



National Textile University

Department of Computer Science

Subject:

Operating System

Submitted to:

Sir Nasir Mehmood

Submitted by:

Akasha Fatima

Reg. number:

23-NTU-CS-FL-1132

Semester:

5th - A

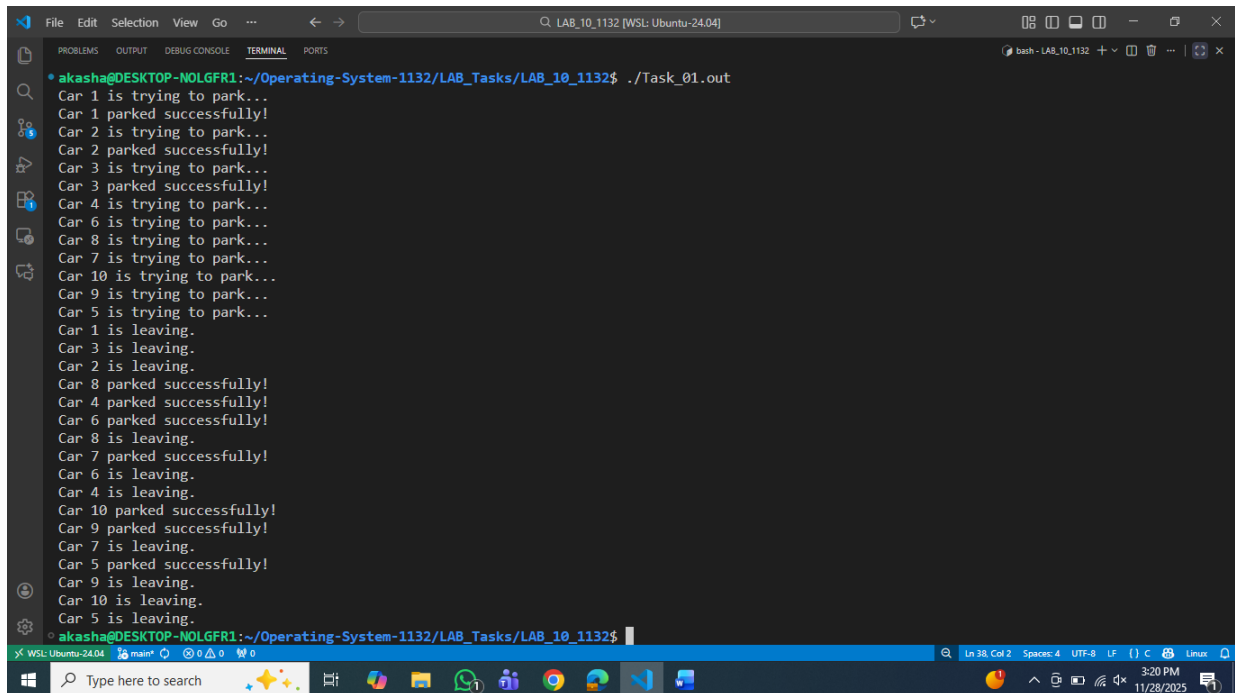
LAB_10

Task_01:

CODE:

```
1
2 // Parking Problem
3
4 #include <stdio.h>
5 #include <pthread.h>
6 #include <semaphore.h>
7 #include <unistd.h>
8
9 sem_t parking_spaces;
10 void* car(void* arg) {
11     int id = *(int*)arg;
12     printf("Car %d is trying to park...\n", id);
13     sem_wait(&parking_spaces); // Try to get a space
14     printf("Car %d parked successfully!\n", id);
15     sleep(2); // Stay parked for 2 seconds
16     printf("Car %d is leaving.\n", id);
17     sem_post(&parking_spaces); // Free the space
18     return NULL;
19 }
20
21 int main() {
22     pthread_t cars[10];
23     int ids[10];
24     // Initialize: 3 parking spaces available
25     sem_init(&parking_spaces, 0, 3);
26     // Create 10 cars (more than spaces!)
27     for(int i = 0; i < 10; i++) {
28         ids[i] = i + 1;
29         pthread_create(&cars[i], NULL, car, &ids[i]);
30     }
31     // Wait for all cars
32     for(int i = 0; i < 10; i++) {
33         pthread_join(cars[i], NULL);
34     }
35
36     sem_destroy(&parking_spaces);
37     return 0;
38 }
```

Output:



```
akasha@DESKTOP-NOLGFR1:~/Operating-System-1132/LAB_Tasks/LAB_10_1132$ ./Task_01.out
Car 1 is trying to park...
Car 1 parked successfully!
Car 2 is trying to park...
Car 2 parked successfully!
Car 3 is trying to park...
Car 3 parked successfully!
Car 4 is trying to park...
Car 6 is trying to park...
Car 8 is trying to park...
Car 7 is trying to park...
Car 10 is trying to park...
Car 9 is trying to park...
Car 5 is trying to park...
Car 1 is leaving.
Car 3 is leaving.
Car 2 is leaving.
Car 8 parked successfully!
Car 4 parked successfully!
Car 6 parked successfully!
Car 8 is leaving.
Car 7 parked successfully!
Car 6 is leaving.
Car 4 is leaving.
Car 10 parked successfully!
Car 9 parked successfully!
Car 7 is leaving.
Car 5 parked successfully!
Car 9 is leaving.
Car 10 is leaving.
Car 5 is leaving.
```

Task_02:

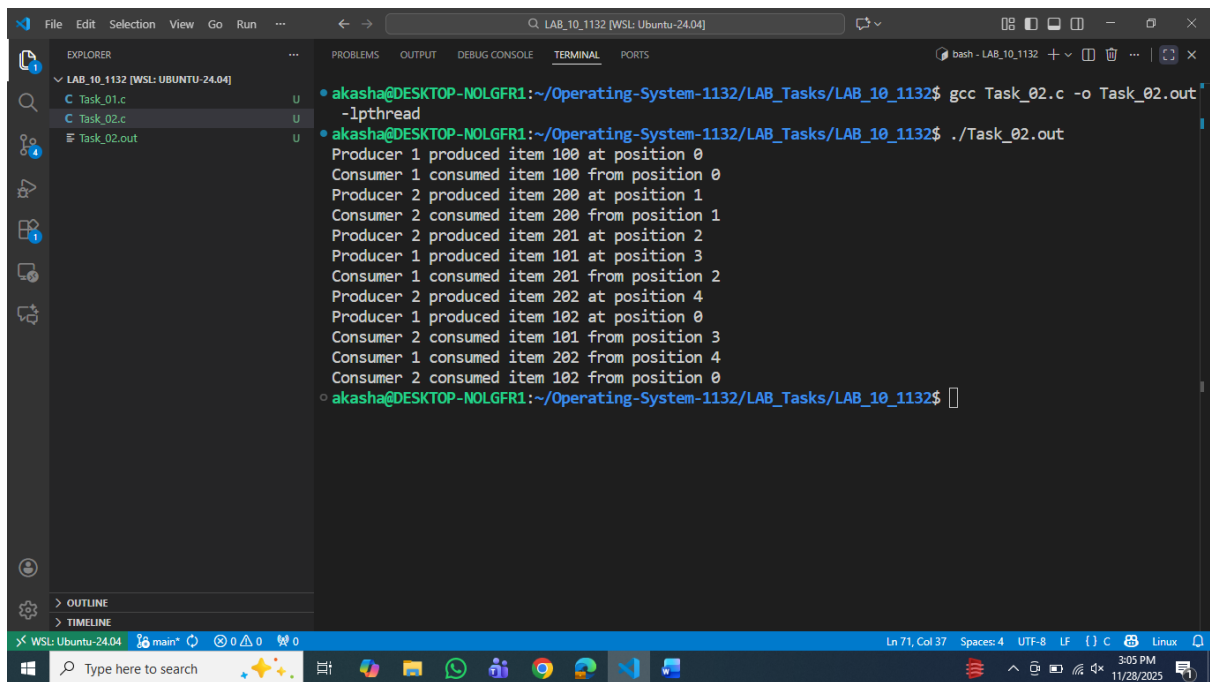
CODE:

```

1
2
3 #include <stdio.h>
4 #include <pthread.h>
5 #include <semaphore.h>
6 #include <unistd.h>
7 #define BUFFER_SIZE 5
8 int buffer[BUFFER_SIZE];
9 int in = 0; // Producer index
10 int out = 0; // Consumer index
11 sem_t empty; // Counts empty slots
12 sem_t full; // Counts full slots
13 pthread_mutex_t mutex;
14 void* producer(void* arg) {
15     int id = *(int*)arg;
16     for(int i = 0; i < 3; i++) { // Each producer makes 3 items
17
18         int item = id * 100 + i;
19         // TODO: Wait for empty slot
20         sem_wait(&empty);
21         // TODO: Lock the buffer
22         pthread_mutex_lock(&mutex);
23         // Add item to buffer
24         buffer[in] = item;
25         printf("Producer %d produced item %d at position %d\n",
26             id, item, in);
27         in = (in + 1) % BUFFER_SIZE;
28         // TODO: Unlock the buffer
29         pthread_mutex_unlock(&mutex);
30         // TODO: Signal that buffer has a full slot
31         sem_post(&full);
32         sleep(1);
33     }
34     return NULL;
35 }
36
37 void* consumer(void* arg) {
38     int id = *(int*)arg;
39     for(int i = 0; i < 3; i++) {
40         // TODO: Students complete this similar to producer
41         sem_wait(&full);
42         pthread_mutex_lock(&mutex);
43         int item = buffer[out];
44         printf("Consumer %d consumed item %d from position %d\n",
45             id, item, out);
46         out = (out + 1) % BUFFER_SIZE;
47         pthread_mutex_unlock(&mutex);
48         sem_post(&empty);
49         sleep(2); // Consumers are slower
50     }
51     return NULL;
52 }
53
54 int main() {
55     pthread_t prod[2], cons[2];
56     int ids[2] = {1, 2};
57     // Initialize semaphores
58     sem_init(&empty, 0, BUFFER_SIZE); // All slots empty initially
59     sem_init(&full, 0, 0);
60     pthread_mutex_init(&mutex, NULL);
61     // No slots full initially
62     // Create producers and consumers
63     for(int i = 0; i < 2; i++) {
64         pthread_create(&prod[i], NULL, producer, &ids[i]);
65         pthread_create(&cons[i], NULL, consumer, &ids[i]);
66     }
67
68     // Wait for completion
69     for(int i = 0; i < 2; i++) {
70         pthread_join(prod[i], NULL);
71         pthread_join(cons[i], NULL);
72     }
73
74     // Cleanup
75     sem_destroy(&empty);
76     sem_destroy(&full);
77     pthread_mutex_destroy(&mutex);
78     return 0;
79 }

```

Output:



```
akasha@DESKTOP-NOLGFR1:~/Operating-System-1132/LAB_Tasks/LAB_10_1132$ gcc Task_02.c -o Task_02.out -lpthread
akasha@DESKTOP-NOLGFR1:~/Operating-System-1132/LAB_Tasks/LAB_10_1132$ ./Task_02.out
Producer 1 produced item 100 at position 0
Consumer 1 consumed item 100 from position 0
Producer 2 produced item 200 at position 1
Consumer 2 consumed item 200 from position 1
Producer 2 produced item 201 at position 2
Producer 1 produced item 101 at position 3
Consumer 1 consumed item 201 from position 2
Producer 2 produced item 202 at position 4
Producer 1 produced item 102 at position 0
Consumer 2 consumed item 101 from position 3
Consumer 1 consumed item 202 from position 4
Consumer 2 consumed item 102 from position 0
akasha@DESKTOP-NOLGFR1:~/Operating-System-1132/LAB_Tasks/LAB_10_1132$
```

Task_03:

Input:

If the “**producer**” is less than “**consumer**”, then the consumer will wait for the producer to produce items. In this case, the consumer will cause the semaphores to be in the deadlock.

Producer[2]: 3

Consumer[2]: 4

Output:

```
akasha@DESKTOP-NOLGFR1:~/Operating-System-1132/LAB_Tasks/LAB_10_1132$ gcc Task_02.c -o Task_02.out
akasha@DESKTOP-NOLGFR1:~/Operating-System-1132/LAB_Tasks/LAB_10_1132$ ./Task_02.out
Producer 1 produced item 100 at position 0
Consumer 2 consumed item 100 from position 0
Producer 2 produced item 200 at position 1
Consumer 1 consumed item 200 from position 1
Producer 1 produced item 101 at position 2
Producer 2 produced item 201 at position 3
Consumer 2 consumed item 101 from position 2
Producer 1 produced item 102 at position 4
Consumer 1 consumed item 201 from position 3
Producer 2 produced item 202 at position 0
Consumer 2 consumed item 102 from position 4
Consumer 1 consumed item 202 from position 0
```

Task_04:

Input:

No. of Threads: 4 – 2, 2 producers, consumers

Total No. of Items: 12

Total No. of Products: 6

Total No. of Consumers: 6

Explanation:

Since 4 threads have been initialized; 2 and 2 for both consumer and products so every single **Item (producer & consumer)** will produce 3,3 items as in the corresponding methods.

Task_05:

Input:

If **No. of threads** is less than **No. of items**, then the remaining items will be in the “wait or blocked” state. Suppose if

No. of Threads: 3

No. of Consumer Item: 6

Results:

The remaining 4 items will be in the **wait** state.

Task_06:

Input:

Technical Working:

Output:

