C MINOR ASSIGNMENT- 10 (Process/Thread Synchronization)

1. Find the critical section and determine the race condition for the given pseudo code. Write program using **fork**() to realize the race condition(**Hint:** *case-1:* Execute P_1 first then P_2, *case-2:* Execute P_2 first then P_1). Here **`shrd`** is a shared variables.

```
P-1 executes:
   x = *shrd;
   x = x + 1;
  sleep(1);
  *shrd = x;
```

```
P-2 executes
   y = *shrd;
   y = y - 1;
   sleep(1);
   *shrd = y;
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>

int main() {
    int shrd = 0;  // Shared variable
    int pid = fork();

    if (pid < 0) {
        perror("Fork failed");
        exit(1);
    }

    if (pid == 0) { // Child process (P1)
        int x = shrd;
        x = x + 1;
        sleep(1);  // Simulate race condition
        shrd = x;
        printf("P1 updated shared variable to: %d\n", shrd);
    }
    else {         // Parent process (P2)
        int y = shrd;
        y = y - 1;
        sleep(1);  // Simulate race condition
        shrd = y;
        printf("P2 updated shared variable to: %d\n", shrd);
        wait(NULL); // Wait for child process
    }
    return 0;
}
```

Explanation: The race condition occurs because both processes access and modify the shared variable without synchronization. The final value of `shrd` depends on the execution order of P1 and P2.

2. Suppose **`process 1`** must execute statement **`a`** before **`process 2`** executes statement **`b`**. The semaphore **`sync`** enforces the ordering in the following pseudocode, provided that **`sync`** is initially `0`.

```
Process 1 executes:                    Process 2 executes:
   a;                                     wait(&sync);
   signal(&sync);                         b;
```

Because **`sync`** is initially **`0`**, **`process 2`** blocks on its **`wait`** until **`process 1`** calls **`signal`**.
Now, You develop a C code using **`POSIX:SEM`** **semaphore** to get the desired result.

Explanation: https://chatgpt.com/c/67764fea-71d4-8001-aefe-7ba3b4bf961f

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

sem_t sync;

void* process1(void* arg) {
    printf("Process 1: Executing statement a\n");
    sem_post(&sync); // Signal
    return NULL;
}
void* process2(void* arg) {
    sem_wait(&sync); // Wait
    printf("Process 2: Executing statement b\n");
    return NULL;
}

int main() {
    pthread_t t1, t2;
    sem_init(&sync, 0, 0); // Initialize semaphore to 0
    pthread_create(&t1, NULL, process1, NULL);
    pthread_create(&t2, NULL, process2, NULL);
    pthread_join(t1, NULL);
    pthread_join(t2, NULL);
    sem_destroy(&sync);
    return 0; }
```

3. Implement the following pseudocode over the semaphores S and Q. State your answer on different initializations of S and Q.

   (a) S=1, Q=1
   (b) S=1, Q=0
   (c) S=0, Q=1
   (d) S=0, Q=0
   (e) S=8, Q=0

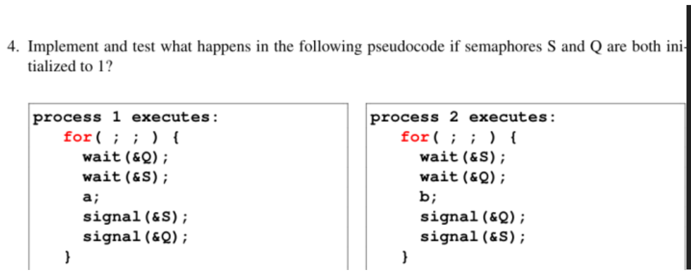
```
Process 1 executes:  Process 2 executes:
for( ; ; ) {           for( ; ; ) {
  wait(&S);                 wait(&Q);
  a;                        b;
  signal(&Q);               signal(&S);
}                       }
```

Explanation: https://chatgpt.com/g/g-F00faAwkE-open-a-i-gpt-3-5/c/67765294-15e4-8001-aad8-660d6d3dd2d7

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

sem_t S, Q;
void* process1(void* arg) {
    while (1) {
        sem_wait(&S);
        printf("Process 1: Executing statement a\n");
        sem_post(&Q);
    }
    return NULL;
}
void* process2(void* arg) {
    while (1) {
        sem_wait(&Q);
        printf("Process 2: Executing statement b\n");
        sem_post(&S);
    }
    return NULL;
}
```

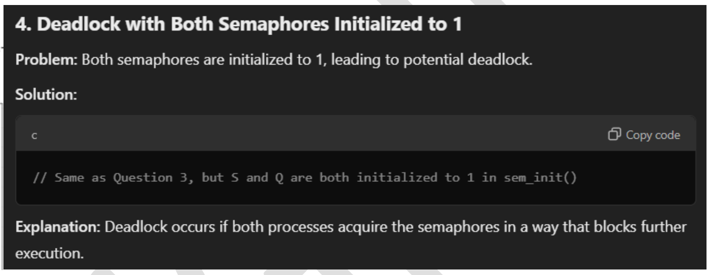
```c
int main() {
    pthread_t t1, t2;
    sem_init(&S, 0, 1); // Initialize S
    sem_init(&Q, 0, 0); // Initialize Q
    pthread_create(&t1, NULL, process1, NULL);
    pthread_create(&t2, NULL, process2, NULL);
    pthread_join(t1, NULL);
    pthread_join(t2, NULL);
    sem_destroy(&S);
    sem_destroy(&Q);
    return 0;
}
```



4. Implement and test what happens in the following pseudocode if semaphores S and Q are both ini-
tialized to 1?

```
process 1 executes:
    for( ; ; ) {
      wait(&Q);
      wait(&S);
      a;
      signal(&S);
      signal(&Q);
    }
```

```
process 2 executes:
    for( ; ; ) {
      wait(&S);
      wait(&Q);
      b;
      signal(&Q);
      signal(&S);
    }
```



4. Deadlock with Both Semaphores Initialized to 1

Problem: Both semaphores are initialized to 1, leading to potential deadlock.

Solution:

```c
// Same as Question 3, but S and Q are both initialized to 1 in sem_init()
```

Explanation: Deadlock occurs if both processes acquire the semaphores in a way that blocks further execution.

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

sem_t S, Q;
void* process1(void* arg) {
    while (1) {
        sem_wait(&S);  // Acquire S first
        sem_wait(&Q);  // Then acquire Q
        printf("Process 1: Executing a\n");
        sem_post(&Q);  // Release Q
        sem_post(&S);  // Release S
    }
    return NULL;
}
void* process2(void* arg) {
    while (1) {
        sem_wait(&S);  // Acquire S first (same order)
        sem_wait(&Q);  // Then acquire Q
        printf("Process 2: Executing b\n");
        sem_post(&Q);  // Release Q
        sem_post(&S);  // Release S
    }
    return NULL;
}
int main() {
    pthread_t t1, t2;
    sem_init(&S, 0, 1); // Initialize S to 1
    sem_init(&Q, 0, 1); // Initialize Q to 1
    pthread_create(&t1, NULL, process1, NULL);
    pthread_create(&t2, NULL, process2, NULL);
    pthread_join(t1, NULL);
    pthread_join(t2, NULL);
    sem_destroy(&S);
    sem_destroy(&Q);
    return 0;
}
```

5. You have three processes/ threads as given in the below diagram. Create your C code to print the sequence gievn as;
XXYZZYXXYZZYXXYZZYXXYZZ. Your are required to choose any one of the process/ thread synchronization mechanism(s).

Process/Thread -1

```
for i = 1 to ?
:
:
display("X");
display("X"); :
:
```

Process/Thread -2

```
for i = 1 to ?
:
:
display("Y");
:
:
```

Process/Thread -3

```
for i = 1 to ?
:
:
display("Z");
display("Z"); :
:
```

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

sem_t semX, semY, semZ;
void* printX(void* arg) {
    for (int i = 0; i < 5; i++) {
        sem_wait(&semX);
        printf("X");
        printf("X");
        sem_post(&semY);
    }
    return NULL;
}
void* printY(void* arg) {
    for (int i = 0; i < 5; i++) {
        sem_wait(&semY);
        printf("Y");
        sem_post(&semZ);
    }
    return NULL;
}
void* printZ(void* arg) {
    for (int i = 0; i < 5; i++) {
        sem_wait(&semZ);
        printf("Z");
        printf("Z");
        sem_post(&semX);
    }
    return NULL;
}
int main() {
    pthread_t t1, t2, t3;
    sem_init(&semX, 0, 1);
    sem_init(&semY, 0, 0);
    sem_init(&semZ, 0, 0);
    pthread_create(&t1, NULL, printX, NULL);
    pthread_create(&t2, NULL, printY, NULL);
    pthread_create(&t3, NULL, printZ, NULL);
    pthread_join(t1, NULL);
    pthread_join(t2, NULL);
    pthread_join(t3, NULL);
    sem_destroy(&semX);
    sem_destroy(&semY);
    sem_destroy(&semZ);
    return 0;}
```

Explanation: https://chatgpt.com/c/677654a9-6ed8-8001-b5c4-acafb8aca84e

6. **Process ordering using semaphore**. Create 6 number of processes (1 parent + 5 children) using **`fork()`** in **`if-elseif-else`** ladder. The parent process will be waiting for the termination of all it's children and each process will display a line of text on the standard error as;

```
P1: Coronavirus
P2: WHO:
P3: COVID-19
P4: disease
P5: pandemic
```

Your program must display the message in the given order
`WHO: Coronavirus disease COVID-19 pandemic`.

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
#include <semaphore.h>
#include <pthread.h>

sem_t sem1, sem2, sem3, sem4, sem5;

void print_message(const char* message, sem_t* my_sem, sem_t* next_sem) {
    sem_wait(my_sem);  // Wait for the turn
    fprintf(stderr, "%s ", message);  // Print message to stderr
    sem_post(next_sem);  // Signal the next process
}
int main() {
    // Initialize semaphores
    sem_init(&sem1, 1, 1);  // First process starts immediately
    sem_init(&sem2, 1, 0);  // All other processes start only when signaled
    sem_init(&sem3, 1, 0);
    sem_init(&sem4, 1, 0);
    sem_init(&sem5, 1, 0);
    pid_t pid1, pid2, pid3, pid4, pid5;
    // Child Process 1: Prints "WHO:"
    if ((pid1 = fork()) == 0) {
        print_message("WHO:", &sem1, &sem2);
        exit(0);
    }
    // Child Process 2: Prints "Coronavirus"
    if ((pid2 = fork()) == 0) {
        print_message("Coronavirus", &sem2, &sem3);
        exit(0);
    }
    // Child Process 3: Prints "disease"
    if ((pid3 = fork()) == 0) {
        print_message("disease", &sem3, &sem4);
        exit(0);
    }
    // Child Process 4: Prints "COVID-19"
    if ((pid4 = fork()) == 0) {
        print_message("COVID-19", &sem4, &sem5);
        exit(0);
    }
    // Child Process 5: Prints "pandemic."
    if ((pid5 = fork()) == 0) {
        print_message("pandemic.", &sem5, NULL);  // No next semaphore
        exit(0);
    }
    // Parent process waits for all children
    for (int i = 0; i < 5; i++) {
        wait(NULL);  // Wait for each child process to terminate
    }
    // Clean up semaphores
    sem_destroy(&sem1);
    sem_destroy(&sem2);
    sem_destroy(&sem3);
    sem_destroy(&sem4);
    sem_destroy(&sem5);

    return 0;
}
```
Explanation: https://chatgpt.com/c/67765581-fa3c-8001-85da-d89290ebd369

```
/* Q7) OPTIONAL: Write a program to give a semaphore-based solution to the producer-consumer problem
 using a bounded buffer scheme*/

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

#define BUFFER_SIZE 5
int buffer[BUFFER_SIZE];
int in = 0, out = 0;
sem_t empty, full;  // Semaphores for empty slots and full slots
pthread_mutex_t mutex;  // Mutex for critical section
void* producer(void* arg) {
    int item;
    for (int i = 0; i < 10; i++) {  // Produce 10 items
        item = i + 1;  // Item to produce
        sem_wait(&empty);  // Wait for an empty slot
        pthread_mutex_lock(&mutex);  // Lock critical section

        buffer[in] = item;  // Add item to buffer
        printf("Producer produced: %d\n", item);
        in = (in + 1) % BUFFER_SIZE;

        pthread_mutex_unlock(&mutex);  // Unlock critical section
        sem_post(&full);  // Signal a full slot

        sleep(1);  // Simulate production time
    }
    return NULL;
}
void* consumer(void* arg) {
    int item;
    for (int i = 0; i < 10; i++) {  // Consume 10 items
        sem_wait(&full);  // Wait for a full slot
        pthread_mutex_lock(&mutex);  // Lock critical section

        item = buffer[out];  // Remove item from buffer
        printf("Consumer consumed: %d\n", item);
        out = (out + 1) % BUFFER_SIZE;

        pthread_mutex_unlock(&mutex);  // Unlock critical section
        sem_post(&empty);  // Signal an empty slot

        sleep(1);  // Simulate consumption time
    }
    return NULL;
}
```

```c
int main() {
    pthread_t prod, cons;

    sem_init(&empty, 0, BUFFER_SIZE);  // Initialize to buffer size
    sem_init(&full, 0, 0);  // Initially no full slots
    pthread_mutex_init(&mutex, NULL);

    pthread_create(&prod, NULL, producer, NULL);
    pthread_create(&cons, NULL, consumer, NULL);

    pthread_join(prod, NULL);
    pthread_join(cons, NULL);

    sem_destroy(&empty);
    sem_destroy(&full);
    pthread_mutex_destroy(&mutex);

    return 0;
}
```

Explanation: https://chatgpt.com/c/6776570b-1cc8-8001-96ac-f4f4fa620d93

```c
/* Q8) OPTIONAL TheSleeping-Barber Problem. A barbershop consists of a waiting room with n chairs
 and a barber room with one barber chair. If there are no customers to be served, the barber goes to
 sleep. If a customer enters the barbershop and all chairs are occupied, then the customer leaves the
 shop. If the barber is busy but chairs are available, then the customer sits in one of the free chairs.
If
 the barber is asleep, the customer wakes up the barber. Write a program to coordinate the barber and
 the customers.*/
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>

#define CHAIRS 3
sem_t customers;
sem_t barber;
pthread_mutex_t mutex;
int waiting = 0;

void* barber_function(void* arg) {
    while (1) {
        sem_wait(&customers);
        pthread_mutex_lock(&mutex);
        waiting--;
        printf("Barber is cutting hair. Customers waiting: %d\n", waiting);
        pthread_mutex_unlock(&mutex);
        sem_post(&barber);
        sleep(2);
    }
    return NULL;
}

void* customer_function(void* arg) {
    pthread_mutex_lock(&mutex);
    if (waiting < CHAIRS) {
        waiting++;
        printf("Customer arrived. Customers waiting: %d\n", waiting);
        pthread_mutex_unlock(&mutex);
        sem_post(&customers);
        sem_wait(&barber);
    } else {
        printf("Customer left. No chairs available.\n");
        pthread_mutex_unlock(&mutex);
    }
    return NULL;
}

int main() {
    pthread_t barber_thread;
    pthread_t customers_threads[10];

    sem_init(&customers, 0, 0);
    sem_init(&barber, 0, 0);
    pthread_mutex_init(&mutex, NULL);

    pthread_create(&barber_thread, NULL, barber_function, NULL);

    for (int i = 0; i < 10; i++) {
        pthread_create(&customers_threads[i], NULL, customer_function, NULL);
        sleep(1);
    }

    for (int i = 0; i < 10; i++) {
        pthread_join(customers_threads[i], NULL);
    }

    pthread_cancel(barber_thread);
    sem_destroy(&customers);
    sem_destroy(&barber);
    pthread_mutex_destroy(&mutex);

    return 0;
}
```

Explanation: https://chatgpt.com/c/677657d8-9ac8-8001-b68b-ed0252c8f701

***OPTIONAL* The Cigarette-Smokers Problem.** Consider a system with three smoker processes and one agent process. Each smoker continuously rolls a cigarette and then smokes it. But to roll and smoke a cigarette, the smoker needs three ingredients: tobacco, paper and matches. One of the smoker processes has paper, another has tobacco, and the third has matches. The agent has an infinite supply of all three materials. The agent places two of the ingredients on the table. The smoker who has the remaining ingredient then makes and smokes a cigarette, signaling the agent on completion. The agent then puts out another two of the three ingredients, and the cycle repeats. Write a program to synchronize the agent and the smokers using any synchronization tool.

Explanation: https://chatgpt.com/c/67765885-4134-8001-9564-aa0c60674358

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

sem_t tobacco, paper, matches;  // Semaphores for the three ingredients
sem_t agent;  // Semaphore for the agent

void *smoker(void *ingredient) {
    while (1) {
        if (ingredient == "tobacco") {
            sem_wait(&tobacco);  // Wait for tobacco
        } else if (ingredient == "paper") {
            sem_wait(&paper);  // Wait for paper
```

```c
        } else {
            sem_wait(&matches);  // Wait for matches
        }

        printf("Smoker with %s is smoking\n", (char *)ingredient);
        sleep(2);  // Simulate smoking time
        printf("Smoker with %s finished smoking\n", (char *)ingredient);

        sem_post(&agent);  // Signal the agent to place more ingredients
    }
}

void *agent_process(void *arg) {
    while (1) {
        sem_wait(&agent);  // Wait until a smoker is done smoking

        // Randomly decide which two ingredients to put on the table
        int choice = rand() % 3;
        if (choice == 0) {
            // Agent puts paper and matches
            printf("Agent puts tobacco on the table\n");
            sem_post(&tobacco);
        } else if (choice == 1) {
            // Agent puts tobacco and matches
            printf("Agent puts paper on the table\n");
            sem_post(&paper);
        } else {
            // Agent puts tobacco and paper
            printf("Agent puts matches on the table\n");
            sem_post(&matches);
        }
    }
}

int main() {
    pthread_t smoker1, smoker2, smoker3, agent_thread;

    // Initialize semaphores
    sem_init(&tobacco, 0, 0);
    sem_init(&paper, 0, 0);
    sem_init(&matches, 0, 0);
    sem_init(&agent, 0, 1);

    // Create smoker threads
    pthread_create(&smoker1, NULL, smoker, (void *)"tobacco");
    pthread_create(&smoker2, NULL, smoker, (void *)"paper");
    pthread_create(&smoker3, NULL, smoker, (void *)"matches");

    // Create agent thread
    pthread_create(&agent_thread, NULL, agent_process, NULL);

    // Join all threads
    pthread_join(smoker1, NULL);
    pthread_join(smoker2, NULL);
    pthread_join(smoker3, NULL);
    pthread_join(agent_thread, NULL);

    // Destroy semaphores
    sem_destroy(&tobacco);
    sem_destroy(&paper);
    sem_destroy(&matches);
    sem_destroy(&agent);    return 0;   }
```