* TRANSFER FILES - Sliding window protocol
* DOCUMENTATIE RETELE SI CALCULATOARE, ECHIPA:
* BACIU H. ALEXANDRU
* CORDUNEANU VLAD
* What are Berkeley sockets?
* Berkeley sockets is an application programming interface (API) for Internet sockets and Unix domain sockets, used for inter-process communication (IPC). It is commonly implemented as a library of linkable modules.
* A socket is an abstract representation (handle) for the local endpoint of a network Communication path. The Berkeley sockets API represents it as a file descriptor (file handle). In the Unix philosophy that provides a common interface for input and output to streams of data.
* The Berkeley socket API typically provides the following functions: https://en.wikipedia.org/wiki/Berkeley\_sockets#Socket\_API\_functions
* **What is UDP**?
* In computer networking, the User Datagram Protocol (UDP) is one of the core members of the Internet protocol suite.  With UDP, computer applications can send messages, in this case referred to as datagrams, to other hosts on an Internet Protocol (IP) network. Prior communications are not required in order to set up communication channels or data paths.
* UDP uses a simple connectionless communication model with a minimum of protocol mechanisms. UDP provides checksums for data integrity, and port numbers for addressing different functions at the source and destination of the datagram. It has no handshaking dialogues, and thus exposes the user's program to any unreliability of the underlying network; there is no guarantee of delivery, ordering, or duplicate protection.
* UDP is suitable for purposes where error checking and correction are either not necessary or are performed in the application; UDP avoids the overhead of such processing in the protocol stack. Time-sensitive applications often use UDP because dropping packets is preferable to waiting for packets delayed due to retransmission, which may not be an option in a real-time system.
* What is TCP flow control?
* TCP is the protocol that guarantees we can have a reliable communication channel over an unreliable network. When we send data from a node to another, packets can be lost, they can arrive out of order, the network can be congested or the receiver node can be overloaded.
* When we are writing an application, though, we usually don’t need to deal with this complexity, we just write some data to a socket and TCP makes sure the packets are delivered correctly to the receiver node.
* Another important service that TCP provides is what is called *Flow Control*. Let’s talk about what that means and how TCP does its magic.
* Flow Control basically means that TCP will ensure that a sender is not overwhelming a receiver by sending packets faster than it can consume. The idea is that a node receiving data will send some kind of feedback to the node sending the data to let it know about its current condition.
* How TCP Flow Control works?
* When we need to send data over a network, this is normally what happens:
* The sender application writes data to a socket, the transport layer (in our case, TCP) will wrap this data in a segment and hand it to the network layer (e.g. IP), that will somehow route this packet to the receiving node.
* On the other side of this communication, the network layer will deliver this piece of data to TCP, that will make it available to the receiver application as an exact copy of the data sent, meaning if will not deliver packets out of order, and will wait for a retransmission in case it notices a gap in the byte stream.
* How TCP Flow Control works?
* If we zoom in, we will see something like in the upper image.
* TCP stores the data it needs to send in the *send buffer*, and the data it receives in the *receive buffer*. When the application is ready, it will then read data from the receive buffer.
* Flow Control is all about making sure we don’t send more packets when the receive buffer is already full, as the receiver wouldn’t be able to handle them and would need to drop these packets.
* To control the amount of data that TCP can send, the receiver will advertise its *Receive Window (rwnd)*, that is, the spare room in the receive buffer.
* Every time TCP receives a packet, it needs to send an ack message to the sender, acknowledging it received that packet correctly, and with this ack message it sends the value of the current receive window, so the sender knows if it can keep sending data.
* **Frames**
* In the OSI model of computer networking, a frame is the protocol data unit at the data link layer. Frames are the result of the final layer of encapsulation before the data is transmitted over the physical layer.
* A frame is „the unit of transmission in a link layer protocol, and consists of a link layer header followed by a packet.” Each frame is separated from the next by an interframe gap.
* A frame is a series of bits generally composed of frame synchronization bits, the packet payload, and a frame check sequence.
* **Packs**
* It turns out that everything you do on the Internet involves packets. For example, every Web page that you receive comes as a series of packets, and every e-mail you send leaves as a series of packets. Networks that ship data around in small packets are called packet switched networks.
* On the Internet, the network breaks an e-mail message into parts of a certain size in bytes. These are the packets. Each packet carries the information that will help it get to its destination – the sender's IP address, the intended receiver's IP address, something that tells the network how many packets this e-mail message has been broken into and the number of this particular packet. The packets carry the data in the protocols that the Internet uses: Transmission Control Protocol/Internet Protocol (TCP/IP).
* Each packet contains part of the body of your message. A typical packet contains perhaps 1,000 or 1,500 bytes. Each packet is then sent off to its destination by the best available route -- a route that might be taken by all the other packets in the message or by none of the other packets in the message. This makes the network more efficient. First, the network can balance the load across various pieces of equipment on a millisecond-by-millisecond basis. Second, if there is a problem with one piece of equipment in the network while a message is being transferred, packets can be routed around the problem, ensuring the delivery of the entire message. Depending on the type of network, packets may be referred to by another name: Frame block cell Segment
* https://computer.howstuffworks.com/question5251.htm
* **How we will do it**
* Technically we have to implement the principles of the SLIDING WINDOW protocol in our own software (the buffers, what information the packages will contain and the logic behind the sending (for both the sender and the receiver).
* First we have to establish an UDP connection using sockets between the instances of the app and then manage the behavior of the transmission via the app itself.
* We will have 2 buffers (one for sender, one for receiver). First, the connection will be established. During this, the receiver will inform the sender about the buffer dimension. After this, the sender will start sending the packages according to the sliding window protocol. The receiver will send an acknowledgement for each package received and this will contain the number of last correct package and the dimension of the available buffer. The sender will modify its sending rate according to the information received in the last acknowledgement . The sender will try for a defined time period to test if the receiver is ready to get messages again.
* To test if the application work perfectly we will use rand to generate a false acknowledgement.
* http://www.ccs-labs.org/teaching/rn/animations/gbn\_sr/ - describes how it should work