QUE 1

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

#define NUM_CHAIRS 5

pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t barber_sleep = PTHREAD_COND_INITIALIZER;

int waiting_customers = 0;

void get_haircut(int customer_id)
{
    pthread_mutex_lock(&mutex);

    if (waiting_customers == 0)
    {
        printf("No customers, barber goes to sleep.\n");
        pthread_cond_wait(&barber_sleep, &mutex);
    }

    waiting_customers++;
    printf("Customer %d takes a seat. %d customer(s) waiting.\n",
customer_id, waiting_customers);

    pthread_mutex_unlock(&mutex);
    pthread_cond_signal(&barber_sleep);

    sleep(rand() % 2 + 1);

    pthread_mutex_lock(&mutex);
    waiting_customers--;
    printf("Barber finishes cutting hair for customer %d. %d
customer(s) waiting.\n", customer_id, waiting_customers);
    pthread_mutex_unlock(&mutex);
}
```

```c
void *customer(void *arg)
{
    int customer_id = *((int *)arg);
    printf("Customer %d arrives at the barber shop.\n", customer_id);
    get_haircut(customer_id);
    return NULL;
}

void *barber(void *arg)
{
    while (1)
    {
        printf("Barber is sleeping.\n");
        pthread_cond_wait(&barber_sleep, &mutex);

        pthread_mutex_lock(&mutex);
        if (waiting_customers == 0)
        {
            printf("Barber wakes up to find no customers.\n");
            pthread_mutex_unlock(&mutex);
            continue;
        }

        pthread_mutex_unlock(&mutex);
        pthread_cond_signal(&barber_sleep);

        printf("Barber is cutting hair for a customer.\n");
        sleep(rand() % 2 + 1);
    }
}

int main()
{
    srand(time(NULL));
    pthread_t barber_thread;
    pthread_create(&barber_thread, NULL, barber, NULL);

    pthread_t customer_threads[10];
```

```
    int i;
    for (i = 0; i < 10; i++)
    {
        pthread_create(&customer_threads[i], NULL, customer, (void
*)&i);
        usleep(rand() % 500000 + 100000);
    }
    pthread_join(barber_thread, NULL);
    for (i = 0; i < 10; i++)
    {
        pthread_join(customer_threads[i], NULL);
    }

    return 0;
}
/*
Barber is sleeping.
Customer 0 arrives at the barber shop.
No customers, barber goes to sleep.
Customer 1 arrives at the barber shop.
No customers, barber goes to sleep.
Customer 2 arrives at the barber shop.
No customers, barber goes to sleep.
Customer 3 arrives at the barber shop.
No customers, barber goes to sleep.
Customer 4 arrives at the barber shop.
No customers, barber goes to sleep.
Customer 5 arrives at the barber shop.
No customers, barber goes to sleep.
Customer 6 arrives at the barber shop.
No customers, barber goes to sleep.
Customer 7 arrives at the barber shop.
No customers, barber goes to sleep.
Customer 8 arrives at the barber shop.
No customers, barber goes to sleep.
Customer 9 arrives at the barber shop.
No customers, barber goes to sleep.
*/
```

QUE 1(B)

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

#define NUM_CHAIRS 5
#define NUM_BARBERS 3

pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t barber_sleep[NUM_BARBERS];

int waiting_customers = 0;

void get_haircut(int customer_id)
{
    pthread_mutex_lock(&mutex);

    for (int i = 0; i < NUM_BARBERS; i++)
    {
        if (waiting_customers == 0)
        {
            printf("No customers, barber %d goes to sleep.\n", i);
            pthread_cond_wait(&barber_sleep[i], &mutex);
            continue;
        }

        waiting_customers++;
        printf("Customer %d takes a seat. %d customer(s) waiting.\n",
customer_id, waiting_customers);

        pthread_mutex_unlock(&mutex);
        pthread_cond_signal(&barber_sleep[i]);

        sleep(rand() % 2 + 1);

        pthread_mutex_lock(&mutex);
        waiting_customers--;
```

```
        printf("Barber %d finishes cutting hair for customer %d. %d
customer(s) waiting.\n", i, customer_id, waiting_customers);
        pthread_mutex_unlock(&mutex);
        pthread_cond_signal(&barber_sleep[i]);

        break;
    }
}

void *customer(void *arg)
{
    int customer_id = *((int *)arg);
    printf("Customer %d arrives at the barber shop.\n", customer_id);
    get_haircut(customer_id);
    return NULL;
}

void *barber(void *arg)
{
    int barber_id = *((int *)arg);

    while (1)
    {
        printf("Barber %d is sleeping.\n", barber_id);
        pthread_cond_wait(&barber_sleep[barber_id], &mutex);

        pthread_mutex_lock(&mutex);
        if (waiting_customers == 0)
        {
            printf("Barber %d wakes up to find no customers.\n",
barber_id);
            pthread_mutex_unlock(&mutex);
            continue;
        }

        pthread_mutex_unlock(&mutex);
        pthread_cond_signal(&barber_sleep[barber_id]);
```

```
        printf("Barber %d is cutting hair for a customer.\n",
barber_id);
        sleep(rand() % 2 + 1);
    }
}

int main()
{
    srand(time(NULL));

    pthread_t barber_threads[NUM_BARBERS];
    pthread_t customer_threads[10];
    int i;

    for (i = 0; i < NUM_BARBERS; i++)
    {
        pthread_cond_init(&barber_sleep[i], NULL);
        pthread_create(&barber_threads[i], NULL, barber, (void *)&i);
    }

    for (i = 0; i < 10; i++)
    {
        pthread_create(&customer_threads[i], NULL, customer, (void
*)&i);
        usleep(rand() % 500000 + 100000);
    }

    for (i = 0; i < NUM_BARBERS; i++)
    {
        pthread_join(barber_threads[i], NULL);
    }
    for (i = 0; i < 10; i++)
    {
        pthread_join(customer_threads[i], NULL);
    }

    return 0;
}
```

```
/*
Barber 0 is sleeping.
Barber 0 is sleeping.
Customer 0 arrives at the barber shop.
No customers, barber 0 goes to sleep.
Barber 0 is sleeping.
Customer 1 arrives at the barber shop.
No customers, barber 0 goes to sleep.
Customer 2 arrives at the barber shop.
No customers, barber 0 goes to sleep.
Customer 3 arrives at the barber shop.
No customers, barber 0 goes to sleep.
Customer 4 arrives at the barber shop.
No customers, barber 0 goes to sleep.
Customer 5 arrives at the barber shop.
No customers, barber 0 goes to sleep.
Customer 6 arrives at the barber shop.
No customers, barber 0 goes to sleep.
Customer 7 arrives at the barber shop.
No customers, barber 0 goes to sleep.
Customer 8 arrives at the barber shop.
No customers, barber 0 goes to sleep.
Customer 9 arrives at the barber shop.
No customers, barber 0 goes to sleep.
*/
```

QUE 2.

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

sem_t agent_sem, tobacco, paper, matches;

void *agent(void *arg)
```

```
{
    while (1)
    {
        sem_wait(&agent_sem);

        // Place two random ingredients on the table
        int rand_num = rand() % 3;
        int rand_num2 = (rand_num + 1) % 3;

        if (rand_num == 0 && rand_num2 == 1)
        {
            printf("Agent places tobacco and paper on the table.\n");
        }
        else if (rand_num == 0 && rand_num2 == 2)
        {
            printf("Agent places tobacco and matches on the
table.\n");
        }
        else
        {
            printf("Agent places paper and matches on the table.\n");
        }

        sem_post(&tobacco);
        sem_post(&paper);
        sem_post(&matches);
    }
}

void *smoker_tobacco(void *arg)
{
    while (1)
    {
        sem_wait(&tobacco);
        sem_wait(&paper);
        printf("Smoker with tobacco rolls and smokes a cigarette.\n");
        sem_post(&agent_sem);
    }
}
```

```c
void *smoker_paper(void *arg)
{
    while (1)
    {
        sem_wait(&paper);
        sem_wait(&matches);
        printf("Smoker with paper rolls and smokes a cigarette.\n");
        sem_post(&agent_sem);
    }
}

void *smoker_matches(void *arg)
{
    while (1)
    {
        sem_wait(&tobacco);
        sem_wait(&matches);
        printf("Smoker with matches rolls and smokes a cigarette.\n");
        sem_post(&agent_sem);
    }
}

int main()
{
    sem_init(&agent_sem, 0, 1);
    sem_init(&tobacco, 0, 0);
    sem_init(&paper, 0, 0);
    sem_init(&matches, 0, 0);

    pthread_t agent_thread, smoker_tobacco_thread,
smoker_paper_thread, smoker_matches_thread;

    pthread_create(&agent_thread, NULL, agent, NULL);
    pthread_create(&smoker_tobacco_thread, NULL, smoker_tobacco,
NULL);
    pthread_create(&smoker_paper_thread, NULL, smoker_paper, NULL);
    pthread_create(&smoker_matches_thread, NULL, smoker_matches,
NULL);
```

```
    pthread_join(agent_thread, NULL);
    pthread_join(smoker_tobacco_thread, NULL);
    pthread_join(smoker_paper_thread, NULL);
    pthread_join(smoker_matches_thread, NULL);

    sem_destroy(&agent_sem);
    sem_destroy(&tobacco);
    sem_destroy(&paper);
    sem_destroy(&matches);

    return 0;
}
/*
Agent places tobacco and paper on the table.
Agent places paper and matches on the table.
Agent places paper and matches on the table.
Agent places paper and matches on the table.
Agent places paper and matches on the table.
Agent places paper and matches on the table.
Agent places paper and matches on the table.
Agent places paper and matches on the table.
Agent places paper and matches on the table.
Agent places paper and matches on the table.
Agent places paper and matches on the table.
Agent places tobacco and paper on the table.
Smoker with tobacco rolls and smokes a cigarette.
Smoker with tobacco rolls and smokes a cigarette.
Smoker with tobacco rolls and smokes a cigarette.
Smoker with tobacco rolls and smokes a cigarette.
Smoker with tobacco rolls and smokes a cigarette.
Smoker with tobacco rolls and smokes a cigarette.
Smoker with tobacco rolls and smokes a cigarette.
Smoker with tobacco rolls and smokes a cigarette.
Smoker with tobacco rolls and smokes a cigarette.
Smoker with tobacco rolls and smokes a cigarette.
Smoker with tobacco rolls and smokes a cigarette.
Smoker with tobacco rolls and smokes a cigarette.
Smoker with tobacco rolls and smokes a cigarette.
```

```
Smoker with tobacco rolls and smokes a cigarette.
Smoker with tobacco rolls and smokes a cigarette.
Smoker with paper rolls and smokes a cigarette.
Smoker with paper rolls and smokes a cigarette.
Smoker with paper rolls and smokes a cigarette.
Smoker with paper rolls and smokes a cigarette.
Smoker with paper rolls and smokes a cigarette.
Smoker with paper rolls and smokes a cigarette.
Smoker with paper rolls and smokes a cigarette.
Smoker with paper rolls and smokes a cigarette.
Smoker with paper rolls and smokes a cigarette.
Smoker with paper rolls and smokes a cigarette.
Smoker with paper rolls and smokes a cigarette.
Smoker with paper rolls and smokes a cigarette.
Smoker with paper rolls and smokes a cigarette.
Smoker with paper rolls and smokes a cigarette.
Smoker with paper rolls and smokes a cigarette.
Smoker with paper rolls and smokes a cigarette.
Smoker with paper rolls and smokes a cigarette.
Smoker with paper rolls and smokes a cigarette.
Smoker with paper rolls and smokes a cigarette.
Smoker with paper rolls and smokes a cigarette.
Smoker with paper rolls and smokes a cigarette.
Smoker with paper rolls and smokes a cigarette.
Smoker with paper rolls and smokes a cigarette.
Smoker with paper rolls and smokes a cigarette.
Smoker with paper rolls and smokes a cigarette.
Smoker with paper rolls and smokes a cigarette.
*/
```

QUE 3.

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

#define NUM_PHILOSOPHERS 5

typedef enum
{
    THINKING,
    HUNGRY,
    EATING
} State;

pthread_mutex_t mutex;
pthread_cond_t condition[NUM_PHILOSOPHERS];
State philosopherStates[NUM_PHILOSOPHERS];

void grab_forks(int philosopher_id)
{
    pthread_mutex_lock(&mutex);
    philosopherStates[philosopher_id] = HUNGRY;
    test(philosopher_id);
    while (philosopherStates[philosopher_id] != EATING)
    {
        pthread_cond_wait(&condition[philosopher_id], &mutex);
    }
    pthread_mutex_unlock(&mutex);
}

void put_away_forks(int philosopher_id)
{
    pthread_mutex_lock(&mutex);
    philosopherStates[philosopher_id] = THINKING;
    test((philosopher_id + 4) % NUM_PHILOSOPHERS);
    test((philosopher_id + 1) % NUM_PHILOSOPHERS);
    pthread_mutex_unlock(&mutex);
```

```
}

void test(int philosopher_id)
{
    if (philosopherStates[(philosopher_id + 4) % NUM_PHILOSOPHERS] !=
EATING &&
        philosopherStates[philosopher_id] == HUNGRY &&
        philosopherStates[(philosopher_id + 1) % NUM_PHILOSOPHERS] !=
EATING)
    {

        philosopherStates[philosopher_id] = EATING;
        pthread_cond_signal(&condition[philosopher_id]);
    }
}

void *philosopher(void *arg)
{
    int philosopher_id = *((int *)arg);

    while (1)
    {
        printf("Philosopher %d is thinking.\n", philosopher_id);
        sleep(rand() % 3);

        grab_forks(philosopher_id);

        printf("Philosopher %d is eating.\n", philosopher_id);
        sleep(rand() % 3);

        put_away_forks(philosopher_id);
    }
}

int main()
{
    pthread_t philosophers[NUM_PHILOSOPHERS];
    int philosopher_ids[NUM_PHILOSOPHERS];
```

```
    pthread_mutex_init(&mutex, NULL);

    for (int i = 0; i < NUM_PHILOSOPHERS; ++i)
    {
        pthread_cond_init(&condition[i], NULL);
        philosopher_ids[i] = i;
        pthread_create(&philosophers[i], NULL, philosopher, (void
*)&philosopher_ids[i]);
    }

    for (int i = 0; i < NUM_PHILOSOPHERS; ++i)
    {
        pthread_join(philosophers[i], NULL);
        pthread_cond_destroy(&condition[i]);
    }

    pthread_mutex_destroy(&mutex);

    return 0;
}
/*
Philosopher 0 is thinking.
Philosopher 1 is thinking.
Philosopher 2 is thinking.
Philosopher 4 is thinking.
Philosopher 0 is thinking.
Philosopher 1 is eating.
Philosopher 1 is thinking.
Philosopher 0 is eating.
Philosopher 0 is thinking.
Philosopher 0 is eating.
Philosopher 3 is thinking.
Philosopher 2 is eating.
Philosopher 0 is thinking.
*/
```