Distributed Systems

Inter-Process Communication

Prof. Dr. Oliver Hahm

Frankfurt University of Applied Sciences
Faculty 2: Computer Science and Engineering
oliver.hahm@fb2.fra-uas.de
https://teaching.dahahm.de

04.05.2023

Agenda

- Processes
- Communication
- Parameter Handling

Agenda

- Processes
- Communication
- Parameter Handling

Programs and Processes

- A Program is an executable piece of software including a set of instructions
- A Process is a program currently executed by an operating system

Program Classification

- Available in a hardware-specific binary format (and thus including the machine instructions) to be directly executable by the Operating System.
  Example: Windows `*.exe` and `*.com` files; UNIX `ELF` and `a.out` files
- Require an additional Interpreter, usually executing the statements sequentially.
  Example: Unix shell scripts, PERL, JAVA scripts
- Available in machine-independent binary format (Byte-code) to be executed within a certain environment: Virtual Machine.
  Example: JAVA `*.jar` files; Python script files

Processes, Threads and LWPs

- Processes:
  - A process possess a environment which is inherited from its parent
  - The OS manages processes
  - Each process contains a Process Control Block PCB) which maintains its attributes
- Threads:
  - Individual tasks within a process may be individual assigned to threads
  - A process can schedule several (concurrent) threads: multithreading
  - Unix Operating Systems supporting POSIX Pthreads

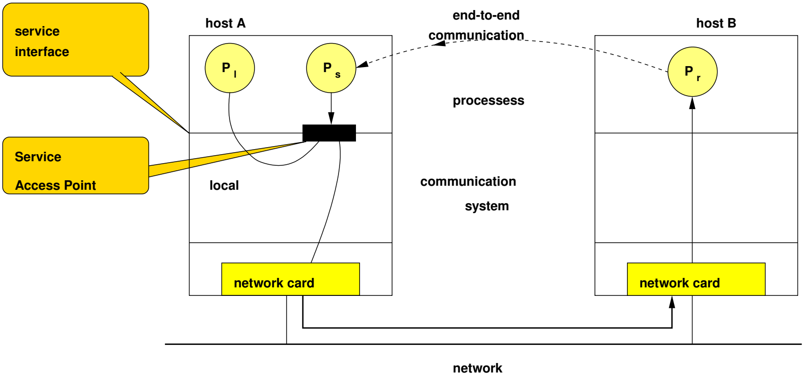
Inter-Process Communication (IPC)

- In order to cooperatively work on a common task processes need to exchange information
- A process shares common resources (e.g., memory) ⇒ threads may access these resources concurrently
- Processes on the same computer also share common resources (e.g., the file system), but in most cases they require support from the OS to exchange information
- Processes in a distributed system have to rely on message passing

What type of information is exchanged?

- Occurrence of events
- Program flow information
- Program data

Generic Model for IPC



- Which properties of a distributed system are affected by the Link Layer?
- What is the impact of packet switching (e.g., compared to circuit switching) on the Network Layer on a distributed system?
- What are the criteria to select the Transport Layer protocol when designing a distributed application?

Agenda

- Processes
- Communication
- Parameter Handling

Which types of IPC do you know?

Types of Inter-Process Communication (IPC)

- Files
  An resource stored in the file system which can be accessed by multiple processes
- Signals and Flags
  Notify another process about the occurrence of an event
- Pipes
  An unidirectional channel between two processes (can be named or anonymous)
- Shared Memory
  A memory block that can be accessed by multiple processes
- Message Queues
  Processes use a queue for message exchange
- (IP and Unix domain) Sockets
  An inode or network based communication end point

Files

- Linux
  - File descriptors represent file handles
  - Part of the POSIX API
  - Per default every process owns three file descriptors (`stdin`, `stdout`, and `stderr`)
  - File descriptors can be used for, e.g., reading, writing, seeking, or truncating a file
- RIOT
  - Virtual File System may be implemented by various backends
  - Not all IoT devices provide persistent memory
  - If available, persistent memory is often realized on flash memory → wear leveling is required

Signals and Flags

- Linux
  - POSIX signals
  - Standardized messages to trigger a certain behaviour
  - The receiver process gets interrupted
  - If a signal is unhandled by the receiver, it will terminate
- RIOT
  - Thread flags
  - Optional kernel feature
  - Notify threads of conditions in a race-free and allocation-less way

Pipes

- Linux
  - A simplex FIFO, i.e., a unidirectional data channel
  - One process accesses the write end, the other the read end of the pipe
  - It can be anonymous or named via an inode in the file system
- RIOT
  - No equivalent available

Shared Memory

- Linux
  - POSIX shared memory objects
  - A shared memory object can be mapped into the process' memory space
  - Shared memory objects are accessed in a similar manner as files
- RIOT
  - Since most MCUs do not provide a MMU, all processes can typically access all memory regions . . .

Message Queues

- Linux
  - POSIX and System V message queues
  - Queues are named and can be shared via this name between processes
  - Message have priorities
- RIOT
  - Kernel messages and mailboxes
  - Optional feature
  - Block and non-block API available
  - A thread may create a message buffer
  - Mailboxes can be accessed by multiple processes

Sockets

- Linux
  - POSIX (or BSD) Sockets
  - Common API for Internet and Unix Domain sockets
  - A socket represents the endpoint of a communication endpoint
- RIOT
  - POSIX Sockets on top of the sock interface
  - sock is currently implemented for . . .
    - TCP
    - UDP
    - Raw IP
    - DTLS
    - DNS
  - More lightweight and custom-tailored ⇒ less generic

Types of Inter-Process Communication (IPC)

Which type of IPC can be used for what?

- Files
- Signals and Flags
- Pipes
- Shared Memory
- Message Queues
- (IP and Unix domain) Sockets

Agenda

- Processes
- Communication
- Parameter Handling

Data Types

- How many bits does an `int` have?

Data Types

- How many bits does an `int` have?
- How is a `string` stored in the programming language C? How in Java?

Data Types

- How many bits does an `int` have?
- How is a `string` stored in the programming language C? How in Java?
- How many digits can be stored after the decimal point in a `float`?

Parameter Handling

- Heterogeneity Problem
  - Different *encodings* (e.g., ASCII, UTF-8)
  - Endianness → little endian vs. big endian
  - Differing number formats
- Possible solutions
  - Mapping between local data representations
    - Sender uses its local representation, the receiver transforms it

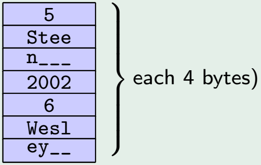
    ⇒ Requires $n \cdot n$ mappings
  - *Canonical network representation* for all types
    - Requires $2n$ mappings (for n local representations)
    - Potentially unnecessary encoding

Common Network Representations: XDR

- *External Data Representation*
  - Defined by Sun as part of SunRPC
  - Mostly Motorola 68000 data formats: ASCII; big endian, two complements; IEEE floating points, . . .
  - Compound data types: arrays, structures, unions
  - No explicit data typing, i.e., no self-describing data

Example

```
struct {
    string author<>;
    int year;
    string publisher<>;
}
```



5
Stee
n___
2002
6
Wesl
ey__

each 4 bytes)

Common Network Representations: ASN.1 BER

- *ISO Abstract Syntax Notation Number 1,*
  Basic Encoding Rules, ISO 8824, 8825, ITU X.409
  - Explicit data types, i.e., the type information precedes all data fields
  - Commonly used: CANopen, LDAP, UMTS/LTE, VoIP, Encryption
  - Standard representation: (type, length, value)
  - Drawback: runtime costly (bit access)

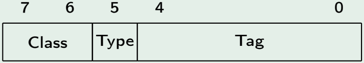
Example

```
count ::=INTEGER
```

0 2	} Type (Identifier)
0 1	} Length
1 A	} Value(26_{10})

each 1 byte (hex)

Type Identifier:



Tag:	1	Boolean
	2	Integer, ...
	16	Sequence
Type:	0	Primitive
	1	Constructed
Class:	00	Universal
	01	Application...

Common Network Representation: CDR

- *Common Data Representation*
  - Defintion in OMG CORBA 2.0
  - Use for CORBA IIOP protocol
  - Sender uses its own format, "‘Receiver makes it right"’
  - Simple types (`short, long, float, char, ...`)
  - Complex types (`sequence, string, union, struct, ...`)
  - Alignment/Padding according to the multiple of the element length
  - Big endian

Example

```
struct <string, unsigned long>
```

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
5				'S'	'T'	'E'	'E'	'N'				2002			
05	00	00	00	53	54	45	45	4E	00	00	00	00	00	07	D2
← Länge →									← Padding →						

Common Network Representations: JSON

- *JavaScript Object Notation* Data Interchange Format
  - Lean, text based exchange format
  - Independent of programming languages
  - RFC 7159, derived from ECMAScript
  - Easy to parse, many parsers available
  - Simple types (string, number, boolean, null)
  - Complex types (object, array)
    - An object is an unordered list of name/value pairs
    - A name is a string and the values may be a simple type, an object, or an array
    - An array is an ordered sequence of values

Example

```
{
"AUTHOR" : "Steen",
"YEAR" : 2002,
"PUBLISHER" : "Wesley"
}
```

Problems

- Complex, compound parameter types
  - e.g., structs, arrays, require rules for serialization
- Addresses in parameters
  - No meaning at the destination's address space
  - Most simple solution: prohibit addresses, only allow call-by-value (e.g., SunRPC)
  - Use of a common, global address space if possible
  - Replace pointers by markers and reconstruct compound data structures at receiver side by pointers (e.g., DCE RPC)

Important takeaway messages of this chapter

- IPC is required to exchange information between processes (or threads)
- Various common concepts exist implemented differently for different operating systems
- If data is exchanged between hosts in the network a common interpretation of the data is required