Chapter 3: Processes and Threads

Nguyen Ngoc Lam - 20162316

Tuesday 21st April, 2020

Contents

1 Compare Process with Thread

Process	Thread
System calls needed.	No system calls involved.
Context switching required.	No context switching required.
Different process have different copies of code and data.	Sharing same copy of code and data can be possible among different threads.
Operating system treats different process differently.	All user level threads treated as single task for operating system.
If a process got blocked, remaining process continue their work	If a user level thread got blocked, all other threads get blocked since they are single task to OS.
Processes are independent.	Threads exist as subsets of a process. They are dependent.
Process run in separate memory space.	Threads run in shared memory space. memory space. And use memory of process which it belong to.
Processes have their own program counter (PC), register set and stack space.	Threads share Code section, data section, address space with other threads.
Communication between processes requires some time.	Communication between processes requires less time than processes.
Processes dont share the memory with any other process.	Threads share the memory with other threads of the same process
Process have overhead.	Threads have no overhead.

2 Limit the Number of Threads in a Server Process

Yes, for two reasons.

- First, threads require memory for setting up their own private stack. Consequently, having many threads may consume too much memory for the server to work properly.
- Secondly, to an operating system, independent threads tend to operate in a chaotic manner. In a virtual memory system it may be difficult to build a relatively stable working set, resulting in many page faults and thus I/O. Having many threads may thus lead to a performance degradation resulting from page thrashing. **(more important)**.

3 Lightweight Process

3.1 As a Solution of Combining Advantages of Deploying Thread Package in User and Kernel Space

- Responsiveness
- Resource Sharing
- Economy
- Utilization of Multiprocessor Architectures

3.2 A Single Lightweight Process per Process Is Not A Good Idea

Such an association effectively reduces to having only kernel-level threads, implying that much of the performance gain of having threads in the first place, is lost.

4 Advantages of Multi-threaded Server Compared to Single-threaded Server

- All the threads of a process share its resources thus making it more economical.
- Program responsiveness allows a program to run even if part of it is blocked using multithreading.
- Increasesing concurrency of the system since each thread can run on different processor.

5 Advantages and Disadvantages of Each of Three Models of Multi-threaded Server

5.1 Thread-per-request

5.1.1 Advantages

- The main advantage of thread-per-request is that it is straightforward to implement.
- This architecture is particularly useful to handle long-duration requests, such as database queries, from multiple clients.

5.1.2 Disadvantages

- Thread-per-request can consume a large number of OS resourcesif many clients make requests simultaneously.
- It is inefficient for short-duration requests because it incurs excessive thread creation overhead.
- It architectures are not suitable for real-time applications since the overhead of spawn a thread for each request can be non-deterministic.

5.2 Thread-per-connection

5.2.1 Advantages

- Thread-per-connection is straightforward to implement.
- It is well suited to perform long-duration conversations with multiple clients.

5.2.2 Disadvantages

- Thread-per-connection does not support load balancing effectively.
- For clients that make only a single request to each server, thread-per-connection is equivalent to the thread-per-request architecture.

5.3 Thread-per-object

5.3.1 Advantages

- Thread-per-object is useful to minimize the amount of rework required to multi-thread existing single-threaded servants given that all methods in a servant only access servant-specific state there is no need for explicit synchronization operations.

5.3.2 Disadvantages

- Thread-per-object does not support load balancing effectively since it serializes request processing in each servant.
- Therefore, if one servant receives considerably more requests than others it can become a performance bottleneck.
- It is prone to deadlock on nested callbacks.

6 Server Following Finite State Machine

6.1 Example

Node.js is an example of a server following Finite state machine

6.2 Reason for its High Scalability

While single threaded, Node.js is asynchronous so a lot of requests can run on the single thread very quickly.

7 The Need of Multi-threaded for Client in Distributed Systems

In a single-process, single-threaded environment all requests to, e.g., the I/O interface blocks any further progress in the process. Any combination of a multi-process or multi-threaded implementation of the library makes provision for the user of the client application to request several independent documents at the same time without getting blocked by slow I/O operations. **Example:** As a World-Wide Web client is expected to use much of the execution time doing I/O operation such as "connect" and "read", a high degree of optimization can be obtained if multiple threads can run at the same time.

8 The Need of Virtualization Technology

- Reduce Downtime
- Save Money and Time
- Better Recovery When Critical Error Happened

9 Reason Why X-Window system Suitable for Thin Client Architecture

- X is an architecture-independent system for remote graphical user interfaces and input device capabilities.
- Each person using a networked terminal has the ability to interact with the display with any type of user input device.
- X provides the basic framework, or primitives, for building such GUI environments: drawing and moving windows on the display and interacting with a mouse, keyboard or touchscreen

- A Thin Client Architecture is system where the clients (the application that runs on a personal computer or workstation and relies on a server to perform some operations) does very little. They basically just act as terminal for the server.
- This architecture allows diskless client and make data storage more centralized.
- Since X is a windowing system, meaning it manages separated screens, it is suitable for thin client architechture as X-Window system does not require much client-side

10 Compare the Daemon Server and Superserver

10.1 Daemon

In multitasking computer operating systems, a daemon is a computer program that runs as a background process, rather than being under the direct control of an interactive user. It means that daemon will always run as long as the system is still up.
When finding a server, since the daemon is always running, you just need to call that particular daemon. However, many daemon may limit the performance of the hardware system. In addition, when needed, application may have to 'remember' the exact daemon for the job.

10.2 Superserver

Superserver is a kind of daemon that called neccesary servers when they are needed.
When finding a server, client will call directly to superserver. The downside of this is that the response time might be high.