**Student Name:** v.sai venkat rushikesh

**Student ID:** 11801889 (59)

**Section:** K18TS

**Email Address:** [rushikeshgoud20@gmail.com](mailto:rushikeshgoud20@gmail.com)

**GitHub Link:** <https://github.com/Rushikesh-20/Rushikesh/tree/master>

Problem: 07

Researchers designed one system that classified interactive and noninteractive processes automatically by looking at the amount of terminal I/O. If a process did not input or output to the terminal in a 1-second interval, the process was classified as noninteractive and was moved to a lower-priority queue. In response to this policy, one programmer modified his programs to write an arbitrary character to the terminal at regular intervals of less than 1 second. The system gave his programs a high priority, even though the terminal output was completely meaningless.

Code :

#include<stdio.h>

int main()

{

int i, type[20],n;

int resptime[20];

printf("Number of process: ");

scanf("%d",&n);

printf("Enter the data\n");

for(i=0;i<n;i++)

{

printf("Response time of P%d (in milliseconds): ",i);

scanf("%d",&resptime[i]);

if(resptime[i]<1000)

{

type[i]=1;

}

else

{

type[i]=0;

}

}

printf("Process Number\tResponse Time\tType\t\tPriority");

for(i=0;i<n;i++)

{

printf("\nP%d\t\t%dms\t\t",i,resptime[i]);

if(type[i]==1)

{

printf("Interactive\tHigh");

}

else

{

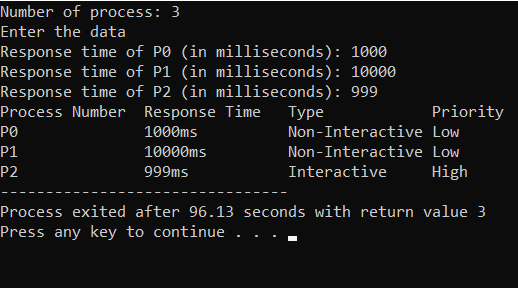
printf("Non-Interactive\tLow");

}

}

}

Output:



**Description :**

**Priority** determines where a task ranks in order relative to all the other tasks that need to be completed. Severity needs to be*considered* when setting priority, but the two are not interchangeable terms. Priority is the measure you’ll use to assign what is most important to get done now and what might be able to wait until later. Priority has the assignments of**High, Medium,**and **Low**.

**High**– An urgent problem that blocks the system use until the issue is resolved.

**Medium**– A core functionality that your product is explicitly supposed to perform is compromised.

**Low**– Should be fixed if time permits but can be postponed.

Priority is, most commonly, set initially by software testers or developers. After triaging and reporting, Product Managers or Owners can adjust priority to best suit big picture goals.

Algorithm :

Characteristics of Priority Scheduling

A CPU algorithm that schedules processes based on priority.

It used in Operating systems for performing batch processes.

If two jobs having the same priority are READY, it works on a FIRST COME, FIRST SERVED basis.

In priority scheduling, a number is assigned to each process that indicates its priority level.

Lower the number, higher is the priority.

In this type of scheduling algorithm, if a newer process arrives, that is having a higher priority than the currently running process, then the currently running process is preempted.

Example of Priority Scheduling

Consider following five processes P1 to P5. Each process has its unique priority, burst time, and arrival time.

| Process | Priority | Burst time | Arrival time |
| --- | --- | --- | --- |
| P1 | 1 | 4 | 0 |
| P2 | 2 | 3 | 0 |
| P3 | 1 | 7 | 6 |
| P4 | 3 | 4 | 11 |
| P5 | 2 | 2 | 12 |

Step 0) At time=0, Process P1 and P2 arrive. P1 has higher priority than P2. The execution begins with process P1, which has burst time 4.

Step 1) At time=1, no new process arrive. Execution continues with P1.

Step 2) At time 2, no new process arrives, so you can continue with P1. P2 is in the waiting queue.

Step 3) At time 3, no new process arrives so you can continue with P1. P2 process still in the waiting queue.

Step 4) At time 4, P1 has finished its execution. P2 starts execution.

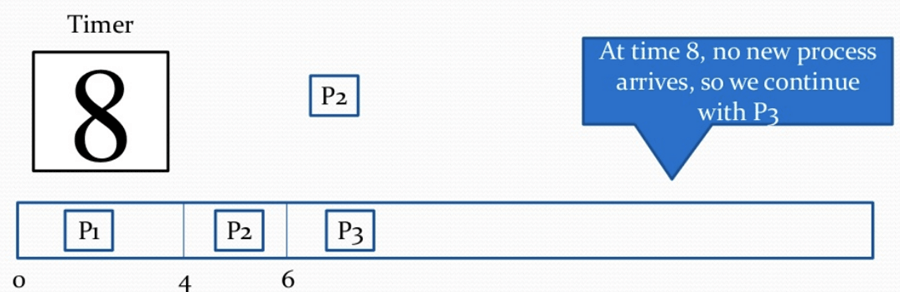
Step 5) At time= 5, no new process arrives, so we continue with P2.

Step 6) At time=6, P3 arrives. P3 is at higher priority (1) compared to P2 having priority (2). P2 is preempted, and P3 begins its execution.

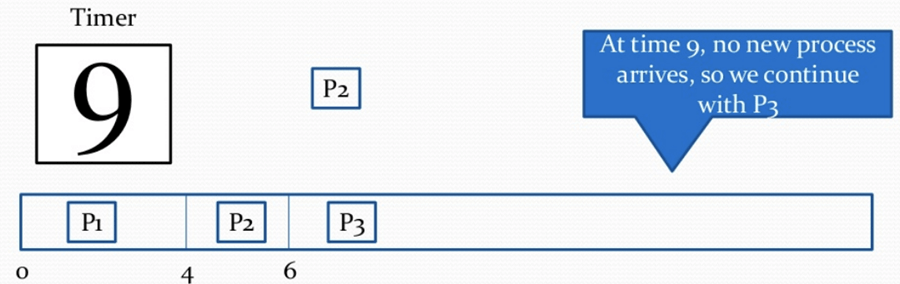
| Process | Priority | Burst time | Arrival time |
| --- | --- | --- | --- |
| P1 | 1 | 4 | 0 |
| P2 | 2 | 1 out of 3 pending | 0 |
| P3 | 1 | 7 | 6 |
| P4 | 3 | 4 | 11 |
| P5 | 2 | 2 | 12 |

Step 7) At time 7, no-new process arrives, so we continue with P3. P2 is in the waiting queue.

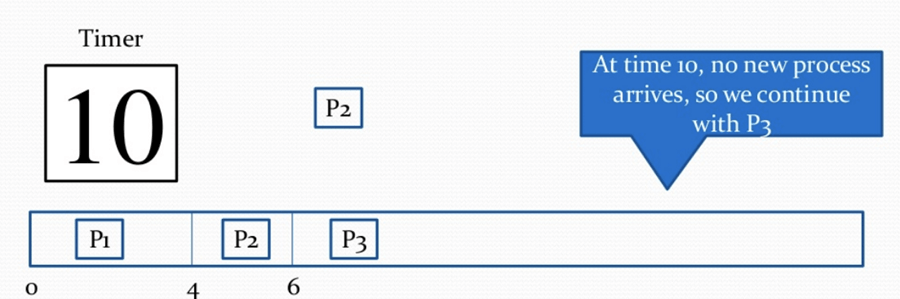
Step 8) At time= 8, no new process arrives, so we can continue with P3.

[](https://www.guru99.com/images/1/122419_0509_PrioritySch9.png)

Step 9) At time= 9, no new process comes so we can continue with P3.

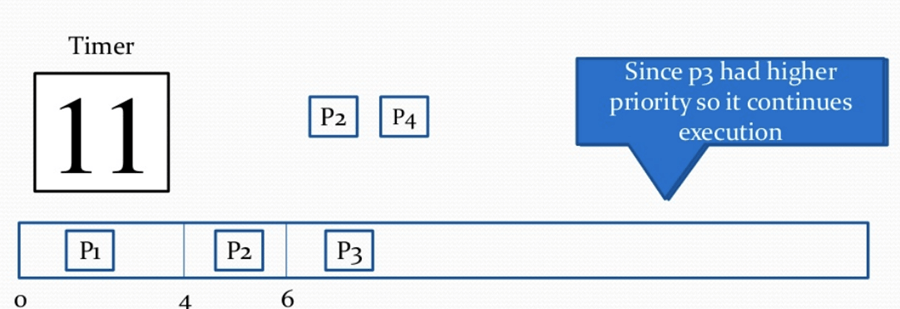
[](https://www.guru99.com/images/1/122419_0509_PrioritySch10.png)

Step 10) At time interval 10, no new process comes, so we continue with P3

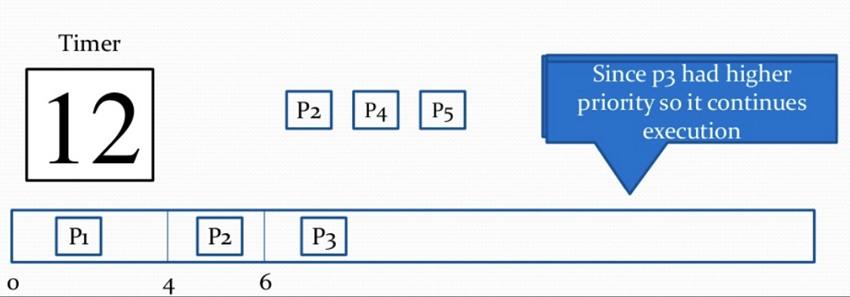
[](https://www.guru99.com/images/1/122419_0509_PrioritySch11.png)

Step 11) At time=11, P4 arrives with priority 4. P3 has higher priority, so it continues its execution.

| Process | Priority | Burst time | Arrival time |
| --- | --- | --- | --- |
| P1 | 1 | 4 | 0 |
| P2 | 2 | 1 out of 3 pending | 0 |
| P3 | 1 | 2 out of 7 pending | 6 |
| P4 | 3 | 4 | 11 |
| P5 | 2 | 2 | 12 |

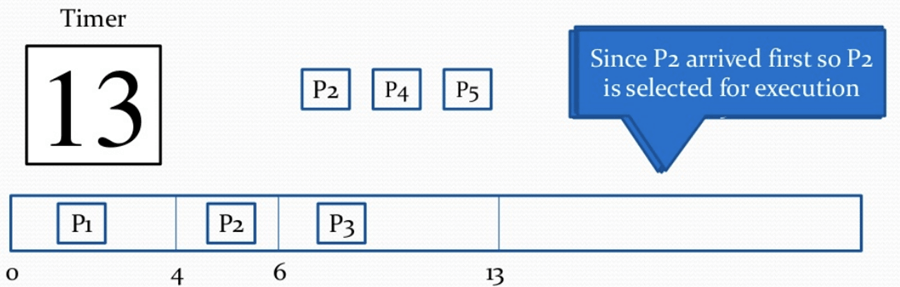
[](https://www.guru99.com/images/1/122419_0509_PrioritySch12.png)

Step 12) At time=12, P5 arrives. P3 has higher priority, so it continues execution.

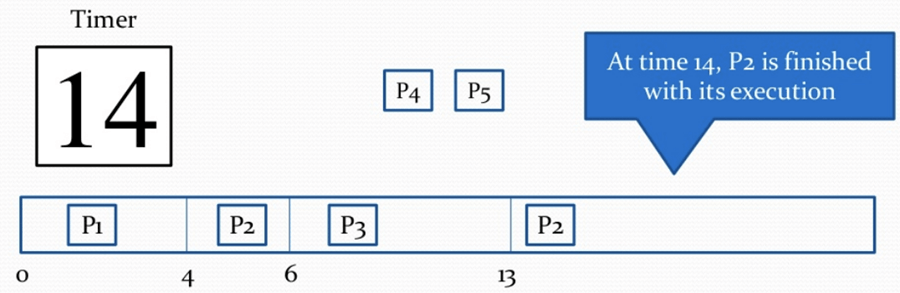
[](https://www.guru99.com/images/1/122419_0509_PrioritySch13.png)

Step 13) At time=13, P3 completes execution. We have P2,P4,P5 in ready queue. P2 and P5 have equal priority. Arrival time of P2 is before P5. So P2 starts execution.

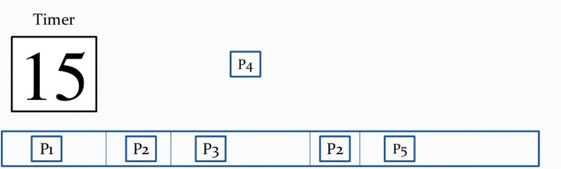
| Process | Priority | Burst time | Arrival time |
| --- | --- | --- | --- |
| P1 | 1 | 4 | 0 |
| P2 | 2 | 1 out of 3 pending | 0 |
| P3 | 1 | 7 | 6 |
| P4 | 3 | 4 | 11 |
| P5 | 2 | 2 | 12 |

[](https://www.guru99.com/images/1/122419_0509_PrioritySch14.png)

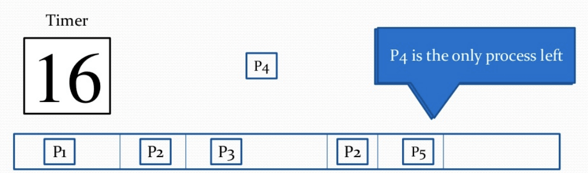
Step 14) At time =14, the P2 process has finished its execution. P4 and P5 are in the waiting state. P5 has the highest priority and starts execution.

[](https://www.guru99.com/images/1/122419_0509_PrioritySch15.png)

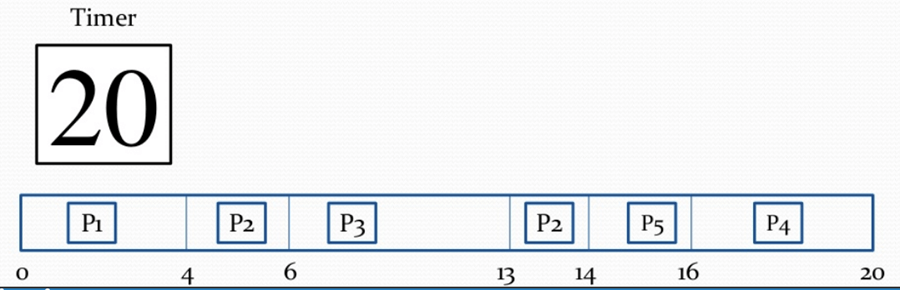
Step 15) At time =15, P5 continues execution.

[](https://www.guru99.com/images/1/122419_0509_PrioritySch16.png)

Step 16) At time= 16, P5 is finished with its execution. P4 is the only process left. It starts execution.

[](https://www.guru99.com/images/1/122419_0509_PrioritySch17.png)

Step 17) At time =20, P5 has completed execution and no process is left.

[](https://www.guru99.com/images/1/122419_0509_PrioritySch18.png)

Step 18) Let's calculate the average waiting time for the above example.

Waiting Time = start time - arrival time + wait time for next burst

P1 = o - o = o

P2 =4 - o + 7 =11

P3= 6-6=0

P4= 16-11=5

Average Waiting time = (0+11+0+5+2)/5 = 18/5= 3.6

**Purpose of use :**

**Severity:** How the bug impacts the application. How critical defect is and what is the impact of the defect on the whole system’s functionality. The severity is a parameter set by the tester while he opens a defect and is mainly in control of the tester.

**Priority:** It defines the priority in which the defects should be resolved. if there are multiple defects, the priority decides which defect has to be fixed and verified immediately versus which defect can be fixed a bit later. It is usually set by the lead

**Examples**  
**High Priority, Low Severity bug :-** If the company name is misspelled in the home page of the website,then the priority is high and severity is low to fix it.

**High Severity, Low Priority :-** Web page not found when user clicks on a link (user&#39;s does not visit that page generally)

**Low Priority, Low Severity :-** Any cosmetic or spelling issues which is within a paragraph or in the report

**High Priority, High Severity :-** An error which occurs on the basic functionality of the application and will not allow the user to use the system (E.g. user is not able to login to the application)

**Test case keep in mind :**

**Prioritizing regression test cases can be done in the following ways:**

Select test cases with frequent defects **:**

Considering the previous regression test cycle experience and learnings, we can select test cases that resulted in generating frequent defects.

Choose test cases with critical functionalities **:**

We can select the test cases that are designed to cover the critical functionality of an application. For e.g. In a mobile banking application, we have major functionalities like ‘Transfer funds’, ‘Bill payments’, ‘View service requests’. We can focus and concentrate on testing these functionalities first

.

Select test cases with frequent code changes**:**

Let’s discuss this with an example — In the mobile banking application under the ‘View service requests’ module, we have the addition of multiple service requests to the existing ones like Credit card limit increase, cheque book request, account linking request, stop cheque payment request, etc. With this example, we understand multiple times the code has been modified for this functionality. That’s the reason we need to prioritize and select the test cases covering this functionality

.

Cover end-to-end test flows**:**

Here we can cover all the end to end integration test cases in which the happy flows of a module are tested right from the beginning to completion of flow. For e.g. End to end testing for placing a fund transfer request or adding a payee under the bill payments section.

Cover field validation test cases **:**

Selection of negative test cases covering the field validations where if we miss inputting the details of a mandatory field in a form, the application displays an error message restricting the user to proceed towards the next section.

20.There are 3 student processes and 1 teacher process. Students are supposed to do their assignments and they need 3 things for that pen, paper and question paper. The teacher has an infinite supply of all the three things. One students has pen,an other has paper and another has question paper. The teacher places two things on a shared table and the student having the third complementary thing makes the assignment and tells the teacher on completion. The teacher then places another two things out of the three and again the student having the third thing makes the assignment and tells the teacher on completion. This cycle continues. WAP to synchronize the teacher and the students.

CODE:

#include<iostream>

using namespace std;

#include<stdbool.h>

class Requirement

{

public: bool pen ;

public: bool paper ;

public: bool question\_paper ;

public: bool all\_three ;

};

int main()

{

int n=3;

Requirement R[n];

R[0].pen=true;

R[0].paper = false;

R[0].question\_paper = false;

R[0].all\_three= false;

R[1].pen=false;

R[1].paper = true;

R[1].question\_paper = false;

R[1].all\_three = false;

R[2].pen=false;

R[2].paper = false;

R[2].question\_paper = true;

R[2].all\_three = false ;

while(R[0].all\_three==false||R[1].all\_three==false||R[2].all\_three==false)

{

int ch1,ch2;

cout<<"\nResources:\n1.pen\n2.paper\n3.question paper\n Enter the two things which is to be placed on the shared table: ";

cin>>ch1>>ch2;

if(ch1==1 && ch2==2 && R[2].all\_three==false)

{

R[2].all\_three=true ;

cout<<"Third Student has completed the task\n";

}

if(ch1==2 && ch2==3 && R[0].all\_three==false)

{

R[0].all\_three=true;

cout<<"First Student has completed the task\n";

}

if(ch1==1 && ch2==3 && R[1].all\_three==false)

{

R[1].all\_three=true;

cout<<"Second Student has completed the task\n";

}

}

cout<<"All the students now have completed their respective tasks succesfully\n";

return 0;

}

**Explanation:**

1. In the given solution I have taken all the student processes and resources in 2D array and initialized then to 0.
2. I have made 3 student processes in three different functions which will be executed by single s\_thread and one t\_thread for execution of teacher process.
3. User will get a menu to select any two out of three resources that are to be placed on shared table.
4. If one process is completed there will be a message printed on the screen saying process is completed.
5. When one process is executing no other student or teacher process will execute and for achieving this I have used Mutex lock.
6. When a process starts to execute it acquires the lock and when it completes the execution releases the lock.

After completion of all the three processes the program will end

OUTPUT:

# Screenshot (7).png

