National Textile University, Faisalabad

Department of Computer Science

Name	Dawood Saif
Class	SE-5th (A)
Reg. No.	23-NTU-CS-1145
Course	Operating system
Submitted To	Sir Nasir Mehmood
Submission Date	28/11/2025
Lab No.	10 (After Mids.)

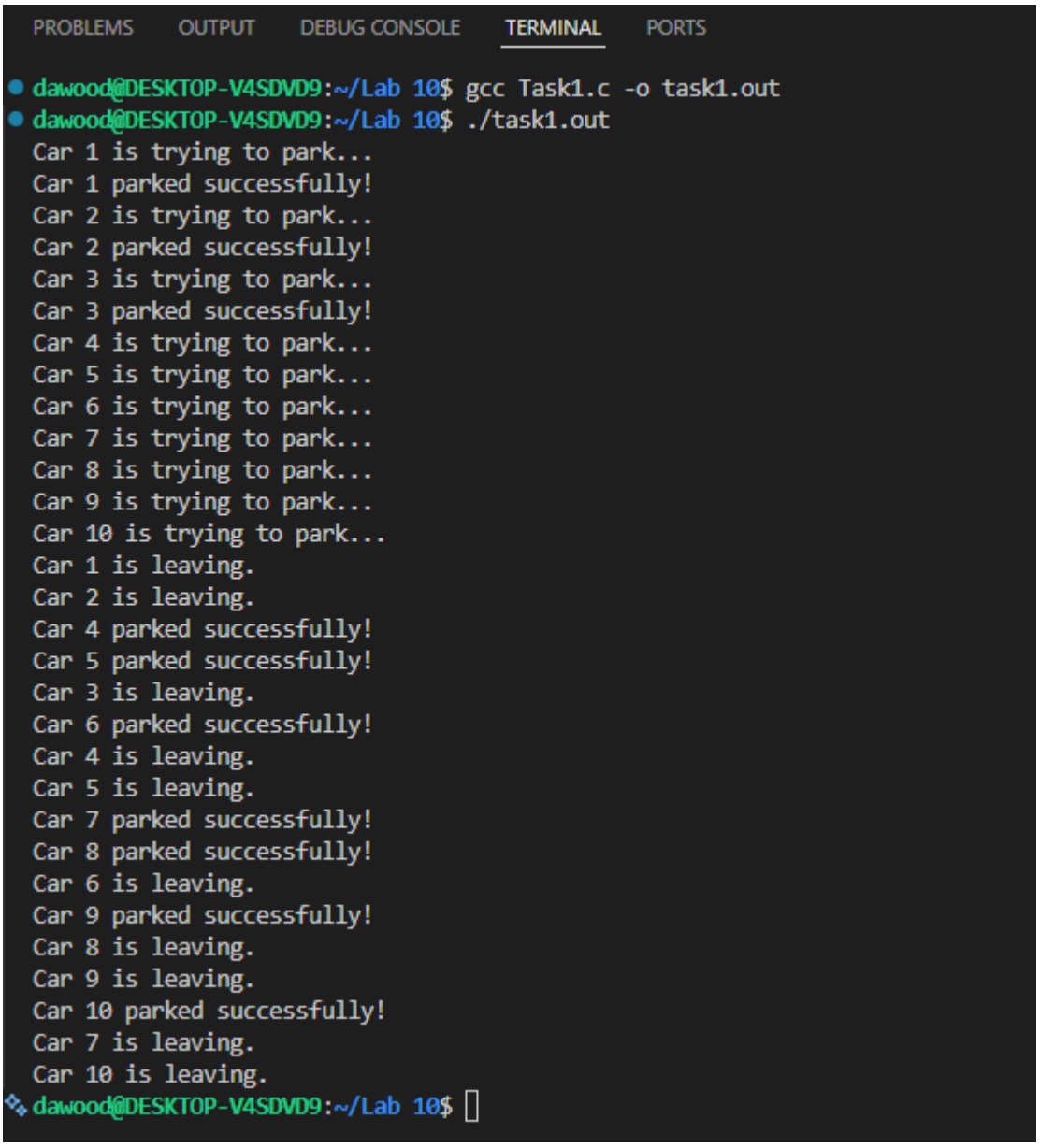
Lab No. 10 Operating system:

Counting Semaphores:

Task 1:

```c
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>

sem_t parking_spaces;
void* car(void* arg) {
int id = *(int*)arg;
printf("Car %d is trying to park...\n", id);
sem_wait(&parking_spaces); // Try to get a space
printf("Car %d parked successfully!\n", id);
sleep(2); // Stay parked for 2 seconds
printf("Car %d is leaving.\n", id);
sem_post(&parking_spaces); // Free the space
return NULL;
}

int main() {
pthread_t cars[10];
int ids[10];
// Initialize: 3 parking spaces available
sem_init(&parking_spaces, 0, 3);
// Create 10 cars (more than spaces!)
for(int i = 0; i < 10; i++) {
ids[i] = i + 1;
pthread_create(&cars[i], NULL, car, &ids[i]);
}
// Wait for all cars
for(int i = 0; i < 10; i++) {
pthread_join(cars[i], NULL);
}
sem_destroy(&parking_spaces);
return 0;
}
```

Output:

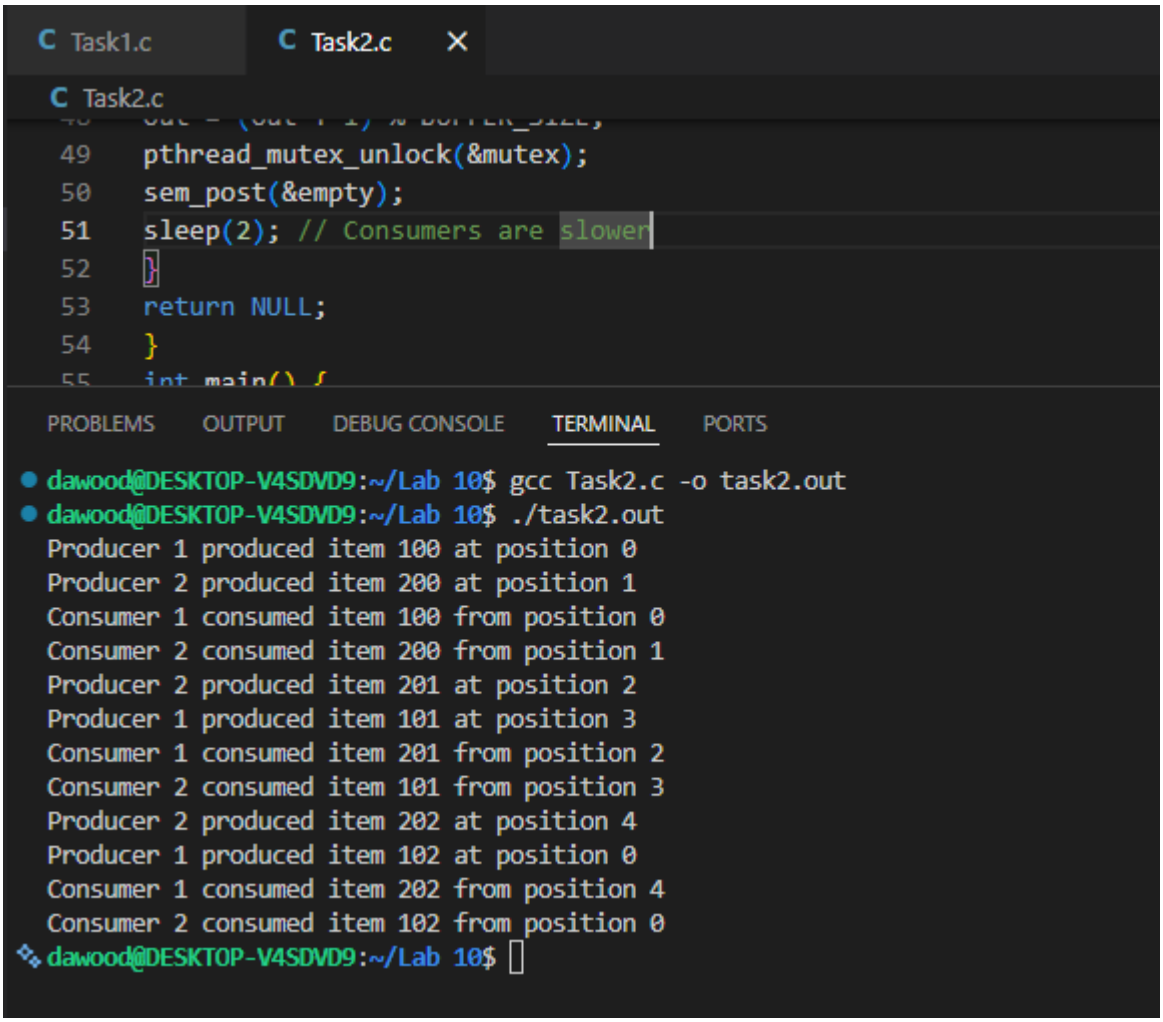

PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
dawood@DESKTOP-V4SDVD9:~/Lab 10$ gcc Task1.c -o task1.out
dawood@DESKTOP-V4SDVD9:~/Lab 10$ ./task1.out
Car 1 is trying to park...
Car 1 parked successfully!
Car 2 is trying to park...
Car 2 parked successfully!
Car 3 is trying to park...
Car 3 parked successfully!
Car 4 is trying to park...
Car 5 is trying to park...
Car 6 is trying to park...
Car 7 is trying to park...
Car 8 is trying to park...
Car 9 is trying to park...
Car 10 is trying to park...
Car 1 is leaving.
Car 2 is leaving.
Car 4 parked successfully!
Car 5 parked successfully!
Car 3 is leaving.
Car 6 parked successfully!
Car 4 is leaving.
Car 5 is leaving.
Car 7 parked successfully!
Car 8 parked successfully!
Car 6 is leaving.
Car 9 parked successfully!
Car 8 is leaving.
Car 9 is leaving.
Car 10 parked successfully!
Car 7 is leaving.
Car 10 is leaving.
dawood@DESKTOP-V4SDVD9:~/Lab 10$

Task 2:

```c
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>


#define BUFFER_SIZE 5
int buffer[BUFFER_SIZE];
int in = 0; // Producer index
int out = 0; // Consumer index
sem_t empty; // Counts empty slots
sem_t full; // Counts full slots
pthread_mutex_t mutex;


void* producer(void* arg) {
    int id = *(int*)arg;
for(int i = 0; i < 3; i++) { // Each producer makes 3 items
int item = id * 100 + i;        // 6 max values

// TODO: Wait for empty slot
sem_wait(&empty);
// TODO: Lock the buffer
pthread_mutex_lock(&mutex);
// Add item to buffer
buffer[in] = item;
printf("Producer %d produced item %d at position %d\n",
id, item, in);
in = (in + 1) % BUFFER_SIZE;
// TODO: Unlock the buffer
pthread_mutex_unlock(&mutex);
// TODO: Signal that buffer has a full slot
sem_post(&full);
sleep(1);
}
return NULL;
}
void* consumer(void* arg) {
int id = *(int*)arg;
for(int i = 0; i < 3; i++) {

// TODO: Students complete this similar to producer
sem_wait(&full);
```

```
pthread_mutex_lock(&mutex);
int item = buffer[out];
printf("Consumer %d consumed item %d from position %d\n",
id, item, out);
out = (out + 1) % BUFFER_SIZE;
pthread_mutex_unlock(&mutex);
sem_post(&empty);
sleep(2); // Consumers are slower
}
return NULL;
}
int main() {
    pthread_t prod[2], cons[2];
int ids[2] = {1, 2};
// Initialize semaphores
sem_init(&empty, 0, BUFFER_SIZE); // All slots empty initially
sem_init(&full, 0, 0); // No slots full initially
pthread_mutex_init(&mutex, NULL);
// Create producers and consumers
for(int i = 0; i < 2; i++) {
pthread_create(&prod[i], NULL, producer, &ids[i]);
pthread_create(&cons[i], NULL, consumer, &ids[i]);
}
// Wait for completion
for(int i = 0; i < 2; i++) {
pthread_join(prod[i], NULL);
pthread_join(cons[i], NULL);
}
// Cleanup
sem_destroy(&empty);
sem_destroy(&full);
pthread_mutex_destroy(&mutex);
return 0;
}
```

Output:



Remarks:

In this code there are two Semaphores Empty and Full

- Empty Values has been initialized with buffer size which is 5
- While Full Values has been initialized with 0

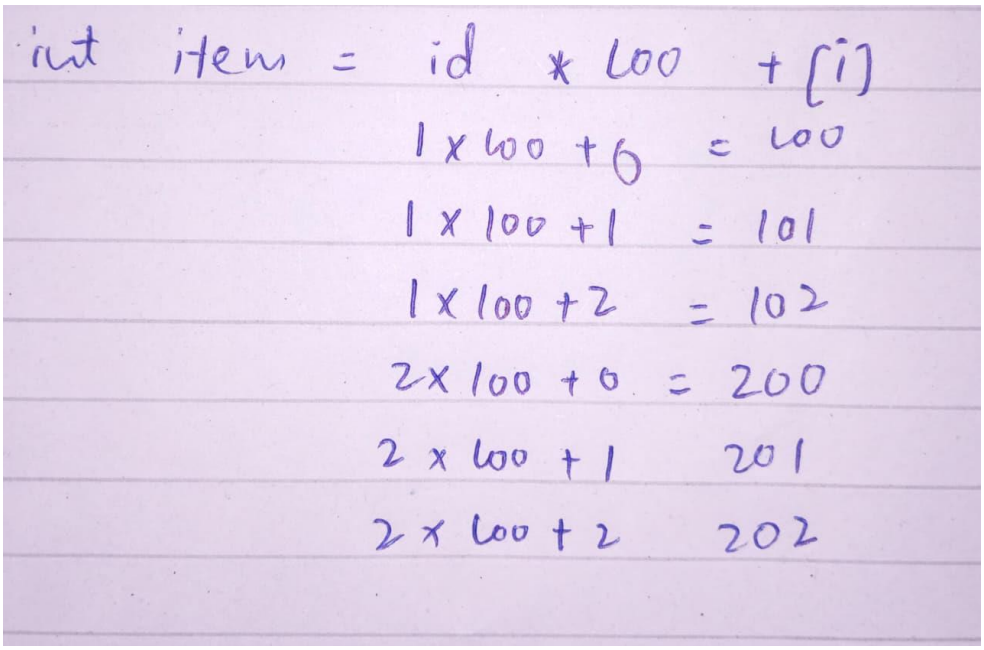
So now, **When the Producer** will run, it will create 3 items [i], Then it will wait for empty slots, then it will lock buffer

Perform its task – that is critical section, after that it will unlock buffer, then it will increment the count of full slots by Sem_post(&Full)

Similarly**, In Consumer**, each consumer will create 4 items, then wait for the full slot and it will lock buffer then

After performing task or critical section it will unlock the buffer and after consuming it will increment empty slots by Sem_post (&empty slots)

Line int item = id * 100 + i; will make 6 Possible Values:



Each Has 5 Main Steps