

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:

#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);

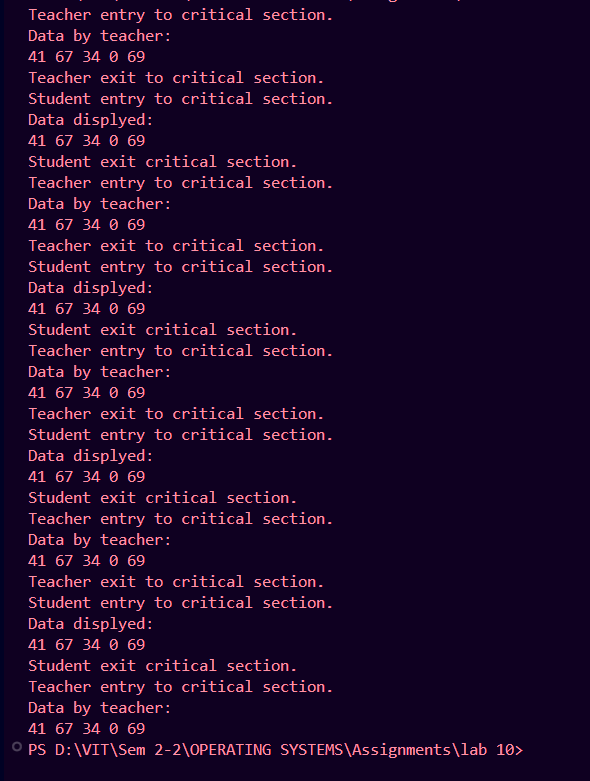
    }

    sem\_destroy(&mutex);

    sem\_destroy(&teach);

    return 0;

}

Output:

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

Code:

#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");

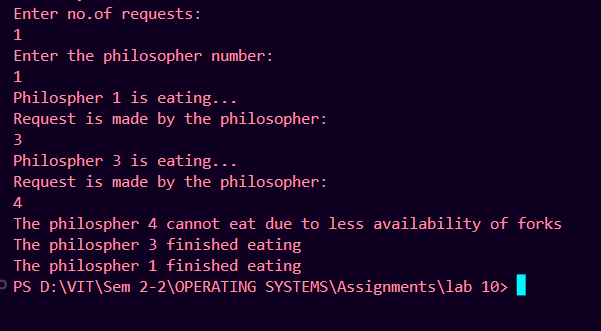
        scanf("%d", &j);

        i = j - 1;

        philosopher(i);

    }

}

Output:

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

Code:

#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");

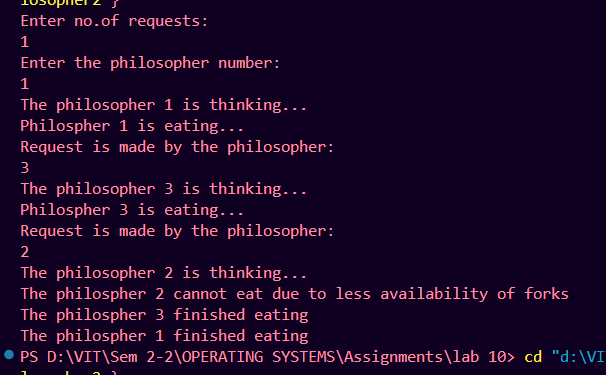
        scanf("%d", &j);

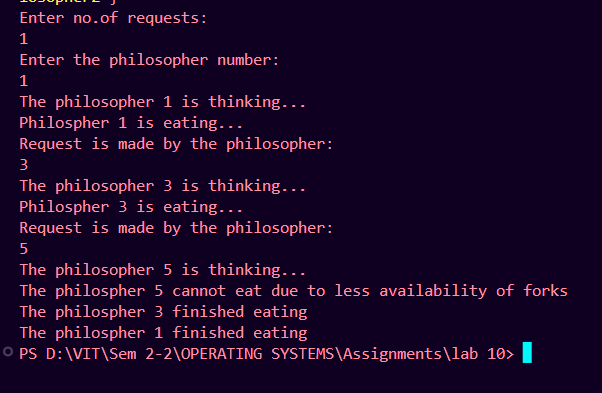
        i = j - 1;

        philosopher(i);

    }

}

Output:



1. 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");

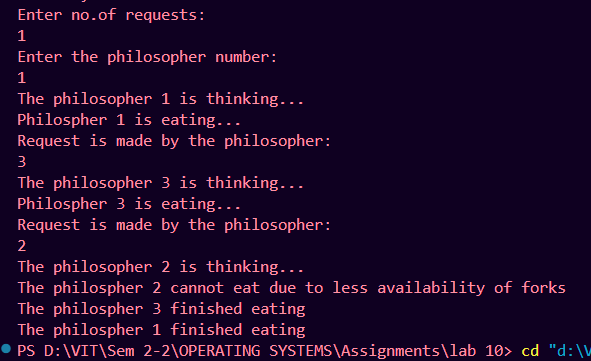
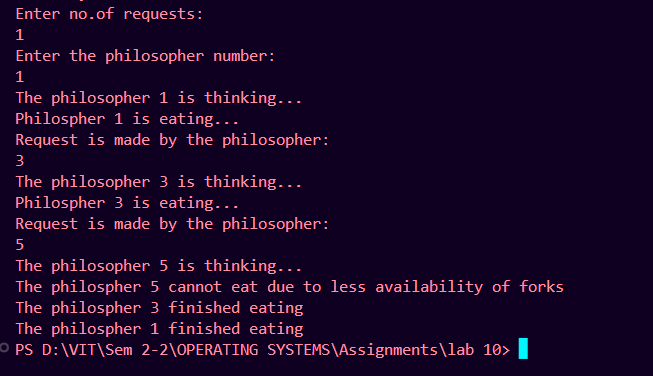
        scanf("%d", &j);

        i = j - 1;

        philosopher(i);

    }

}

Output:

1. Develop a Dining philosopher problem using Mutex

Code:

#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);

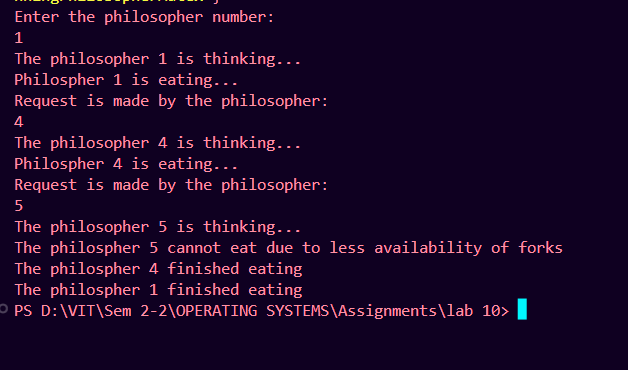
    i = j - 1;

    pthread\_create(&t, NULL, (void \*)philosopher, (void \*)i);

    pthread\_join(t, NULL);

    sem\_destroy(&mutex);

}

Output:

1. 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");

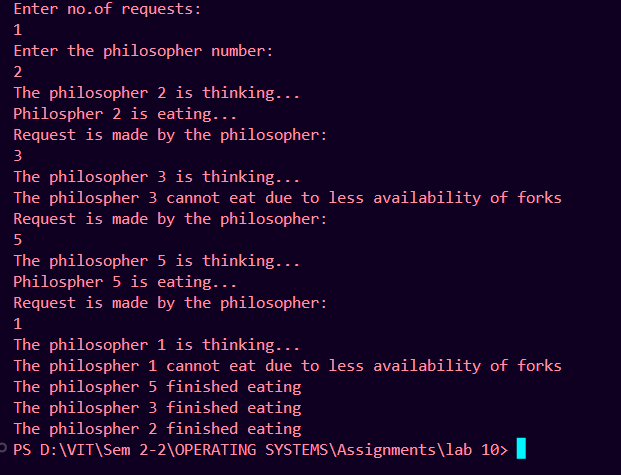
        scanf("%d", &j);

        i = j - 1;

        philosopher(i);

    }

}

 Output: