**WEEK 6**

**Zayd Ahmed**

**1BM21CS254**

**19-07-2023**

**Q: 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\n");

while (1)

{

printf("\nEnter your choice:");

scanf("%d", &n);

switch (n)

{

case 1:

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

producer();

else

printf("buffer is full\n");

break;

case 2:

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

consumer();

else

printf("buffer is empty\n");

break;

case 3:

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("\Producer produces the item %d", x);

mutex = signal(mutex);

}

void consumer()

{

mutex= wait(mutex);

full = wait(full);

empty = signal(empty);

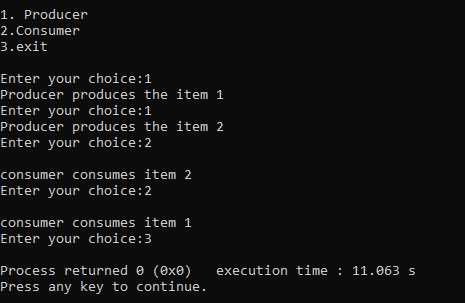
printf("\nconsumer consumes item %d", x);

x--;

mutex = signal(mutex);

}

**OUTPUT:**

****

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

#include <pthread.h>

#include <semaphore.h>

#include <stdio.h>

#define N 5

#define THINKING 2

#define HUNGRY 1

#define EATING 0

#define LEFT (phnum + 4) % N

#define RIGHT (phnum + 1) % N

int state[N];

int phil[N] = { 0, 1, 2, 3, 4 };

sem\_t mutex;

sem\_t S[N];

void test(int phnum)

{

if (state[phnum] == HUNGRY

&& state[LEFT] != EATING

&& state[RIGHT] != EATING) {

// state that eating

state[phnum] = EATING;

sleep(2);

printf("Philosopher %d takes fork %d and %d\n",

phnum + 1, LEFT + 1, phnum + 1);

printf("Philosopher %d is Eating\n", phnum + 1);

// sem\_post(&S[phnum]) has no effect

// during takefork

// used to wake up hungry philosophers

// during putfork

sem\_post(&S[phnum]);

}

}

// take up chopsticks

void take\_fork(int phnum)

{

sem\_wait(&mutex);

// state that hungry

state[phnum] = HUNGRY;

printf("Philosopher %d is Hungry\n", phnum + 1);

// eat if neighbours are not eating

test(phnum);

sem\_post(&mutex);

// if unable to eat wait to be signalled

sem\_wait(&S[phnum]);

sleep(1);

}

// put down chopsticks

void put\_fork(int phnum)

{

sem\_wait(&mutex);

// state that thinking

state[phnum] = THINKING;

printf("Philosopher %d putting fork %d and %d down\n",

phnum + 1, LEFT + 1, phnum + 1);

printf("Philosopher %d is thinking\n", phnum + 1);

test(LEFT);

test(RIGHT);

sem\_post(&mutex);

}

void\* philosopher(void\* num)

{

while (1) {

int\* i = num;

sleep(1);

take\_fork(\*i);

sleep(0);

put\_fork(\*i);

}

}

int main()

{

int i;

pthread\_t thread\_id[N];

// initialize the semaphores

sem\_init(&mutex, 0, 1);

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

sem\_init(&S[i], 0, 0);

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

// create philosopher processes

pthread\_create(&thread\_id[i], NULL,

philosopher, &phil[i]);

printf("Philosopher %d is thinking\n", i + 1);

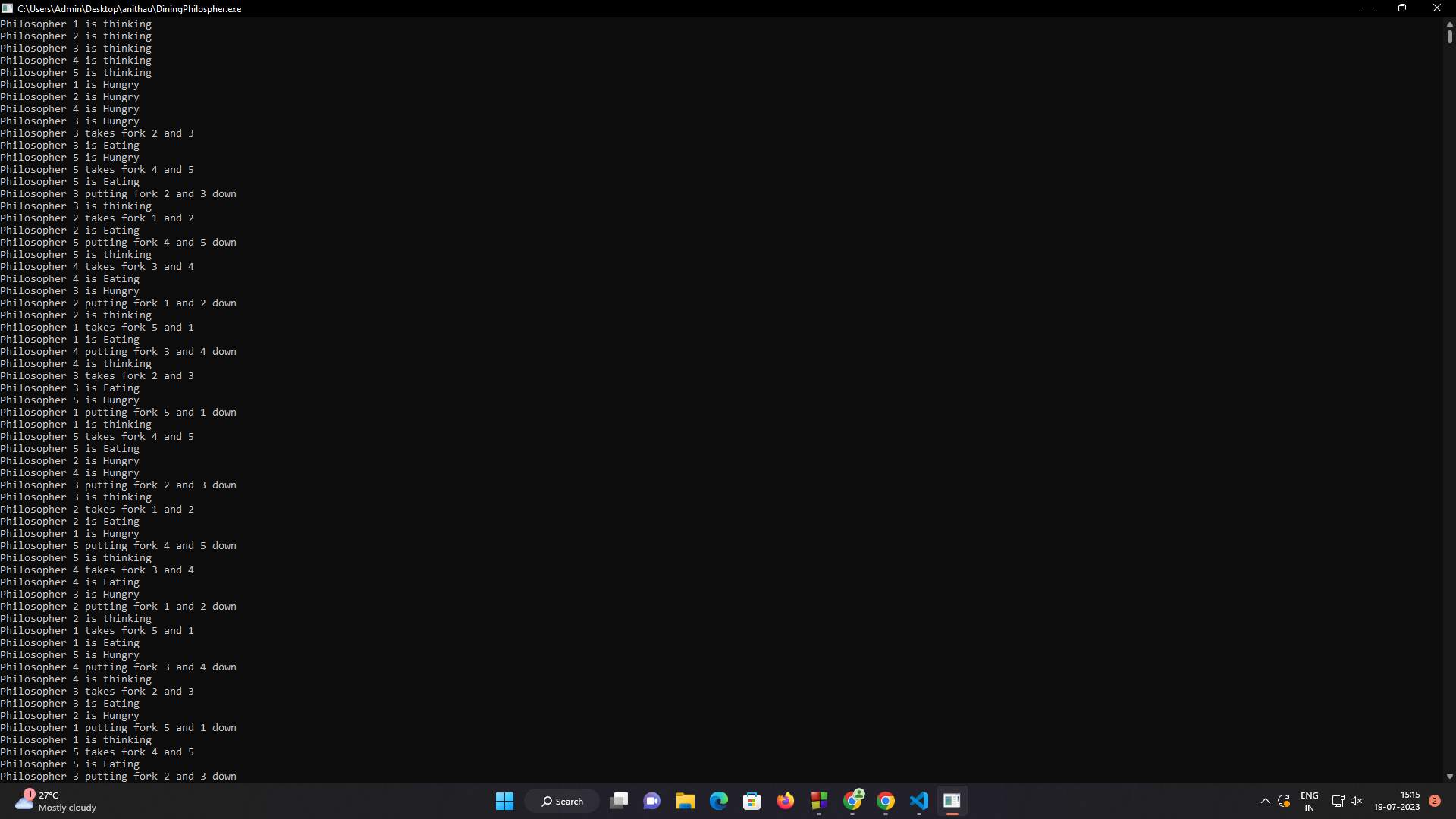
}

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

pthread\_join(thread\_id[i], NULL);

}

**OUTPUT:**

****