DOI: 10.46299/ISG.2025.MONO.TECH.2.5.1

5.1 Simulation of visualization processes of ram management via VN network

One of the problems in studying operating systems is that the processes taking place during their operation are invisible to the human eye [165]. Therefore, it is relevant to visualize the processes in the computer when the operating system is running. Consequently, it is necessary to develop appropriate approaches and models to improve the learning process.

A VN network has been developed to simulate the execution of the process and the allocation of the necessary RAM. The VN network is a triple [166],

VN = (R, D, F)

R is the set of informational positions (nodes), R = {r_1, r_2, r_3, r_4, r_5, r_6, r_7, r_8, r_9, r_{10}}; D is the set of transitions, D = {d_1, d_2, d_3, d_4, d_5, d_6}; F is the set of functions assigned to transitions, F = {f_1, f_2, f_3, f_4,f_5, f_6}.

Each process has an m marker, $m \in M$. Each marker has four attributes. The first attribute is the pending time, the second is the active time, the third is the requested memory size, and the fourth is the priority. These attributes are arguments for the transition functions.

The VN network contains ten types of information positions (Fig. 1). The position of the first type (r_1) contains markers corresponding to the processes entered into the system. The position of the second type (r_2) contains markers corresponding to the processes ready to be executed. In this position, the markers are ordered on a FIFO basis if the processes have equal priorities, or by priority if the processes have unequal priorities.

The third position (r_3) contains markers corresponding to the processes to which the RAM area will be allocated according to the "First suitable" strategy. The fourth position (r_4) contains markers corresponding to the processes to which the RAM area will be allocated according to the "Most suitable" strategy. The fifth position (r_5) contains markers corresponding to the processes to which the RAM area will be

allocated according to the "Less suitable" strategy.

The sixth position (r_6) contains markers corresponding to the process in progress. The position of the seventh type (r_7) contains markers corresponding to processes that have left the system.

The position of the eighth type (r_8) contains markers corresponding to the processes that released the occupied memory area because the quantum of time allocated for their execution has expired. The position of the ninth type (r_9) contains markers corresponding to the processes by which the freed memory area should be combined with a non-neighboring area. The position of the tenth type (r_{10}) contains markers corresponding to the processes by which the freed memory area should be combined with the neighboring area.

The VN network contains six types of transitions. On the first type of transition (d_1), function f_1 works, which places markers in position r_2 according to the FIFO principle, if the markers have the same priorities, or places them according to priority if the markers have unequal priorities. The function checks the pending time for each marker. Depending on it, the marker is moved to position r_2, that is, it is placed in the queue of ready processes.

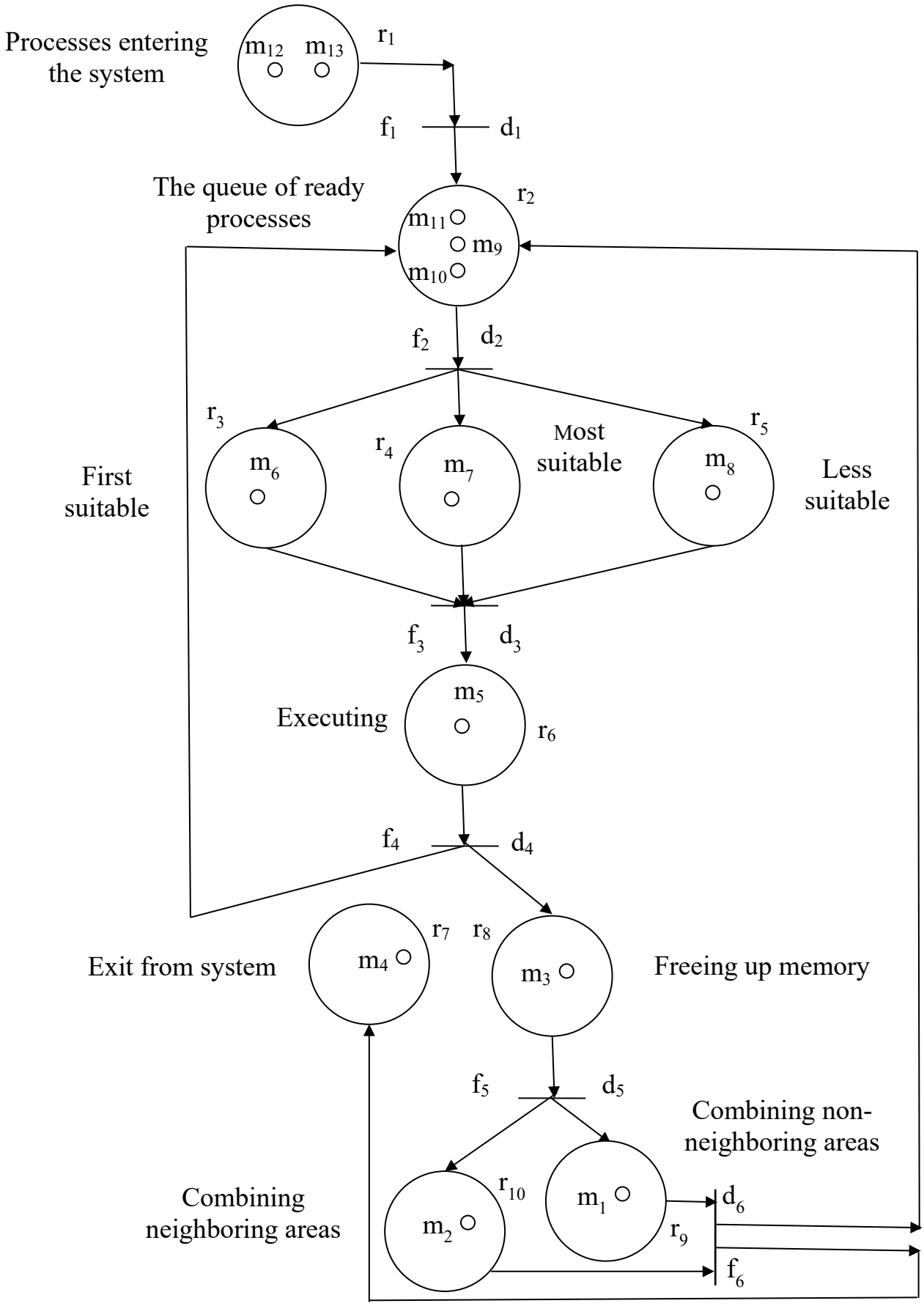

Figure 1. The VN-Network

On the second type of transition (d_2), the function f_2 works, which selects the marker corresponding to the first process from position r_2 (from the queue of ready processes) and places it in position r_3, r_4, or r_5. It depends on the memory allocation strategy. The function checks the value of the T_q quantum. If $T_q = 0$, then the first marker in the sequence is moved to position r_3, r_4, or r_5.

On the third type of transition (d_3), the f_3 function works, which puts markers from positions r_3, r_4, or r_5 to position r_6, as soon as a RAM area is allocated to the corresponding processes. Processes start executing.

On the fourth type of transition (d_4), the function f_4 works, which reduces the time quantum T_q by one unit. As soon as the quantum runs out, the marker moves to position r_8.

On the fifth type of transition (d_5), the f_5 function works, which sends the marker to the r_9 or r_{10} position. It depends on which area of memory was freed. If there is a free area next to the freed area, then the marker will move to position r_{10}; otherwise, to position r_9.

On the sixth type of transition (d_6), the function f_6 works, which combines adjacent or non-adjacent areas of memory and sends the marker to the position r_7. The simulation process ends when the last element of the sequence contains "*".

To simulate RAM management, we must make the following assumptions. [167-168]: a) We know the entry time of each process in the system; b) We know when and how much memory each process requires; c) We know the size of the memory allocated by the operating system for the process.

Taking into account these assumptions, we can present the pending processes in the form of the following sequence:

$P_1(t_{o1}, t_{a1}, z_1)$	$P_2(t_{o2}, t_{a2}, z_2)$	$P_3(t_{o3}, t_{a3}, z_3)$	. . .	$P_i(t_{oi}, t_{ai}, z_i)$	*

Each process has the following features: pending time, active time, requested memory size, and priority. As we can see, the first element of the sequence contains information about the first process, the second element - about the second process, and so on. Here, P_i is the identifier of the process i. t_{pi} is the pending time of the P_i process.

t_{ai} — the time during which the process P_i occupies the area of memory with the size of z_i. Consider an example. Suppose we have the following sequence of processes:

$P_1(0,3,100)$	$P_2(0,1,250)$	$P_3(2,5,230)$	$P_4(1,2,110)$	$P_5(0,1,180)$	*

"$P_1(0,3,100)$" is placed in the first element of the sequence. This means that process P_1 requests 100 units of memory during 3 units of time and gets it if there is a free memory area of the appropriate size. 100 units can be 100 kilobytes or 100 megabytes, etc. Process P_1 is placed in the queue of ready processes immediately because its pending time is 0. Process P_2 also enters the system immediately, which requests 250 units of memory for 1 unit of time. The P_2 process gets it if there is a free area of this size. Otherwise, the process waits for an area of the required size to be freed. After 2 units of time, process P_3 enters the system. After another 1 unit of time has passed, process P_4 enters the system. Along with process P_4, process P_5 also enters the system because its pending time is equal to 0. As soon as 3 units of time allocated to process P_1 expire, it releases the 100 units of memory it occupies and exits the system.

Algorithms for RAM allocation between processes have been developed for cases when processes have equal and unequal priorities. A set of algorithms has been developed for both cases: $L = \{ L_1, L_2, L_3, L_4, L_5, L_6, L_7 \}$.

Here, L_1 is the algorithm for executing the process from the moment it enters the queue of pending processes until the execution is completed, L_2 is a memory allocation algorithm for a process based on the "First suitable" principle, L_3 is a memory allocation algorithm for a process according to the "Most suitable" discipline, L_4 is an algorithm for allocating memory for a process according to the "Least suitable" discipline, L_5 is an algorithm for freeing memory by the process, L_6 is an algorithm for combining two free adjacent areas of memory, and L_7 is an algorithm for combining non-adjacent free areas of memory.

Assume the processes have equal priorities: $\Pi_1 = \Pi_2 = \ldots = \Pi_l$. Accordingly, the processes included in the list of ready processes are queued according to the FIFO discipline. First, let us consider the L_2 algorithm for allocating memory according to the "First suitable" discipline for a process. It consists of the following steps:

1. Sorting of free memory areas according to increasing F_n starting addresses will be performed.

2. The size z_n of the process P_i will be compared to each element of the set of sizes of free areas - $\{ F_1, F_2, F_3, \dots , F_j \}$.

3. The first acceptable one is selected from the set - F_m, the size of which is greater than or equal to the size of the process P_n, $F_1 \geq z_n$.

4. The P_i process will be allocated an area of size F_1, the starting address of which is $A_{in} = F_{i1}$, and the last address is $A_{fi} = A_{in} + z_n - 1$.

5. The size of the remaining free memory will change - $F_k = F_k - z_n$. Its initial address will be $F_{i1} = A_{fn} + 1$, and the last address F_{fl} will remain the same.

Consider the L3 algorithm for allocating memory according to the "Most suitable" discipline for the process. It consists of the following steps:

1. Free memory areas are sorted in ascending order of starting addresses F_i.

2. The differences between the size of each free area and the size of a given process are calculated, $D_k = F_k - z_n$. A set of differences is obtained: $\{ D_1, D_2, D_3, \dots , D_j \}$.

3. The P_n process will be allocated a region of minimum size:

$F^{min}_k = \min\{ D_1, D_2, D_3,..., D_j \}$.

4. The Pn process is allocated an area of size F_k, the starting address of which is $A_{in} = F_{ik}$, and the last address will be – $A_{fn} = A_{in} + z_n - 1$.

5. The size of the remaining free memory will change - $F_k = F_k - z_n$. Its starting address will be $F_{ik} = A_{fn} + 1$, and the last address will remain the same $F_{fk} - k_n$.

Consider the L4 algorithm for allocating memory according to the "Least suitable" discipline for the process. It consists of the following steps:

1. Free memory areas are sorted according to increasing F_i starting addresses.

2. The differences between the size of each free area and the size of a given process are calculated, $D_k = F_k - z_n$. A set of differences is obtained $\{ D_1, D_2, D_3, ..., D_j \}$.

3. The P_i process will be allocated the maximum size area:

$F^{max}_k = \max \{ D_1, D_2, D_3, \dots , D_j \}$.

4. The P_i process is allocated an area of size F_k, whose starting address is $A_{in} = F_{ik}$, and the last address is – $A_{fn} = A_{fn} + z_n - 1$.

5. The size of the remaining free memory will change - $F_k = F_k – z_n$. Its starting address will be $F_{ik} = A_{fn} + 1$, and the last address F_{fk} will remain the same.

Consider the L_5 algorithm for freeing memory by the process. It consists of the following steps:

1. A new area will be added to the set of sizes of the free area F, the size of which is equal to the size of the process P_i, and A_{in} and A_{fn}, are the start and last addresses, $F_{ik}= A_{in}$ and $F_{fk} = A_{fn}$, $F_k= z_n$.

2. It is checked whether, as a result of freeing the memory area, two neighboring areas were placed next to each other or not.

3. If two free memory areas are adjacent to each other, then the L_6 algorithm starts working.

Consider the L_6 algorithm for the combining of two free neighboring areas. It consists of the following steps:

1. The last address of the first free area will be compared with the starting address of the second free area. If the last address is one more than the first, then these two free areas are adjacent to each other - $F_{ks} = F_{k-1b} + 1$.

2. They are combined, that is, the addresses are rearranged. In particular, the starting address of the first neighboring area remains the same, and the last address of the second area will become the last address of the first area, $F_{k-1i} = F_{kf}$.

Consider the L_7 algorithm for the combining non-adjacent free areas of memory. It consists of the following steps:

1. The starting address of the first free area of memory will be compared to the starting address of the memory area occupied by each process. The area whose address is closest to the starting address of the first free area is selected. Also, the starting address of the free area must be less than the starting address of the area occupied by the process, $A_{in} > A_{ik}$.

2. These two memory areas swap places, which is done by changing their start and last addresses: $A_{in} = F_{ik}$, $A_{fn} = A_{in} + z_n - 1$, $F_{ik} = A_{fn} + 1$, $F_{fk} = F_{ik} + F_k - 1$.

3. This process is repeated until two free areas are found next to each other. In this case, the L_6 algorithm of combining two neighboring free areas starts working.

4. The mentioned three steps are repeated until one area of the memory remains free.

Consider the L_1 algorithm. It consists of the following steps:

1. From the queue of pending processes, the corresponding elements of the P_n process will be read. If its pending time is zero, $t_{pn} = 0$, then it is placed in the ready processes list (queue). If $t_{pn} \neq 0$, then it remains in the pending process queue until the pending time expires.

2. It is checked if the last element of the sequence is "*". If there is, it means that we have no more pending processes. Otherwise, proceed to the 3rd step.

3. The P_n process is placed at the end of the list of ready processes.

4. All the above steps are repeated for each process.

5. If process P_n is first in the list of pending processes and the operating system has enough memory to allocate for it, then it will be allocated an area of size z_n in memory.

6. At this moment, the L_2, L_3, or L_4 memory allocation algorithm for the process starts working.

7. Steps 5 and 6 are repeated for each process. As a result, the memory area allocated for processes will be filled.

8. After that, the count of time starts. The activity time of each process t_{an} simultaneously decreases by a unit of time until the activity time of one of them expires, $t_{an} = 0$.

9. If $t_{an} = 0$, then P_n frees the occupied memory area and exits the system.

10. The L_5 algorithm for freeing memory by the process starts working.

11. It is checked whether there are free neighboring areas in the memory. If there is, then the L_6 algorithm of combining two free neighboring areas starts working.

12. The list of ready processes is checked. If there is a process in it, it checks if there is a free area in memory to place it. If there is, then the process will be placed in memory. If it is not, then the combining free non-adjacent areas of memory is

completed, i.e., the L_7 algorithm starts working. If there is still no free memory area of sufficient size, then the process remains on the list of ready processes.

13. If there is at least one process in memory, then proceed to the 8^{th} step. Otherwise, the algorithm stops working.

For processes with unequal priority, $\Pi_1 > \Pi_2 > \ldots > \Pi_l$. Accordingly, the processes included in the list of ready processes are queued according to priority. The above algorithms are valid even in the case of unequal priorities.

Based on the proposed visualization model and the presented algorithms, the corresponding software training system and the corresponding graphic interface are realized. In the first window of the simulator (Fig. 2), the student selects the number of processes, their priorities, the amount of memory they should occupy, as well as active time and pending time. In this case, we have five processes that have equal priority. The student chooses one of the strategies for allocating processes in memory and the amount of memory allocated to processes. These settings can also be generated with a random number generator. Then, the student chooses the process execution mode: automatic or step-by-step. In the case of automatic mode, the interval for changing the state of the process is indicated, for example, 5 seconds. This means that after every 5 seconds, a change of situation will be made: memory allocation/release by a process, or combining neighboring/non-neighboring free areas. Here, the student chooses an interval that makes it easier for him to observe the memory. In step-by-step mode, the situation changes every time the student presses the "Next" button (Fig. 3 - 5).

#1 _ ☐ ☒

Number of processes	5	Memory size	700	Strategy		
				Most suitable		
Processes	Priorities		Pending time	Activity time	Requested memory	
P_1	1		0	4	160	
P_2	1		0	3	150	
P_3	1		0	8	130	
P_4	1		0	5	110	
P_5	1		1	7	120	

Next	End	Generate		
○	Auto mode	5 sec.	⊙	Step mode

Figure 2. The first window of the software simulator.

2 _ ☐ ☒

		Set of pending processes: $P_1(0,4,160)$ $P_2(0,3,150)$ $P_3(0,8,130)$ $P_4(0,5,110)$ $P_5(1,9,120)$ *								
		List of ready processes *								
RAM		Memory allocation table								
Free memory - 700		F_i	F_f	F		A_i	A_f	Z	t_A	P
		1	700	700						
Next		End		Generate						

Figure 3. Formation of a set of pending processes.

# 3									_	☐	☒
		Set of pending processes: P5(1,9,120) *									
		List of ready processes: P1(0,4,160) P2(0,3,150) P3(0,8,130) P4(0,5,110) *									
RAM		Memory allocation table									
Free memory - 700		F_i	F_f	F		A_i	A_f	Z	t_A	P	
		1	700	700							
Next		End		Generate							

Figure 4. Formation of the list of finished processes.

# 4									_	☐	☒
$t_{A3} = t_{a3} - 2 =$ 2 - 2 = 0, $t_{p10} = t_{p10} - 2 =$ 1 - 2 = 0		Set of pending processes: P_5(1,9,120) *									
		List of ready processes: P_2(0,3,150) P_3(0,8,130) P_4(0,5,110) *									
RAM		Memory allocation table									
P_1 - 160		F_i	F_f	F		A_i	A_f	Z	t_A	P	
		161	700	540		1	160	160	4	P_1	
Free memory - 540											
Next		End		Generate							

Figure 5. The first process occupies an area of the requested size in memory.

At the start of work, the sequence elements of each process are blue. The moment the element is read, it starts to flash. When a process is in the list of ready processes, then the corresponding item turns yellow. If the process is present in the list of pending

processes, then the corresponding item turns red. When a process occupies memory, the corresponding item is green. Free areas are white.

Thus, the monograph offers algorithms for visualizing RAM management processes, using which it is possible to simulate RAM allocation strategies: "first suitable", "most suitable", and "least suitable". Algorithms are developed for processes with equal and unequal priority. To simulate the mentioned processes, the VN network is used, which provides the possibility of visualizing several functions performed by the operating system. Based on the VN network, RAM management processes are simulated and visualized. Based on the algorithms mentioned, a software training system is built, which provides the opportunity to teach the processes of RAM allocation effectively. As a result, the student sees the running processes on the screen, changes the parameters, observes the obtained results, and actively participates in the process of studying the mentioned issues.