Programiz C Online Compiler

Programiz P

main.c

Share Run

Output

```c
#include <stdio.h>
#include <stdbool.h>
#define P 5
#define R 3
int allocation[P][R], max[P][R], need[P][R], available[R];
bool isSafe() {
    int work[R], finish[P] = {0};
    for (int i = 0; i < R; i++) work[i] = available[i];
    while (1) {
        bool found = false;
        for (int p = 0; p < P; p++) {
            if (!finish[p]) {
                bool canAllocate = true;
                for (int r = 0; r < R; r++) {
                    if (need[p][r] > work[r]) {
                        canAllocate = false;
                        break;
                    }
                }
                if (canAllocate) {
                    for (int r = 0; r < R; r++) work[r] +=
                        allocation[p][r];
                    finish[p] = 1;
                    found = true;
                }
            }
        }
```

```
System is in a safe state.


=== Code Execution Successful ===
```

Programiz C Online Compiler

main.c

Share Run

```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <semaphore.h>
#define BUFFER_SIZE 5
int buffer[BUFFER_SIZE];
int in = 0, out = 0;
sem_t empty, full;
pthread_mutex_t mutex;
void* producer(void* arg) {
    for (int i = 0; i < 10; i++) {
        sem_wait(&empty);
        pthread_mutex_lock(&mutex);
        buffer[in] = i;
        in = (in + 1) % BUFFER_SIZE;
        printf("Produced: %d\n", i);
        pthread_mutex_unlock(&mutex);
        sem_post(&full);
    }
    return NULL;
}
void* consumer(void* arg) {
    for (int i = 0; i < 10; i++) {
        sem_wait(&full);
        pthread_mutex_lock(&mutex);
        int item = buffer[out];
        out = (out + 1) % BUFFER_SIZE;
```

Output

```
Produced: 0
Produced: 1
Produced: 2
Produced: 3
Produced: 4
Consumed: 0
Consumed: 1
Consumed: 2
Consumed: 3
Consumed: 4
Produced: 5
Produced: 6
Produced: 7
Produced: 8
Consumed: 5
Consumed: 6
Consumed: 7
Consumed: 8
Produced: 9
Consumed: 9


=== Code Execution Successful ===
```