**Name: Varun Menon**

**Reg No: 19BCE1438**

**Course Code: CSE4001**

**Faculty: Dr. Harini S**

**Lab Experiment 6**

1. Sample critical without parallel program

Code:

#include<stdio.h>

#include<stdlib.h>

#include<omp.h>

#include<math.h>

int main()

{

    int shared = 0;

    for(int i = 0; i < 10; i++)

    {

        #pragma omp critical

            shared++;

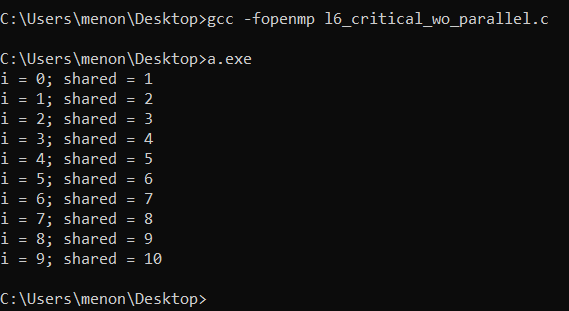
        printf("i = %d; shared = %d\n", i, shared);

    }

    return 0;

}

Output:



1. Sample static schedule program

Code:

#include<stdio.h>

#include<stdlib.h>

#include<omp.h>

#include<math.h>

int main()

{

    int a = 0;

    int i;

    #pragma omp parallel for schedule(static)

    for(i = 0; i < 10; i++) {

        a++;

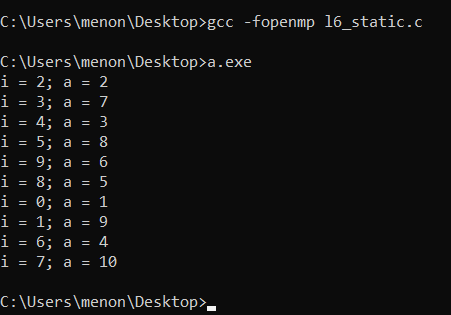
        printf("i = %d; a = %d\n", i, a);

    }

    return 0;

}

Output:



1. Sample dynamic schedule program

Code:

#include<stdio.h>

#include<stdlib.h>

#include<omp.h>

#include<math.h>

int main()

{

    int a = 4;

    #pragma omp parallel for schedule(dynamic)

    for(int i = 0; i < 10; i++) {

        a++;

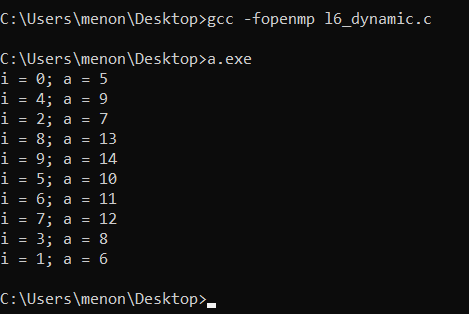
        printf("i = %d; a = %d\n", i, a);

    }

    return 0;

}

Output:



1. Sample profile program with omp\_get\_wtime()

Code:

#include <omp.h>

#include <stdio.h>

int main()

{

    double start = omp\_get\_wtime();

    int sum = 0;

    #pragma omp parallel for

    for (int i = 0; i < 10000000; i++)

    {

        sum += i;

    }

    double end = omp\_get\_wtime();

    printf("Time taken - parallel for: %fs\n", end-start);

    sum = 0;

    double start2 = omp\_get\_wtime();

    #pragma omp parallel

    {

        for (int i = 0; i < 10000000; i++)

        {

            sum += i;

        }

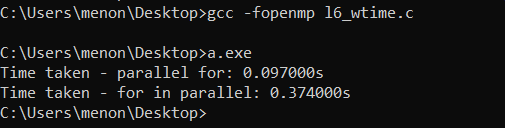
    }

    double end2 = omp\_get\_wtime();

    printf("Time taken - for in parallel: %fs", end2-start2);

}

Output:



1. Sample program to set threads using directive clause

Code:

#include <omp.h>

#include <stdio.h>

int main()

{

    #pragma omp parallel num\_threads(7)

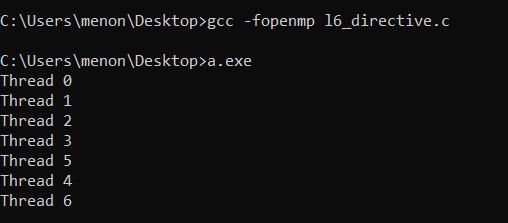
    {

        printf("Thread %d\n", omp\_get\_thread\_num());

    }

}

Output:



1. Sample program to set threads using omp\_set\_threads()

Code:

#include <omp.h>

#include <stdio.h>

int main()

{

    omp\_set\_num\_threads(7);

    #pragma omp parallel

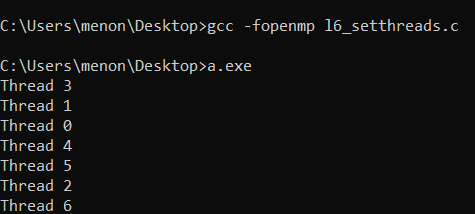
    {

        printf("Thread %d\n", omp\_get\_thread\_num());

    }

}

Output:



1. Optional : 3 Hot Questions
   1. Consider a priority queue that has four levels of priority. Each priority level can have at the maximum of ‘n’ tasks. Each task will have a priority.  Depending on the priority, the tasks are inserted in the queue. For the dispatch of tasks, the tasks from the highest priority queue should be dispatched every clock cycle, from the second highest priority queue after every 2 clock cycles, from the third highest priority queue after every 3 clock cycles and so on. The insertion and dispatch can happen in parallel. Write an efficient OpenMP algorithm to emulate the above priority queue of an operating system. Every clock cycle, your program should print the following.
      1. Tasks in each priority level
      2. Task ID last dispatched /currently getting dispatched

Task ID’s and its priority can be randomly generated using rand()functions. Based on the set of random inputs simulate the above scenario and display the results.

Code:

#include<omp.h>

#include<stdio.h>

int main()

{

    int n;

    printf("Enter number of tasks: ");

    scanf("%d", &n);

    int tasks[n][2];

    int front[4] = {0, 0, 0, 0}, rear[4] = {0, 0, 0, 0};

    int q[4][1000];

    printf("Enter task id and priority for each task (1 to 4 (highest priority)):\n");

    for(int i = 0; i < n; i++)

    {

        scanf("%d %d", &tasks[i][0], &tasks[i][1]);

        q[tasks[i][1]-1][rear[tasks[i][1]-1]++] = tasks[i][0]; // enqueue to the specified queue

    }

    for(int i = 1; i < 3\*n + 1; i++)

    {

        #pragma omp parallel sections

        {

            #pragma omp section

            {

                if(front[3] < rear[3])

                {

                    printf("Completed task %d which had priority 4; Clock cycles taken: %d\n", q[3][front[3]], i);

                    #pragma omp critical

                        front[3]++;

                }

            }

            #pragma omp section

            {

                if(i % 2 == 0 && front[2] < rear[2])

                {

                    printf("Completed task %d which had priority 3; Clock cycles taken: %d\n", q[2][front[2]], i);

                    #pragma omp critical

                        front[2]++;

                }

            }

            #pragma omp section

            {

                if(i % 3 == 0 && front[1] < rear[1])

                {

                    printf("Completed task %d which had priority 2; Clock cycles taken: %d\n", q[1][front[1]], i);

                    #pragma omp critical

                        front[1]++;

                }

            }

            #pragma omp section

            {

                if(i % 4 == 0 && front[0] < rear[0])

                {

                    printf("Completed task %d which had priority 1; Clock cycles taken: %d\n", q[1][front[1]], i);

                    #pragma omp critical

                        front[0]++;

                }

            }

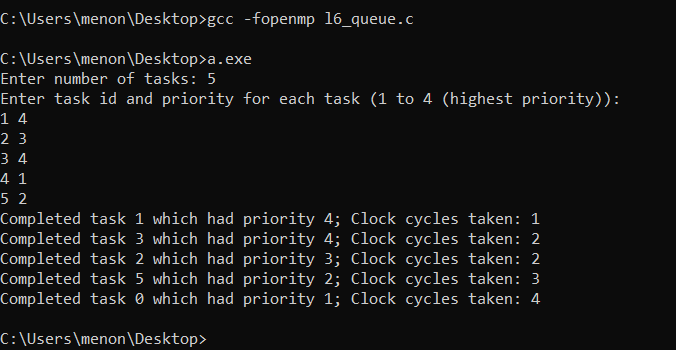
        }

    }

    return 0;

}

Output:



* 1. Write a parallel program using OpenMP to implement the following series, 1/2+1/4+1/8+1/16+………..1/16384

Find the sum of the series and print it along with the thread id and the last value in the series i.e., (1/16384).

Code:

#include <omp.h>

#include <stdio.h>

#include <math.h>

int main()

{

    double x;

    float s;

    #pragma omp parallel for lastprivate(x) reduction(+:s)

    for(int i = 1; i <= 14; i++)

    {

        x = pow(2,i);

        s = (1/x);

        printf("x = %f; Sum = %f\n", x, s);

    }

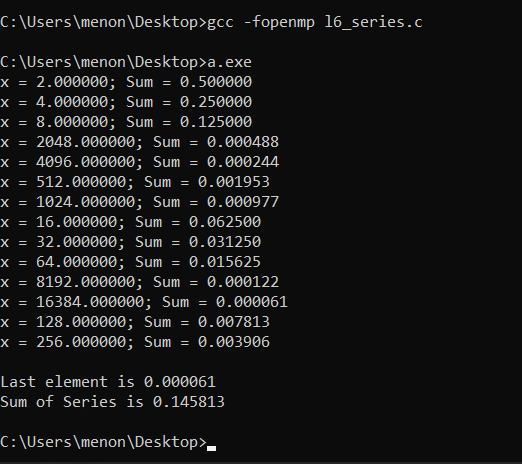
    printf("\n");

    printf("Last element is %f\n", (1/x));

    printf("Sum of Series is %f\n", s);

}

Output:



* 1. There are four rail tracks that pass through a railway station. The trains can use the same track with a delay of 60 minutes. Depending on the above conditions, the signal for any of the track will be updated by a control system. The signal will be displayed red for time delay less than 60. If the time delay exceeds 60, the signal will be changed to green. The above tasks are done in parallel to avoid any mishaps. Implement the above scenario in C and parallelize it using OpenMP. Let the train arrival times are randomly generated in your program and number of trains that are going to considered can be obtained from the user. The track to be used by a train can also be set by the user prior to the simulation. Whenever a signal is set, the same has to be displayed in your screen (Eg: Track 1: red, Track2 :Green etc) by your program.

Code:

#include <omp.h>

#include <stdio.h>

int main()

{

    int n;

    printf("Enter number of trains: ");

    scanf("%d", &n);

    int trackNum[n];

    int arrivalTime[n]

    for (int i = 0; i < n; i++)

    {

        printf("Enter track number of train number (1 to 4) %d:", i);

        scanf("%d", &trackNum[i]);

        arrivalTime[i] = rand();

    }

    int signals = {0,0,0,0}; // 0 is green and 1 is red

    int last\_time[4] = {omp\_get\_wtime(),omp\_get\_wtime(),omp\_get\_wtime(),omp\_get\_wtime()};

    omp\_set\_num\_threads(4);

    for (int i = 0; i < n; i++)

    {

        #pragma omp parallel sections

            #pragma omp section

            {

                if (trackNum[i] == 1 && ((last\_time[0] - omp\_get\_wtime()) <= 60\*60))

                {

                    #pragma omp critical

                    {

                        signals[0] = 1; //red

                        last\_time[0] = omp\_get\_wtime();

                    }

                    for(int j = 0; j < 4; j++)

                    {

                        if(signals[j] == 0)

                        {

                            printf("Signal %d: Green\n", (j+1));

                        }

                        else

                        {

                            printf("Signal %d: Green\n", (j+1));

                        }

                    }

                }

                else if (((last\_time[0] - omp\_get\_wtime()) <= 60\*60))

                {

                    signals[0] = 0; //green

                    for(int j = 0; j < 4; j++)

                    {

                        if(signals[j] == 0)

                        {

                            printf("Signal %d: Green\n", (j+1));

                        }

                        else

                        {

                            printf("Signal %d: Green\n", (j+1));

                        }

                    }

                }

            }

            #pragma omp section

            {

                if (trackNum[i] == 2 && ((last\_time[1] - omp\_get\_wtime()) <= 60\*60))

                {

                    #pragma omp critical

                    {

                        signals[1] = 1; //red

                        last\_time[1] = omp\_get\_wtime();

                    }

                    for(int j = 0; j < 4; j++)

                    {

                        if(signals[j] == 0)

                        {

                            printf("Signal %d: Green\n", (j+1));

                        }

                        else

                        {

                            printf("Signal %d: Green\n", (j+1));

                        }

                    }

                }

                else if (((last\_time[1] - omp\_get\_wtime()) <= 60\*60))

                {

                    signals[1] = 0; //green

                    for(int j = 0; j < 4; j++)

                    {

                        if(signals[j] == 0)

                        {

                            printf("Signal %d: Green\n", (j+1));

                        }

                        else

                        {

                            printf("Signal %d: Green\n", (j+1));

                        }

                    }

                }

            }

            #pragma omp section

            {

                if (trackNum[i] == 3 && ((last\_time[2] - omp\_get\_wtime()) <= 60\*60))

                {

                    #pragma omp critical

                    {

                        signals[2] = 1; //red

                        last\_time[2] = omp\_get\_wtime();

                    }

                    for(int j = 0; j < 4; j++)

                    {

                        if(signals[j] == 0)

                        {

                            printf("Signal %d: Green\n", (j+1));

                        }

                        else

                        {

                            printf("Signal %d: Green\n", (j+1));

                        }

                    }

                }

                else if (((last\_time[2] - omp\_get\_wtime()) <= 60\*60))

                {

                    signals[2] = 0; //green

                    for(int j = 0; j < 4; j++)

                    {

                        if(signals[j] == 0)

                        {

                            printf("Signal %d: Green\n", (j+1));

                        }

                        else

                        {

                            printf("Signal %d: Green\n", (j+1));

                        }

                    }

                }

            }

            #pragma omp section

            {

                if (trackNum[i] == 4 && ((last\_time[3] - omp\_get\_wtime()) <= 60\*60))

                {

                    #pragma omp critical

                    {

                        signals[3] = 1; //red

                        last\_time[3] = omp\_get\_wtime();

                    }

                    for(int j = 0; j < 4; j++)

                    {

                        if(signals[j] == 0)

                        {

                            printf("Signal %d: Green\n", (j+1));

                        }

                        else

                        {

                            printf("Signal %d: Green\n", (j+1));

                        }

                    }

                }

                else if (((last\_time[3] - omp\_get\_wtime()) <= 60\*60))

                {

                    signals[3] = 0; //green

                    for(int j = 0; j < 4; j++)

                    {

                        if(signals[j] == 0)

                        {

                            printf("Signal %d: Green\n", (j+1));

                        }

                        else

                        {

                            printf("Signal %d: Green\n", (j+1));

                        }

                    }

                }

            }

    }

}

Output:

