ASSIGNMENT 8
07/11/23

NAME : SHRESTH SONKAR
REGNO : 20214272
GROUP : CS5D
TOPIC : OS LAB
CODE : CS-15203

```c
/*
 * Q1a :
 * Write a program to synchronize the sleeping barber
problem to prevent any race conditions.
 */

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

#define N 5

sem_t *barberSem, *customerSem, *accessSeatsSem,
*barberSleepSem;
int waitingCustomers = 0;

void *barber(void *arg) {
    while (1) {
        sem_wait(barberSem);
        sem_wait(accessSeatsSem);

        waitingCustomers--;

        sem_post(barberSleepSem);
        sem_post(accessSeatsSem);

        printf("\n\tBarber is cutting hair.\n\n");
        sleep(2);
        printf("\n\tBarber finished cutting hair.
\n\n");
    }
}

void *customer(void *arg) {
    sem_wait(accessSeatsSem);

    if (waitingCustomers < N) {
        waitingCustomers++;
        printf("\n\tCustomer entered the waiting room.
\n\tTotal customers waiting: %d\n\n",
waitingCustomers);

        sem_post(barberSem);
```

```
        sem_post(accessSeatsSem);

        sem_wait(barberSleepSem);
        printf("\n\tCustomer is getting a haircut.
\n\n");
    } else {

        printf("\n\tNo available seats.\n\tCustomer is
leaving.\n\n");
        sem_post(accessSeatsSem);
    }
}

int main() {
    pthread_t barberThread, customerThread;

    barberSem = sem_open("/barberSem", O_CREAT, 0644,
0);
    customerSem = sem_open("/customerSem", O_CREAT,
0644, 0);
    accessSeatsSem = sem_open("/accessSeatsSem",
O_CREAT, 0644, 1);
    barberSleepSem = sem_open("/barberSleepSem",
O_CREAT, 0644, 0);

    pthread_create(&barberThread, NULL, barber, NULL);

    for (int i = 0; i < 10; i++) {
        pthread_create(&customerThread, NULL, customer,
NULL);
        sleep(1);
    }

    pthread_join(barberThread, NULL);

    sem_close(barberSem);
    sem_close(customerSem);
    sem_close(accessSeatsSem);
    sem_close(barberSleepSem);

    sem_unlink("/barberSem");
    sem_unlink("/customerSem");
    sem_unlink("/accessSeatsSem");
    sem_unlink("/barberSleepSem");
```

```
    return 0;
}

/*
 * Q1b :
 * Consider the Sleeping-Barber Problem with the
modification that there are k barbers and k barber
chairs
 * Write a program to coordinate the barbers and the
customers.
 */

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

#define N 5
#define K 3

sem_t *customerSem, *barberSem, *accessSeatsSem,
*barberSleepSem;
int waitingCustomers = 0;

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

    while (1) {
        sem_wait(customerSem);
        sem_wait(accessSeatsSem);

        if (waitingCustomers > 0) {
            waitingCustomers--;

            sem_post(barberSem);
            sem_post(accessSeatsSem);


            printf("\n\tBarber %d is cutting hair.\n",
id);
            sleep(2);
```

```
            printf("\n\tBarber %d finished cutting
hair.\n", id);
        } else {
            sem_post(accessSeatsSem);
            printf("\n\tBarber %d is sleeping.\n", id);
            sem_wait(barberSleepSem);
        }
    }
}

void *customer(void *arg) {
    sem_wait(accessSeatsSem);

    if (waitingCustomers < N) {
        waitingCustomers++;
        printf("\n\tCustomer entered the waiting room.
\n\tTotal customers waiting: %d\n", waitingCustomers);

        sem_post(customerSem);
        sem_post(accessSeatsSem);

        sem_wait(barberSem);
        printf("\n\tCustomer is getting a haircut.\n");
    } else {
        printf("\n\tNo available seats.\n\tCustomer is
leaving.\n");
        sem_post(accessSeatsSem);
    }
}

int main() {
    pthread_t barberThreads[K], customerThread;
    int barberIDs[K];

    customerSem = sem_open("/customerSem", O_CREAT,
0644, 0);
    barberSem = sem_open("/barberSem", O_CREAT, 0644,
0);
    accessSeatsSem = sem_open("/accessSeatsSem",
O_CREAT, 0644, 1);
    barberSleepSem = sem_open("/barberSleepSem",
O_CREAT, 0644, 0);

    for (int i = 0; i < K; i++) {
        barberIDs[i] = i + 1;
```

```
        pthread_create(&barberThreads[i], NULL, barber,
&barberIDs[i]);
    }

    for (int i = 0; i < 10; i++) {
        pthread_create(&customerThread, NULL, customer,
NULL);
        sleep(1);
    }

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

    sem_close(customerSem);
    sem_close(barberSem);
    sem_close(accessSeatsSem);
    sem_close(barberSleepSem);

    sem_unlink("/customerSem");
    sem_unlink("/barberSem");
    sem_unlink("/accessSeatsSem");
    sem_unlink("/barberSleepSem");

    return 0;
}
```

```c
/*
 * Q2 :
 * 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, signalling 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.
 */

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

sem_t *agentSem, *tobaccoSem, *paperSem, *matchesSem;

void *agent(void *arg) {
    while (1) {
        sem_wait(agentSem);

        int random = rand() % 3;
        if (random == 0) {
            sem_post(tobaccoSem);
            sem_post(paperSem);
            printf("Agent placed tobacco and paper on the table.\n");
        } else if (random == 1) {
            sem_post(paperSem);
            sem_post(matchesSem);
```

```
            printf("Agent placed paper and matches on
the table.\n");
        } else {
            sem_post(tobaccoSem);
            sem_post(matchesSem);
            printf("Agent placed tobacco and matches on
the table.\n");
        }
    }
}

void *smoker_tobacco(void *arg) {
    while (1) {
        sem_wait(tobaccoSem);
        sem_wait(paperSem);
        printf("Smoker with tobacco is rolling and
smoking the cigarette.\n");
        sem_post(agentSem);
    }
}

void *smoker_paper(void *arg) {
    while (1) {
        sem_wait(paperSem);
        sem_wait(matchesSem);
        printf("Smoker with paper is rolling and
smoking the cigarette.\n");
        sem_post(agentSem);
    }
}

void *smoker_matches(void *arg) {
    while (1) {
        sem_wait(tobaccoSem);
        sem_wait(matchesSem);
        printf("Smoker with matches is rolling and
smoking the cigarette.\n");
        sem_post(agentSem);
    }
}

int main() {
    pthread_t agentThread, smokerTobaccoThread,
smokerPaperThread, smokerMatchesThread;
```

```
    agentSem = sem_open("/agentSem", O_CREAT, 0644, 1);
    tobaccoSem = sem_open("/tobaccoSem", O_CREAT, 0644,
0);
    paperSem = sem_open("/paperSem", O_CREAT, 0644, 0);
    matchesSem = sem_open("/matchesSem", O_CREAT, 0644,
0);

    pthread_create(&agentThread, NULL, agent, NULL);
    pthread_create(&smokerTobaccoThread, NULL,
smoker_tobacco, NULL);
    pthread_create(&smokerPaperThread, NULL,
smoker_paper, NULL);
    pthread_create(&smokerMatchesThread, NULL,
smoker_matches, NULL);

    pthread_join(agentThread, NULL);
    pthread_join(smokerTobaccoThread, NULL);
    pthread_join(smokerPaperThread, NULL);
    pthread_join(smokerMatchesThread, NULL);

    sem_close(agentSem);
    sem_close(tobaccoSem);
    sem_close(paperSem);
    sem_close(matchesSem);

    sem_unlink("/agentSem");
    sem_unlink("/tobaccoSem");
    sem_unlink("/paperSem");
    sem_unlink("/matchesSem");

    return 0;
}
```

```c
/*
 * Q3 :
 * Implement a solution to the Dining Philosophers
problem using a monitor.
 */

#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>
#include <sys/wait.h>
#include <sys/types.h>
#include <semaphore.h>
#include <sys/signal.h>

#define N 7
#define THINKING 0
#define HUNGRY 1
#define EATING 2
#define LEFT (i + N - 1) % N
#define RIGHT (i + 1) % N

void initialization();

void test(int i);

void take_chopsticks(int i);

void put_chopsticks(int i);

sem_t *mutex;
sem_t *next;

int next_count = 0;
int state[N];
int turn[N];

typedef struct {
    sem_t *sem;

    int count;
} condition;

condition x[N];
```

```
void wait_cust(int i) {
    x[i].count++;
    if (next_count > 0)
        sem_post(next);
    else
        sem_post(mutex);
    sem_wait(x[i].sem);
    x[i].count--;
}

void signal_cust(int i) {
    if (x[i].count > 0) {
        next_count++;
        sem_post(x[i].sem);
        sem_wait(next);
        next_count--;
    }
}

void test(int i) {
    if (state[i] == HUNGRY && state[LEFT] != EATING &&
state[RIGHT] != EATING && turn[i] == i && turn[LEFT] ==
i) {
        state[i] = EATING;
        signal_cust(i);
    }
}

void take_chopsticks(int i) {
    sem_wait(mutex);
    state[i] = HUNGRY;
    test(i);

    while (state[i] == HUNGRY)
        wait_cust(i);

    if (next_count > 0)
        sem_post(next);
    else
        sem_post(mutex);

}

void put_chopsticks(int i) {
```

```
    sem_wait(mutex);
    state[i] = THINKING;

    turn[i] = RIGHT;
    turn[LEFT] = LEFT;

    test(LEFT);
    test(RIGHT);

    if (next_count > 0)
        sem_post(next);
    else
        sem_post(mutex);
}

void initialization() {
    int i;

    mutex = sem_open("/mutex", O_CREAT | O_EXCL, 0644,
1);
    next = sem_open("/next", O_CREAT | O_EXCL, 0644,
0);

    for (i = 0; i < N; i++) {
        state[i] = THINKING;
        char sem_name[20];
        snprintf(sem_name, sizeof(sem_name), "/sem_%d",
i);
        x[i].sem = sem_open(sem_name, O_CREAT | O_EXCL,
0644, 0);
        x[i].count = 0;
        turn[i] = i;
    }

    turn[1] = 2;
    turn[3] = 4;
    turn[6] = 0;
}

void cleanup() {
    int i;

    sem_close(mutex);
    sem_close(next);
```

```
    for (i = 0; i < N; i++) {
        char sem_name[20];
        snprintf(sem_name, sizeof(sem_name), "/sem_%d",
i);
        sem_close(x[i].sem);
        sem_unlink(sem_name);
    }

    sem_unlink("/mutex");
    sem_unlink("/next");
}

void *philosopher(void *i) {
    while (1) {
        int self = *(int *) i;
        int j, k;

        j = rand();
        j = j % 11;

        printf("\n\tPhilosopher %d is thinking for %d
secs\n", self, j);
        sleep(j);
        take_chopsticks(self);

        k = rand();
        k = k % 4;

        printf("\n\tPhilosopher %d is eating for %d
secs\n", self, k);
        sleep(k);
        put_chopsticks(self);
    }
}

int main() {
    int i, pos[N];
    pthread_t thread[N];
    pthread_attr_t attr;

    initialization();
    pthread_attr_init(&attr);

    for (i = 0; i < N; i++) {
        pos[i] = i;
```

```c
        pthread_create(&thread[i], NULL, philosopher,
    (int *) &pos[i]);
    }

    for (i = 0; i < N; i++)
        pthread_join(thread[i], NULL);
    cleanup();
    return 0;
}
```


.../sem5/os/2023-11-07
→ ~/desktop/cse/ASSGN/sem5/os/2023-11-07 $ clang q2.c -o q2 2> /dev/null
→ ~/desktop/cse/ASSGN/sem5/os/2023-11-07 $ ./q2
Agent placed paper and matches on the table.
Smoker with paper is rolling and smoking the cigarette.
Agent placed paper and matches on the table.
Smoker with paper is rolling and smoking the cigarette.
Agent placed tobacco and matches on the table.
^C
→ ~/desktop/cse/ASSGN/sem5/os/2023-11-07 $


.../sem5/os/2023-11-07
→ ~/desktop/cse/ASSGN/sem5/os/2023-11-07 $ clang q3.c -o q3
→ ~/desktop/cse/ASSGN/sem5/os/2023-11-07 $ ./q3
Philosopher 0 is thinking for 10 secs
Philosopher 1 is thinking for 1 secs
Philosopher 2 is thinking for 0 secs
Philosopher 2 is eating for 2 secs
Philosopher 5 is thinking for 8 secs
Philosopher 3 is thinking for 3 secs
Philosopher 6 is thinking for 2 secs
Philosopher 4 is thinking for 0 secs
Philosopher 4 is eating for 3 secs
Philosopher 2 is thinking for 4 secs
Philosopher 3 is eating for 0 secs
Philosopher 3 is thinking for 10 secs
Philosopher 4 is thinking for 6 secs
Philosopher 5 is eating for 2 secs
Philosopher 0 is eating for 3 secs
Philosopher 4 is eating for 3 secs
Philosopher 5 is thinking for 1 secs
Philosopher 6 is eating for 1 secs
Philosopher 1 is eating for 0 secs
Philosopher 1 is thinking for 6 secs
Philosopher 2 is eating for 1 secs
Philosopher 0 is thinking for 4 secs
Philosopher 4 is thinking for 1 secs
Philosopher 5 is eating for 1 secs
Philosopher 6 is thinking for 4 secs
Philosopher 3 is eating for 1 secs
Philosopher 2 is thinking for 1 secs
Philosopher 5 is thinking for 5 secs
Philosopher 3 is thinking for 0 secs
Philosopher 4 is eating for 0 secs
Philosopher 4 is thinking for 8 secs
^C
→ ~/desktop/cse/ASSGN/sem5/os/2023-11-07 $