

## Semaphore

### 1. `sem_init`

- `int sem_init(sem_t *sem, int pshared, unsigned int value);`
- Initializes a new unnamed semaphore or reinitializes an existing semaphore.

### 2. `sem_wait`

- `int sem_wait(sem_t *sem);`
- Waits (blocks) until the specified semaphore's value is greater than zero, then decrements the semaphore.

### 3. `sem_post`

- `int sem_post(sem_t *sem);`
- Increments the value of the specified semaphore. If blocked threads are waiting, one of them will be unblocked.

### 4. `sem_destroy`

- `int sem_destroy(sem_t *sem);`
- Destroys the specified semaphore, making it unusable thereafter.

## Problem 0

Write a C program to create 2 threads and update a global variable `sum` access to which is controlled by a semaphore.

## Solution

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <semaphore.h>

int sum = 0;
sem_t sem;

void *solve(void *p) {
    int val = *(int *)p;
    sem_wait(&sem); // Wait for the semaphore
    // Start of Critical Section
    sum += val;
    printf("Value: %d\n", sum);
    // End of Critical Section
    sem_post(&sem); // Signal the semaphore
    pthread_exit(NULL);
}

int main(int argc, char* argv[]) {
```

```
pthread_t thread1, thread2;

// Initialize Semaphore with an initial value of 1
sem_init(&sem, 0, 1);

int i = 1, j = 2;
pthread_create(&thread1, NULL, solve, &i);
pthread_create(&thread2, NULL, solve, &j);

pthread_join(thread1, NULL);
pthread_join(thread2, NULL);

// Destroy Semaphore
sem_destroy(&sem);

return 0;
}
```