Definition of the term

Socket Primitive

In Distributed Systems

Florian Willich
Hochschule für Technik und Wirtschaft Berlin
University of Applied Sciences Berlin
Course: Distributed Systems
Lecturer: Prof. Dr. Christin Schmidt

July 12, 2015

Abstract

Abstract...

Contents

1 Introduction

During my work on a ?technical report? I came across the term *Socket Primitive* which was a heavily used term when describing operations with the Transmission Control Protocol / Internet Protocol (TCP/IP) (Tanenbaum & Steen, 2007, p. 141, ch. 4.3.1):

> [...] A socket forms an abstraction over the actual communication end point that is used by the local operating system for a specific transport protocol. In the following text, we concentrate on the socket primitives for TCP, which are shown in Fig. 4-14. [...]

Inevitably the question came up what exactly the term *Socket Primitive* defines? With this technical report, I will first clarify what a *Socket* is and what *Primitive* means in such context. Furthermore, I will describe the term *Socket Primitive* with examples and more detail and define the term.

I also would like to give special thanks to Maarten van Steen who wrote me some more clarification details in the beginning of my research.

Please keep in mind that although I am writing in a more abstract and general way about *Socket Primitives*, it is always considered to operate with a UNIX Operating System (OS).

1.1 Socket

A socket is a communication end point of a computer system. Whenever two computer systems shall communicate with each other, a connection between those systems has to be established by using interconnected sockets (Tanenbaum & Wetherall, 2011, p. 553, ch. 6.5.2).

A socket is always considered to be used associated with a protocol that implements a model of communication. Inter-Process Communication (IPC) will only be successful when both processes use the same protocol. Common protocols are the User Datagram Protocol (UDP) or TCP/IP for example. The used protocol determines what low-level mechanism is used to transmit and receive data (Loosemore *et al.*, 2015, p. 427 - 427, ch. 16).

1.2 Primitive

1.2.1 Meaning in Computer Science

In computer science, the word *primitive* is often used to name an instruction which represents a self-contained unit on the current abstraction level. This also means that there is probably no need for further description of what this

instruction is composed of rather to describe the semantic, logic or functionality the *primitive* introduces.

For example: The GNU C Reference Manual uses the term *Primitive Data Types* to describe the built-in data types in the programming language C (Rothwell & Youngman, 2015, p. 8, ch. 2). The following table shows a selection of primitive data types of a C language, their characteristics and the memory allocation on 64-bit systems. Please see Appendix A for the implementation:

Primitive	Stores	Memory Allocation
char	ASCII characters	1 byte
int	integers	4 byte
float	floating point real numbers	4 byte
double	double precision floating point real numbers	8 byte

As long as a programmer knows what each built-in *Primitive Data Type* represents and how to use it, he must not necessarily know what assembly code the compiler is generating to write code in a high level language. On the contrary, a compiler builder has to.

1.2.2 Meaning in Linguistic Science

The *Oxford Advanced Learners Dictionary* has no definition of the word *primitive* that fits for the use in computer science (A S Hornby, 2005, p. 1197). Nevertheless, the online version `http://www.oxfordlearnersdictionaries.com` offers an additional word origin (Original URL: `http://www.oxfordlearnersdictionaries.com/definition/english/primitive_1?q=primitive`):

> late Middle English (in the sense 'original, not derivative'): from Old French primitif, -ive, from Latin primitivus 'first of its kind', from primus 'first'.

In computer science, *Primitive* is often used when describing a low-level operation.

2 TCP/IP Socket Primitives

As already mentioned above (chapter 1.1), a socket communicates always associated with a protocol. For the programmer in a high level programming language, protocols specify the interface to operate with sockets.

The following table lists the most common TCP/IP *Socket Primitives* with a short description (Anupama, 2007) & (Tanenbaum & Steen, 2007, p. 142, ch. 4.3.1):

Primitive	Service description
Socket	creates a new socket e.g. communication end point
Bind	associates a local address with a socket
Listen	allows to accept new incoming connections to a socket
Accept	blocks until a connection request
Connect	connects to a socket
Send	sends a message to a socket
Receive	reads a message from a socket
Close	aborts a connection

The following definition of the function *send()* is the implementation of the *Socket Primitive Send* by the GNU C Library (Loosemore *et al.*, 2015, p. 457, ch. 16.9.5.1):

```
ssize_t send(   int socket,
                const void *buffer,
                size_t size,
                int flags)
```

A short description for the parameters:

- socket: The socket file descriptor.
- buffer: Memory address to the data which shall be send.
- size: The size in bytes of the data that shall be send.
- flags: Additional information for the send instruction (Loosemore *et al.*, 2015, p. 459, ch. 16.9.5.3)

The function returns the number of bytes which have been transmitted or -1 if there occurred any error. For more detailed information reading the referenced GNU C Library Reference Manual (Loosemore *et al.*, 2015) on page 457 in chapter 16.9.5.1 is recommended.

The *send()* function is called by the application in user mode which will then call a number of other functions in kernel mode. More detailed, the *send()* function is called in the process that runs in user mode. Afterwards, functions in the socket layer, protocol layer and interface layer are called in kernel mode (Anupama, 2007, p. 2, 16). This implicitly means that even if the interface for sending a message to a socket is relatively easy to use, the actual call stack and possible errors introduces another level of complexity in application logic. But as long as the programmer does not want to treat every possible occurring error he has not to understand how the *send()* function actually performs the desired instruction but instead is able to completely rely on its provided functionality.

It is also important to mention that this was an example of how the *Send* primitive could be defined as a function. Other C libraries can differ in their definition and provided functionality not to mention other programming languages.

3 Definition

Now that the term *Socket Primitive* is more clarified, the following definition of primitives by Andrew S. Tanenbaum gives a very good summary (Tanenbaum & Wetherall, 2011, p. 38, ch. 1.3.4):

> A service is formally specified by a set of primitives (operations) available to user processes to access the service. These primitives tell the service to perform some action or report on an action taken by a peer entity. If the protocol stack is located in the operating system, as it often is, the primitives are normally system calls. These calls cause a trap to kernel mode, which then turns control of the machine over to the operating system to send the necessary packets.

The following is the working definition of a *Socket Primitive*:

> *A Socket Primitive is a socket related function that is provided by various high-level programming language libraries. A socket is a communication endpoint of a UNIX OS. Operating with sockets requires the implementation of a protocol and its interface. Socket Primitives are called primitive because they represent self-contained low-level units that operate with sockets.*

A Primitive Data Types in C++

```cpp
#include <iostream>

/**
 * Prints the size of the transferred value in bytes as follows:
 *
 * Size of 'type_name': X byte
 *
 * Where X is the size that is returned by the function sizeof(T).
 *
 * @param   type_name        The name of the type.
 */
template <typename T>
void print_size(std::string type_name)
{
    std::cout << "Size of " << type_name << ": " << sizeof(T) << " byte" << std::endl;
}

/**
 * Prints out the following integer types:
 *
 * - signed char
 * - unsigned char
 * - char
 * - short
 * - short int
 * - unsigned short int
 * - int
 * - unsigned int
 * - long int
 * - unsigned long int
 * - long long int
 * - unsigned long long int
 */
void print_integer_types()
{
    print_size<signed char>             ("signed char");
    print_size<unsigned char>           ("unsigned char");
    print_size<char>                    ("char");
    print_size<short>                   ("short");
    print_size<short int>               ("short int");
    print_size<unsigned short int>      ("unsigned short int");
    print_size<int>                     ("int");
    print_size<unsigned int>            ("unsigned int");
    print_size<long int>                ("long int");
    print_size<unsigned long int>       ("unsigned long int");
    print_size<long long int>           ("long long int");
    print_size<unsigned long long int> ("unsigned long long int");
}

/**
 * Prints out the following real number types:
 *
 * - float
 * - double
```

```
 * - long double
 */
void print_real_number_types()
{
    print_size<float>       ("float");
    print_size<double>      ("double");
    print_size<long double> ("long double");
}

/**
 * This application prints all listed primitive data types of the
 * GNU C Reference Manual [1] excluding the complex number types
 * with information on their allocated memory:
 *
 * Integer Types:
 * - signed char
 * - unsigned char
 * - char
 * - short
 * - short int
 * - unsigned short int
 * - int
 * - unsigned int
 * - long int
 * - unsigned long int
 * - long long int
 * - unsigned long long int
 *
 * Real Number Types:
 * - float
 * - double
 * - long double
 *
 * [1] Rothwell, Trevis, & Youngman, James. 2015. The GNU C
     Reference Manual.
 * http://www.gnu.org/software/gnu-c-manual/gnu-c-manual.pdf . Free
     Software
 * Foundation, Inc. [Online. Accessed 1st July 2015].
 *
 * @author Florian Willich
 */
int main()
{
    print_integer_types();
    print_real_number_types();

    return 0;
};
```

Acronyms

ASCII American Standard Code for Information Interchange. 2

IPC Inter-Process Communication. 1

OOP Object-oriented programming. 2

OS Operating System. 1, 4

TCP/IP Transmission Control Protocol / Internet Protocol. 1, 2, 3

UDP User Datagram Protocol. 1

References

A S Hornby. 2005. *Oxford Advanced Learner's Dictionary*. 7th edn. Oxford University Press, Cornelsen. ISBN 0-19-431655-6.

Anupama, Bindu. 2007. *Know your TCP system call sequences*. `http://www.ibm.com/developerworks/aix/library/au-tcpsystemcalls/au-tcpsystemcalls-pdf.pdf`. IBM Corporation. [Online. Accessed 30th June 2015].

Loosemore, Sandra, Stallman, Richard M., McGrath, Roland, Oram, Andrew, & Drepper, Ulrich. 2015. *The GNU C Library Reference Manual*. `http://www.gnu.org/software/libc/manual/pdf/libc.pdf`. Free Software Foundation, Inc. [Online. Accessed 1st July 2015].

Rothwell, Trevis, & Youngman, James. 2015. *The GNU C Reference Manual*. `http://www.gnu.org/software/gnu-c-manual/gnu-c-manual.pdf`. Free Software Foundation, Inc. [Online. Accessed 1st July 2015].

Tanenbaum, Andrew S., & Steen, Maarten Van. 2007. *Distributed Systems: Principles and Paradigms*. 2nd edn. Pearson Prentice Hall. ISBN 0-13-239227-5.

Tanenbaum, Andrew S., & Wetherall, David J. 2011. *Computer Networks*. 5th edn. Prentice Hall. ISBN 978-0132126953.

This document was written with LaTeX
Typeface: Open Sans by Steve Matteson.