CSCI4430 Data Communication and Computer Networks

# Pthread Programming

ZHANG, Mi

Jan. 26, 2017

## Outline

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

## Introduction

- Socket programming

➢ Server accepts connection requests

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

➢ Exchange data

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

# Introduction

- Recall the blocking functions in the last tutorial. If we do not have multi-thread programming:
  - The whole program will be blocked waiting for incoming connection requests and data.
  - We cannot handle both with only One thread.

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

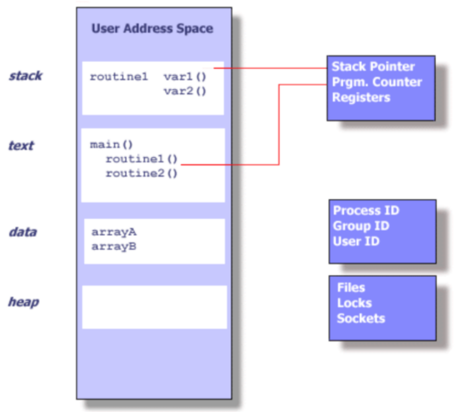
```
while(1){
        int sd = accept(…);
        …
}
```

# What is Multi-thread Programming

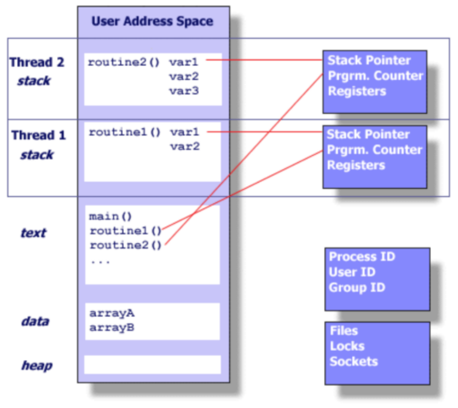
- A **thread** is a sequence of instructions within a program that can be executed independently of other code.
- Thread
  - Exists within one process.
  - Has independent flow of control.
    - Duplicates the essential resources only.
    - May share the process resources.
  - Dies if the parent dies.
  - 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.


User Address Space
stack
routine1 var1 () var2 ()
text
main () routine1 () routine2 ()
data
arrayA arrayB
heap
Stack Pointer Prgm. Counter Registers
Process ID Group ID User ID
Files Locks Sockets

UNIX PROCESS


User Address Space
Thread 2 stack
routine2 () var1 var2 var3
Thread 1 stack
routine1 () var1 var2
text
main () routine1 () routine2 () ...
data
arrayA arrayB
heap
Stack Pointer Prgrm. Counter Registers
Stack Pointer Prgrm. Counter Registers
Process ID User ID Group ID
Files Locks Sockets

THREADS WITHIN A UNIX PROCESS

# 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:

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

Thread 2:

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

## Basic Pthread Programming

- To implement a multi-threads program using pthread library:
  - #include <pthread.h>
  - pthread_t, to define a thread id
  - pthread_create, to create a thread
  - pthread_join, join with a terminated thread
  - pthread_mutex_t, to create a mutex in pthread
  - pthread_mutex_lock, to lock a mutex in pthread
  - pthread_mutex_unlock, unlock a mutex in pthead
- While compiling your program, you should use "-lpthread" flag
  - gcc -o main main.c -lpthread

# 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.

## 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

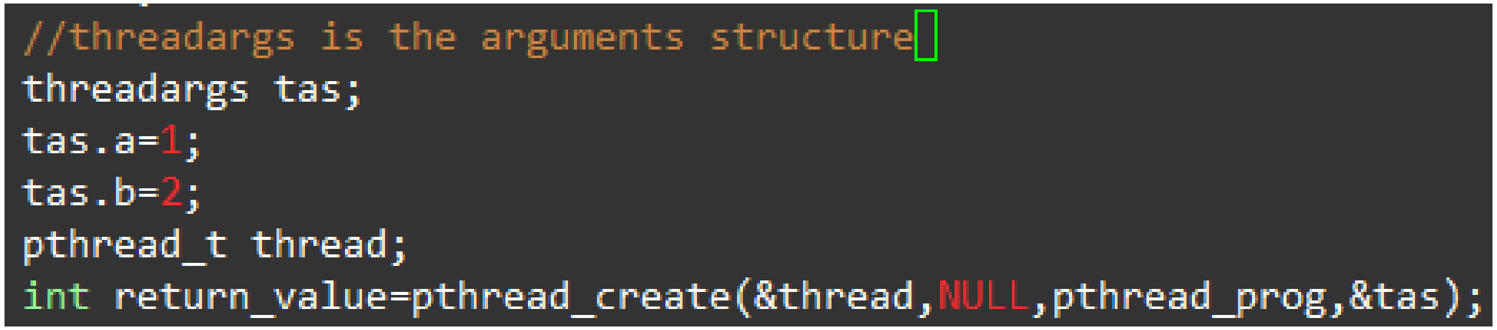
```
pthread_exit(NULL);
```

## Basic Pthread Programming

- Return value of pthread_exit()
  - pthread_exit() will kill the thread and 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()

# 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);
```

# Recommended Materials

- Here are some links from which you can get more guidance on pthread programming
  - POSIX Threads Programming
  - POSIX thread (pthread) libraries
  - Wikipedia
- Always take Manual for reference.
  - man pthread_create