Distributed memory management

Egyed László Attila
Szántó Zoltán
Information Technology
1st year at M. Sc.

2010

Table of Contents

Abstract

Overgrowing your storage area is always a problem. This document show a way how you can cope with this problem using memory management services through a network. We are using a set of service providers along with a name server that can orientate a client to a server. Our system contains memory management services, security protocols and also a name server. These components are connected by our own communication layer.

1. Terminology

Here are a few terms used in this document:

Memory – We refer to memory as a storage area that allows the user to store any information on it.

Memory mapping – Is a way of extending your memory. An address in the memory map its a logical address representing a physical location. The data is stored in the physical memory. Memory mapping defines a way to translate from a logical to a physically address and backwards.

MMS – or Memory Management Service offers users to store and manage data.

Name Server – Knows the every service provider's location. When a client connects to the Name Server it will be redirected to the appropriate service. A service provider can subscribe to the Name Server so its services are available to clients.

2. Introduction

According to Wikipedia a memory is an organism ability to store, retain and recall information. In computer science the term memory is referred to as the part of the computer or electronic device that capable of storing data.

When we talk about a memory devices there are a some features that we have to look out for. Here are some examples: is it writable more than once? Or is it read-only? How fast can I get a stored data? etc. This project focuses on the size issue because when we want store data we need to know exactly how much data can be stored on a specific memory device.

People in general when assigned to certain 'space' they tend to overgrow it. To get an idea of this problem lets look at the following images.

On the first picture the user is granted to 1Mb of physical memory. Everything fine until he realizes that this is not enough for him. But there is a problem. Because he is using all of the

available physical memory..it is physically impossible to extend this.

Note that the user when addressing to a location in the memory is using the actual physical address.

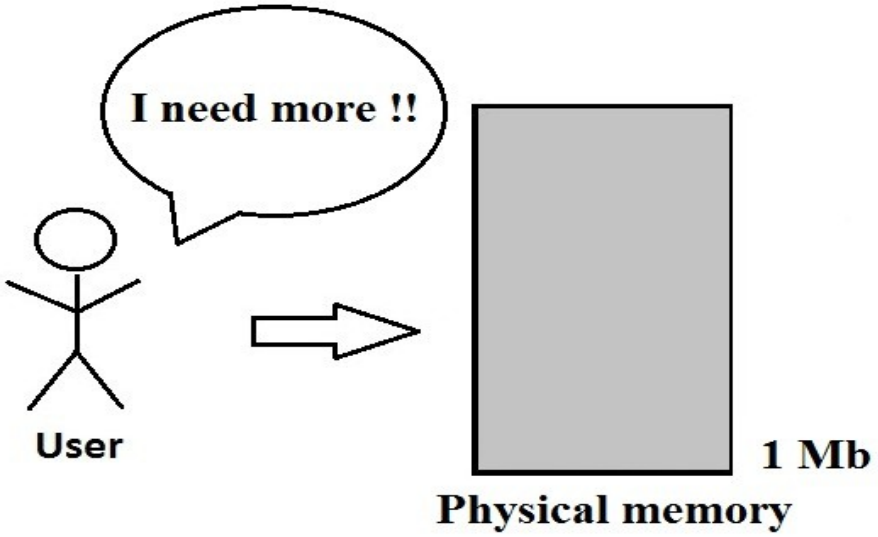

Figure 1: Memory needs

The problem above cannot be solved locally. But what if we had network and we could use the free space on somebody's storage area. In this case we could expand the size of our memory.

But first we need to map out the so that we know the available memory. Then we can create the memory map of the system. From the users point of view there is no difference between the memory map and the physical memory. But if we look closer the memory map contains the the

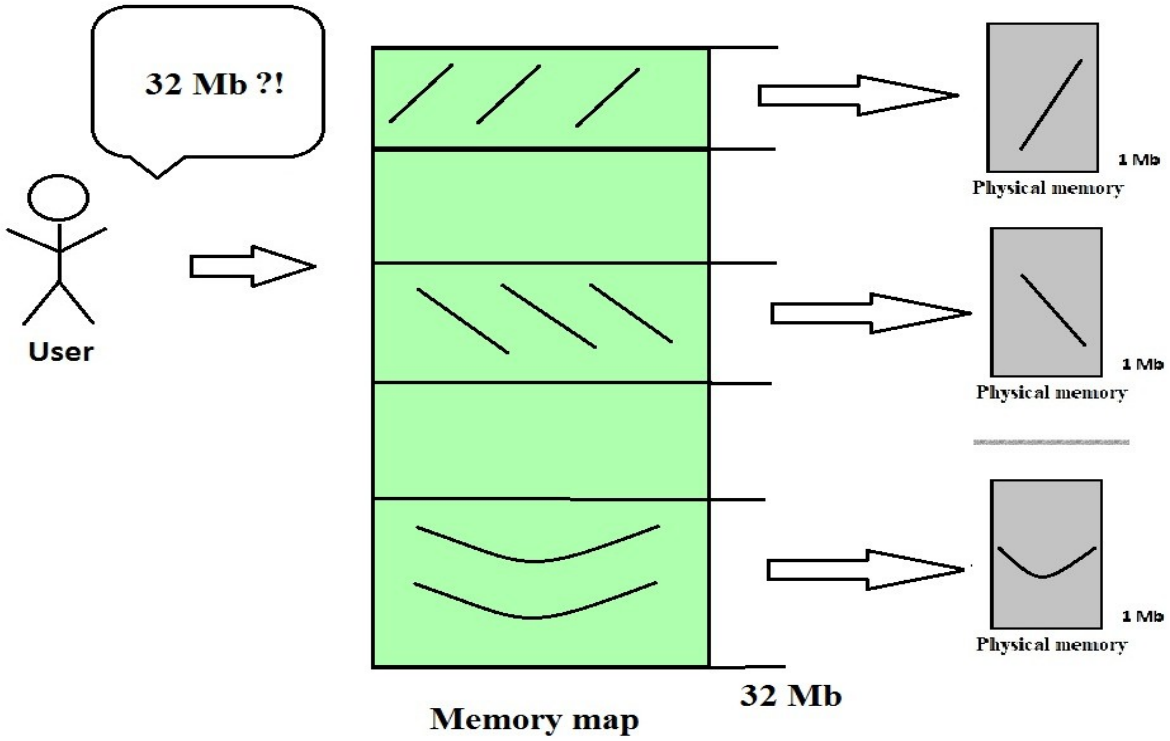

Figure 2: Memory map

addresses of physical memories. The information is stored in the physical memories that aren't even on the same computer. Thus the relative memory of the system can be increased.

3. Structure

The structure of the system is illustrated bellow. Each component uses the same Communication layer to interact with others.

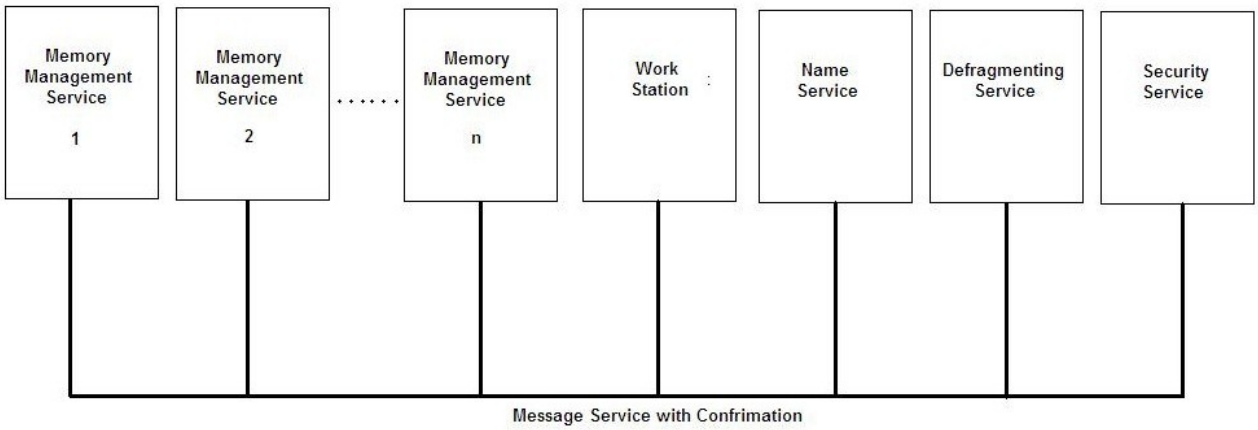

Figure 3: System configuration

In the following chapters the structure and meaning of each component is explained.

4. Communication Layer

The Communication layer is one of the most important layers of the system. Because other components such as Name Server or Memory Management Service are not on the same computer we need to implement a reliable communication layer.

After examining the the messages and their types that we intend to use with this communication layer we decided to work with UDP. The main factor behind this idea is that the system's communication is based on simply 'announce like' messages. I referrer to 'announce like' messages when for example a server announces it's presents to a Name Server. In this case there is no reason to build a TCP like connection because it is a request-response type communication and neither of the two parts know when will the next information exchange take place.

Choosing UDP comes with a price: safety. Sending a message with UDP is fairly easy but we never know for sure what is happening on the other side. A message sent might never reach the its destination. Also if we send a list of messages their receive order might not be the same as the send order. In order to control this problem we will introduce a new layer on top of the simple UDP message layer. This new layer is called the Message Service with Confirmation. The purpose of this

layer is to force the receiver to send feedback to the sender. This way the sender knows that the message arrived to the recipient.

Now that we have a reliable communication we can focus more on the data it self. Because this layer is used to send both user and system messages security is a big issue. Therefore we add another layer to the construction which is the Security Protocols layer. This layer will code the message. If a message is intercepted its still useless until its decoded.

The Communication layer with all the sub-layers can be viewed on the next figure.

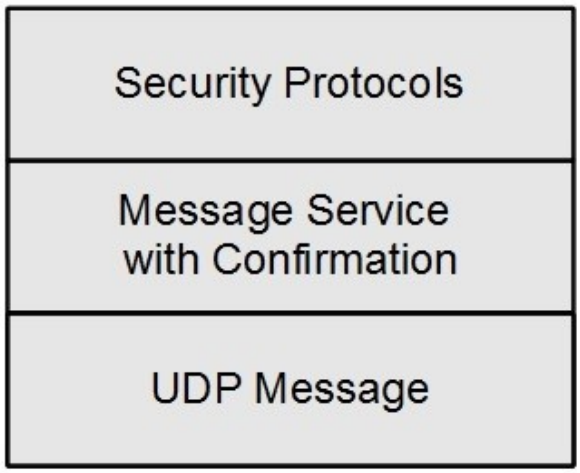

Figure 4: Communication Layer

In the following parts we will discuss more on these sub-layers.

4.1 UDP Message

The UDP Message layer uses UDP connection to send the data. UDP connections are not stateful connections, but rather stateless. This is mainly because there is no connection establishment, nor closing, and most importantly when receiving UDP datagrams we never know the order in which they were sent.

The term datagram defines a packet that is sent containing the address of the recipient and the data. Now this information packet has a storage limit and if the amount of data that we are trying to send exceeds this limit than the data needs to be cut into several pieces that fit. Because our messages don't exceed this limit we skipped the fragmentation process but in case of future development this is a must.

4.2 Message Service with Confirmation

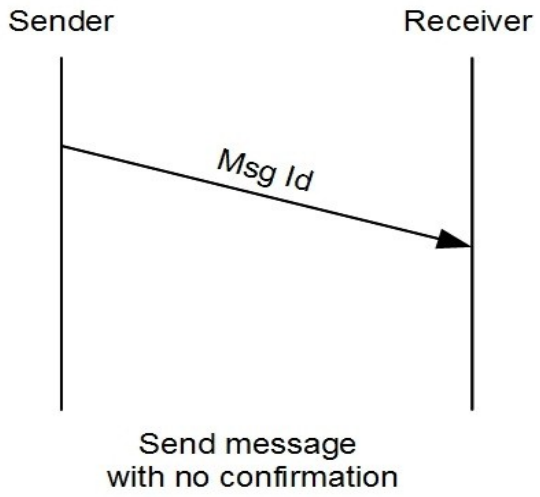

Figure 5: No Confirmation

Using the UDP Message layer only to send information across the network does not promise anything about the the packets reaching their destination, nor about the order they arrive.

If we try to send a file through the network it is vital that the packages arrive in the correct order, otherwise the revived file might not look like the one sent. But than again there are times were speed is more important. If we are broadcasting live from a camera, on the receiver's side it is more important that a new image is received rather then a the correct order.

This layer defines three message types: 'No Confirmation' , 'Confirmation 2' and 'Confirmation 3'. Their meaning and usage is explained bellow.

4.2.1 No Confirmation

It might sound interesting that the 'Message Service with Confirmation' layer has a sublayer that is called 'No Confirmation' but we cannot enforce confirmation mechanism to all our messages.

This message type is used to send messages which don't require confirmation. Again speed is dominant factor here. The sender sends the message identified by 'Msg Id' not knowing when will the receiver get this message. The message might get lost and we will not be notified.

4.2.2 Confirmation 2

The second type is the 'Confirmation 2' message. In this case the sender gets a confirmation message, 'Ack', from the receiver. After sending the message the sender waits a certain amount of time and if confirmation fails to arrive the message will be resent otherwise the transfer is over.

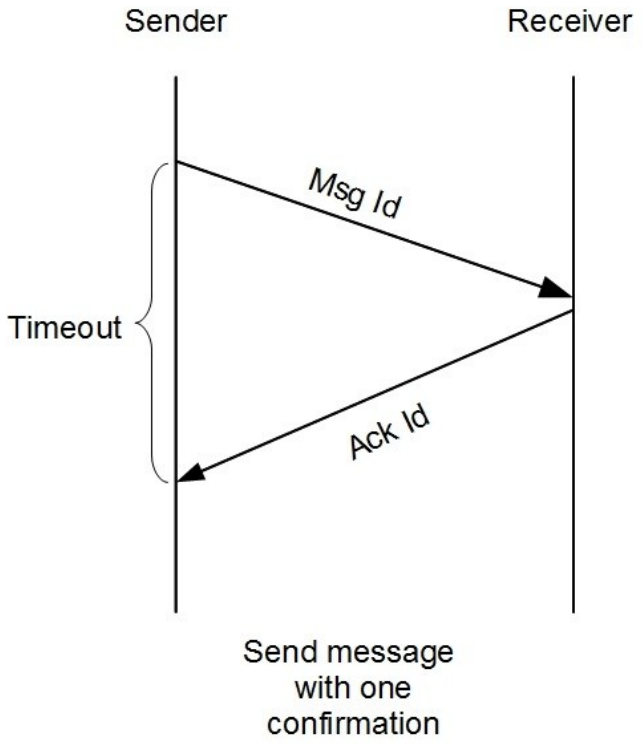

Figure 6: Confirmation 2

If we use 'Confirmation 2' the sending mechanism will have the following state-machine:

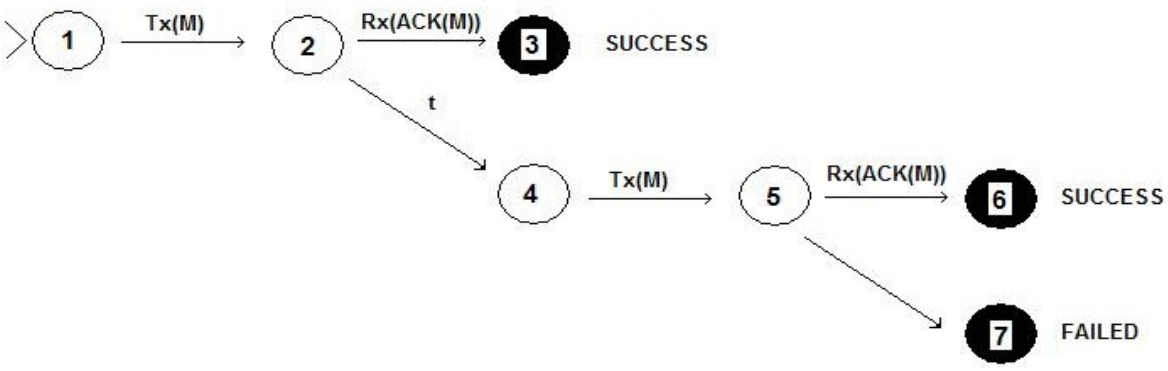

Figure 7: Confirmation 2, send state-machine

The corresponding receive state-machine is:

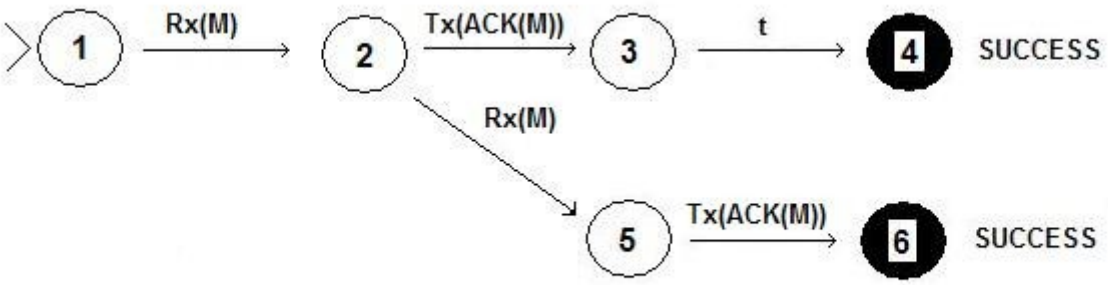

Figure 8: Confirmation 2, receive state-machine

4.2.3 Confirmation 3

Although a message sent with 'Confirmation 2' provides a way to send feedback to the sender there is no information about the 'Ack' or confirmation message ever reaches the sender. To handle this situation another confirmation message is introduced. This way the sender acknowledges the sent 'Ack' message.

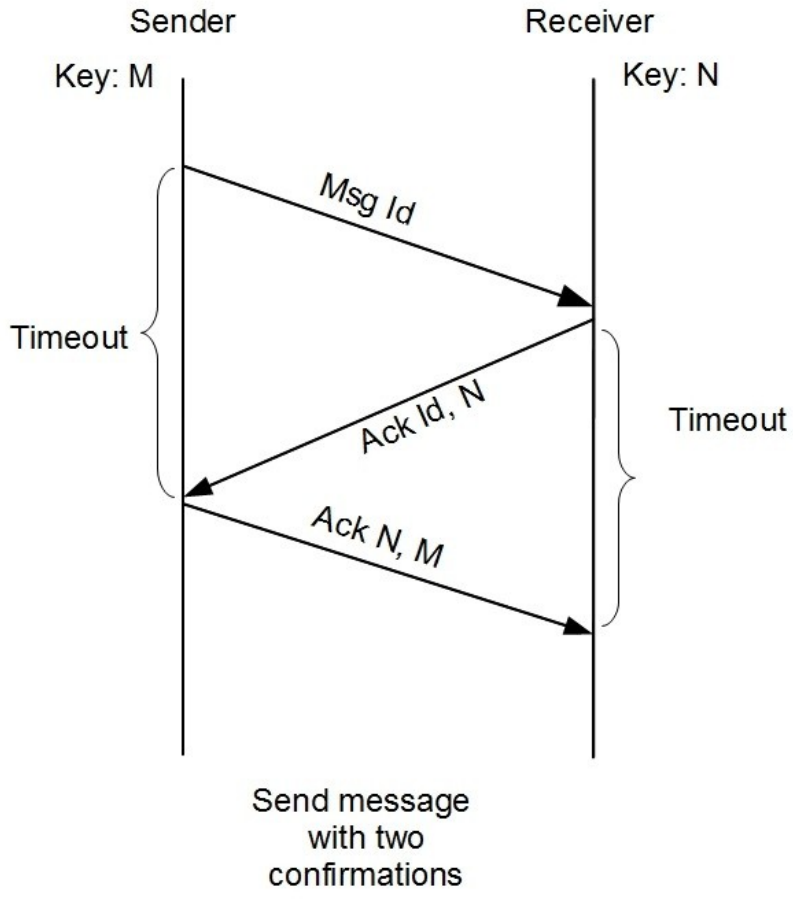

Figure 9: Confirmation 3

The process is similar to 'Confirmation 2'. Suppose that the receiver has a private key N which is sent together with the 'Ack' over the message identified by the 'Id'. When the sender gets this respons from the receiver it will create a new acknowledgment message over the receiver's key (N), also introducing his own key (M) as well.

4.3 Security Layer

Although security is a major concern in data transfer and communication we intentionally left this section empty mainly because of time issues but also because we do not intend to use secrete data just plain text. Therefore implementing the security layer is left for future development.

4.4 Message structure

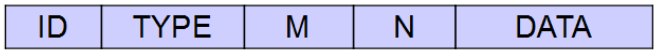

Figure 10: Message structure

Message header contains:

- **ID**: is the unique number that identifies the message, it is a 4 byte unsigned int
- **TYPE**: represents the type of the message, it's enum, which is 1 byte . The type values are:
  - **SEND**: simple message without confirmation
  - **SEND1**: message with one confirmation, equivalent to 'Confirmation 2'
  - **ACK1**: acknowledge message to SEND1 message
  - **SEND2**: message with two confirmations, equivalent to 'Confirmation 3'
  - **ACK2_1** : acknowledge message for SEND2 message
  - **ACK2_2**: acknowledge message for ACK2_1 message
  - **M** : represents the sender key, 4 byte unsigned int
  - **N** : represents the receiver key, 4 byte unsigned int
  - **DATA**: represents the message text which will be sent, it's a char array

5. Name Service

The Name Service is a static component. Its sole purpose is to redirect clients to the appropriate servers. The Name Service has an public address, everybody can connect to it. The server components register to the Name Service so their services are visible. Clients on the other

hand chose the best service from a the list and will try to establish connection with it.

6. Memory Management Service

Memory Management Service is a service that provides storage area for clients. After registering to the Name Server, they wait for incoming connections. If they are contacted by a client they offer storage area along with some functions like memory allocation, read or write. If the service is no longer possible, the MMS will unregister from the Name Server.

7. Workstation - Client

The Client component is more of a test component that we use to test other components. Will attempt to connect to the Name Server and get some services. After that execute some simple tasks such as writing and reading to a certain memory block provided by one of the MMS services.

The interesting part with the workstation is the 'ClientAPI'. This small component hides the memory locations, the communication etc. The client working with the 'ClientAPI' has a set of functions (alloc, free, read, write) that he can use in his work. The rest is done by this API. The user does not have any information about any server or whatsoever and uses the available memory as if it was just local memory.

The 'ClientAPI' connects to the Name Server and gets the available memory of the entire system. From this it creates a memory map and allows the user to work on it.

8. Security Protocols Service

This services is left for future development. It has nothing to do with the communication security. The purpose of this services is to provide security for the stored information such as access keys or encryption.

9. Communication protocols for components

The communication between components is based on request-response. If one component wants to exchange information with another component, it first initiates a request and waits for the response. Getting a request means that we have to process the request and send a response to the caller. This response can be an acknowledge message, information or an error message.

The communication packet is no standard packet for all types of requests and responses. The first 3 fields are the same in every packet, but the rest is dependent on the message type:

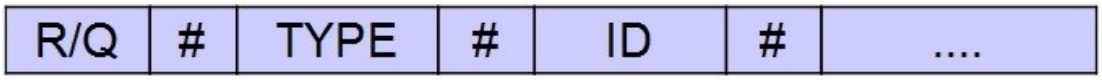

Figure 11: Protocol message header

- we use '#' character as separator
- **Q**: for request
- **R**: for response
- **TYPE**: is a char array (string), which contains the request/response type
- **ID**: is the request/response identity, for example if we send a request to a service with ID = 6 the response ID will 6 too.

The last part of the packet is not filled because this contains request or response specific elements that change whenever a components changes, for example Name Service – MMS, or MMS – Workstation.

9.1 Name Service – MMS

The Name Service is a unique component because it is static. After starting it just waits for incoming requests. Handles these request and send the formed response. But it never actually sends any request to anybody.

When a Memory Management Service wants to register to the Name Service it will send the following message:

Figure 12: MMS request to Name Service

Where :

- IP is the Memory Management Service IP address,
- PORT is the port number where this service is waiting for message
- Service Type is a string which contains the service name, for example in this case is MMS_SERVICE

The Service Type field is left open for other future services. So if let's say a 'Memory Cleaner Service' wants to register, it will use a 'MCS_SERVICE' type.

The Name Service after receiving this request, will try to register the service in the services list. This is done by adding a new element which contains the service address and type. The generated response:

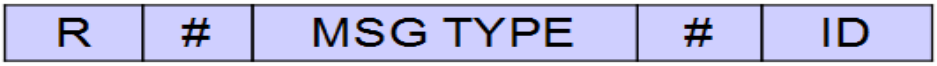

Figure 13: Name Service response to MMS

Where:

- **ID** is the same as in the request
- **MSG TYPE** can be:
  - **R_REG_SERVICE_OK:** if everything went well
  - **R_REG_SERVICE_ERR:** is returned if an error occurred

Withdrawing services from the Name Service is done in the same manner. The only difference is that the 'MSG TYPE' will be: **Q_UNREG_SERVER**. In this case the response would have **R_UNREG_SERVICE_OK** or **R_UNREG_SERVICE_ERR** message type value.

9.2 Name Service – Workstation or Client

The Client Terminal can has to locate were the available services are. This means creating a request to the Name Server in which the services list is asked:

Figure 14: Client request: service list

The Name Server will send following response:

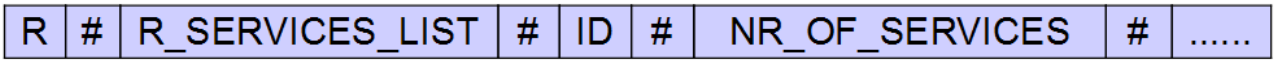

Figure 15: Name Service response to Client

Where:

- **ID** : is the same as in the request
- **NR_OF_SERVICES** : the number of the services registered to the name service
- if there is no service available NR_OF_SERVICES = 0 and the last separator character won't be at the end of the message
- the services list if not empty is listed the following way:

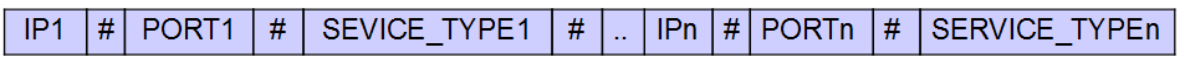

Figure 16: Service list for Client

There is also a way for the Client to ask information about a specific service. Because we only use one type of service, MMS this part is left for future development.

9.3 MMS – Workstation

The Client Terminal can send the following types of requests to the MMS:

- **Q_GET_NR_OF_FREE_BLOCKS**: to get the number of free blocks on the MMS. The response to this request is: **R_NR_OF_FREE_BLOCKS**
- **Q_ALLOC_A_BLOCK**: to allocate a memory block on the MMS. The response is memory block ID or in case of error an error message
  - **R_ALLOC_BLOCK_OK**
  - **R_ALLOC_BLOCK_ERR**
- **Q_FREE_A_BLOCK**: to free a memory block on the MMS. The Client will send the memory block ID. MMS will send the operation result:
  - **R_FREE_BLOCK_OK**
  - **R_FREE_BLOCK_ERR**
- **Q_WRITE_A_BLOCK**: to write a memory block on the MMS. The Client send the block ID and the block data and the MMS will send back the operation result:
  - **R_WRITE_BLOCK_OK**
  - **R_WRITE_BLOCK_ERR**
- **Q_READ_A_BLOCK** : to read a memory block the CT will send the memory block ID and the MMS will send back the memory block or an error message:
  - **R_READ_BLOCK_OK**
  - **R_READ_BLOCK_ERR**

10. User guide

The developed system is not a complete system all by it self. Its more like a sub-system on top of which a bigger distributed system is built, ex. distributed data storing software.

Although the working of the system is briefly explained the usage is a bit complicated. Therefore we thought that it would be easier to work with if there was a help menu associated with every executable. So before you run a program make sure you type 'ProgramName – help' to learn how to use it properly.

11. Notes

Figure list

12. References

1. http://www.ibs.ro/~bela/Teachings/DOS/project.html