LTAT.06.007 Distributed Systems

Lecture 3 – Processes I (Fundamentals)

Huber Flores, PhD
ASSOCIATE PROFESSOR

Tartu, Estonia 22/02/2021
Recap

• Architecture
  ▪ Roles and behaviors

• Software architecture

• System architecture
  ▪ Centralized
  ▪ Decentralized
Agenda

• Processes
  - Client
  - Server

• Modelling
  - Server performance
  - Client/Server applications
Fundamentals
# Process

## Definition

A program in execution in a virtual processor created by the operating system.

## Characteristics

- Processes are executed on top of the OS
- Concurrency transparency of multiple processes enforced by OS (PCB)
  - Independent state information
  - Independent address spaces
  - Interact only via IPC (inter-process communication) mechanism
  - Expensive context switch (CPU, memory, address caches)
- A process typically comprises of multiple (parallel) threads providing finer granularity of control
### PCB

**Definition**

PCB -> Process Control Block
Threads

Basic idea

We build *virtual processors* in software, on top of *physical processors*:

**Processor:** provides a set of instructions along with the capability of automatically executing a series of those instructions.

**Thread:** a minimal software processor in whose *context* a series of instructions can be executed. Saving a thread context implies stopping the current execution and saving all the data needed to continue the execution at a later stage.

**Process:** a software processor in whose *context* one or more threads may be executed. Executing a thread, means executing a series of instructions in the *context* of that thread.
Client (runtime)

Lightweight

- User applications
- OS
- Underlying hardware

PCB
- Virtual processor
- Physical Processor
Server (runtime)

Heavy

- Daemon (admin GUI)
- OS
- Underlying hardware
Server (runtime)

- Underlying hardware
- OS
- Daemon (admin GUI)

→

- Process threads
- Process threads
- OS
- Virtualization
- OS
## Context switching

### Contexts

**Processor context:** The minimal collection of values stored in the registers of a processor used for the execution of a series of instructions (e.g., stack pointer, addressing registers, program counter).

**Thread context:** The minimal collection of values stored in registers and memory, used for the execution of a series of instructions (i.e., processor context, state).

**Process context:** The minimal collection of values stored in registers and memory, used for the execution of a thread (i.e., thread context, but now also at least MMU register values).
Server (runtime)
## Client vs Server

### Differences
- Processing resources (CPU)
- Memory

### Deployment
- Client -> Single machine
- Server -> Multiple machines

### Computing workload
- Client -> Lightweight processes -> Threads
- Server -> Processes -> Threads
Using threads at the client side

Multithreaded web client

Hiding network latencies:
- Web browser scans an incoming HTML page, and finds that more files need to be fetched.
- Each file is fetched by a separate thread, each doing a (blocking) HTTP request.
- As files come in, the browser displays them.

Multiple request-response calls to other machines (RPC)

- A client does several calls at the same time, each one by a different thread.
- It then waits until all results have been returned.
- Note: if calls are to different servers, we may have a linear speed-up.
Using threads at the server side

<table>
<thead>
<tr>
<th>Multithreaded web client</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Starting a thread is cheaper than starting a new process.</td>
</tr>
<tr>
<td>• Having a single-threaded server prohibits simple scale-up to a multiprocessor system.</td>
</tr>
<tr>
<td>• As with clients: hide network latency by reacting to next request while previous one is being replied.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Multiple request-response calls to other machines (RPC)</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Most servers have high I/O demands. Using simple, well-understood blocking calls simplifies the overall structure.</td>
</tr>
<tr>
<td>• Multithreaded programs tend to be smaller and easier to understand due to simplified flow of control.</td>
</tr>
</tbody>
</table>
Why multithreading is popular: organization

Dispatcher/worker model

Overview

<table>
<thead>
<tr>
<th>Model</th>
<th>Characteristics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multithreading</td>
<td>Parallelism, blocking system calls</td>
</tr>
<tr>
<td>Single-threaded process</td>
<td>No parallelism, blocking system calls</td>
</tr>
<tr>
<td>Finite-state machine</td>
<td>Parallelism, nonblocking system calls</td>
</tr>
</tbody>
</table>
## Basic model

A process implementing a specific service on behalf of a collection of clients. It waits for an incoming request from a client and subsequently ensures that the request is taken care of (transaction), after which it waits for the next incoming request.
Contacting a server

Observation: most services are tied to a specific port

<table>
<thead>
<tr>
<th>Service</th>
<th>Port</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ftp-data</td>
<td>20</td>
<td>File Transfer [Default Data]</td>
</tr>
<tr>
<td>ftp</td>
<td>21</td>
<td>File Transfer [Control]</td>
</tr>
<tr>
<td>telnet</td>
<td>23</td>
<td>Telnet</td>
</tr>
<tr>
<td>smtp</td>
<td>25</td>
<td>Simple Mail Transfer</td>
</tr>
<tr>
<td>www</td>
<td>80</td>
<td>Web (HTTP)</td>
</tr>
</tbody>
</table>

Dynamically assigning an end point
## Out-of-band communication

<table>
<thead>
<tr>
<th>Issue</th>
</tr>
</thead>
<tbody>
<tr>
<td>Is it possible to interrupt a server once it has accepted (or is in the process of accepting) a service request?</td>
</tr>
</tbody>
</table>
# Out-of-band communication

## Issue

Is it possible to interrupt a server once it has accepted (or is in the process of accepting) a service request?

## Solution 1: Use a separate port for urgent data

- Server has a separate thread/process for urgent messages
- Urgent message comes in → associated request is put on hold
- Note: we require OS supports priority-based scheduling
Servers and state

Stateless servers

Never keep accurate information about the status of a client after having handled a request:

- Don’t record whether a file has been opened (simply close it again after access)
- Don’t promise to invalidate a client’s cache
- Don’t keep track of your clients
Servers and state

Stateless servers

Never keep accurate information about the status of a client after having handled a request:
• Don’t record whether a file has been opened (simply close it again after access)
• Don’t promise to invalidate a client’s cache
• Don’t keep track of your clients

Consequences

• Clients and servers are completely independent
• State inconsistencies due to client or server crashes are reduced
• Possible loss of performance because, e.g., a server cannot anticipate client behavior (think of prefetching file blocks)
Servers and state

Stateless servers

Never keep accurate information about the status of a client after having handled a request:
• Don’t record whether a file has been opened (simply close it again after access)
• Don’t promise to invalidate a client’s cache
• Don’t keep track of your clients

Consequences
• Clients and servers are completely independent
• State inconsistencies due to client or server crashes are reduced
• Possible loss of performance because, e.g., a server cannot anticipate client behavior (think of prefetching file blocks)

Question
Does connection-oriented communication fit into a stateless design?
Servers and state

Stateful servers

Keeps track of the status of its clients:
- Record that a file has been opened, so that prefetching can be done
- Knows which data a client has cached, and allows clients to keep local copies of shared data
Servers and state

Stateful servers

Keeps track of the status of its clients:
- Record that a file has been opened, so that prefetching can be done
- Knows which data a client has cached, and allows clients to keep local copies of shared data

Observation

The performance of stateful servers can be extremely high, provided clients are allowed to keep local copies. As it turns out, reliability is often not a major problem.
Three different tiers

Server cluster organization

Question

The first tier is generally responsible for passing requests to an appropriate server: request dispatching
Observation

Having the first tier handle all communication from/to the cluster may lead to a bottleneck.

A solution: TCP handoff
Modelling performance
Server workload

Observation

• A server does not handle a single request, but many concurrently.

• Workload depicts a collection of requests/transactions/processes that the server executes
## Server performance

### Essence

The ability to operate (execute processes/transactions/jobs/etc) reliably and dependably to meet interaction and behavior expectations.

### Quality of Service (QoS)

System attributes to fulfill QoS

- Response time
- Throughput
- Availability
- Reliability
- Security
- Scalability
- Extensibility
Server performance

QoS attributes

- **Response time**: the time it takes a system to react to a human request
- **Throughput**: the rate at which requests are completed from a computer system and is measured in operations per unit time.
- **Availability**: the fraction of time that a system is up and available to its customers
- **Reliability**: the probability that it functions properly and continuously over a fixed period of time. Reliability and availability are closely related concepts but are different. When the time period during which the reliability is computed becomes very large, the reliability tends to the availability.
- **Security**: A combination of Confidentiality, Data integrity and Non-repudiation
- **Scalability**: A system is said to be *scalable* if its performance does not degrade significantly as the number of users, or equivalently, the load on the system increases
- **Extensibility**: is the property of a system to easily evolve to cope with new functional and performance requirements.
Assume that an I/O operation at a disk in an OLTP system takes 10 msec on average. If the disk is constantly busy (i.e., its utilization is 100%), then it will be executing I/O operations continuously at a rate of one I/O operation every 10 msec or 0.01 sec. So, what is the maximum throughput of the disk?
Example: Throughput

Assume that an I/O operation at a disk in an OLTP system takes 10 msec on average. If the disk is constantly busy (i.e., its utilization is 100%), then it will be executing I/O operations continuously at a rate of one I/O operation every 10 msec or 0.01 sec. So, what is the maximum throughput of the disk?

100 (= 1 / .01) I/Os per second.

But if the rate at which I/O requests are submitted to the disk is less than 100 requests/sec, then its throughput will be equal to the rate at which requests are submitted. This leads to the expression

\[
\text{throughput} = \text{minimum } [\text{servercapacity, offeredworkload}]
\]
Example: Availability

If the availability of a system is 99.99% over a period of thirty days, how long the system was unavailable?
Example: Availability

If the availability of a system is 99.99% over a period of thirty days, how long the system was unavailable?

\[
(1 - 0.9999) \times 30 \text{ days} \times 24 \text{ hours/day} \times 60 \text{ min/hr} = 4.32 \text{ minutes}
\]
Modelling server performance

Models

- Approximation
- Simulation
- Analytic
- And others (do not rely on intuitive ones)

Note

- A model should not be made more complex than is necessary to achieve its goals.
Modelling server performance

Analytic

A distributed computer system is composed of a collection of resources, where each resource usage is regulated by a queue.

- Network of queues, or Queuing Network (QN)
- Request/transaction/process → Customer

Essence

- Analytic models are composed of a set of formular and/or computational algorithms that provide the values of desired performance measures as a function of a set of input workload parameters.
Modelling server performance

(a) Single queue with one resource server (b) Single queue with m resource servers.

Service demand
The total average service time of a transaction (single class) is called its service demand.
Modelling server performance

Example: Queuing network for a simple database server
## Modelling multiple request types

### Essence

The workload consists of different types of transactions (Multiclass QN model)

### Characteristics

**Heterogeneous service demands**: the requests that form the workload can be clustered into groups

**Different types of workload**: the requests in the workload are different in nature

**Different service level objectives**: the requirements of each group of requests is different
Modelling multiple request types

Example: Summary statistics for the database server

<table>
<thead>
<tr>
<th>Transaction Group</th>
<th>Percentage of Total</th>
<th>Average CPU Time (sec)</th>
<th>Avg. Number of I/Os</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trivial</td>
<td>45%</td>
<td>0.04</td>
<td>5.5</td>
</tr>
<tr>
<td>Medium</td>
<td>25%</td>
<td>0.18</td>
<td>28.9</td>
</tr>
<tr>
<td>Complex</td>
<td>30%</td>
<td>1.20</td>
<td>85.0</td>
</tr>
</tbody>
</table>

Overall arrival rate = 1.5 tps
Arrival rate per type of request? (class)
Modelling multiple request types

Example: Summary statistics for the database server

<table>
<thead>
<tr>
<th>Transaction Group</th>
<th>Percentage of Total</th>
<th>Average CPU Time (sec)</th>
<th>Avg. Number of I/Os</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trivial</td>
<td>45%</td>
<td>0.04</td>
<td>5.5</td>
</tr>
<tr>
<td>Medium</td>
<td>25%</td>
<td>0.18</td>
<td>28.9</td>
</tr>
<tr>
<td>Complex</td>
<td>30%</td>
<td>1.20</td>
<td>85.0</td>
</tr>
</tbody>
</table>

Overall arrival rate = 1.5 tps
Arrival rate per type of request? (class)

0.675 (=1.5 x 0.45) tps, 0.375 (=1.5 x 0.25) tps, 0.45 (=1.5 x 0.30) tps
Modelling other request types

Essence

Open class
• Workload intensity is specified by an arrival rate
• Unbounded number of customers in the system
• Throughput is an input parameter

Closed class
• Workload intensity specified by the customer population (not transactions, but batch jobs)
• Bounded and known number of customers in the system
• Throughput is an output parameter
Modelling other request types

Example: Queuing network for a database server with closed workload
Mixed classes

SLA (Service Level Agreement)

Performance goals of servers based on different metrics, including response time, throughput and availability, among others.

Example: SLAs for the Database Server

- 99.99% availability during 8:00 am – 11:00 pm, and 99.9% at other times
- Minimum throughput of 2,000 page downloads per second

<table>
<thead>
<tr>
<th>Transaction Group</th>
<th>Maximum Average Response Time (sec)</th>
<th>Minimum Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trivial</td>
<td>1.2</td>
<td>-</td>
</tr>
<tr>
<td>Medium</td>
<td>2.5</td>
<td>-</td>
</tr>
<tr>
<td>Complex</td>
<td>8.0</td>
<td>-</td>
</tr>
<tr>
<td>Batch Reports</td>
<td>-</td>
<td>20 per hour</td>
</tr>
</tbody>
</table>
Mixed classes

Mixed queuing network for a database server
# Modelling different types of resources

## Types of resources

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load independent (LI)</td>
<td>These resources have a constant service rate that does not depend on the load (e.g., CPU, disk)</td>
</tr>
<tr>
<td>Load dependent (LD)</td>
<td>The service rate is a function of the number of requests in the queue (e.g., LAN)</td>
</tr>
<tr>
<td>Delay (D)</td>
<td>There is no waiting line. A request that arrives at a delay resource is served immediately (e.g., client)</td>
</tr>
</tbody>
</table>
Modelling different types of resources

QN for client/server applications (database server with clients and LAN)

No queue (think/wait)

Arrow across
Modelling different types of resources

Example: Summary Statistics for the Database Server

<table>
<thead>
<tr>
<th>Transaction Group</th>
<th>Percentage of Total</th>
<th>Average CPU Time (sec)</th>
<th>Avg. Number of I/Os</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trivial</td>
<td>45%</td>
<td>0.04</td>
<td>5.5</td>
</tr>
<tr>
<td>Medium</td>
<td>25%</td>
<td>0.18</td>
<td>28.9</td>
</tr>
<tr>
<td>Complex</td>
<td>30%</td>
<td>1.20</td>
<td>85.0</td>
</tr>
</tbody>
</table>

Overall arrival rate = 1.5 tps
Arrival rate per class = 0.675 (=1.5 x 0.45) tps, 0.375 (=1.5 x 0.25) tps, 0.45 (=1.5 x 0.30) tps

Average I/O time = 0.01 seconds
Service demand at disk per each class?
Modelling different types of resources

Example: Summary Statistics for the Database Server

**Service demand at disk per each class?**

*Service demand = average number of I/Os x average time per I/O*

<table>
<thead>
<tr>
<th>Class (r)</th>
<th>Type</th>
<th>$\lambda_r$ (tps)</th>
<th>$D_{disk,r}$ (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (Trivial)</td>
<td>open</td>
<td>0.675</td>
<td>0.055</td>
</tr>
<tr>
<td>2 (Medium)</td>
<td>open</td>
<td>0.375</td>
<td>0.289</td>
</tr>
<tr>
<td>3 (Complex)</td>
<td>open</td>
<td>0.450</td>
<td>0.850</td>
</tr>
</tbody>
</table>
Summary

• Fundamentals
  • Decomposed behavior and execution of client and servers
  • Client and servers experience workload (different type of workload)

• Modelling
  • QoS attributes
  • Performance metrics
  • Model workload (Classes)
Next lecture

Processes II (From Systems to Descriptive Models)
Questions?

E-mail: huber.flores@ut.ee