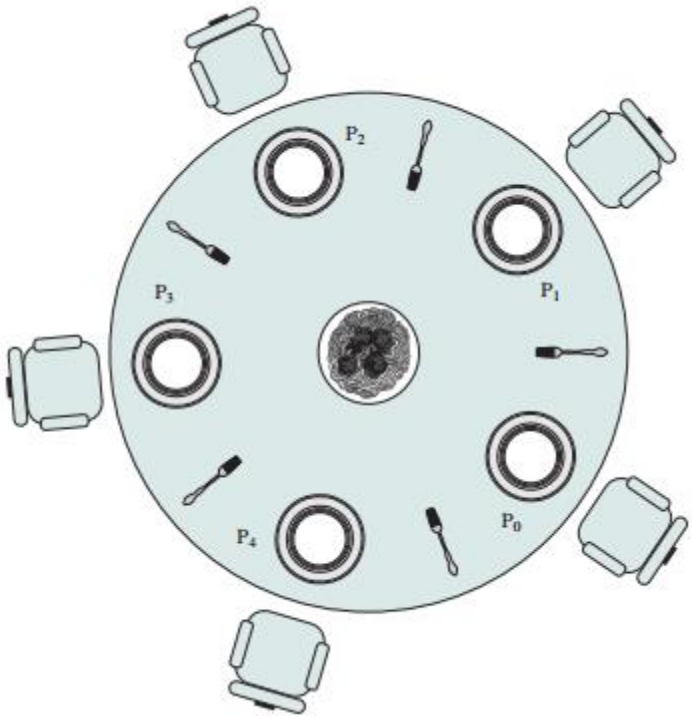
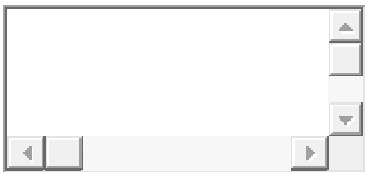
What is Dining Philosophers Problem?
There are some Philosophers whose work is just thinking and eating. Let there are 5 (for example) philosophers. They sat at a round table for dinner. To complete dinner each must need two Forks (spoons). But there are only 5 Forks available (Forks always equal to no. of Philosophers) on table. They take in such a manner that, first take left Fork and next right Fork. But problem is they try to take at same time. Since they are trying at same time, Fork 1, 2, 3, 4, 5 taken by Philosopher 1, 2, 3, 4, 5 respectively (since they are left side of each). And each one tries to ta ke right side Fork. But no one found available Fork. And also that each one thinks that someone will release the Fork and then I can eat. This continuous waiting leads to Dead Lock situation.



Dining Arrangement
Solution: To solve this Dead Lock situation, Last philosopher (any one can do this) first try to take right side fork and then left side fork. i.e in our example 5th person tries to take 4th Fork instead of 5th one. Since 4th Fork already taken by 4th the person, he gets nothing. But he left 5th Fork. Now the first person will take this 5th Fork and complete dinner and make 1st and 5th available for remaining people. Next 2nd person takes 1st fork and completes and releases 1st and 2nd. This continuous until all finishes dinner.
Operating System
In Operating System, this concept used in process synchronization. Same problem but instead of Philosophers processes are there and instead of Forks Resources are there. We follow above solution to avoid dead lock condition.

Program for Dining Philosophers Problem in C



```
1  #include<stdio.h>
2
3  #define n 4
4
5  int compltedPhilo = 0,i;
6
7  struct fork{
8              int taken;
9  }ForkAvil[n];
10
11 struct philosp{
12             int left;
13             int right;
14 }Philostatus[n];
15
16 void goForDinner(int philID){ //same like threads concept here cases implemented
17             if(Philostatus[philID].left==10 && Philostatus[philID].right==10)
18         printf("Philosopher %d completed his dinner\n",philID+1);
19             //if already completed dinner
20             else if(Philostatus[philID].left==1 && Philostatus[philID].right==1){
21            //if just taken two forks
22            printf("Philosopher %d completed his dinner\n",philID+1);
23
```

```
24          Philostatus[philID].left = Philostatus[philID].right = 10; //remembering that he completed dinner by
25 assigning value 10
26          int otherFork = philID-1;
27
28          if(otherFork== -1)
29              otherFork=(n-1);
30
31          ForkAvil[philID].taken = ForkAvil[otherFork].taken = 0; //releasing forks
32          printf("Philosopher %d released fork %d and fork %d\n",philID+1,philID+1,otherFork+1);
33          compltedPhilo++;
34      }
35      else if(Philostatus[philID].left==1 && Philostatus[philID].right==0){ //left already taken, trying for right
36 fork
37              if(philID==(n-1)){
38                  if(ForkAvil[philID].taken==0){ //KEY POINT OF THIS PROBLEM, THAT LAST
39 PHILOSOPHER TRYING IN reverse DIRECTION
40                      ForkAvil[philID].taken = Philostatus[philID].right = 1;
41                      printf("Fork %d taken by philosopher %d\n",philID+1,philID+1);
42                  }else{
43                      printf("Philosopher %d is waiting for fork %d\n",philID+1,philID+1);
44                  }
45              }else{ //except last philosopher case
46                  int dupphilID = philID;
47                  philID-=1;
48
49                  if(philID== -1)
```

```
50                  philID=(n-1);
51
52              if(ForkAvil[philID].taken == 0){
53                  ForkAvil[philID].taken = Philostatus[dupphilID].right = 1;
54                  printf("Fork %d taken by Philosopher %d\n",philID+1,dupphilID+1);
55              }else{
56                  printf("Philosopher %d is waiting for Fork %d\n",dupphilID+1,philID+1);
57              }
58            }
59          }
60          else if(Philostatus[philID].left==0){ //nothing taken yet
61              if(philID==(n-1)){
62                  if(ForkAvil[philID-1].taken==0){ //KEY POINT OF THIS PROBLEM, THAT LAST
63 PHILOSOPHER TRYING IN reverse DIRECTION
64                      ForkAvil[philID-1].taken = Philostatus[philID].left = 1;
65                      printf("Fork %d taken by philosopher %d\n",philID,philID+1);
66                  }else{
67                      printf("Philosopher %d is waiting for fork %d\n",philID+1,philID);
68                  }
69              }else{ //except last philosopher case
70                  if(ForkAvil[philID].taken == 0){
71                      ForkAvil[philID].taken = Philostatus[philID].left = 1;
72                      printf("Fork %d taken by Philosopher %d\n",philID+1,philID+1);
73                  }else{
74                      printf("Philosopher %d is waiting for Fork %d\n",philID+1,philID+1);
75                  }
```

```
76                  }
77          }else{}
78 }
79
80 int main(){
81              for(i=0;i<n;i++)
82          ForkAvil[i].taken=Philostatus[i].left=Philostatus[i].right=0;
83
84              while(compltedPhilo<n){
85                      /* Observe here carefully, while loop will run until all philosophers complete dinner
86                      Actually problem of deadlock occur only thy try to take at same time
87                      This for loop will say that they are trying at same time. And remaining status will print by go
88 for dinner function
89                      */
90                      for(i=0;i<n;i++)
91              goForDinner(i);
                        printf("\nTill now num of philosophers completed dinner are %d\n\n",compltedPhilo);
                }


              return 0;
    }
```

Output

Fork 1 taken by Philosopher 1
Fork 2 taken by Philosopher 2
Fork 3 taken by Philosopher 3
Philosopher 4 is waiting for fork 3
Till now num of philosophers completed dinner are 0
Fork 4 taken by Philosopher 1
Philosopher 2 is waiting for Fork 1
Philosopher 3 is waiting for Fork 2
Philosopher 4 is waiting for fork 3

Till now num of philosophers completed dinner are 0
Philosopher 1 completed his dinner
Philosopher 1 completed his dinner
Philosopher 1 released fork 1 and fork 4
Fork 1 taken by Philosopher 2
Philosopher 3 is waiting for Fork 2
Philosopher 4 is waiting for fork 3
Till now num of philosophers completed dinner are 1
Philosopher 1 completed his dinner
Philosopher 2 completed his dinner
Philosopher 2 released fork 2 and fork 1
Fork 2 taken by Philosopher 3
Philosopher 4 is waiting for fork 3
Till now num of philosophers completed dinner are 2
Philosopher 1 completed his dinner
Philosopher 2 completed his dinner
Philosopher 3 completed his dinner
Philosopher 3 released fork 3 and fork 2
Fork 3 taken by philosopher 4
Till now num of philosophers completed dinner are 3
Philosopher 1 completed his dinner
Philosopher 2 completed his dinner
Philosopher 3 completed his dinner
Fork 4 taken by philosopher 4
Till now num of philosophers completed dinner are 3
Philosopher 1 completed his dinner
Philosopher 2 completed his dinner
Philosopher 3 completed his dinner
Philosopher 4 completed his dinner
Philosopher 4 released fork 4 and fork 3
Till now num of philosophers completed dinner are 4

Producer Consumer Problem in C

Here you will learn about producer consumer problem in C.
Producer consumer problem is also known as bounded buffer problem. In this problem we have two processes, producer and consumer, who share a fixed size buffer. Producer work is to produce data or items and put in buffer. Consumer work is to remove data from buffer and consume it. We have to make sure that producer do not produce data when buffer is full and consumer do not remove data when buffer is empty.

The producer should go to sleep when buffer is full. Next time when consumer removes data it notifies the producer and producer starts producing data again. The consumer should go to sleep when buffer is empty. Next time when producer add data it notifies the consumer and consumer starts consuming data. This solution can be achieved using semaphores.

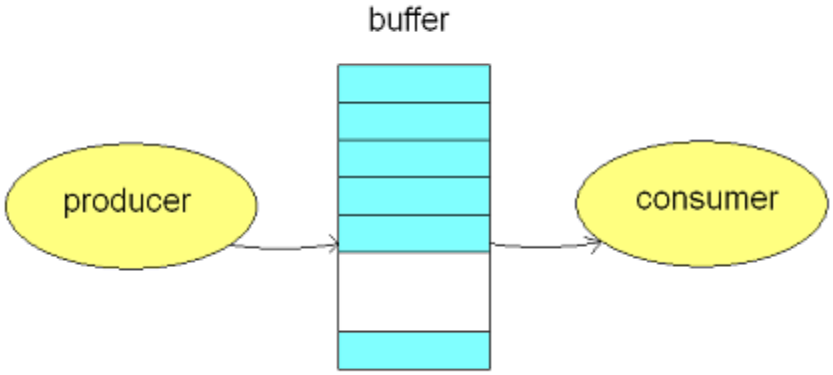


Image Source

Producer Consumer Problem in C

Below is the program to implement this problem.

```
1  #include<stdio.h>
2  #include<stdlib.h>
3
4  int mutex=1,full=0,empty=3,x=0;
5
6  int main()
7  {
8          int n;
9          void producer();
10         void consumer();
11         int wait(int);
12         int signal(int);
```

```
13          printf("\n1.Producer\n2.Consumer\n3.Exit");
14          while(1)
15          {
16                  printf("\nEnter your choice:");
17                  scanf("%d",&n);
18                  switch(n)
19                  {
20                          case 1:  if((mutex==1)&&(empty!=0))
21                                                  producer();
22                                          else
23                                                  printf("Buffer is full!!");
24                                          break;
25                          case 2:  if((mutex==1)&&(full!=0))
26                                                  consumer();
27                                          else
28                                                  printf("Buffer is empty!!");
29                                          break;
30                          case 3:
31                                          exit(0);
32                                          break;
33                  }
34          }
35
36          return 0;
37 }
38
```

```
39 int wait(int s)
40 {
41         return (--s);
42 }
43
44 int signal(int s)
45 {
46         return(++s);
47 }
48
49 void producer()
50 {
51         mutex=wait(mutex);
52         full=signal(full);
53         empty=wait(empty);
54         x++;
55         printf("\nProducer produces the item %d",x);
56         mutex=signal(mutex);
57 }
58
59 void consumer()
60 {
61         mutex=wait(mutex);
62         full=wait(full);
63         empty=signal(empty);
64         printf("\nConsumer consumes item %d",x);
```

```
65          x--;
66          mutex=signal(mutex);
67 }
```

Output

1.Producer
2.Consumer
3.Exit
Enter your choice:1
Producer produces the item 1
Enter your choice:2
Consumer consumes item 1
Enter your choice:2
Buffer is empty!!
Enter your choice:1
Producer produces the item 1
Enter your choice:1
Producer produces the item 2
Enter your choice:1
Producer produces the item 3
Enter your choice:1
Buffer is full!!
Enter your choice:3