Multithreading

Tran Giang Son, tran-giang.son@usth.edu.vn

ICT Department, USTH

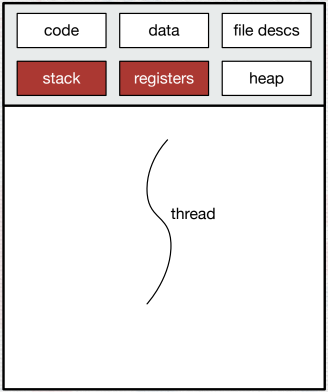
Contents

- What?
- Why?
- Examples
- Which types?
- How?

What?

Thread & Single-threaded process

- Thread
  - a single flow of execution
  - belongs to a process
  - can be considered as lightweight process
- Single-threaded process
  - Default
  - Only one thread per process



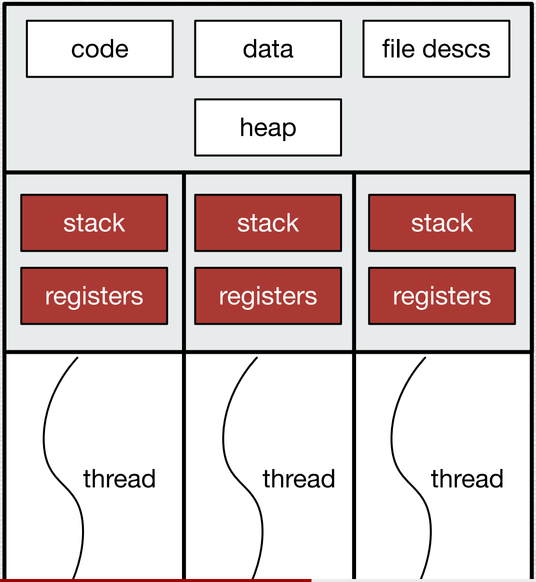
Single-threaded process

- Single stack
- Single text section (code)
- Single data section (global data)
- Single heap (dynamic allocation)

Multi-threaded process

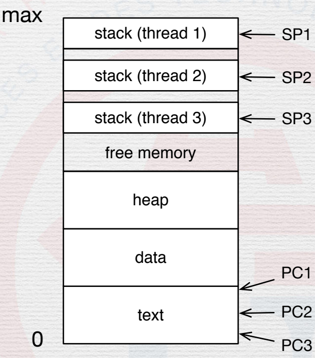
- More than one thread per process
- Share the same PCB among threads
  - Process state
  - Memory allocation (heap, global data)
  - File descriptors (files, sockets, etc.)
  - Scheduling information
  - Accounting information
- **Different** processor state (program counter, registers)
- **Different** stack

Multi-threaded process



Multi-threaded process

- Each thread has:
  - Private stack
  - Private stack pointer
  - Private program counter
  - Private register values
  - Private scheduling policies
- Share:
  - Common text section (code)
  - Common data section (global data)
  - Common heap (dynamic allocation)
  - File descriptors (opened files)
  - Signals...



Process memory space

Multi-threaded process vs Multi process

- Same goals

Multi-threaded process vs Multi process

- Same goals
  - Do several things at the same time

Multi-threaded process vs Multi process

- Same goals
  - Do several things at the same time
  - Increase CPU utilization

Multi-threaded process vs Multi process

- Same goals
  - Do several things at the same time
  - Increase CPU utilization
  - Increase responsiveness

Multi-threaded process vs Multi process

- Same goals
  - Do several things at the same time
  - Increase CPU utilization
  - Increase responsiveness
- What is the principal difference between these two types of process?

Multi-threaded process vs Multi process

- Same goals
  - Do several things at the same time
  - Increase CPU utilization
  - Increase responsiveness
- What is the principal difference between these two types of process?
  - Multi-process with `fork()`: «resource cloning»

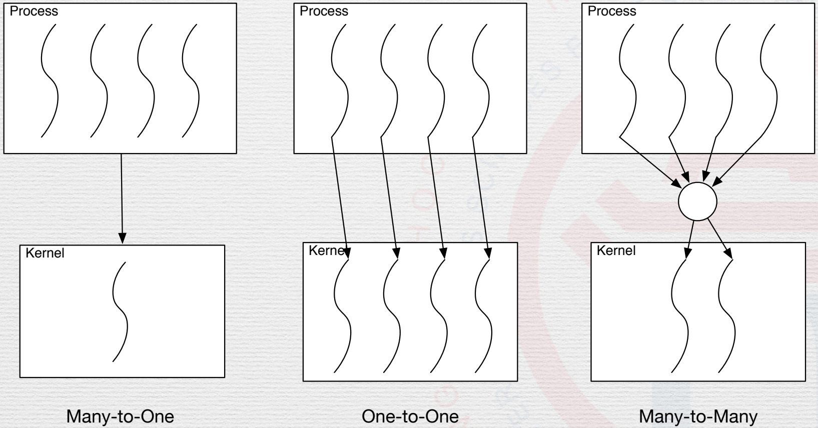
Multi-threaded process vs Multi process

- Same goals
  - Do several things at the same time
  - Increase CPU utilization
  - Increase responsiveness
- What is the principal difference between these two types of process?
  - Multi-process with `fork()`: «resource cloning»
  - Multi-thread process: «resource sharing»

User threads & kernel threads

- User threads
  - POSIX pthread (UNIX/**Linux**/BSD/macOS)
  - **Win32** thread
  - Java thread
- Kernel threads
  - **Windows**
  - **Linux**
  - macOS

Multithreading models



Why?

Why?

- Responsiveness
- Performance
- Resource Sharing
- Scalability

Responsiveness

- Perform different tasks **at the same time**

Responsiveness

- Perform different tasks **at the same time**
  - Several operations can block (e.g. network, disk I/O)

Responsiveness

- Perform different tasks **at the same time**
  - Several operations can block (e.g. network, disk I/O)
  - UI needs responsiveness

→ one thread for UI, other threads for background tasks

Performance

- Creating (`fork()`) a new process is slower than a thread
- Terminating a process is also slower than a thread
- Switching between processes is slower than between threads

Resource Sharing

- Memory is always shared
  - Heap
  - Global data
- All file descriptors are also shared
  - Open files
  - TCP sockets
  - UNIX sockets
  - Devices
- No need to use `shm*()`

Scalability

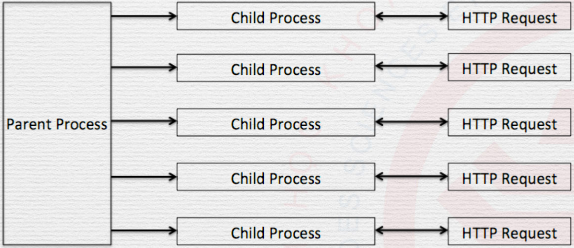
- More CPU cores: simply increase number of threads
- Don't create too many threads
  - Overhead
  - Synchronization

Why **NOT** multi-thread?

- Threads are evil
  - Nondeterministic
  - Synchronization
  - Deadlocks
- Complication

Examples

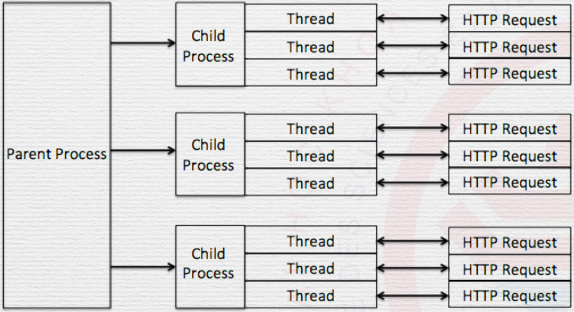
Multi-process real world app



Apache HTTPD Prefork Model[1]

[1] Image courtesy of Toni Miu's blog

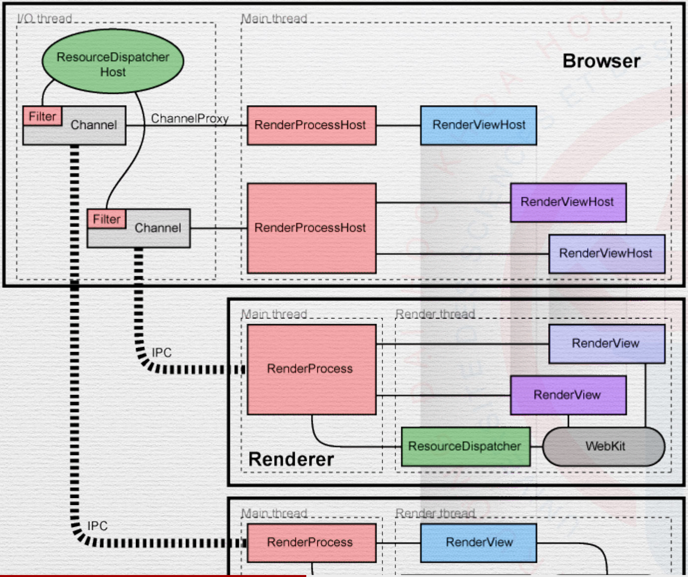
Multi-thread, multi-process, real world app



Apache HTTPD Worker Model[2]

[2] Image courtesy of Toni Miu's blog

Multi-thread, multi-process, real world app



How?

How?

- 2 «How» questions:

How?

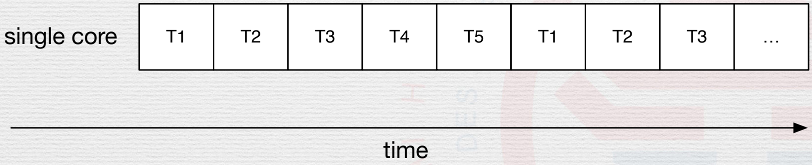
- 2 «How» questions:
  - Q1: How does thread achieve concurrency?

How?

- 2 «How» questions:
  - Q1: How does thread achieve concurrency?
  - Q2: How to use thread?

How (Q1): Concurrency on Single Core

- Q1: How does thread achieve concurrency?



How (Q1): Concurrency on Multi Cores

- Q1: How does thread achieve concurrency?

core 0	T1	T5	T4	T3	T2	T1	T5	T4	...
core 1	T2	T1	T5	T4	T3	T2	T1	T5	...
core 2	T3	T2	T1	T5	T4	T3	T2	T1	...
core 3	T4	T3	T2	T1	T5	T4	T3	T2	...

time

How (Q2): Using thread

- Q2: How to use thread?
- 2 main libraries
  - Win32 thread on Windows
  - POSIX pthread on UNIX/Linux/BSD/macOS

How (Q2a): Using Win32 thread

```
HANDLE WINAPI CreateThread(
  _In_opt_  LPSECURITY_ATTRIBUTES  lpThreadAttributes,
  _In_      SIZE_T                 dwStackSize,
  _In_      LPTHREAD_START_ROUTINE lpStartAddress,
  _In_opt_  LPVOID                 lpParameter,
  _In_      DWORD                  dwCreationFlags,
  _Out_opt_ LPDWORD                lpThreadId
);
```

Source: MSDN

How (Q2a): Using Win32 thread

```
DWORD WINAPI MyThreadFunction(LPVOID lpParam) {
    // do something in the background
}
int _tmain() {
    // create a background thread to execute MyThreadFunction
    DWORD dwThreadId;
    HANDLE threadId = CreateThread(
            NULL,                   // default security attributes
            0,                      // use default stack size
            MyThreadFunction,       // thread function name
            NULL,                   // argument to thread function
            0,                      // use default creation flags
            &dwThreadId);           // returns the thread identifier

    // main thread execution continues here
    // ...

    // [optional] wait for thread to finish
    WaitForSingleObject(threadId, INFINITE);
}
```

How (Q2b): Using POSIX pthread on UNIX

```
#include <pthread.h>

int pthread_create(
    pthread_t *thread,                  // returns the thread identifier
    const pthread_attr_t *attr,         // thread attributes
    void *(*start_routine) (void *),    // thread function
    void *arg);                         // argument to thread function
```

How (Q2b): Using POSIX pthread on UNIX

```c
#include <pthread.h>
void *threadFunction(void *param) {
    // do something in the background
}
int main() {
    // create a background thread to execute threadFunction
    pthread_t tid;
    pthread_create(
        &tid,                          // get thread id
        NULL,                          // skip the attributes
        threadFunction,                // thread function name
        NULL);                         // argument to thread function

    // main thread execution continues here
    // ...

    // [optional] wait for thread to finish
    pthread_join(tid, NULL);
}
```

Practical Work 8: Threading with pthread

- Make a copy your practical work 7
  - Name it « 08.practical.work.shell.pthread.c »
  - Use pthread
    - Create a new thread for each command to `fork()` and `exec()`
    - The main thread is used only for inputing command
- Push your C program to corresponding forked Github repository