|  |
| --- |
| ASU faculty of engineering |
| Operating Systems project 1 |
| https://lh4.googleusercontent.com/5aths2DE5VJwVbzlyOZu4GwMrl_rGjIbwbZUZjeLEJPZB8oXWAuBYTkFt1nfC6a9DShabdUNrFUG3jHE20XnUIdj8Brx0romx2ZTe-Qk_23l5cAQs5eJ2vgwD7Cq_5DDazjZ-1sOlM43NpIi |
|  |

**Submitted By:**

Mostafa Ahmed Mohamed Abd Elghany Elkady 17p6040

|  |
| --- |
|  |

Hazem Hamada Abdellatif Mohamed 16p3100

Shehab Ahmed Hassan Kotb 16p6014

**Banker’s Algorithm:**

Description:

a)Intro:

This is a program that implements the Banker’s algorithm.Several processes request and release resources and the algorithm will grant a request only if it leaves the system in a safe state. A request that leaves the system in an unsafe state will be denied. The algorithm will consider requests from n customers for m resources types. Processes will continually loop requesting and releasing resources from the system.The processes request and then release random numbers of resources, which are bounded by their respective values in the need array. You should invoke your program by passing the number of resources of each type on the command line.

b)Algorithm:

safe:

*1)Let Work and Finish be vectors of length ‘m’ and ‘n’ respectively*

*Initialize:*

*Work = Available*

*Finish[i] = false; for i=1, 2, 3, 4….n*

*2) Find an i such that both*

*a) Finish[i] = false  
b) Needi <= Work*

*if no such i exists goto step (4)*

*3) Work = Work + Allocation  
Finish[i] = true*

*goto step (2)*

*4) if finish [i] = true for all i  
then the system is in a safe state.*

Implementation:

After checking that need is less than the work using need\_less\_available variable we increase the work by the resources in the allocation and set the process to be finished and set foundprocess to true cause we have the one we want now.

(need\_less\_available) {

for (int j = 0; j < m; j++) {

work[j] = work[j] + allocation[i][j];

}

finish[i] = true;

foundprocess = true; } }

After that we reset i to make the loop continue after a process has finished we put I = -1 that in the next iteration when i increases in the for loop it starts from the first process again and reset the variable found process to false to search for another process.

If(i == n-1 && foundprocess) {

i = -1;

foundprocess = false;

}

At the end we use alltrue variable to check that all our processes has finished correctly if so the function will return true.

Resource\_request:

Let Requesti be the request array for process Pi. Requesti[j] = k means process Pi wants k instances of resource type Rj. When a request for resources is made by process Pi,

1. *If Requesti <= Needi*

*Goto step (2) ; otherwise, raise an error condition, since the process has exceeded its maximum claim.*

1. *If Requesti <= Available Goto step (3); otherwise, Pi must wait, since the resources are not available.*
2. *Have the system pretend to have allocated the requested resources to process Pi by modifying the state as  
   follows:*

*Available = Available – Requesti  
Allocationi = Allocationi + Requesti  
Needi = Needi– Requesti*

Implementation:

We just looping on the resources to check that the request is less than or equal the available and need.

If the process is not safe we print not safe and inverse the request else print that the process is safe.

If(!safe()) {

request\_inv(request, pnum);

System.out.println(“not safe reversing state”);

printmats();

}

else {

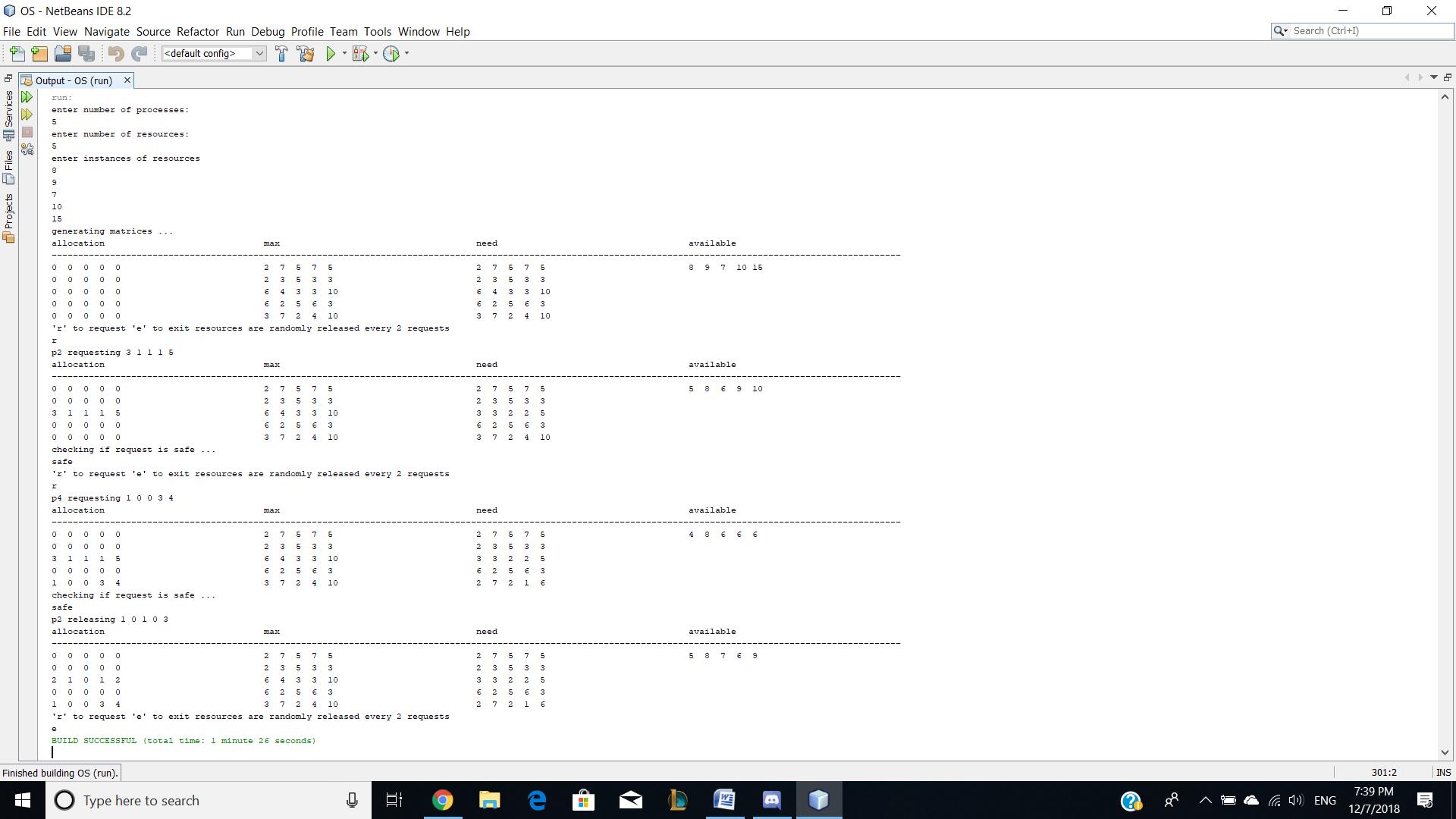
System.out.println(“safe”);

}

c)Inputs:

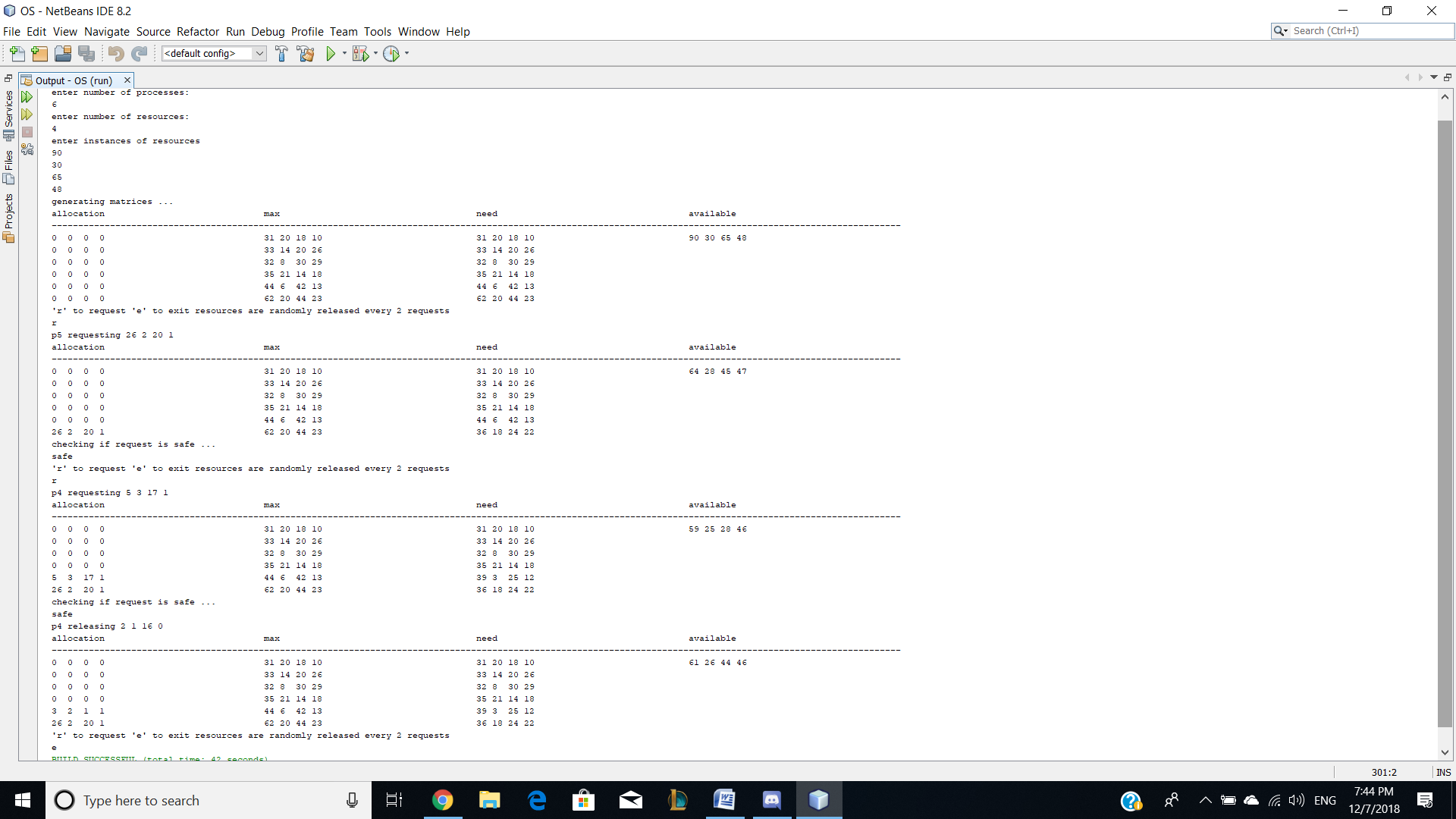
1. At the start of the program you will be asked to enter number of processes you have.
2. Second you will be asked to enter number of resources and instances of each resource.
3. Max and need matrices are randomly generated by the program
4. Then you will be asked to choose to make a request or to exit the program.
5. Every two times you choose r for request a process will release.
6. Pressing e means you want to exit so the program terminates.

TestCases:

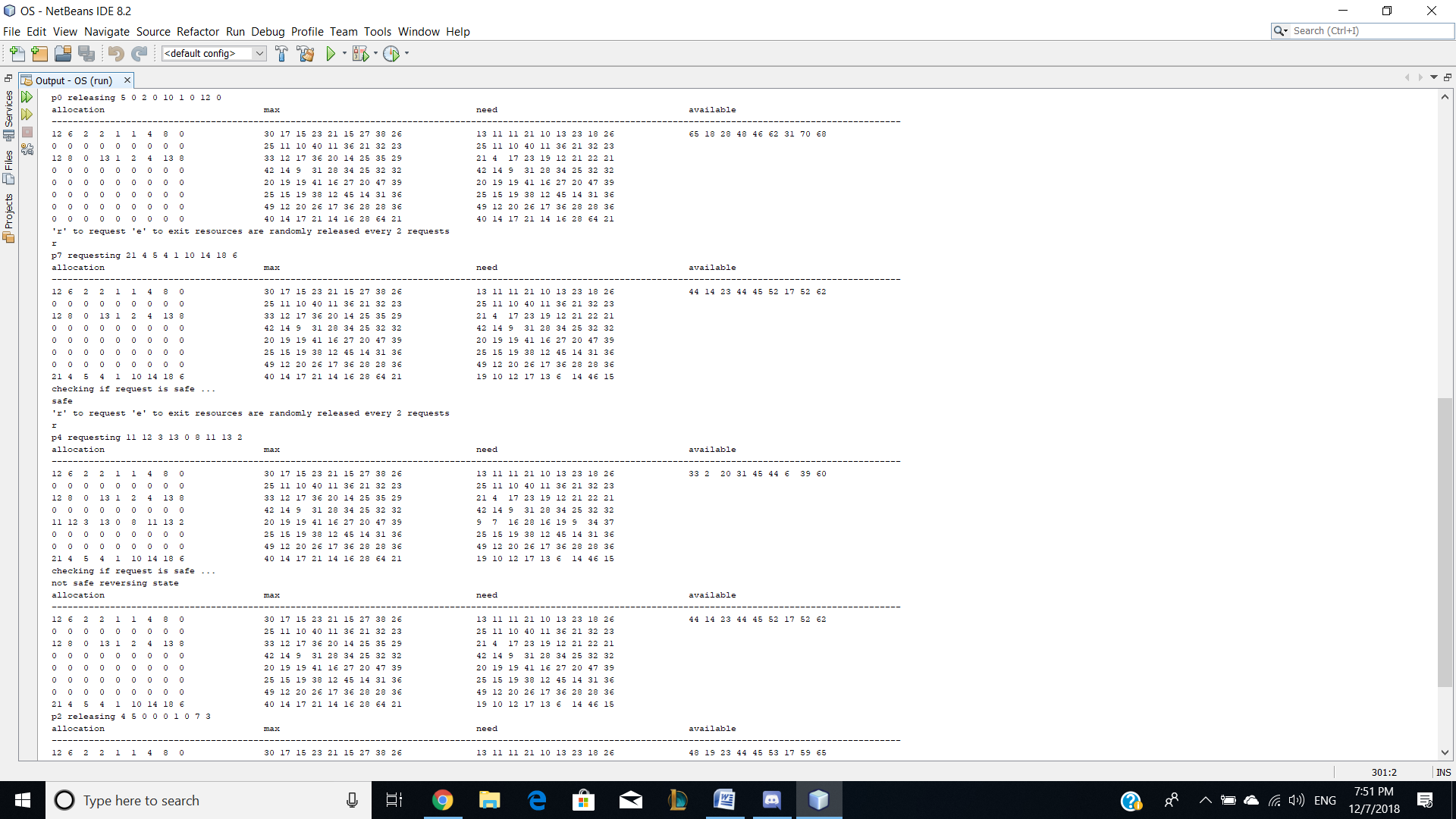
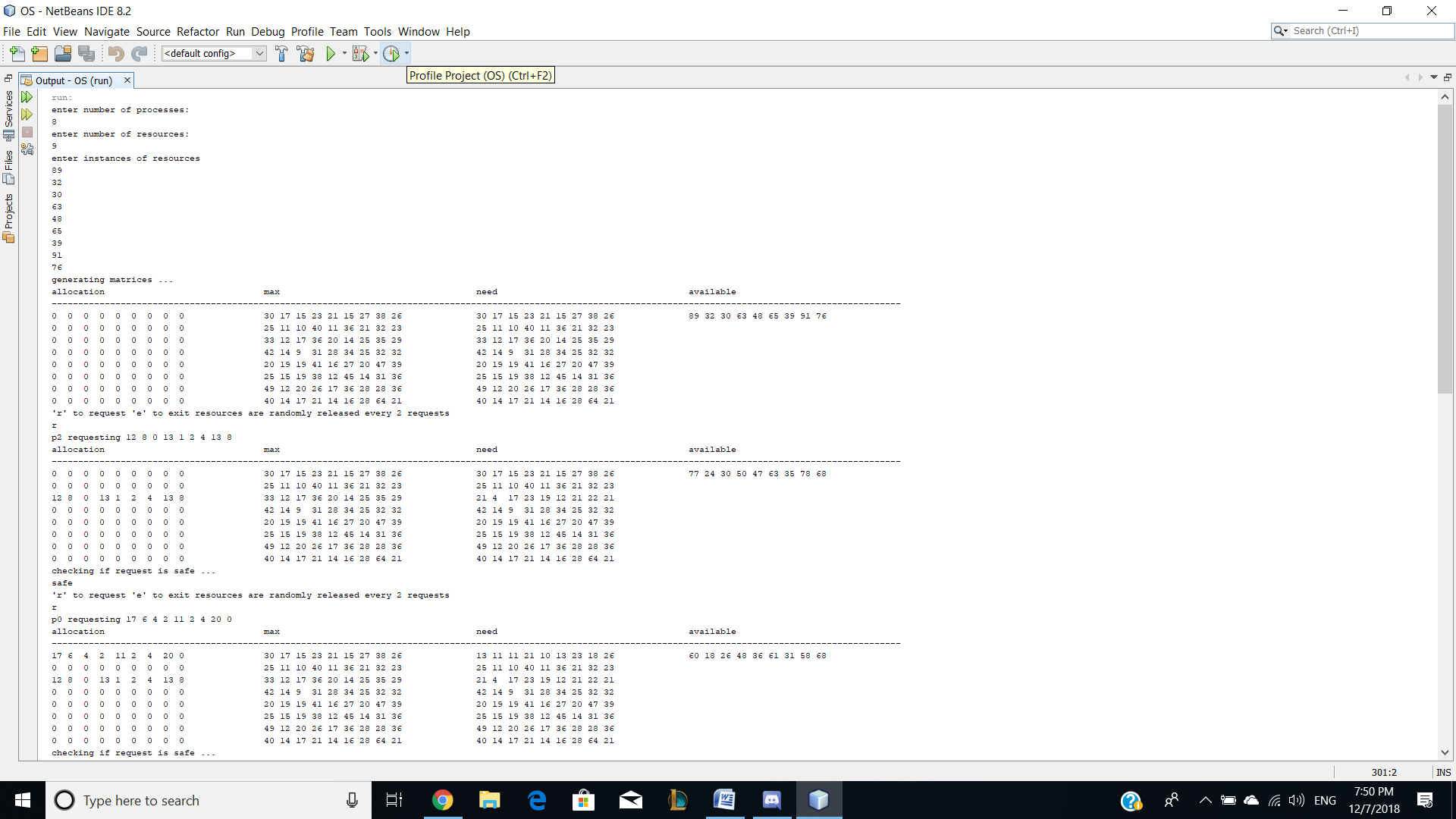


In this case we have 5 processes and 5 resources with instances

(8,9,7,10,15)

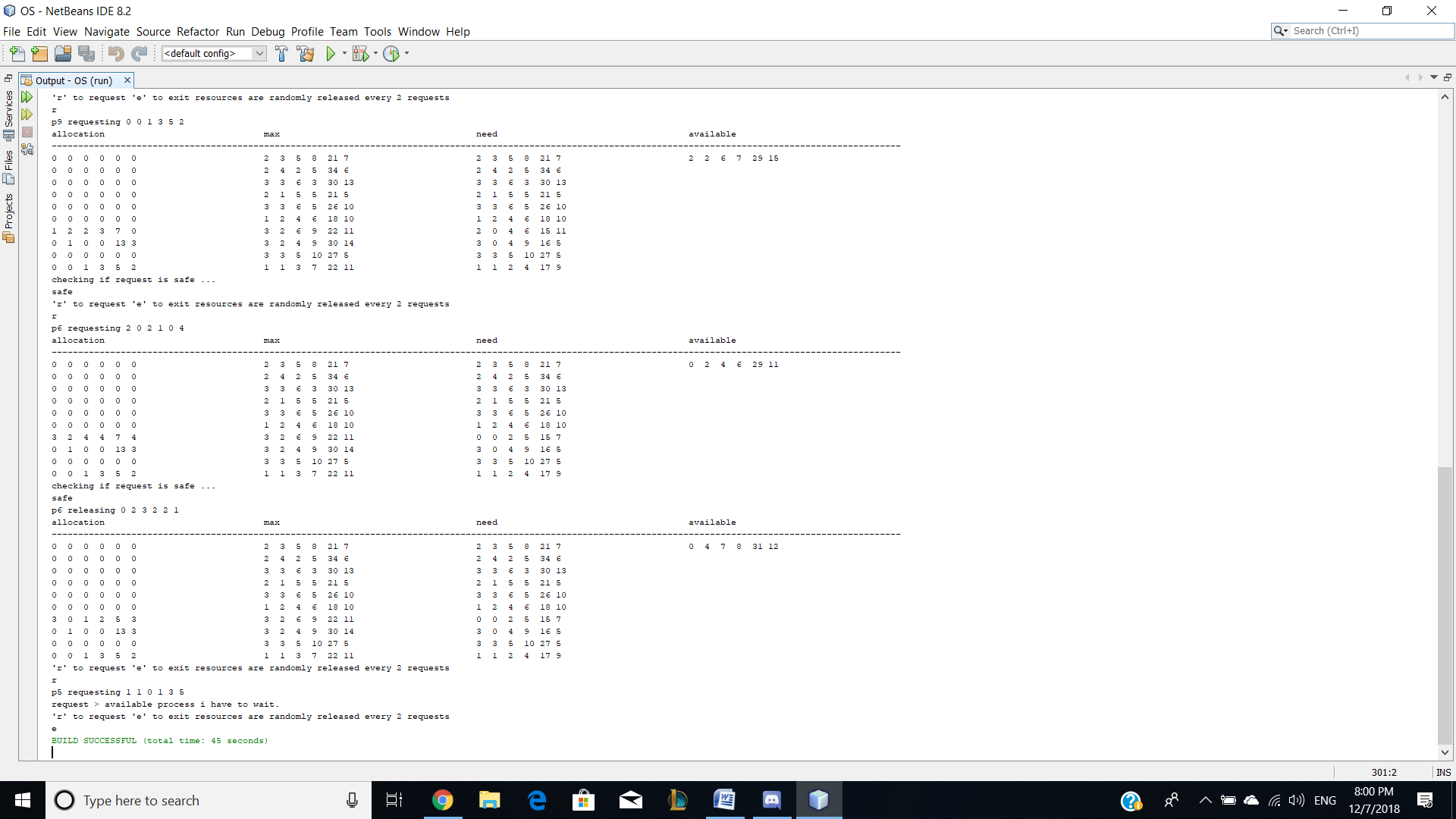
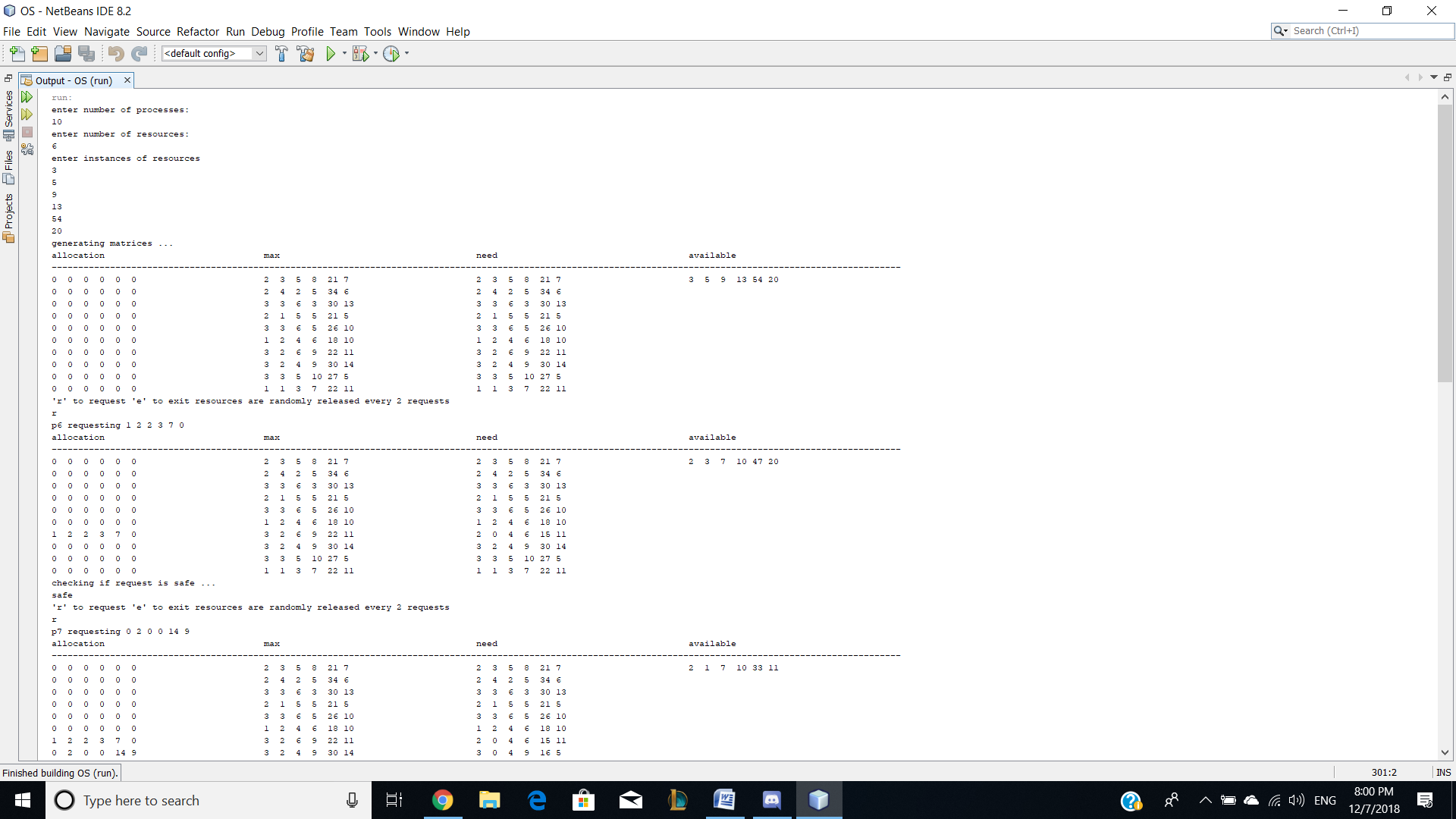


This one we have 6 processes and 4 resources with instances (90, 30, 65, and 48)



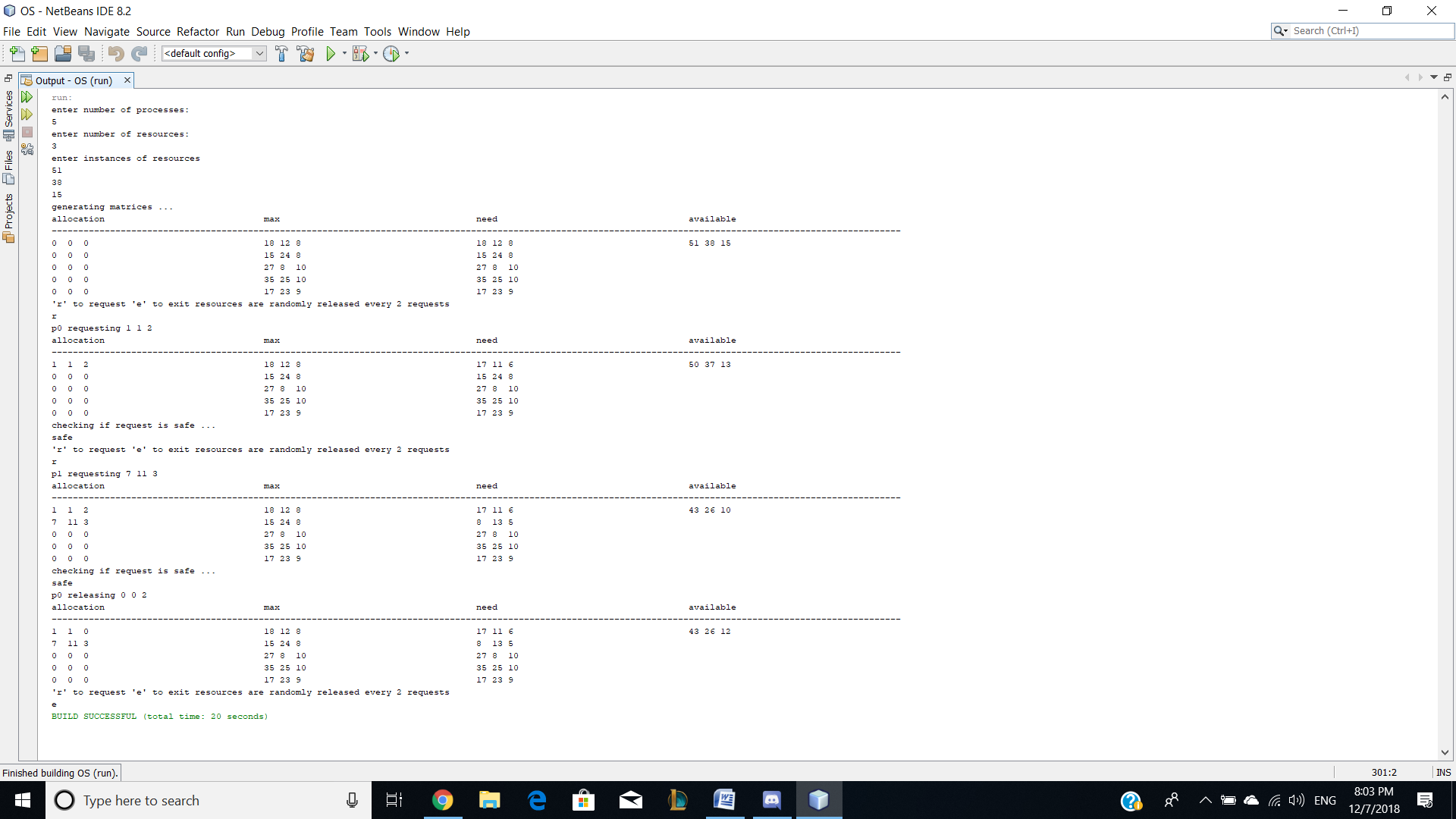
This time 8 processes and 9 resources (89, 32, 30, 63, 48, 65, 39, 91 and 76)

Notice: here we have a not safe request when p4 requested for resources.



10 processes and 6 resources (3, 5, 9, 13, 54 and 20)

Notice: we have a process (p5) that has to wait.



Last one we has processes and 3 resources (51, 38 and 15).