thread, setting to NULL will be OK for most cases
  - Example

```
pthread_exit(NULL);
```

Basic Pthread Programming

- Return value of the thread
  - pthread_exit() will kill the thread and will never return. Thus,
    - Remember that the return value cannot be of local scope, otherwise when the thread terminates, the return value will not exist.
  - This value can be get and examined by some other thread with function pthread_join()

Basic Pthread Programming

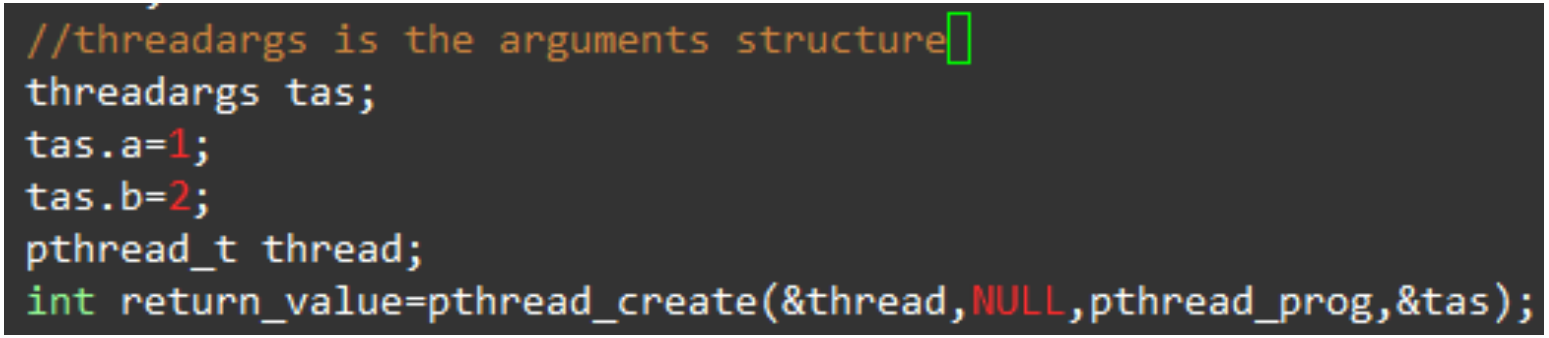
- Creating mutex
  - Statically, when it is declared. E.g.,
    - `pthread_mutex_t mymutex = PTHREAD_MUTEX_INITIALIZER;`
  - Dynamically, with
    - `pthread_mutex_init(mutex, attr)`
- Destroying mutex
  - `pthread_mutex_destroy(mutex)` should be used to free a mutex object which is no longer needed.

Basic Pthread Programming

- Locking mutex
  - `int pthread_mutex_lock(pthread_mutex_t *mutex)`
  - If the mutex is already locked by another thread, this call will block the calling thread until the *mutex* is unlocked.
- Unlocking mutex
  - `int pthread_mutex_unlock(pthread_mutex_t *mutex)`
  - Release the *mutex* after a thread has completed the use of protected data.

Transfer Data Among Threads

- Using global variable.
  - Do not forget mutex.
- Initialize the worker threads with arguments.
  - pthread_create()
    - Multiple arguments for start_routine
      - Always using a structure to pass the arguments
      - Example:



```
//threadargs is the arguments structure
threadargs tas;
tas.a=1;
tas.b=2;
pthread_t thread;
int return_value=pthread_create(&thread,NULL,pthread_prog,&tas);
```

Compiling Flags

- While compiling your program, you should use "-lpthread" flag (pthread library)
  - gcc -o main main.c -lpthread

Recommended Materials

- Here are some links from which you can get more guidance on pthread programming
  - https://computing.llnl.gov/tutorials/pthreads/
  - http://www.yolinux.com/TUTORIALS/LinuxTutorialPosixThreads.html
- Always take Manual for reference.
  - man pthread_create