**NAME: SHASHWAT SHAH**

**SAP ID : 60004220126**

**DIV: B3**

**EXPERIMENT 3 : Implement the Producer Consumer Problem**

**OPERATING SYSTEMS – EXPERIMENT 5**

**AIM**:

To implement producer consumer problem.

**THEORY:**

There are one or more producers generating some type of data (records, characters) and

placing the data in a buffer.

There is a single consumer that is taking items out of the buffer one at a time.

The system is to be constrained to prevent the overlap of buffer operations. That is, only

one agent (producer or consumer) may access the buffer at any one time.

The problem is to make sure that the producer won’t try to add data into the buffer if it’s

full and that the consumer won’t try to remove data from an empty buffer.

The solution to this problem is that if buffer is empty, the consumer goes to sleep until the

producer puts some data in the buffer. Similarly, the producer is put to sleep when the

buffer is full and is not woken up until the consumer removes data from the buffer.

The solution to producer consumer problem uses three semaphores, for showing if buffer is

full, for showing if buffer is empty and for mutual exclusion, i.e., for accessing the critical

section(buffer).

**CODE:**

// Shashwat Shah 60004220126

#include <stdio.h>

#include <stdlib.h>

int mutex = 1;

int products = 0;

int emptyspace = 5, x = 0;

int Wait(int n)

{

--n;

return n;

}

int Signal(int n)

{

++n;

return n;

}

void producer()

{

mutex = Wait(mutex);

products = Signal (products);

emptyspace = Wait (empty space);

x = Signal(x);

printf("Producer produces item : %d\n",x);

mutex = Signal(mutex);

}

void consumer()

{

mutex = Wait (mutex);

products = Wait (products);

emptyspace = Signal (emptyspace);

printf("Consumer consumes item : %d\n", x);

X =Wait(x);

mutex = Signal(mutex);

}

int main()

int n, i=1;

printf("\n1. Press 1 for Producer\n2. Press 2 for Consumer\n3. Press 3 for Exit");

while(i>0)

{

printf("\nEnter your choice: ");

scanf("%d", &n);

switch (n)

{

case 1:

if ((mutex == 1) && (emptyspace != 0))

producer();

else

printf("Buffer is Full\nProducer cannot produce more items\n");

break;

case 2:

if((mutex == 1) && (products != 0))

consumer();

else

printf(“Buffer is empty”);

break;

}

}

}

**Conclusion :** Hereby, we have implemented the above CPU scheduling algorithms.