Student’s name: Nguyễn Đức Dũng

Class: ICT.02-K61

**Class Exercises**

**Module: Distributed Systems**

**Chapter 3: Process and Thread**

**Theoretical Exercises:**

**Question 1:** Compare Process with Thread.

|  |  |
| --- | --- |
| Process | Thread |
| A program in execution | A segment of a process |
| Is not lightweight | Is lightweight |
| The process is mostly isolated | Threads share memory |
| Does not share data | Share data with each other |
| Process switching uses interface in operating system | Thread switching does not require to call a operating system and cause an interrupt to the kernel |
| If one server process is blocked no other server process can execute until the first process unblocked | Second thread in the same task could run, while one server thread is blocked |

**Question 2:** Would it make sense to 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. Another, more serious reason, is that, to an operating system, independent threads tend to operate in a chaotic man- ner. 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. Even in those cases where everything fits into memory, we may easily see that memory is accessed following a chaotic pattern rendering caches useless. Again, performance may degrade in comparison to the single-threaded case.

**Question 3:** Why do we consider lightweight process as a solution of combining advantages of deploying thread package in user and kernel space? Having only a single lightweight process per process is also not such a good idea. Why not?

We consider lightweight process as a solution of combining advantages of deploying thread package in user and kernel space because:

An LWP runs in the context of a single (heavy-weight) process, and there can be several LWPs per process. In addition to having LWPs, a system also offers a user-level thread package, offering applications the usual operations for creating and destroying threads. In addition, the package provides facilities for thread synchronization, such as mutexes and condition variables. The important issue is that the thread package is implemented entirely in user space. In other words, all operations on threads are carried out without intervention of the kernel.

Having only a single lightweight process per process is also not such a good idea since in this scheme, we effectively have only user-level threads, meaning that any blocking system call will block the entire process.

**Question 4:** What are the advantages of multi-threaded server compared to single-threaded server?

  The advantages of multi-threaded server compared to single-threaded server are:

* Improved performance and concurrency
* Simplified coding of remote procedure calls and conversations
* Simultaneous access to multiple applications
* Reduced number of required servers

**Question 5:** Give the advantages and disadvantages of each of three models of multithreaded server:Thread-per-request, Thread-per-connection, and Thread-per-object?

|  |  |  |
| --- | --- | --- |
| Model | Advantages | Disadvantages |
| Thread-per-request | The main advantage of thread-per-request is that it is straightforward to implement. This architecture is particularly useful for ORBs that handle long-duration requests, such as database queries, from multiple clients. | The disadvantage with thread-perrequest is that it can consume a large number of OS resources if many clients make requests simultaneously. Moreover, it is inefficient for short-duration requests because it incurs excessive thread creation overhead. In addition, thread-per-request architectures are not suitable for real-time applications since the overhead of spawn a thread for each request can be nondeterministic. |
| Thread-per-connection | Like thread-per-request, thread-perconnection is straightforward to implement. It is well suited for ORBs that perform long-duration conversations with multiple clients. | The primary disadvantage with threadper-connection is that it does not support load balancing effectively. Moreover, for clients that make only a single request to each server, thread-per-connection is equivalent to the threadper-request architecture. |
| Thread-per-object | Thread-per-object is useful for programmers who want to minimize the amount of rework required to multi-thread existing single-threaded servants. So long as all methods in a servant only access servant-specific state there is no need for explicit synchronization operations. | A disadvantage with thread-per-object is that it 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. In addition, thread-per-object designs are prone to deadlock on nested callbacks. |

**Question 6:** Give an example of a server following Finite state machine. Explain why this kind of server is single-thread but high scalability?

Example of a server following finite state machine:

Node.js

The server following Finite state machine is single-thread but high scalability since:

* It also has only 1 thread
* It is non-blocking (asynchronous)
* It records the state of the current request in a table
* It simulates threads and their stacks
* It achieves high performance through parallelism

**Question 7:** Why do the client in Distributed Systems also need to be multi-threaded? Give an example.

The client in Distributed Systems also need to be multi-threaded since:

* Separate UI and processing task
* Increasing the system performance while working with many servers

Example:

The web browser is implemented multi-threaded, so when the website is loading or sending data with its server, we can still use the other features of this website normally

**Question 8:** Why do we need the virtualization technology?

The reasons we need the virtualization technology are:

* It allows legacy software to run on expensive mainframe hardware
* Since the hardware change reasonably fast
* Diversity of platforms and machines can be reduced by letting each app run on its own virtual machine, which run on a common platform
* Virtualization provides a high degree of portability and flexibility

**Question 9:** Why is X-Window system suitable for thin client architecture?

The X Window System, generally referred to simply as X, is used to control bit-mapped terminals, which include a monitor, keyboard, and a pointing device such as a mouse. In a sense, X can be viewed as that part of an operating system that controls the terminal. The heart of the system is formed by what we shall call the X kernel. It contains all the terminal-specific device drivers, and as such, is generally highly hardware dependent.

The X kernel offers a relatively low-level interface for controlling the screen, but also for capturing events from the keyboard and mouse. This interface is made available to applications as a library called Xlib.

**Question 10:** Considering the problem of finding server, compare the deamon server and superserver!

|  |  |
| --- | --- |
| Daemon server | Super-server |
| The daemon keeps track of the current end point of each service implemented by a co-located server.  The daemon itself listens to a well-known end point.  A client will first contact the daemon, request the end point, and then contact the specific server. | It is common to have lots of servers running simultaneously, with most of them passively waiting until a client request comes in.  Instead of having to keep track of so many passive processes, it is often more efficient to have a single superserver listening to each end point associated with a specific service. |