Synchronization Problems

- **Producer-Consumer Problem Using Mutex**

Code:

```
#include <pthread.h>

#include <semaphore.h>

#include <stdlib.h>

#include <stdio.h>


#define MaxItems 5

#define BufferSize 5


sem_t empty;

sem_t full;

int in = 0;

int out = 0;

int buffer[BufferSize];

pthread_mutex_t mutex;


void *producer(void *pno)

{

        int item;
```

```
        for(int i = 0; i < MaxItems; i++) {
        item = rand();
        sem_wait(&empty);
        pthread_mutex_lock(&mutex);
           buffer[in] = item;
        printf("Producer %d: Insert Item %d at %d\n", *((int *)pno),buffer[in],in);
        in = (in+1)%BufferSize;
        pthread_mutex_unlock(&mutex);
        sem_post(&full);
        }
}

void *consumer(void *cno)
{
        for(int i = 0; i < MaxItems; i++) {
        sem_wait(&full);
        pthread_mutex_lock(&mutex);
        int item = buffer[out];
        printf("Consumer %d: Remove Item %d from %d\n",*((int *)cno),item, out);
        out = (out+1)%BufferSize;
        pthread_mutex_unlock(&mutex);
        sem_post(&empty);
        }
```

```
}

int main()
{

        pthread_t pro[5],con[5];
        pthread_mutex_init(&mutex, NULL);
        sem_init(&empty,0,BufferSize);
        sem_init(&full,0,0);

        int a[5] = {1,2,3,4,5};

        for(int i = 0; i < 5; i++) {
        pthread_create(&pro[i], NULL, (void *)producer, (void *)&a[i]);
        }
        for(int i = 0; i < 5; i++) {
        pthread_create(&con[i], NULL, (void *)consumer, (void *)&a[i]);
        }

        for(int i = 0; i < 5; i++) {
        pthread_join(pro[i], NULL);
        }
        for(int i = 0; i < 5; i++) {
```

```
        pthread_join(con[i], NULL);
        }

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

        return 0;

}
```

OUTPUT:

```
Producer 4: Insert Item 41 at 4
Consumer 3: Remove Item 41 from 0
Consumer 3: Remove Item 18467 from 1
Consumer 1: Remove Item 41 from 2
Producer 2: Insert Item 18467 at 0
Consumer 3: Remove Item 41 from 3
Consumer 4: Remove Item 41 from 4
Producer 4: Insert Item 18467 at 1
Consumer 5: Remove Item 18467 from 0
Producer 1: Insert Item 6334 at 2
Consumer 1: Remove Item 18467 from 1
Producer 5: Insert Item 41 at 3
Consumer 2: Remove Item 6334 from 2
Producer 2: Insert Item 6334 at 4
Consumer 4: Remove Item 41 from 3
Producer 5: Insert Item 18467 at 0
Consumer 3: Remove Item 6334 from 4
Producer 1: Insert Item 26500 at 1
Producer 3: Insert Item 18467 at 2
Producer 4: Insert Item 6334 at 3
Consumer 5: Remove Item 18467 from 0
Producer 2: Insert Item 26500 at 4
Consumer 1: Remove Item 26500 from 1
Consumer 2: Remove Item 18467 from 2
Consumer 4: Remove Item 6334 from 3
```

- **Producer-Consumer using Semaphore:**

Code:

```
#include <stdio.h>

#include <stdlib.h>


#define N 5

int semaphore = 1, full = 0, empty = N, x = 0;


int wait(int s){

	return (--s);

}


int signal(int s){

	return(++s);

}


void producer()

{

	semaphore = wait(semaphore);

	full = signal(full);

	empty = wait(empty);


	x++;

	printf("Producer produces the item : %d\n", x);
```

```
        semaphore = signal(semaphore);
}

void consumer(){
        semaphore = wait(semaphore);
        full = wait(full);
        empty = signal(empty);

        printf("Consumer consumes the item : %d\n",x);
        x--;

        semaphore = signal(semaphore);
}

int main(){
        int n;

        printf("1. Producer\n2.Consumer\n3. Exit\n");

        while(1){
        printf("\nEnter your choice : ");
        scanf("%d", &n);
```

```
switch(n){

case 1:
        if((semaphore == 1) && (empty != 0))
        producer();
        else
        printf("Buffer is full !!\n");
        break;

case 2:
        if((semaphore == 1) && (full != 0))
      consumer();
        else
        printf("Buffer is empty !!\n");
        break;

case 3:
        exit(0);
        break;
default:
        printf("Wrong choice....choose again !!\n");
}
}
```

```
        return 0;
}
```

OUTPUT:

```
Enter your choice : 1
Producer produces the item : 1

Enter your choice : 2
Consumer consumes the item : 1

Enter your choice : 1
Producer produces the item : 1

Enter your choice : 2
Consumer consumes the item : 1

Enter your choice : 2
Buffer is empty !!

Enter your choice : 1
Producer produces the item : 1

Enter your choice : 1
Producer produces the item : 2

Enter your choice : 1
Producer produces the item : 3

Enter your choice : 1
Producer produces the item : 4

Enter your choice : 2
Consumer consumes the item : 4

Enter your choice : 2
Consumer consumes the item : 3

Enter your choice : 2
Consumer consumes the item : 2

Enter your choice : 2
Consumer consumes the item : 1

Enter your choice : 2
Buffer is empty !!
```

Reader-Writer Problem Using Mutex:

Code:

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

```
#include <semaphore.h>

#include <stdlib.h>

#include <stdio.h>


#define MaxItems 5

#define BufferSize 5


sem_t empty;

sem_t full;

int in = 0;

int out = 0;

int buffer[BufferSize];

pthread_mutex_t mutex;


void *producer(void *pno)

{

	int item;

	for(int i = 0; i < MaxItems; i++) {

	item = rand();

	sem_wait(&empty);

	pthread_mutex_lock(&mutex);

	buffer[in] = item;

	printf("Producer %d: Insert Item %d at %d\n", *((int *)pno),buffer[in],in);
```

```
        in = (in+1)%BufferSize;
        pthread_mutex_unlock(&mutex);
        sem_post(&full);
        }
}

void *consumer(void *cno)
{
        for(int i = 0; i < MaxItems; i++) {
        sem_wait(&full);
        pthread_mutex_lock(&mutex);
        int item = buffer[out];
      printf("Consumer %d: Remove Item %d from %d\n",*((int *)cno),item, out);
        out = (out+1)%BufferSize;
        pthread_mutex_unlock(&mutex);
        sem_post(&empty);
        }
}
 int main()
{

        pthread_t pro[5],con[5];
        pthread_mutex_init(&mutex, NULL);
```

```
sem_init(&empty,0,BufferSize);
sem_init(&full,0,0);

int a[5] = {1,2,3,4,5};

for(int i = 0; i < 5; i++) {
pthread_create(&pro[i], NULL, (void *)producer, (void *)&a[i]);
}
for(int i = 0; i < 5; i++) {
pthread_create(&con[i], NULL, (void *)consumer, (void *)&a[i]);
}

for(int i = 0; i < 5; i++) {
pthread_join(pro[i], NULL);
}
for(int i = 0; i < 5; i++) {
pthread_join(con[i], NULL);
}

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

```
    return 0;

}
```

OUTPUT:

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

PS D:\Sem 4\DS\LAB\Assignments> cd "d:\Sem 4\DS\LAB\Assignments\" ; if ($?) { gcc code.c -o code } ; if ($?) { .\code }
Producer 1: Insert Item 41 at 0
Producer 2: Insert Item 41 at 1
Producer 3: Insert Item 41 at 2
Producer 4: Insert Item 41 at 3
Producer 5: Insert Item 41 at 4
Consumer 1: Remove Item 41 from 0
Consumer 2: Remove Item 41 from 1
Consumer 3: Remove Item 41 from 2
Consumer 4: Remove Item 41 from 3
Consumer 5: Remove Item 41 from 4
Producer 1: Insert Item 18467 at 0
Producer 2: Insert Item 18467 at 1
Producer 3: Insert Item 18467 at 2
Producer 4: Insert Item 18467 at 3
```

- **Reader Writer Problem using Semaphore:**

  Code

```
#include <pthread.h>

#include <semaphore.h>

#include <stdlib.h>

#include <stdio.h>


#define MaxItems 5

#define BufferSize 5


sem_t empty;

sem_t full;

int in = 0;

int out = 0;

int buffer[BufferSize];

pthread_mutex_t mutex;


void *producer(void *pno)

{


        int item;

        for(int i = 0; i < MaxItems; i++) {

        item = rand();

        sem_wait(&empty);

        pthread_mutex_lock(&mutex);
```

```
        buffer[in] = item;
        printf("Producer %d: Insert Item %d at %d\n", *((int *)pno),buffer[in],in);
        in = (in+1)%BufferSize;
        pthread_mutex_unlock(&mutex);
        sem_post(&full);
        }
}

void *consumer(void *cno)
{

        for(int i = 0; i < MaxItems; i++) {
        sem_wait(&full);
        pthread_mutex_lock(&mutex);
        int item = buffer[out];
        printf("Consumer %d: Remove Item %d from %d\n",*((int *)cno),item, out);
        out = (out+1)%BufferSize;
        pthread_mutex_unlock(&mutex);
        sem_post(&empty);
        }
}

int main()
```

```
{

    pthread_t pro[5],con[5];
    pthread_mutex_init(&mutex, NULL);
    sem_init(&empty,0,BufferSize);
    sem_init(&full,0,0);

    int a[5] = {1,2,3,4,5};

    for(int i = 0; i < 5; i++) {
    pthread_create(&pro[i], NULL, (void *)producer, (void *)&a[i]);
    }

    for(int i = 0; i < 5; i++) {
    pthread_create(&con[i], NULL, (void *)consumer, (void *)&a[i]);
    }

    for(int i = 0; i < 5; i++) {
    pthread_join(pro[i], NULL);
    }

    for(int i = 0; i < 5; i++) {
    pthread_join(con[i], NULL);
```

```
        }

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

        return 0;

}
```

OUTPUT:

Dining Philosopher Problem:
Code:

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

```
#include <semaphore.h>

#include <stdio.h>


#define N 5

#define THINKING 2

#define HUNGRY 1

#define EATING 0

#define LEFT (phnum + 4) % N

#define RIGHT (phnum + 1) % N


int state[N];

int phil[N] = { 0, 1, 2, 3, 4 };


sem_t mutex;

sem_t S[N];


void test(int phnum)

{

	if (state[phnum] == HUNGRY

		&& state[LEFT] != EATING

		&& state[RIGHT] != EATING)

		{
```

```
                state[phnum] = EATING;

                sleep(2);

                printf("Philosopher %d takes fork %d and %d\n",
                                phnum + 1, LEFT + 1, phnum + 1);

                printf("Philosopher %d is Eating\n", phnum + 1);

                sem_post(&S[phnum]);
        }
}

void take_fork(int phnum)
{

        sem_wait(&mutex);
        state[phnum] = HUNGRY;
        printf("Philosopher %d is Hungry\n", phnum + 1);
        test(phnum);
        sem_post(&mutex);
        sem_wait(&S[phnum]);
        sleep(1);
```

```
}

void put_fork(int phnum)
{

        sem_wait(&mutex);
        state[phnum] = THINKING;
        printf("Philosopher %d putting fork %d and %d down\n",
                phnum + 1, LEFT + 1, phnum + 1);
        printf("Philosopher %d is thinking\n", phnum + 1);

        test(LEFT);
        test(RIGHT);

        sem_post(&mutex);
}

void* philosopher(void* num)
{

        while (1) {

                int* i = num;
```

```
            sleep(1);

            take_fork(*i);

            sleep(0);

            put_fork(*i);
        }
}

int main()
{

        int i;
        pthread_t thread_id[N];

        sem_init(&mutex, 0, 1);

        for (i = 0; i < N; i++)

            sem_init(&S[i], 0, 0);
```

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

        pthread_create(&thread_id[i], NULL,
                                    philosopher, &phil[i]);

        printf("Philosopher %d is thinking\n", i + 1);
    }

    for (i = 0; i < N; i++)

        pthread_join(thread_id[i], NULL);
}
```

OUTPUT:

```
Philosopher 2 is thinking
Philosopher 3 is thinking
Philosopher 4 is thinking
Philosopher 5 is thinking
Philosopher 4 is Hungry
Philosopher 5 is Hungry
Philosopher 3 is Hungry
Philosopher 2 is Hungry
Philosopher 1 is Hungry
Philosopher 1 takes fork 5 and 1
Philosopher 1 is Eating
Philosopher 1 putting fork 5 and 1 down
Philosopher 1 is thinking
Philosopher 5 takes fork 4 and 5
Philosopher 5 is Eating
```