CS550 Homework 1

1. **(10 points)** In this task, you are asked to evaluate the efficiency of reading a file with a single threaded and a multithreaded file server. It takes 25 milliseconds to process a request, dispatch it,and perform the necessary tasks if the required data is stored in the main memory cache. If a disk operation is needed, which occurs 1/3 of the time, an additional 75 milliseconds is required, during which the thread goes to sleep.

a) What is the maximum number of requests per second that the single-threaded server can

handle?

P(with\_disk\_operation)=1/3

P(without\_disk\_operation)=2/3

E(one\_request)=1/3\*(25+75)+2/3\*25=50

number=1000/50=20 requests per second

b) What is the maximum number of requests per second that the multithreaded server can

handle?

The maximum number of requests per second that the multithreaded server can handle is uncertain. In theory, it can approach 50 times the number of threads. However, there are many other factors that could impact the performance, including but not limited to process scheduling overhead, whether disk operations are locked or not, etc.

2. (**10 points**) Imagine a series of processes P1, P2, ..., Pn that implement a multitiered client

server architecture. Process P(i) acts as a client to process P(i+1), and P(i) only sends a reply to P(i-1) after receiving a response from P(i+1). What are the main challenges with this setup when examining the request-reply performance at process P1?

For this type of problem about the performance of this processes, I understand the core issue is its uncontrollability.

For any process like P1, its processing capability is fully coupled with external other processes. The performance test result may fluctuate greatly each time, completely depending on the other process single run performance and whether errors occur in its upstream steps.

3. **(10 points)** Not every node in a peer-to-peer network should become superpeer. What are

reasonable requirements that a superpeer should meet?

In a hierarchical p2p network, the super peer is responsible for receiving information within the network of super peers and delivering it to weak peers. Therefore, in a hierarchical p2p network:

Firstly, not all peers will become super peers.

Secondly, super peers should have better bandwidth and transmission capabilities to handle information dissemination to multiple weak peers.

1. **(10 points)** Enumerate all components of the program state that are shared among threads in a multithreaded process.
   1. The code for the running program
   2. The static data
   3. Space for dynamic data (heap)
   4. A set of OS resources
2. (**10 points)** Would it make sense to limit the number of threads in a server process?

It is meaningful because the accessible resources are limited. The CPU, memory that can be read and written within a unit of time all have upper limits. Therefore, too many threads will only lead to a large amount of waiting, locking issues, and the performance consumed by more frequent context switching operations.

1. (**10 points)** Outline the pros and cons of utilizing multiple processes versus multiple threads.

I understand the core differences between mutiprocesses and mutithreads lie in the issues of independence and data sharing.

For multi-processes:

The advantages are independence and security. Processes are isolated from each other, resources are independent between different processes, and the OS knows the existence and running status of each process. To achieve such advantages, the same disadvantage is that more context switching overhead needs to be paid.

For multi-threads:

The advantages are data sharing. The space for concurrent operations is larger within a process, and resources are shared among multiple threads of the same process. Context switching overhead is relatively small. The user can independently determine locking-related issues and freely maintain communication between threads. To achieve such advantages, the same disadvantage is that complexity is increased and security is sacrificed. Some thread operations may affect other threads.

7. (**10 points**) Outline the pros and cons of preemptive scheduling versus non-preemptive

scheduling.

The advantages and disadvantages of preemptive scheduling and non-preemptive scheduling are obvious:

Preemptive scheduling will continuously and dynamically replace the currently executing task according to the computed weight during the task running. The advantage is that it pursues the extreme in resource utilization and is closer to the concept of global optimal. The corresponding disadvantage is that the computation of various weights itself takes time, and in general, it increases the frequency of context switching and has certain overhead.

In contrast, non-preemptive scheduling will not interrupt the execution of tasks during task running, and will only execute the next task after a task is completed.The advantages are that the losses in weight calculation and context switching are much less. The same disadvantage is that the sorting of some tasks may not be reasonable enough, leading to the average response time of all processes being longer.

8. **(10 points)** Is a multi-threaded approach always advantageous in terms of performance?

Provide an explanation with supporting reasons.

The multi-threaded approach does not necessarily suit all scenarios and does not always bring better performance.

For example, when multithreads concurrently access a file for I/O operations, the number of times this file can be opened within a unit of time is limited. Too many threads can only queue up and also increase the time of context switching.

For a process, reading n lines of a file in a loop, the time needed to read the entire file is A,

For multiple threads, each thread separately opens and reads a line of the same file, the time needed to read the entire file is B

Often A < B

9. **(10 points)** List two distinctions between user-level and kernel-level threads. When is one

preferred over the other?

The operating system is know of the existence of kernel threads, but it does not know about user threads.

Therefore, user threads are fully controlled by the user, and the operating system only manages the process above the user threads. Kernel threads are controlled by the operating system.

Therefore, when a user thread is blocked, the entire process will be blocked. However, for a kernel thread, it actually only blocks itself, and the operating system is know of its existence and will allow other threads to continue executing.

Therefore, multiple user threads are also considered as a single process by the operating system, and they do not have the ability to utilize multiple CPUs for parallel execution.

So if it is a scenario that requires frequent blocking or utilizes a multi-CPU environment, kernel threads are more suitable.

10. **(5 points)** Describe a simple scheme in which there are as many lightweight processes as

there are runnable threads.

To achieve this effect, ultimately it is to have a lightweight process corresponding to each user thread.

Specifically, we can roughly implement multi user-threads using the thread library of the language itself, and then use system calls to create LWP (lightweight process) in each user thread to finally achieve this effect.

1. **(5 points)** Explain the difference between a process and a thread, including which uses more resources.

A process is the basic unit that the operating system runs. A process uses independent resources including CPU, memory address space, etc. While a thread shares resources within a process.

Because a process owns its own address space and resources, the overhead of simultaneously switching processes is greater than switching threads.

It is also because of the differences in whether resources are shared or not, mutiprocess communication is more complex, while mutithread communication only needs to handle shared memory properly.