MOSCOW STATE UNIVERSITY

FACULTY [OF COMPUTATIONAL MATHEMATICS AND CYBERNETICS](https://www.msu.ru/en/address/#vmk)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**RAM management and data allocation**

A report by a student

of the 201st group,

Parygin Artiom

Moscow

2019

# Contents

[What is RAM 3](#_Toc9501903)

[Static and dynamic RAM 4](#_Toc9501904)

[Data allocation 5](#_Toc9501905)

[Single continuous memory distribution 6](#_Toc9501906)

[Unmovable sections distribution 6](#_Toc9501907)

[Movable sections distribution 8](#_Toc9501908)

[Page distribution 9](#_Toc9501909)

[Segment distribution 11](#_Toc9501910)

[Segment-page distribution 12](#_Toc9501911)

[Conclusion 13](#_Toc9501912)

[Source list 14](#_Toc9501913)

What is RAM

**Random-access memory** (**RAM)** is a form of computer data storage that is capable of storing the executed machine code.

A random-access memory device allows data items to be read or written in almost the same amount of time irrespective of the physical location of data inside the memory. In contrast, with other direct-access data storage media such as hard disks, CD-RWs, DVD-RWs and the older magnetic tapes and drum memory, the time required to read and write data items varies significantly depending on their physical locations on the recording medium, due to mechanical limitations such as media rotation speeds and arm movement.

RAM contains multiplexing and demultiplexing circuitry, to connect the data lines to the addressed storage for reading or writing the entry. “Multiplexing” means that the cirquit gives one output out of several; “demultiplexing” cirquit gets an input and gives one of several outputs. Usually more than one bit of storage is accessed by the same address, and RAM devices often have multiple data lines and are said to be "8-bit" or "16-bit", etc. devices.

In today's technology, random-access memory takes the form of integrated circuits. RAM is normally associated with volatile types of memory, where stored information is lost if power is removed, although non-volatile RAM has also been developed. Other types of non-volatile memories exist that allow random access for read operations, but either do not allow write operations or have other kinds of limitations on them. These include most types of ROM and a type of flash memory called *NOR-Flash*.

Static and dynamic RAM

The two widely used forms of modern RAM are static RAM (SRAM) and dynamic RAM (DRAM). In SRAM, a bit of data is stored using the state of a six-transistor memory cell. This form of RAM is more expensive to produce, but is generally faster and requires less dynamic power than DRAM. In modern computers, SRAM is often used as cache memory for the CPU. DRAM stores a bit of data using a transistor and capacitor pair, which together comprise a DRAM cell. The capacitor holds a high or low charge (1 or 0, respectively), and the transistor acts as a switch that lets the control circuitry on the chip read the capacitor's state of charge or change it. As this form of memory is less expensive to produce than static RAM, it is the predominant form of computer memory used in modern computers.

Both static and dynamic RAM are considered *volatile*, as their state is lost or reset when power is removed from the system. By contrast, read-only memory (ROM) stores data by permanently enabling or disabling selected transistors, such that the memory cannot be altered. Writeable variants of ROM (such as EEPROM and flash memory) share properties of both ROM and RAM, enabling data to persist without power and to be updated without requiring special equipment. These persistent forms of semiconductor ROM include USB flash drives, memory cards for cameras and portable devices, and solid-state drives.

ECC memory (which can be either SRAM or DRAM) includes special circuitry to detect and/or correct random faults (memory errors) in the stored data, using parity bits or error correction codes.

In general, the term *RAM* refers solely to solid-state memory devices (either DRAM or SRAM), and more specifically the main memory in most computers. In optical storage, the term DVD-RAM is somewhat of a misnomer since, unlike CD-RW or DVD-RW it does not need to be erased before reuse. Nevertheless, a DVD-RAM behaves much like a hard disc drive if somewhat slower.

Data allocation

**Data allocation** in the operational memory is an essential problem in developing an operational system, as it defines the way calculating resources are being used. It is important for knowing the state of each memory cell, whether it is free or being used. When the strategy of RAM arrangement is chosen, the main question solved is for which process and for how much time the resources should be assigned.

Another important part of data allocation is freeing memory: there should be a strategy of freeing the memory when the process is ending and a strategy of freeing the memory for a different process (by moving the data to external memory). In the second case, the solution becomes less obvious: the process to be downloaded has to be chosen as well as the part of the memory to free.

There are several models of RAM management, which include different hardware, typical algorithms, their own advantages and disadvantages.

Single continuous memory distribution

This model of RAM distribution is one of the simplest. RAM is divided into two parts: the one for the operational system and the other one for user processes. In this model, only one registry is needed to store the border between two parts. If the user tries to access the OS part, the attempt is being terminated.

The main advantage of this model is simplicity. For instance, Microsoft DOS didn’t even have the border between two parts. The disadvantages, though, make it impossible to be used for modern computers: firstly, a great amount of memory isn’t being used for most of the time, as the process takes it all for the time it runs, though it usually requires to access only a few certain parts. Consequently, this model is inefficient, and it makes multiprocessing completely impossible. Also, the model restricts the size of the user-run process itself, as it can’t be expanded.

Unmovable sections distribution

This memory allocation type is very close to single continuous distribution. Just like it, this distribution has the separation between the OS processes and user processes. However, the second part is split to a certain number of sections, each of which has a fixed size. That setting is included in the operational system, so that the division is completed before the processes start. The queue of user-run processes is divided between those sections.

There are several ways of using this model in real computers. In one, two registries are used for storing the beginning and the ending addresses for each process. The alternative way is the use of PSWs – processor status words. Each part of the RAM gets is own protection “key”. Each process has its its own protection key, which is stored in a registry. Memory access is allowed if the protection key of the process matches with the key in the memory itself.

There are also several ways of how the memory is allocated for this distribution model. The queue of processes is sorted into several queues, each of which leads to a certain memory section. During the division, each process is matched with the smallest section that is big enough to hold it.

The model with a single queue is more flexible in terms of memory distribution. It does, however, cause a problem of choosing which process should take the part of the memory recently turned free from the previous processes.

One of existing solutions suggests that the first process from the queue which can fit in that memory section should take it; it is rather simple, but often the size of the processes doesn’t match to the size of the memory allocated for them.

In the other solution the place in RAM is taken by the process which needs the biggest amount of memory of all the processes which could fit inside that section. It is usually more effective than the second one, but the small processes rarely get the memory.

The issue of processes being ignored can be solved by a “discrimination counter” variable. A discrimination of a process is a case when the section for this process has been freed, but the process is rejected to get memory. While choosing the process for the section, the scheduler checks the counter value.

*Unmovable sections* *distribution* can be brought to life with very simple algorithms. However, the internal fragmentation of each process makes a lot of space unused, since the sizes of processes almost never match the sizes of memory sections. It is also inefficient as the whole process has to be in memory, though usually it works only with a much smaller part.

Movable sections distribution

This distribution model solves the problem of loading a wandering number of processes, and each process gets a section with the matching memory amount. Consequently, the system allows moving processes and sections in the memory, which lets to get rid of fragmentation.

This model of RAM usage has a special “compression” process, which displaces the processes in memory, uniting all “small” memory fragments into one, which can hold bigger processes.

In one approach to the model, local compression is done with a few (1-2) processes to free the required space. It doesn’t usually help with allocating memory for bigger processes. Also, it is turns out to be very slow when many processes require the memory to be freed. In a different one, all processes are stopped to be moved to free all the memory that can be freed.

Getting rid of memory fragments makes this model far more effective in terms of computer memory usage, but much slower due to requirement of constant defragmentation.

Page distribution

*Page distribution* is based on the principle of dividing the addressed space into blocks of fixed size defined as “pages”. It usually has a virtual page workspace (used by the program) and a physical page workspace (the one which exists in the computer). The so-called “page table” included in hardware links the virtual pages and physical ones, and considerably boosts the access to memory.

At the beginning of this distribution, when the implementation of the concept just begun, a big page table caused a list of issues, including the size of the table and the requirement to reload the whole table during the change contexts. Keeping the table in RAM isn’t affordable, and loading it by parts would take too long. That is the reason page table was suggested to be used with cache memory.

One of the first decisions made for optimizing memory was the creation *Translation Look-aside Buffer (TLB).* This buffer stores a certain number of recently used virtual page addresses. The TLB is of limited size, and when it cannot satisfy a given request (this sityation is called a TLB miss) the page tables must be searched manually (either in hardware or software, depending on the architecture) for the correct mapping. Larger page sizes mean that a TLB cache of the same size can keep track of larger amounts of memory, which avoids the costly TLB misses.

In case of a miss, the oldest string is deleted from TLB and replaced with the requested and found one.

While the size of TLB still appeared to be too big, the hierarchical page table organization was created. This page table type is more complex, as the information about the page is represented in a combination of numbers instead of a single one. Each number is used for addressing its level of tables. The hierarchy can have 2, 3 and 4 levels.

Another solution for minimizing the size of the page table is the use of hashing (hash-tables). The method is based on a special “hash” function, which projects a number of virtual page number values to a smaller number of values. The function allows value collisions (situations, where a single cluster has several different values). Each string in hash-table includes a list of its collisions, which helps to find the right page faster.

Final solution is the inverted page table. In this case, the processor has to work with process id’s. Virtual address consists of three values – PID, virtual page number and the offset in it. This approach uses a single table for the whole system, and each line of the table matches a physical page. Consequently, the update is much faster than in other models.

Segment distribution

The main disadvantage of page distribution is the single range of virtual addresses from zero to a certain limit. That becomes an issue for using *heap* and *stack* memory structures, as they get a limited part of memory not included in the pages, and even if many memory pages are free, there might not be enough disk space for them.

Segment memory distribution model is supposed to get rid of those problems. Each process is represented as a combination of segments, each of which can have its size and functions: there could be stack segments, static data segments etc.

This model also includes virtual page table, which just has segment numbers and addresses where segments start. Virtual addresses consist of the number of segment and the offset in it.

The main advantage of this model is simplicity. The disadvantage is the mentioned early localization issue (each segment has to be in memory as a solid block). Swapping also becomes an issue as it’s done by the whole process or at least one segment, which becomes slow for large segments. That also limits the size of a segment because it has to be fit in the memory.

Segment-page distribution

Segment-page distribution is a next step in development of the segment distribution model. This model defines the virtual address as a combination of the number of segment and the offset in it. However, there is also another segment table, in which the linear address, which consists of the page number and the offset in it. Basically, the segments are divided into pages.

This model has both the advantages of logical segmenting (memory separation by the functions) and the advantages of page segmenting (allowing to work with certain memory pages without loading the whole segment in RAM).

An example of segment-page distribution has been suggested by Intel. In the model, a virtual address is presented in a selector and the offset in the selector. Selector contains information about the segment’s localization, described either in LDT *(Local Descriptor Table)*, or GDT *(Global Descriptor Table).* Elements of the Local Descriptor Table is available for only one process. Global Descriptor Table has descriptors which can be separated between a few. Each line from both GDT and LDT contains full information about the segment (base address, size, type etc.)

One interesting feature about this model is the possibility to turn off the page function and turning it to work as a segment distribution. It is also possible to make it work like the page distribution model by disabling the segments. That makes it most universal way of managing RAM due to its compatibility with many different tasks and optimization.

Conclusion

Random access memory is one of the most important parts of the computers. The computational capabilities of the whole computer are defined not only its size and speed, but also the way those resources are used.

In the report, a few data allocation models have been described. In fact, creating those have been a part of some of the major steps in achieving the possibilities we possess. Without any doubt, computer science will keep developing in this area as the demands for RAM keep increasing on a regular basis.

Source list

1. Mashechkin, I.V. "Operational Systems – abstract of lectures", 2017
2. Gallagher, Sean (2013-04-04). ["Memory that never forgets: non-volatile DIMMs hit the market"](https://arstechnica.com/information-technology/2013/04/memory-that-never-forgets-non-volatile-dimms-hit-the-market/). Ars Technica. [Archived](https://web.archive.org/web/20170708073138/https:/arstechnica.com/information-technology/2013/04/memory-that-never-forgets-non-volatile-dimms-hit-the-market/) from the original on 2017-07-08.
3. Bellis, Mary. ["The Invention of the Intel 1103"](http://inventors.about.com/library/weekly/aa100898.htm).
4. ["IBM Archives -- FAQ's for Products and Services"](http://www-03.ibm.com/ibm/history/reference/faq_0000000011.html). ibm.com. [Archived](https://web.archive.org/web/20121023184527/http:/www-03.ibm.com/ibm/history/reference/faq_0000000011.html) from the original on 2012-10-23.
5. Napper, Brian, [Computer 50: The University of Manchester Celebrates the Birth of the Modern Computer](https://web.archive.org/web/20120504133240/http:/www.computer50.org/), archived from [the original](http://www.computer50.org/) on 4 May 2012, retrieved 26 May 2012