Threads and Multithread Model

Hui Chen [a]

[a]CUNY Brooklyn College

October 9, 2024

Outline

Outline

Process

Recall our discussion that multiple processes run concurrently ...

- P_1 on CPU, context switch, P_2 on CPU, context switch, P_3 on CPU
- The OS save the context of the current process, and load the context of the next process ...
- The OS maintains Process Control Blocks (PCB) for the processes where a process consists of,
  - an execution context, and
  - an address space (program text, data, stack, and heap).

What are the implications from the user's perspective?

- Observing example programs using `fork` and `clone`.

Process and Threads

How about we let a process have

- *multiple execution contexts*, and
- an address space (program text, data, stack, and heap)?

What are the implications from the user's perspective?

- Observing example programs using `clone`.

Benefits of Threads

- Can ease resource sharing (a single address space)

  Example. Consider writing an application that has multiple *cooperating* processes vs. multiple *cooperating* threads

- Can be made more economic (less overhead)

  Example. Consider the *cost of creating* multiple processes vs. multiple threads

- Can be more scalable (to multicore architecture)

  Example. Consider creating a parallel program.

- Can improve responsiveness

  Example. Consider a GUI program.

Outline

Multithread vs Multiprocess Architecture

A programming model (thread-per-request or process-per-request).

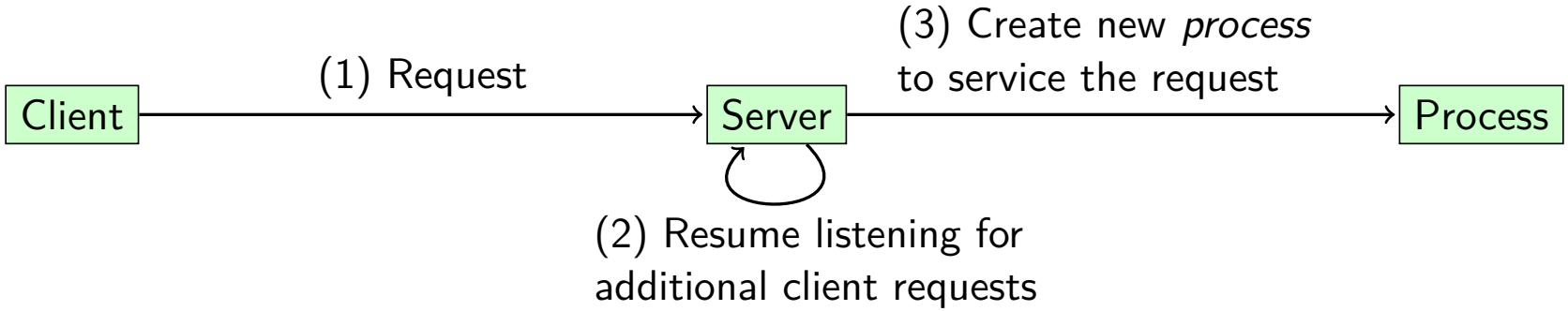

Figure: Multiprocess architecture server

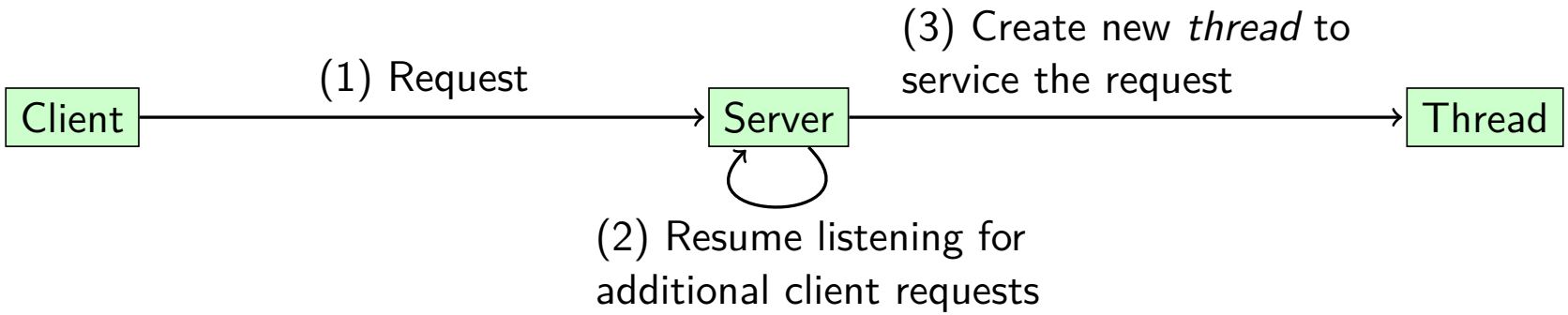

Figure: Multithread architecture server

Discussion Question

1. What benefits can we obtain from multithread architecture, but not from multiprocess architecture?
2. What benefits can we obtain from multiprocess architecture, but not from multithread architecture?

Let's analyze a few examples ...

Outline

Concurrency and Parallelism

Discuss,

- What concurrency is?
- What parallelism is?

Data and Task Parallelism

- Data parallelism. Distributes subsets of the same data across multiple cores, same operation on each
- Task parallelism. Distributing threads across cores, each thread performing unique operation

Amdahl's Law

$$\text{speedup} = \frac{1}{S + \frac{1-S}{N}} \tag{1}$$

where S is serial portion and N processing cores

- It identifies performance gains from adding additional cores to an application that has both serial and parallel components

Let's examine a few examples ...

- Estimate the speed-up factor
- Estimate the serial portion or the parallel portion

Multicore Programming

Multicore or multiprocessor systems putting pressure on programmers, challenges include:

- Dividing activities
- Balance
- Data splitting
- Data dependency
- Testing and debugging

Outline

Kernel and User Threads

- Kernel threads
  - Management done by the kernel
  - TCBs in the kernel
  - User threads can be blocked by the process, less concurrency, in particular, on multiprocessor/multicore systems
- User threads.
  - Management done by user-level threads library.
  - Thread Control Blocks (TCBs) in user process
  - Kernel threads are more expensive to create, and can support multiple processors
    But how much more?

Multithreading Models

- One-to-One (1 user thread / 1 kernel thread)
  - Most OSes, such as Linux and Windows
- Many-to-One (N user threads / 1 kernel thread)
  - User level threads library, such as Green Threads in early Java.
- Many-to-Many (M user threads / N kernel threads)
  - Early versions of Sun Solaris Operating System
  - Java Virtual Threads (JDK 21)
  - Goroutines

Outline

Thread Libraries and APIs

- Explicit vs. implicit threads
- Kernel vs. user threads
- Thread library and API examples
  - (UNIX) POSIX threads (Pthread)
  - Windows threads
  - Java threads
- Implicit threads examples
  - Android thread pools
  - Windows thread pools
  - Java thread pools (`Executors`)
  - Fork-Join
    - Java fork-join API (`ForkJoinPool`)
  - OpenMP
  - Grand Central Dispatch (OS X)
  - Intel thread building blocks