Write a c program to simulate
Producer consumer problem using semaphore
Concept of dining philosopher problem

a

Write a C program to simulate producer-consumer problem using semaphores.

```
#include <stdio.h>
#include <stdlib.h>
int mutex=1, full=0, empty=3, x=0;
int main()
{ int n;
  void producer();
  void consumer();
  int wait(int);
  int signal(int);
printf("\n1. Producer \n2. Consumer \n3. Exit");
  while(1)
{ printf("\n Enter your choice:");
  scanf("%d", &n);
switch(n)
{ case1: if((mutex==1)&&(empty!=0))
            producer();
         else
            printf("Buffer is full!!");
         break;
  case2: if((mutex==1)&&(full!=0))
            consumer();
         else
            printf("Buffer is empty!!");
         break;
  case3:
         exit(0);
         break;
    }
}
```

```
return 0;
}
int wait (int s)
{ return (--s);
}
int signal (int s)
{ return (++s);
}
void producer()
{ mutex = wait (mutex);
  full = signal (full);
  empty = wait (empty);
  x++;
printf ("\n Producer produces the item %d", x);
  mutex = signal (mutex);
}
void consumer()
{ mutex = wait (mutex);
  full = wait (full);
  empty = signal (empty);
printf ("\n Consumer consumes item %d", x);
  x--;
mutex = signal (mutex);
}
```

Output

1. Producer
2. Consumer
3. Exit

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: 2
Consumer consumes item 2
Enter your choice: 2
Consumer consumes item 1
Enter your choice: 3

/* c program to implement dining philosophers

#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#define NUM_PHIL 5
#define NUM_CHOP 5

void dine (int n);
pthread_t phil [NUM_PHIL];
pthread_mutex_t chop [NUM_CHOP];

int main ()
{
int i, status;
void *msg;
for (i=1; i<NUM_CHOP; i++)
{
status = pthread_mutex_init (&chop[i], NULL);
if (status == -1)
{ printf ("mutex failed!\n");
exit (1);
}
}
for (i=1; i<= NUM_PHIL; i++)
{
status = pthread_create (&phil[i], NULL,
(void*) dine, (int*) i);
if (status != 0)
printf ("Thread creation error\n");
exit (1);
}
for (i=1; i<NUM_PHIL; i++)
{
status = pthread_join (phil[i], &msg);
if (status != 0)
{ printf ("Thread join failed\n");

```
exit (1);
  }
}
for (i=1; i<NUM_CHOP; i++)
  { status = pthread_mutex_ destroy
    if (status != 0)
    { printf ("mutex destroyed!")
      exit (1);
    }
  }
  return 0;
}
void dine (int n)
{ printf ("\n Philosopher %d [thi
  pthread_mutex_lock [4 chop
    pthread_mutex_lock [4 chops

printf ("Philosopher %d thinking",
```

philosophum 1 is eating
philosophum 3 is eating
philosophum 2 finished eating
philosophum 4 finished eating
philosophum 5 finished eating

(& chop[1]).

g M', M).
ch [M]).
[M+1) % NUM_
CHOP]].

M]).
(M+1) %
M-CHOP]).
, M).

```
1.Producer
2.Consumer
3.Exit
Enter your choice:1

Producer produces the item 1
Enter your choice:1

Producer produces the item 2
Enter your choice:2

Consumer consumes item 2
Enter your choice:2

Consumer consumes item 1
Enter your choice:
```

```
Philosopher 1 is thinking
Philosopher 2 is thinking
Philosopher 3 is thinking
Philosopher 4 is thinking
Philosopher 5 is thinking
Philosopher 1 is Hungry
Philosopher 2 is Hungry
Philosopher 3 is Hungry
Philosopher 4 is Hungry
Philosopher 5 is Hungry
Philosopher 5 takes fork 4 and 5
Philosopher 5 is Eating
Philosopher 5 putting fork 4 and 5 down
Philosopher 5 is thinking
Philosopher 4 takes fork 3 and 4
Philosopher 4 is Eating
Philosopher 1 takes fork 5 and 1
Philosopher 1 is Eating
Philosopher 4 putting fork 3 and 4 down
Philosopher 4 is thinking
Philosopher 3 takes fork 2 and 3
Philosopher 3 is Eating
Philosopher 5 is Hungry
Philosopher 1 putting fork 5 and 1 down
Philosopher 1 is thinking
Philosopher 5 takes fork 4 and 5
Philosopher 5 is Eating
Philosopher 4 is Hungry
Philosopher 3 putting fork 2 and 3 down
```