

Bansilal Ramnath Agarwal Charitable Trust's
Vishwakarma Institute of Technology, Pune-37

(Anautonomous Institute of Savitribai Phule Pune University)



Department of Multidisciplinary Engineering

Division	CS-A
Batch	B1
Roll no.	90
Name	Aditya Shrinivas Kurapati
Subject	OS

1. Producer Consumer PROBLEM USING C PROGRAM

```
#include <pthread.h>
```

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
#include <unistd.h>
```

```
#define BUFFER_SIZE 5
```

```
typedef struct {
```

```
    int buf[BUFFER_SIZE];
```

```
    size_t len;
```

```
    pthread_mutex_t mutex;
```

```
    pthread_cond_t can_produce;
```

```
    pthread_cond_t can_consume;
```

```
} buffer_t;
```

```
void *producer(void *arg) {
```

```
    buffer_t *buffer = (buffer_t *)arg;
```

```
    while (1) {
```

```
#ifdef UNDERFLOW
```

```
    sleep(5);
```

```
#endif
```

```
    pthread_mutex_lock(&buffer->mutex);
```

```
    if (buffer->len == BUFFER_SIZE) {
```

```
        pthread_cond_wait(&buffer->can_produce, &buffer->mutex);
```

```
}

int t = rand();

printf("Produced %d\n ", t);

buffer->buf[buffer->len] = t;
++buffer->len;

pthread_cond_signal(&buffer->can_consume);
pthread_mutex_unlock(&buffer->mutex);

printf("\nFULL: %d\n", buffer->len);
printf("\nEmpty: %d\n", (BUFFER_SIZE - buffer->len));
}

return NULL;
}

void *consumer(void *arg) {
    buffer_t *buffer = (buffer_t *)arg;

    while (1) {

#ifdef OVERFLOW
        sleep(5);
#endif

        pthread_mutex_lock(&buffer->mutex);
```

```

    if (buffer->len == 0) {
        pthread_cond_wait(&buffer->can_consume, &buffer->mutex);
    }
    --buffer->len;
    printf("Consumed %d\n", buffer->buf[buffer->len]);
    pthread_cond_signal(&buffer->can_produce);
    pthread_mutex_unlock(&buffer->mutex);

    printf("\nFULL: %d\n", buffer->len);
    printf("\nEmpty: %d\n", (BUFFER_SIZE - buffer->len));
}
return NULL;
}

int main(int argc, char *argv[]) {
    buffer_t buffer = {.len = 0,
        .mutex = PTHREAD_MUTEX_INITIALIZER,
        .can_produce = PTHREAD_COND_INITIALIZER,
        .can_consume = PTHREAD_COND_INITIALIZER};

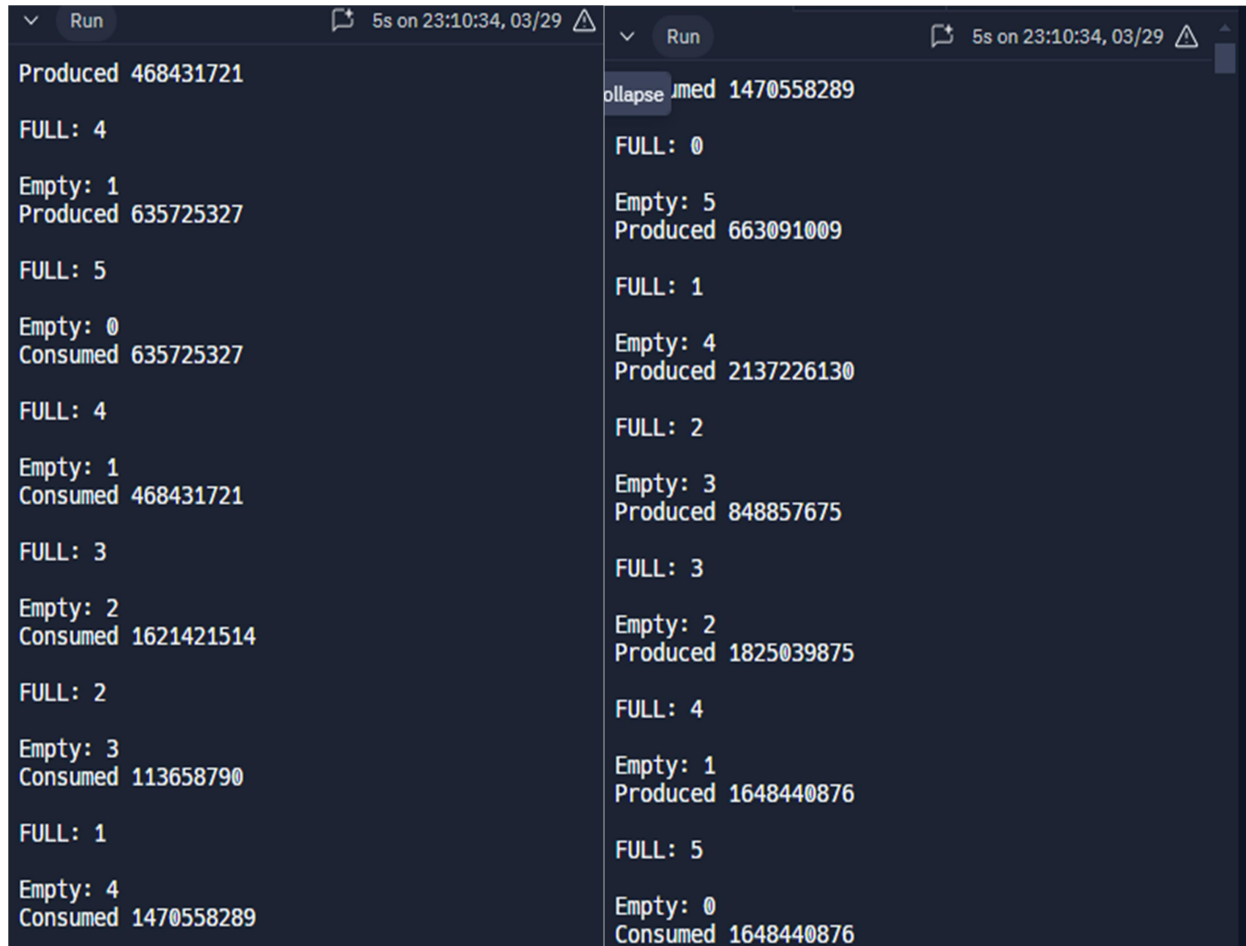
    pthread_t prod, cons;
    pthread_create(&prod, NULL, producer, (void *)&buffer);
    pthread_create(&cons, NULL, consumer, (void *)&buffer);

    pthread_join(prod, NULL);
    pthread_join(cons, NULL);
    return 0;
}

```

}

OUTPUT:-



```
Produced 468431721
FULL: 4
Empty: 1
Produced 635725327
FULL: 5
Empty: 0
Consumed 635725327
FULL: 4
Empty: 1
Consumed 468431721
FULL: 3
Empty: 2
Consumed 1621421514
FULL: 2
Empty: 3
Consumed 113658790
FULL: 1
Empty: 4
Consumed 1470558289

collapse
Empty: 1470558289
FULL: 0
Empty: 5
Produced 663091009
FULL: 1
Empty: 4
Produced 2137226130
FULL: 2
Empty: 3
Produced 848857675
FULL: 3
Empty: 2
Produced 1825039875
FULL: 4
Empty: 1
Produced 1648440876
FULL: 5
Empty: 0
Consumed 1648440876
```

2. READER_WRITER PROBLEM USING C PROGRAM

```
#include <pthread.h>
```

```
#include <semaphore.h>
```

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
#include <unistd.h>
```

```
sem_t mutex, writeblock;
```

```
int data = 0, rcount = 0;
```

```
void *reader(void *arg) {  
    int f;  
    f = *((int *)arg);  
    sem_wait(&mutex);  
    rcount++;  
    if (rcount == 1) {  
        sem_wait(&writeblock);  
    }  
    sem_post(&mutex);  
    printf("Reader %d is reading data %d\n", f, data);  
    sleep(2);  
    sem_wait(&mutex);  
    rcount--;  
    if (rcount == 0) {  
        sem_post(&writeblock);  
    }  
    sem_post(&mutex);  
    return NULL;  
}
```

```
void *writer(void *arg) {  
    int f;  
    f = *((int *)arg);  
    sem_wait(&writeblock);
```

```
data++;

printf("Data Written By The Writer Is %d\n", data);

sleep(1);

sem_post(&writeblock);

return NULL;
}

int main() {
    int i;

    pthread_t rtid[3], wtid[3];

    sem_init(&mutex, 0, 1);

    sem_init(&writeblock, 0, 1);

    while (1) {
        for (i = 0; i < 3; i++) {
            pthread_create(&wtid[i], NULL, writer, &i);

            pthread_create(&rtid[i], NULL, reader, &i);
        }

        for (i = 0; i < 3; i++) {
            pthread_join(wtid[i], NULL);

            pthread_join(rtid[i], NULL);
        }

        sleep(5); // Delay for 5 seconds before running the loop again

        printf("/n/n");
    }

    return 0;
}
```

OUTPUT:-

```
Run 52s on 23:15:00, 03/29
Data Written By The Writer Is 1
Data Written By The Writer Is 2
Data Written By The Writer Is 3
Reader 0 is reading data 3
Reader 0 is reading data 3
Reader 0 is reading data 3
/n/nData Written By The Writer Is 4
Reader 0 is reading data 4
Reader 0 is reading data 4
Reader 0 is reading data 4
Data Written By The Writer Is 5
Data Written By The Writer Is 6
/n/nData Written By The Writer Is 7
Reader 1 is reading data 7
Reader 2 is reading data 7
Reader 0 is reading data 7
Data Written By The Writer Is 8
Data Written By The Writer Is 9
/n/nData Written By The Writer Is 10
Data Written By The Writer Is 11
Reader 0 is reading data 11
Reader 0 is reading data 11
Reader 0 is reading data 11
Data Written By The Writer Is 12
/n/nData Written By The Writer Is 13
Reader 1 is reading data 13
Reader 2 is reading data 13
Reader 0 is reading data 13
Data Written By The Writer Is 14
Data Written By The Writer Is 15
/n/nData Written By The Writer Is 16
```


3.Dinning Philosophers Problem

```
#include <pthread.h>
```

```
#include <semaphore.h>
```

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
#include <unistd.h>
```

```
#define MIN(a, b) (a < b ? a : b)
```

```
#define MAX(a, b) (a > b ? a : b)
```

```
sem_t chopsticks[5];
```

```
void *eat(void *arg) {
```

```
    int i = (int)arg;
```

```
    int count = 2;
```

```
    while (count > 0) {
```

```
        sem_wait(&chopsticks[MIN(i, (i + 1) % 5)]);
```

```
        sem_wait(&chopsticks[MAX(i, (i + 1) % 5)]);
```

```
        printf("Philosopher %d is eating\n", i);
```

```
        sem_post(&chopsticks[MIN(i, (i + 1) % 5)]);
```

```
        sem_post(&chopsticks[MAX(i, (i + 1) % 5)]);
```

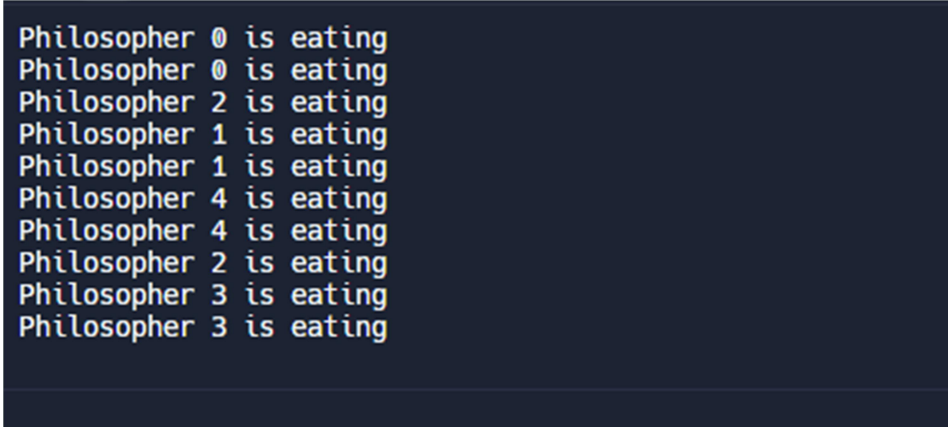
```
        count--;
```

```
    }
```

```
    return NULL;
```

```
}
```

```
int main() {  
    pthread_t philosophers[5];  
    int i;  
    for (i = 0; i < 5; i++) {  
        sem_init(&chopsticks[i], 0, 1);  
    }  
    for (i = 0; i < 5; i++) {  
        pthread_create(&philosophers[i], NULL, eat, (void *)i);  
    }  
    for (i = 0; i < 5; i++) {  
        pthread_join(philosophers[i], NULL);  
    }  
    return 0;  
}
```

A terminal window with a dark blue background and white text. It displays 12 lines of output, each stating 'Philosopher X is eating', where X is a number from 0 to 4. The numbers 0, 1, 2, 3, and 4 each appear three times, indicating that each of the five philosophers has eaten three times.

```
Philosopher 0 is eating  
Philosopher 0 is eating  
Philosopher 2 is eating  
Philosopher 1 is eating  
Philosopher 1 is eating  
Philosopher 4 is eating  
Philosopher 4 is eating  
Philosopher 2 is eating  
Philosopher 3 is eating  
Philosopher 3 is eating
```