
# OPERATING SYSTEMS ASSIGNMENT BASED ON PROCESS SYNCHRONISATION

**Course Code:** CSE 316

**Course Name:** OPERATING SYSTEMS

**Submitted To**: Manjit kaur mam

**Submitted by:** M.Trinath Yadav

**Registration no:** 11718564

**Roll no:** 32

**Mail:**trinathyadavmatta@gmail.com

**QUESTION:**

21. You are asked to design and implement a new synchronization primitive that will allow multiple processes to block on an event until some other process signals the event. When a process signals the event, all processes that are blocked on the event are unblocked. If no processes are blocked on an event when it is signaled, then the signal has no effect. Implement the following using C functions.

- intdoeventopen(); Creates a new event, returning event ID on success, -1 on failure.
- intdoeventclose(inteventID); Destroy the event with the given event ID and signal any processes waiting on the event to leave the event. Return number of processes signaled on success and -1 on failure.
- intdoeventwait(inteventID); Blocks process until the event is signaled. Return 1 on success and -1 on failure.
- intdoeventsig(inteventID); Unblocks all waiting processes; ignored if no processes are blocked. Return number of processes signaled on success and -1 on failure.

# SOLUTION:

## Code in C Language:

```
#include <stdio.h>

#define capacity 20

int event[capacity];

int eventblocked[capacity];

int eventId=0;

int doeventopen(){

 //creates a new event, returning eventId on success, -1 on failure

 if(eventId!=capacity)

 {

  event[eventId]=1;

  eventId++;
```

```
 return eventId;
 }
 else
 return -1;
}

int doeventclose(int eventId){
 //Destroy the event with the given eventId and signal any processes waiting on the event to leave the event. Return number of processes signalled on success and -1 on failure.
 if(event[eventId]==1)
 {
  event[eventId]=0;
  return eventId--;
 }
 else
 return -1;
}

int doeventwait(int eventId){
 //Blocks the processes until the event is signalled. Return 1 on success and -1 on failure.
 if(eventId<=capacity&&eventId>=0)
 {
  eventblocked[eventId]=1;
```

```
 return 1;
 }
 else
 return -1;
}

int doeventsig(int eventId){
 //Unblocks all waiting processes; ignored if no processes are blocked. Return number of processes signalled on success and -1 on failure.
 if(eventId<=capacity&&eventId>=0)
 {
  eventblocked[eventId]=0;
  return 1;
 }else
 return -1;
}

int main(){
int i;
 for(i=0;i<23;i++)
 {
     int k=doeventopen();
     if(k!=-1)    //Success
     {
      printf("Process created successfully with event ID : %d\n",k);
```

```
    }
    else      //failure
    {
    printf("Process failed to create\n");
    }
}

for(i=1;i<24;i++)
{
    int k=doeventclose(i);
    if(k!=-1)    //Success
    {
     printf("Process closed successfully with event ID : %d\n",k);
    }
    else      //failure
  {
    printf("Process failed to close\n");
    }
}

for(i=1;i<23;i++)
{
 int k=doeventwait(i);
 if(k!=-1)  //blocked event successfully
```

```
 {
  printf("process blocked successfully\n");
 }
 else  //failed to block the process
 printf("process failed to block\n");
}

for(i=1;i<23;i++)
{
 int k=doeventsig(i);
 if(k!=-1)  //Unblocked event successfully
 {
  printf("process unblocked successfully\n");
 }
 else  //failed to unblock the process
 printf("process failed to unblock\n");
 }

int n,m;
printf("**************************\n");
printf("Please enter your choice:\n");
printf("1. Create new event\n");
printf("2. Close the event with eventID\n");
printf("3. Block an event with eventID\n");
```

```
printf("4. Unblock an event with eventID\n\n");
printf("5. Check whether an event is closed\n");
printf("6. Check if an event is blocked\n");
printf("**************************\n\n");
scanf("%d",&n);
switch(n)
{
 case 1:
    if(doeventopen()!=1)    //Success
    {
     printf("Process created successfully\n");
    }
    else      //failure
    {
    printf("Process failed to create\n");
    }
    break;
          case 2:
    printf("Please enter eventID(1 to 20):\n");
    scanf("%d",&m);
     if(m<1||m>20){
    printf("I already said eventID should be between 1 to 20\n");break;}
    if(doeventclose(m)!=-1)    //Success
    {
```

```
     printf("Process closed successfully\n");
    }
    else      //failure
    {
    printf("Process failed to close\n");
    printf("Try opening an event before closing it!\n");
    }
    break;

     case 3:
    printf("Please enter eventID(1 to 20):\n");
    scanf("%d",&m);
    if(m<1||m>20){
    printf("I already said eventID should be between 1 to 20\n");break;}
if(doeventwait(m)!=-1)  //blocked event successfully
{
 printf("process blocked successfully\n");
 }
else  //failed to block the process
printf("process failed to block\n");
   break;

case 4:
printf("Please enter eventID(1 to 20):\n");
```

```
   scanf("%d",&m);
 if(m<1||m>20){
   printf("I already said eventID should be between 1 to 20\n");break;}
if(doeventsig(m)!=-1)  //Unblocked event successfully
{
 printf("process unblocked successfully\n");
}
else  //failed to unblock the process
printf("process failed to unblock\n");
break;

case 5:
printf("Please enter eventID(1 to 20):\n");
   scanf("%d",&m);
 if(m<1||m>20){
   printf("I already said eventID should be between 1 to 20\n");break;}
if(event[m]!=1)
{
printf("No the event is not closed!\n");
}
else
printf("Yes the event is closed\n");
break;
```

```
 case 6:
 printf("Please enter eventID(1 to 20):\n");
    scanf("%d",&m);
  if(m<1||m>20){
    printf("I already said eventID should be between 1 to 20\n");break;}
 if(eventblocked[m]!=1)
 {
  printf("No the event is not blocked!\n");
 }
 else
 printf("Yes the event is blocked!\n");
 break;

 }
 return 0;
}
```

## problem in terms of Operating system concept:

**Description:** The problem given is to implement four methods in c language. In terms of operating system, process synchronization is the one word which can be suited to describe the problem.

# algorithm for proposed solution of the assigned problem:

## Algorithms:

**Algorithm 1 (int doeventopen()) :**

Step 1: if eventId not equal to capacity

Step 2 : Then event[eventId]=1 and eventId++ and return eventId

Step 3: else return -1

**Algorithm 2 (int doeventclose(int eventId) ) :**

Step 1: if event[eventId] is equal to 1

Step 2: then event[eventId]=0 and return eventId

Step 3: else return -1

**Algorithm 3 (int doeventwait(int eventId)) :**

Step 1: if eventId is less than or equal to capacity and eventId is greater than or equal to zero

Step 2: then eventblocked[eventId]=1 and return 1

Step 3: else return-1

**Algorithm 4 (int doeventsig(int eventId)) :**

Step 1: if eventId is less than or equal to capacity and eventId is greater than or equal to zero

Step 2: then eventblocked[eventId]=0 and return 1

Step 3: else return -1

## the complexity of proposed algorithm:

For each line, the time complexity is constant or O(1) .

But for testing, since I have used for loop it is O(n).

Although it looks as less complexity but my code has space complexity.

This is mainly because I have used two arrays one is for events and the other one for flags which represent either the process is busy or free.

I thought of using linked list, but in linked list for travelling to the nth node requires n steps whereas in arrays we can directly use indexing ( for example arr[n] will get the required item)

Primary reason behind using arrays is to reduce time complexity and make the code easier to implement and debug.

## the constraints given in the problem. Attach the code snippet of implemented constraint:

**First constraint:** int doeventopen(): creates a new event, returning eventId on success, -1 on failure

```
6  int doeventopen(){
7   //creates a new event, returning eventId on success, -1 on failure
8   if(eventId!=capacity)
9   {
10   event[eventId]=1;
11   eventId++;
12   return eventId;
13  }
14  else
15  return -1;
16 }
17
```

**Code snippet:**

**Code snippet for testing:**

```
50 int main(){
51     for(int i=0;i<23;i++)
52     {
53           int k=doeventopen();
54           if(k!=-1)     //Success
55           {
56              printf("Process created successfully
                   with event ID : %d\n",k);
57           }
58           else       //failure
59           {
60           printf("Process failed to create\n");
61           }
62     }
63
```

**Code spinnet for results:**

```
Process created successfully with event ID : 1
Process created successfully with event ID : 2
Process created successfully with event ID : 3
Process created successfully with event ID : 4
Process created successfully with event ID : 5
Process created successfully with event ID : 6
Process created successfully with event ID : 7
Process created successfully with event ID : 8
Process created successfully with event ID : 9
Process created successfully with event ID : 10
Process created successfully with event ID : 11
Process created successfully with event ID : 12
Process created successfully with event ID : 13
Process created successfully with event ID : 14
Process created successfully with event ID : 15
Process created successfully with event ID : 16
Process created successfully with event ID : 17
Process created successfully with event ID : 18
Process created successfully with event ID : 19
Process created successfully with event ID : 20
Process failed to create
Process failed to create
Process failed to create
```

**Second constraint:** int doeventclose(int eventId): Destroy the event with the given eventId and signal any processes waiting on the event to leave the event. Return number of processes signalled on success and -1 on failure.

**Code Snippet:**

```
17
18 int doeventclose(int eventId){
19     //Destroy the event with the given eventId and
         signal any processes waiting on the event to
         leave the event. Return number of processes
         signalled on success and -1 on failure.
20     if(event[eventId]==1)
21     {
22         event[eventId]=0;
23         return eventId--;
24     }
25     else
26     return -1;
27 }
28
```

**Code snippet for testing:**

```
63
64     for(int i=1;i<24;i++)
65     {
66          int k=doeventclose(i);
67          if(k!=-1)    //Success
68          {
69             printf("Process closed successfully
                  with event ID : %d\n",k);
70          }
71          else      //failure
72          {
73          printf("Process failed to close\n");
74          }
75     }
76
```

**Code Snippet for results**:

```
Process closed successfully with event ID : 1
Process closed successfully with event ID : 2
Process closed successfully with event ID : 3
Process closed successfully with event ID : 4
Process closed successfully with event ID : 5
Process closed successfully with event ID : 6
Process closed successfully with event ID : 7
Process closed successfully with event ID : 8
Process closed successfully with event ID : 9
Process closed successfully with event ID : 10
Process closed successfully with event ID : 11
Process closed successfully with event ID : 12
Process closed successfully with event ID : 13
Process closed successfully with event ID : 14
Process closed successfully with event ID : 15
Process closed successfully with event ID : 16
Process closed successfully with event ID : 17
Process closed successfully with event ID : 18
Process closed successfully with event ID : 19
Process failed to close
Process failed to close
Process failed to close
Process failed to close
```

**Third constraint:** int doeventwait(int eventId) :Blocks the processes until the event is signalled. Return 1 on success and -1 on failure.

Code snippet:

```
28
29 int doeventwait(int eventId){
30     //Blocks the processes until the event is
         signalled. Return 1 on success and -1 on
         failure.
31     if(eventId<=capacity&&eventId>=0)
32     {
33         eventblocked[eventId]=1;
34         return 1;
35     }
36     else
37     return -1;
38 }
39
```

**Fourth constraint:** int doeventsig(int eventId): Unblocks all waiting processes; ignored if no processes are blocked. Return number of processes signalled on success and -1 on failure.

Code snippet:

```
39
40 int doeventsig(int eventId){
41     //Unblocks all waiting processes; ignored if
         no processes are blocked. Return number of
         processes signalled on success and -1 on
         failure.
42     if(eventId<=capacity&&eventId>=0)
43     {
44         eventblocked[eventId]=0;
45         return 1;
46     }else
47     return -1;
48 }
49
```

**Code snippet for testing:**

```
87
88     for(int i=1;i<23;i++)
89     {
90         int k=doeventsig(i);
91         if(k!=-1)  //Unblocked event successfully
92         {
93             printf("process unblocked successfully
                 \n");
94         }
95         else  //failed to unblock the process
96         printf("process failed to unblock\n");
97     }
98
```

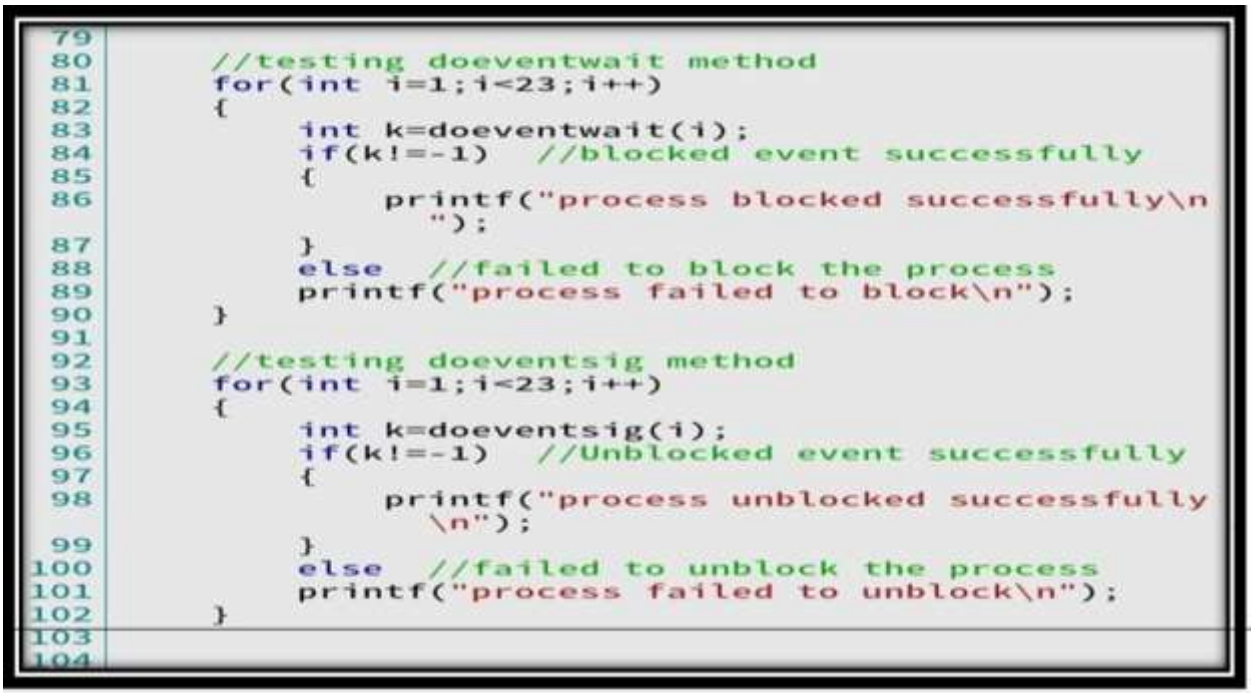
**Code snippet for results:**

```
process unblocked successfully
process unblocked successfully
process unblocked successfully
process unblocked successfully
process unblocked successfully
process unblocked successfully
process unblocked successfully
process unblocked successfully
process unblocked successfully
process unblocked successfully
process unblocked successfully
process unblocked successfully
process unblocked successfully
process unblocked successfully
process unblocked successfully
process unblocked successfully
process unblocked successfully
process unblocked successfully
process unblocked successfully
process unblocked successfully
process unblocked successfully
process failed to unblock
process failed to unblock
```

**aditional algorithm to support the solution:**

Yes, for testing purpose I have used four for loops each one to test the given c methods

```
50  int main(){
51
52      //testing doeventopen method
53      for(int i=0;i<23;i++)
54      {
55          int k=doeventopen();
56          if(k!=-1)     //Success
57          {
58              printf("Process created successfully
                    with event ID : %d\n",k);
59          }
60          else        //failure
61          {
62          printf("Process failed to create\n");
63          }
64      }
65
66      //testing doeventclose method
67      for(int i=1;i<24;i++)
68      {
69          int k=doeventclose(i);
70          if(k!=-1)     //Success
71          {
72              printf("Process closed successfully
                    with event ID : %d\n",k);
73          }
74          else        //failure
75          {
76          printf("Process failed to close\n");
77          }
78      }
79
```



```
79
80      //testing doeventwait method
81      for(int i=1;i<23;i++)
82      {
83          int k=doeventwait(i);
84          if(k!=-1)  //blocked event successfully
85          {
86              printf("process blocked successfully\n
                  ");
87          }
88          else  //failed to block the process
89          printf("process failed to block\n");
90      }
91
92      //testing doeventsig method
93      for(int i=1;i<23;i++)
94      {
95          int k=doeventsig(i);
96          if(k!=-1)  //Unblocked event successfully
97          {
98              printf("process unblocked successfully
                  \n");
99          }
100         else  //failed to unblock the process
101         printf("process failed to unblock\n");
102     }
103
104
```

**the boundary conditions of the implemented code :**

One condition is the array index may go out of bounds. This may lead to runtime error or unexpected behaviour. To prevent this before adding elements I have checked whether the index is below the capacity.