Synchronization with Semaphore

The pair of functions which control the value of the semaphore are declared as follows:

```
#include <semaphore.h>
int sem_wait(sem_t * sem);
int sem_post(sem_t * sem);
```

Both these funtions take a pointer to the semaphore object initialized by a call to sem_init. The sem_post function atomically increases the value of the semaphore by 1. The sem_wait function atomically decreases the value of the semaphore by one, but always waits until the semaphore has a nonzero count first. Thus, if you call sem_wait on a semaphore with a value of 2, the thread will continue executing but the semaphore will be decreased to 1. If sem_wait is called on a semaphore with a value of 0, the function will wait until some other thread has incremented the value so that it is no longer 0.

Q1. WAP to Demonstrate a basic semaphore implementation.

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

sem_t mutex; // Semaphore for mutual exclusion

void* thread_function() {
    sem_wait(&mutex); // Acquire the semaphore

    // Critical section where shared resources are accessed
    printf("Thread %lu is in the critical section\n", pthread_self());

    sem_post(&mutex); // Release the semaphore
}

int main() {
    pthread_t thread1, thread2;

    // Initialize the semaphore
    sem_init(&mutex, 0, 1);

    // Create threads
    pthread_create(&thread1, NULL, thread_function, NULL);
    pthread_create(&thread2, NULL, thread_function, NULL);

    // Wait for threads to finish
    pthread_join(thread1, NULL);
    pthread_join(thread2, NULL);
    // Destroy the semaphore
    sem_destroy(&mutex);

    return 0;
}
```

Q2. WAP to demostrate counting semaphore.

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

sem_t semaphore; // Semaphore for counting

void* thread_function() {
    sem_wait(&semaphore); // Decrement the semaphore

    // Critical section where shared resources are accessed
    printf("Thread %lu is in the critical section\n", pthread_self());
    sleep(5);

    sem_post(&semaphore); // Increment the semaphore
}

int main() {
    int num_threads = 5;

    // Initialize the semaphore with an initial value of 3
    sem_init(&semaphore, 0, 3);

    // Create threads
    for (int i = 0; i < num_threads; i++) {
        pthread_t thread;
        pthread_create(&thread, NULL, thread_function, NULL);
        pthread_detach(thread); // Detach threads for asynchronous execution
    }

    // Main thread continues while detached threads execute
    // ...

    return 0;
}
```

Q3. Write C program using thread concept to count no of character entered from the keyboard and also utilize the concept of semaphore for thread synchronization.

```
#include<stdio.h>
#include<sys/types.h>
#include<string.h>
#include<stdlib.h>
#include<unistd.h>
#include<pthread.h>
#include<semaphore.h>
void *count(void *arg);
sem_t s;                              Declare the semaphore, s, as a global variable.
char area [1024];
int main()
{
        int res;
        pthread_t tid;
        void *result;

        res=sem_init(&s,0,0);         Initialize the semaphore, s, to 0.

        if(res!=0)
        {
                perror("Semaphore creation failed");
                exit(EXIT_FAILURE);
        }
        res=pthread_create(&tid,NULL,count,NULL);
        if(res!=0)
        {
                perror("Thread creation failed");
                exit(EXIT_FAILURE);
        }
        printf("Enter some text. Enter end to finish\n");
        while(strncmp("end",area,3)!=0)
        {
                fgets(area,1024,stdin);
                sem_post(&s);         Unlock the semaphore, s, by incrementing the value from 0 to 1
        }
        printf("\nWaiting for thread to finish\n");
        res=pthread_join(tid,&result);
        if(res!=0)
        {
                perror("Thread joined failure");
                exit(EXIT_FAILURE);
        }
        printf("Thread joined%s\n",(char *)result);
        sem_destroy(&s);
        exit(EXIT_SUCCESS);
}
void *count(void *arg)
{
        sem_wait(&s);                 Wait on the semaphore.
```

```
	while(strncmp("end",area,3)!=0)
	{
		printf("You input %ld Character\n",strlen(area)-1);
		sem_wait(&s);          Wait on the semaphore.
	}
	pthread_exit("\nThnaks for CPU time & Count funtion running with semamphore\n");
}
```

Q4. Write a program in C, that creates two threads, one of which reads a message and the other writes or displays the same on screen. Synchronise these threads with semaphores to ensure that read comes before write.

Solution: Try on your own.