**Program 3**

**AIM:** To Write a C program to simulate producer-consumer problem using semaphores.

**DESCRIPTION**

Producer consumer problem is a synchronization problem. There is a fixed size buffer where the producer produces items and that is consumed by a consumer process. One solution to the producer- consumer problem uses shared memory. To allow producer and consumer processes to run concurrently, there must be available a buffer of items that can be filled by the producer and emptied by the consumer. This buffer will reside in a region of memory that is shared by the producer and consumer processes. The producer and consumer must be synchronized, so that the consumer does not try to consume an item that has not yet been produced.

#include <stdio.h>

int buffer[10], bufsize = 10, in = 0, out = 0;

void produce()

{

int produce;

if ((in + 1) % bufsize == out)

{

printf("\nBuffer is Full");

return;

}

printf("\nEnter the value: ");

scanf("%d", &produce);

buffer[in] = produce;

in = (in + 1) % bufsize;

}

void consume()

{

int consume;

if (in == out) {

printf("\nBuffer is Empty");

return;

}

consume = buffer[out];

printf("\nThe consumed value is %d", consume);

out = (out + 1) % bufsize;

}

int main()

{

int choice = 0;

while (choice != 3)

{

printf("\n1. Produce\t2. Consume\t3. Exit");

printf("\nEnter your choice: ");

scanf("%d", &choice);

switch (choice)

{

case 1:

produce();

break;

case 2:

consume();

break;

case 3:

break;

default:

printf("\nInvalid choice! Enter again.");

}

}

return 0;

}

# PROGRAM 4

**AIM:** To Write a C program to simulate the concept of Dining-Philosophers problem.

# DESCRIPTION

The dining-philosophers problem is considered a classic synchronization problem because it is an example of a large class of concurrency-control problems. It is a simple representation of the need to allocate several resources among several processes in a deadlock-free and starvation-free manner. Consider five philosophers who spend their lives thinking and eating. The philosophers share a circular table surrounded by five chairs, each belonging to one philosopher. In the center of the table is a bowl of rice, and the table is laid with five single chopsticks. When a philosopher thinks, she does not interact with her colleagues. From time to time, a philosopher gets hungry and tries to pick up the two chopsticks that are closest to her (the chopsticks that are between her and her left and right neighbors). A philosopher may pick up only one chopstick at a time. Obviously, she cam1ot pick up a chopstick that is already in the hand of a neighbor. When a hungry philosopher has both her chopsticks at the same time, she eats without releasing her chopsticks. When she is finished eating, she puts down both of her chopsticks and starts thinking again. The dining-philosophers problem may lead to a deadlock situation and hence some rules have to be framed to avoid the occurrence of deadlock.

#include <stdio.h>

#include <stdlib.h>

int tph, philname[20], status[20], howhung, hu[20], cho;

void one()

{

// Implement logic for allowing one philosopher to eat at a time

printf("\nAllow one philosopher to eat at any time\n");

for (int pos = 0; pos < howhung; pos++)

{

printf("\nP %d is granted to eat", philname[hu[pos]]);

for (int x = pos + 1; x < howhung; x++)

{

printf("\nP %d is waiting", philname[hu[x]]);

}

}

}

void two()

{

// Implement logic for allowing two philosophers to eat at the same time

printf("\nAllow two philosophers to eat at the same time\n");

int s = 0;

for (int i = 0; i < howhung; i++)

{

for (int j = i + 1; j < howhung; j++)

{

if (abs(hu[i] - hu[j]) >= 1 && abs(hu[i] - hu[j]) != 4)

{

printf("\n\nCombination %d\n", (s + 1));

int t = hu[i];

int r = hu[j];

s++;

printf("P %d and P %d are granted to eat\n", philname[hu[i]],philname[hu[j]]);

for (int x = 0; x < howhung; x++)

{

if ((hu[x] != t) && (hu[x] != r))

{

printf("P %d is waiting\n", philname[hu[x]]);

}

}

}

}

}

}

int main() {

int i;

printf("\nDINING PHILOSOPHER PROBLEM\n");

printf("Enter the total number of philosophers: ");

scanf("%d", &tph);

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

{

philname[i] = (i + 1);

status[i] = 1;

}

printf("How many are hungry: ");

scanf("%d", &howhung);

if (howhung == tph)

{

printf("\nAll are hungry..\nDeadlock stage will occur\nExiting\n");

return 0;

} else {

for (i = 0; i < howhung; i++) {

printf("Enter philosopher %d position: ", (i + 1));

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

status[hu[i]] = 2;

}

do {

printf("\n1. One can eat at a time\n2. Two can eat at a time\n3. Exit\n");

printf("Enter your choice: ");

scanf("%d", &cho);

switch (cho)

{

case 1:

one();

break;

case 2:

two();

break;

case 3:

exit(0);

break;

default:

printf("\nInvalid option..\n");

break;

}

} while (1);

}

return 0;

}

**OUTPUT**

DINING PHILOSOPHER PROBLEM

Enter the total number of philosophers: 5

How many are hungry: 3

Enter philosopher 1 position: 2

Enter philosopher 2 position: 4

Enter philosopher 3 position: 5

1. One can eat at a time

2. Two can eat at a time

3. Exit

Enter your choice: 1

Allow one philosopher to eat at any time

P 3 is granted to eat

P 5 is waiting

P 0 is waiting

P 5 is granted to eat

P 0 is waiting

P 0 is granted to eat

1. One can eat at a time

2. Two can eat at a time

3. Exit

Enter your choice: 2

Allow two philosophers to eat at the same time

Combination 1

P 3 and P 5 are granted to eat

P 0 is waiting

Combination 2

P 3 and P 0 are granted to eat

P 5 is waiting

Combination 3

P 5 and P 0 are granted to eat

P 3 is waiting

1. One can eat at a time

2. Two can eat at a time

3. Exit

Enter your choice: