PROCESS SYNCHRONIZATION 2

BCSE303P Operating Systems

Name: Chillara V L N S Pavana Vamsi
Register Number: 21BCE5095

1. Maths teacher gives homework to the students. There are twenty students in the class. She has a box with five divisions in it. In that division she will write a integer number. Any student who is ready, may get the number and display the number name of it. At the maximum, teacher can give 5 numbers in 5 divisions. The same way student can consume 5 integer numbers which in turn will be displayed as number names. Develop a C program for this scenario and justify the output generated by your code.

Code:

```c
#include <stdio.h>
#include <semaphore.h>
#include <pthread.h>
#include <stdbool.h>
#include <unistd.h>
#include <stdlib.h>
int readcount = 1, teachercount = 0, i = 0, j = 0, k = 0, arr[20];
sem_t teach, mutex;
void *teacher()
{
    sem_wait(&teach);
    printf("Teacher entry to critical section.\n");
    printf("Data by teacher:\n");
    for (i = 0; i < 5; i++)
    {
        arr[j++] = rand() % 100;
        printf("%d ", arr[j - 1]);
    }
    sleep(1);
    sem_post(&teach);
    printf("\nTeacher exit to critical section.\n");
}
void *student()
{
    readcount++;
    if (readcount == 1)
    {
        sem_wait(&teach);
    }
    sem_post(&mutex);
    printf("Student entry to critical section.\n");
    printf("Data displyed:\n");
    for (i = 0; i < 5; i++)
    {
        printf("%d ", arr[k++]);
    }
    printf("\n");
    sleep(1);
    sem_wait(&mutex);
    printf("Student exit critical section.\n");
    readcount--;
    if (readcount == 0)
    {
        sem_post(&teach);
    }
    sem_post(&mutex);
```

```
}
int main()
{
    sem_init(&mutex, 0, 0);
    sem_init(&teach, 0, 1);
    pthread_t t[10];
    for (int i = 0; i < 5; i++)
    {
        pthread_create(&t[0], NULL, teacher, NULL);
        sleep(2);
        pthread_create(&t[1], NULL, student, NULL);
        sleep(2);
        pthread_join(t[0], NULL);
        pthread_join(t[1], NULL);
    }
    sem_destroy(&mutex);
    sem_destroy(&teach);
    return 0;
}
```

Output:

```
Teacher entry to critical section.
Data by teacher:
41 67 34 0 69
Teacher exit to critical section.
Student entry to critical section.
Data displyed:
41 67 34 0 69
Student exit critical section.
Teacher entry to critical section.
Data by teacher:
41 67 34 0 69
Teacher exit to critical section.
Student entry to critical section.
Data displyed:
41 67 34 0 69
Student exit critical section.
Teacher entry to critical section.
Data by teacher:
41 67 34 0 69
Teacher exit to critical section.
Student entry to critical section.
Data displyed:
41 67 34 0 69
Student exit critical section.
Teacher entry to critical section.
Data by teacher:
41 67 34 0 69
Teacher exit to critical section.
Student entry to critical section.
Data displyed:
41 67 34 0 69
Student exit critical section.
Teacher entry to critical section.
Data by teacher:
41 67 34 0 69
PS D:\VIT\Sem 2-2\OPERATING SYSTEMS\Assignments\lab 10>
```

2. Develop a dining philosopher problem using C without synchronization.

Code:

```c
#include <stdio.h>
#include <stdbool.h>
bool state[5];
int forks[5], count = 0;
int think(int i)
{
    state[i] = true;
}
int eat(int i)
{
    state[i] = false;
    printf("Philospher %d is eating...\n", i + 1);
}
int takeforks(int i)
{
    if (state[i] != false && forks[i] != 0)
    {
        forks[i] = 0;
        return 1;
    }
    else
    {
        return 0;
    }
}
void putforks(int i, int k)
{
    state[i] = true;
    forks[i] = 1;
    forks[k] = 1;
    printf("The philospher %d finished eating\n", i + 1);
}
void philosopher(int i)
{
    int k;
    while (1)
    {
        if (i > 4)
        {
            printf("No philosopher availalbe:\n");
            break;
        }
        else
        {
            think(i);
            int a = takeforks(i);
            int b = takeforks((i + 1) % 5);
            if (a == 1 && b == 1)
            {
                eat(i);
```

```
            }
            else
            {
                printf("The philospher %d cannot eat due to less availability of forks\n",
i + 1);
                break;
            }
            if (count == 0)
            {
                printf("Request is made by the philosopher:\n");
                scanf("%d", &k);
                philosopher(k - 1);
                count = count + 1;
            }
            putforks(i, (i + 1) % 5);
            break;
        }
    }
}
int main()
{
    int j, i;
    for (i = 0; i < 5; i++)
    {
        state[i] = true;
        forks[i] = 1;
    }
    int n;
    printf("Enter no.of requests:\n");
    scanf("%d", &n);
    for (i = 0; i < n; i++)
    {
        printf("Enter the philosopher number:\n");
        scanf("%d", &j);
        i = j - 1;
        philosopher(i);
    }
}
```

Output:

```
Enter no.of requests:
1
Enter the philosopher number:
1
Philospher 1 is eating...
Request is made by the philosopher:
3
Philospher 3 is eating...
Request is made by the philosopher:
4
The philospher 4 cannot eat due to less availability of forks
The philospher 3 finished eating
The philospher 1 finished eating
PS D:\VIT\Sem 2-2\OPERATING SYSTEMS\Assignments\lab 10>
```

3. Develop a dining philosopher problem with if condition and constant sleep time using C without synchronization construct.

Code:

```c
#include <stdio.h>
#include <unistd.h>
#include <stdbool.h>
bool state[5];
int forks[5], count = 0;
int think(int i)
{
    state[i] = true;
    printf("The philosopher %d is thinking...\n", i + 1);
}
int eat(int i)
{
    state[i] = false;
    printf("Philospher %d is eating...\n", i + 1);
}
int takeforks(int i)
{
    if (state[i] != false && forks[i] != 0)
    {
        forks[i] = 0;
        return 1;
    }
    else
    {
        return 0;
    }
}
void putforks(int i, int k)
{
    state[i] = true;
    forks[i] = 1;
    forks[k] = 1;
    printf("The philospher %d finished eating\n", i + 1);
}
```

```
void philosopher(int i)
{
    int k;
    while (1)
    {
        if (i > 4)
        {
            printf("No philosopher availalbe:\n");
            break;
        }
        else
        {
            think(i);
            int a = takeforks(i);
            int b = takeforks((i + 1) % 5);
            if (b == 1)
            {
                if (a == 1)
                {
                    eat(i);
                }
                else
                {
                    printf("The philospher %d cannot eat due to less availability of
forks\n", i + 1);
                    sleep(2);
                }
            }
            else
            {
                printf("The philospher %d cannot eat due to less availability of forks\n",
i + 1);
                break;
            }
            if (count == 0)
            {
                printf("Request is made by the philosopher:\n");
                scanf("%d", &k);
                philosopher(k - 1);
                count = count + 1;
            }
            putforks(i, (i + 1) % 5);
            break;
        }
    }
}
int main()
{
    int j, i;
    for (i = 0; i < 5; i++)
    {
        state[i] = true;
        forks[i] = 1;
    }
```

```
    int n;
    printf("Enter no.of requests:\n");
    scanf("%d", &n);
    for (i = 0; i < n; i++)
    {
        printf("Enter the philosopher number:\n");
        scanf("%d", &j);
        i = j - 1;
        philosopher(i);
    }
}
```

Output:

```
Enter no.of requests:
1
Enter the philosopher number:
1
The philosopher 1 is thinking...
Philospher 1 is eating...
Request is made by the philosopher:
3
The philosopher 3 is thinking...
Philospher 3 is eating...
Request is made by the philosopher:
2
The philosopher 2 is thinking...
The philospher 2 cannot eat due to less availability of forks
The philospher 3 finished eating
The philospher 1 finished eating
PS D:\VIT\Sem 2-2\OPERATING SYSTEMS\Assignments\lab 10> cd "d:\VI
```

```
Enter no.of requests:
1
Enter the philosopher number:
1
The philosopher 1 is thinking...
Philospher 1 is eating...
Request is made by the philosopher:
3
The philosopher 3 is thinking...
Philospher 3 is eating...
Request is made by the philosopher:
5
The philosopher 5 is thinking...
The philospher 5 cannot eat due to less availability of forks
The philospher 3 finished eating
The philospher 1 finished eating
PS D:\VIT\Sem 2-2\OPERATING SYSTEMS\Assignments\lab 10>
```

4. Modify the question 3 with random sleep time

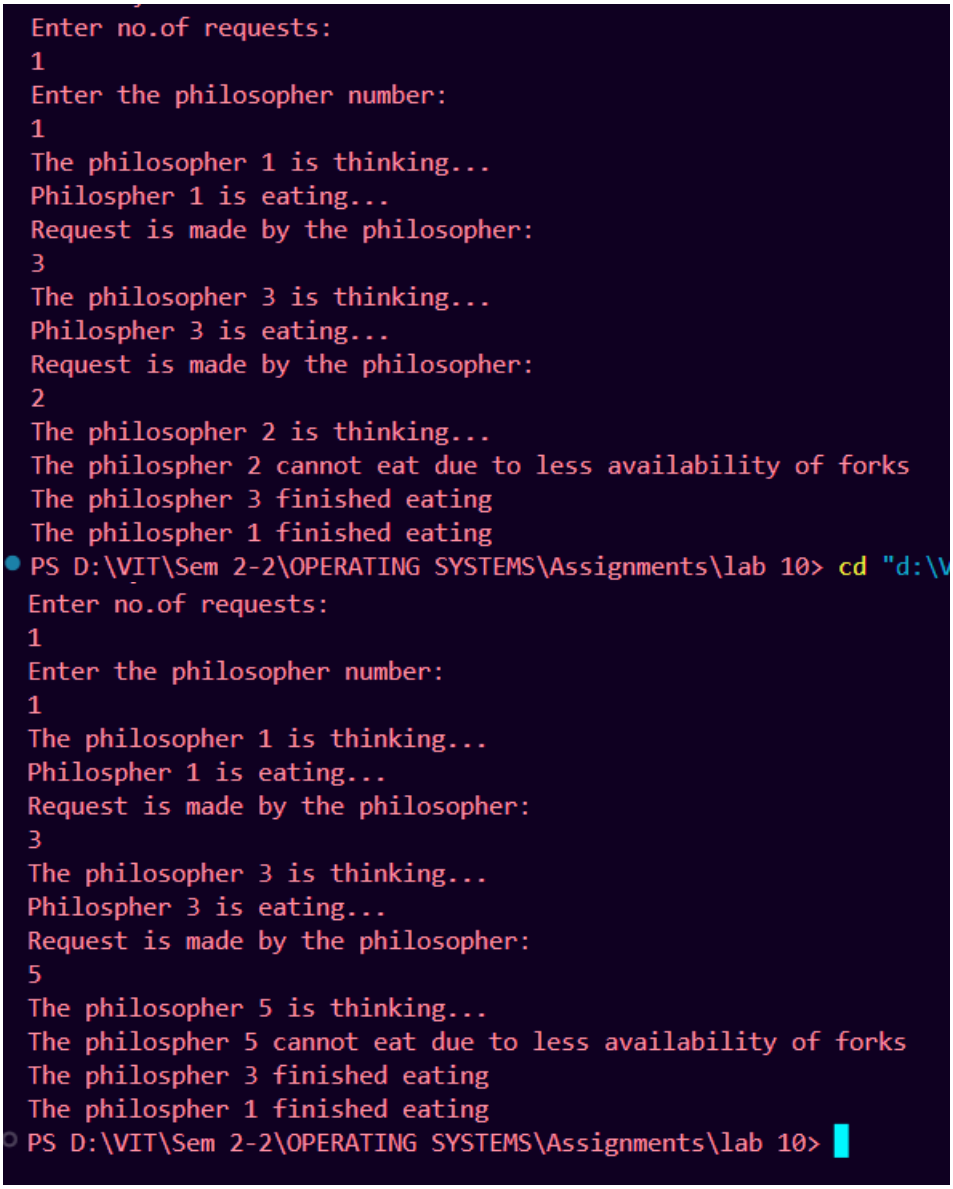
Code:

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <time.h>
#include <stdbool.h>
bool state[5];
int forks[5], count = 0;
int think(int i)
{
    state[i] = true;
    printf("The philosopher %d is thinking...\n", i + 1);
}
int eat(int i)
{
    state[i] = false;
    printf("Philospher %d is eating...\n", i + 1);
}
int takeforks(int i)
{
    if (state[i] != false && forks[i] != 0)
    {
        forks[i] = 0;
        return 1;
    }
    else
    {
        return 0;
    }
}
void putforks(int i, int k)
{
    state[i] = true;
    forks[i] = 1;
    forks[k] = 1;
    printf("The philospher %d finished eating\n", i + 1);
}
void philosopher(int i)
{
    int k;
    while (1)
    {
        if (i > 4)
        {
            printf("No philosopher availalbe:\n");
            break;
        }
        else
        {
            think(i);
            int a = takeforks(i);
```

```
            int b = takeforks((i + 1) % 5);
            if (b == 1)
            {
                if (a == 1)
                {
                    eat(i);
                }
                else
                {
                    printf("The philospher %d cannot eat due to less availability of
forks\n", i + 1);
                    sleep(rand());
                }
            }
            else
            {
                printf("The philospher %d cannot eat due to less availability of forks\n",
i + 1);
                break;
            }
            if (count == 0)
            {
                printf("Request is made by the philosopher:\n");
                scanf("%d", &k);
                philosopher(k - 1);

                count = count + 1;
            }
            putforks(i, (i + 1) % 5);
            break;
        }
    }
}
int main()
{
    int j, i;
    for (i = 0; i < 5; i++)
    {
        state[i] = true;
        forks[i] = 1;
    }
    int n;
    printf("Enter no.of requests:\n");
    scanf("%d", &n);
    for (i = 0; i < n; i++)
    {
        printf("Enter the philosopher number:\n");
        scanf("%d", &j);
        i = j - 1;
        philosopher(i);
    }
}
```

Output:



5. Develop a Dining philosopher problem using Mutex

Code:

```c
#include <stdio.h>
#include <unistd.h>
#include <semaphore.h>
#include <pthread.h>
#include <stdbool.h>
bool state[5];
int forks[5], count = 0;
sem_t mutex;
int think(int i)
{
    state[i] = true;
    printf("The philosopher %d is thinking...\n", i + 1);
}
int eat(int i)
{
    state[i] = false;
    printf("Philospher %d is eating...\n", i + 1);
```

```
}
int takeforks(int i)
{
    sem_wait(&mutex);
    if (state[i] != false && forks[i] != 0)
    {
        forks[i] = 0;
        sem_post(&mutex);
        return 1;
    }
    else
    {
        sem_post(&mutex);
        return 0;
    }
}
void putforks(int i, int k)
{
    sem_wait(&mutex);
    state[i] = true;
    forks[i] = 1;
    forks[k] = 1;
    printf("The philospher %d finished eating\n", i + 1);
    sem_post(&mutex);
}
void philosopher(int p)
{
    int i = (int *)p;
    int k;
    while (1)
    {
        if (i > 4)
        {
            printf("No philosopher availalbe:\n");
            break;
        }
        else
        {
            think(i);
            int a = takeforks(i);
            int b = takeforks((i + 1) % 5);
            if (b == 1)
            {
                if (a == 1)
                   eat(i);
                else
                {
                    printf("The philospher %d cannot eat due to less availability of
forks\n", i + 1);
                    sleep(2);
                }
            }
            else
            {
```

```
                printf("The philospher %d cannot eat due to less availability of forks\n",
i + 1);
                break;
            }
            if (count == 0)
            {
                printf("Request is made by the philosopher:\n");
                scanf("%d", &k);
                philosopher(k - 1);
                count = count + 1;
            }
            putforks(i, (i + 1) % 5);
            break;
        }
    }
}
int main()
{
    int j, i;
    pthread_t t;
    sem_init(&mutex, 0, 1);
    for (i = 0; i < 5; i++)
    {
        state[i] = true;
        forks[i] = 1;
    }
    printf("Enter the philosopher number:\n");
    scanf("%d", &j);
    i = j - 1;
    pthread_create(&t, NULL, (void *)philosopher, (void *)i);
    pthread_join(t, NULL);
    sem_destroy(&mutex);
}
```

Output:

```
Enter the philosopher number:
1
The philosopher 1 is thinking...
Philospher 1 is eating...
Request is made by the philosopher:
4
The philosopher 4 is thinking...
Philospher 4 is eating...
Request is made by the philosopher:
5
The philosopher 5 is thinking...
The philospher 5 cannot eat due to less availability of forks
The philospher 4 finished eating
The philospher 1 finished eating
PS D:\VIT\Sem 2-2\OPERATING SYSTEMS\Assignments\lab 10>
```

6. Develop a Dining problem using semaphore with test(), takefork() and putfork() functions for better synchronization of Dining_philosopher problem.

Code:

```
#include <stdio.h>
#include <unistd.h>
#include <semaphore.h>
#include <stdbool.h>
bool state[5];
int forks[5], count = 0;
sem_t semaphore_1;
int think(int i)
{
    state[i] = true;
    printf("The philosopher %d is thinking...\n", i + 1);
}
int eat(int i)
{
    state[i] = false;
    printf("Philospher %d is eating...\n", i + 1);
}
int takeforks(int i)
{
    sem_wait(&semaphore_1);
    if (state[i] != false && forks[i] != 0)
    {
        forks[i] = 0;
        sem_post(&semaphore_1);
        return 1;
    }
    else
    {
        sem_post(&semaphore_1);
        return 0;
    }
}
void putforks(int i, int k)
{
    sem_wait(&semaphore_1);
    state[i] = true;
    forks[i] = 1;
    forks[k] = 1;
    printf("The philospher %d finished eating\n", i + 1);
    sem_post(&semaphore_1);
}
void philosopher(int i)
{
    int k;
    while (1)
    {
        if (i > 4)
        {
            printf("No philosopher availalbe:\n");
```

```
            break;
        }
        else
        {
            think(i);
            int a = takeforks(i);
            int b = takeforks((i + 1) % 5);
            if (b == 1){
                if (a == 1)
                        eat(i);
                else
                {
                    printf("The philospher %d cannot eat due to less availability of
forks\n", i + 1);
                    sleep(2);
                }
        }
            else
            {

                printf("The philospher %d cannot eat due to less availability of forks\n",
i + 1);
                break;
            }
            if (count == 0)
            {
                printf("Request is made by the philosopher:\n");
                scanf("%d", &k);
                philosopher(k - 1);
                count = count + 1;
            }
            putforks(i, (i + 1) % 5);
            break;
        }    }     }
int main()
{
    int j, i;
    sem_init(&semaphore_1, 0, 1);
    for (i = 0; i < 5; i++)
    {
        state[i] = true;
        forks[i] = 1;
    }
    int n;
    printf("Enter no.of requests:\n");
    scanf("%d", &n);
    for (i = 0; i < n; i++)
    {
        printf("Enter the philosopher number:\n");
        scanf("%d", &j);
        i = j - 1;
        philosopher(i);
    }
}
```

Output:

```
Enter no.of requests:
1
Enter the philosopher number:
2
The philosopher 2 is thinking...
Philospher 2 is eating...
Request is made by the philosopher:
3
The philosopher 3 is thinking...
The philospher 3 cannot eat due to less availability of forks
Request is made by the philosopher:
5
The philosopher 5 is thinking...
Philospher 5 is eating...
Request is made by the philosopher:
1
The philosopher 1 is thinking...
The philospher 1 cannot eat due to less availability of forks
The philospher 5 finished eating
The philospher 3 finished eating
The philospher 2 finished eating
PS D:\VIT\Sem 2-2\OPERATING SYSTEMS\Assignments\lab 10>
```