

OPENMP – SYNCHRONIZATION

LAB 2

Aim: To understand and implement the producer consumer problem using various synchronization constructs (like nowait, single, barrier and critical).

CODE AND OUTPUTS

1. PRODUCER CONSUMER PROBLEM

```
1 #include <stdio.h>
2 #include <unistd.h>
3 #include <omp.h>
4 #include <time.h>
5
6 #define SIZE 4
7 #define NUM 26
8
9 char buffer[SIZE];
10 int in = 0;
11 int out = 0;
12 int count = 0;
13 int empty = 1;
14 int full = 0;
15 int i,j;
16 void put(char item)
17 {
18     buffer[in] = item;
19     in = (in + 1) % SIZE;
20     count++;
21     if (count == SIZE)
22         full = 1;
23     if (count == 1)
24         empty = 0;
25 }
26
27 void producer(int tid)
28 {
29     char item;
30     while( i < NUM)
31     {
32         #pragma omp critical(CRIT)
33         {
34             item = 'A' + (i % 26);
35             put(item);
36             i++;
37             printf("Thread: %d\tProducing\t%c\n",tid, item);
38         }
39         sleep(1);
40     }
41 }
```

```

38     }
39     sleep(1);
40 }
41 }
42
43 char get()
44 {
45     char item;
46     item = buffer[out];
47     out = (out + 1) % SIZE;
48     count--;
49     if (count == 0)
50         empty = 1;
51     if (count == (SIZE-1))
52         full = 0;
53     return item;
54 }
55
56 void consumer(int tid)
57 {
58     char item;
59     while(j < NUM)
60     {
61         #pragma omp critical(CRIT)
62         {
63             j++;
64             item = get();
65             printf("Thread: %d\t Consuming\t%c\n",tid, item);
66         }
67         sleep(1);
68     }
69 }
70
71 int main()
72 {
73     int tid;
74     i=j=0;
75     #pragma omp parallel firstprivate(i,j) private(tid)
76     {
77         tid=omp_get_thread_num();
78         if(tid%2==1)
79         {
80             producer(tid);
81         }
82         else
83         {
84             consumer(tid);
85         }
86     }
87 }

```

OUTPUT

```

dhruv@dhruv-Inspiron-5559: ~/PDC-lab/Lab2
Thread: 3      Producing      E
Thread: 1      Producing      F
Thread: 2      Consuming      E
Thread: 0      Consuming      F
Thread: 3      Producing      G
Thread: 1      Producing      H
Thread: 2      Consuming      G
Thread: 0      Consuming      H
Thread: 3      Producing      I
Thread: 1      Producing      J
Thread: 2      Consuming      I
Thread: 0      Consuming      J
Thread: 1      Producing      K
Thread: 2      Consuming      K
Thread: 3      Producing      L
Thread: 0      Consuming      L
Thread: 1      Producing      M
Thread: 3      Producing      N
Thread: 2      Consuming      M
Thread: 0      Consuming      N
Thread: 1      Producing      O
Thread: 3      Producing      P
Thread: 0      Consuming      O
Thread: 2      Consuming      P

```

RESULT

Thus, the producer consumer problem was successfully implemented using the synchronization constructs in OpenMP.