**LAB BONUS**

**1 Review**

Example1: Pthread Creation and Termination

This simple example code creates 5 threads with the pthread create() routine. Each thread prints a "Hello World!" message, and then terminates with a call to pthread exit().

#include <pthread.h>

#include <stdlib.h>

#include <stdio.h>

#define NUM\_THREADS 10

void \* user\_def\_func(void \* threadID) {

    long TID;

    TID = (long) threadID;

    printf("Hello World! from thread %ld\n", TID);

    pthread\_exit(NULL);

}

int main(int argc, char\* argv) {

    pthread\_t threads[NUM\_THREADS];

    int create\_flag;

    long i;

    for(int i = 0; i < NUM\_THREADS; i++) {

        printf("In main: creating thread %ld\n", i);

        create\_flag = pthread\_create(&threads[i], NULL, user\_def\_func, (void \*)i);

        if (create\_flag) {

            printf("ERROR: return code from pthread\_create() is %d\n", create\_flag);

            exit(-1);

        }

    }

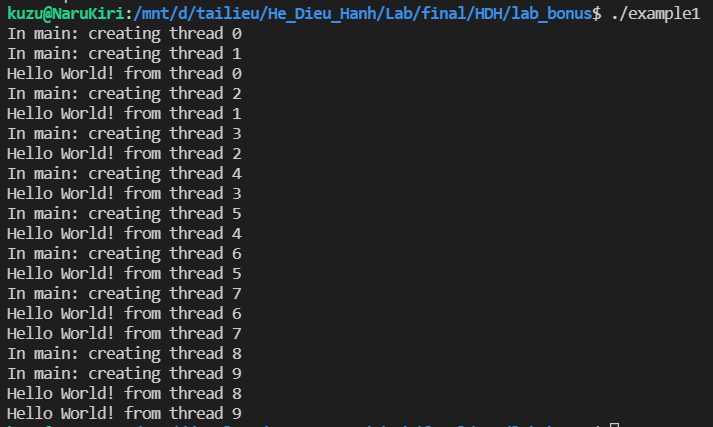
    //free thread

    pthread\_exit(NULL);

    return 0;

}

Kết quả:



Example2: Thread Argument PassingThis code fragment demonstrates how to pass a simple integer to each thread. The calling thread uses a unique data structure for each thread, insuring that each thread’s argument remains intact throughout the program.

#include <pthread.h>

#include <stdlib.h>

#include <stdio.h>

#define NUM\_THREADS 10

void \* user\_def\_func(void \* threadID) {

    long TID;

    TID = (long) threadID;

    printf("Hello World! from thread %ld\n", TID);

    pthread\_exit(NULL);

}

long taskids[NUM\_THREADS];

int main(int argc, char\* argv) {

    pthread\_t threads[NUM\_THREADS];

    int create\_flag;

    long i;

    for(int i = 0; i < NUM\_THREADS; i++) {

        taskids[i] = i;

        printf("In main: creating thread %ld\n", (void\*)taskids[i]);

        create\_flag = pthread\_create(&threads[i], NULL, user\_def\_func, (void \*)i);

        if (create\_flag) {

            printf("ERROR: return code from pthread\_create() is %d\n", create\_flag);

            exit(-1);

        }

    }

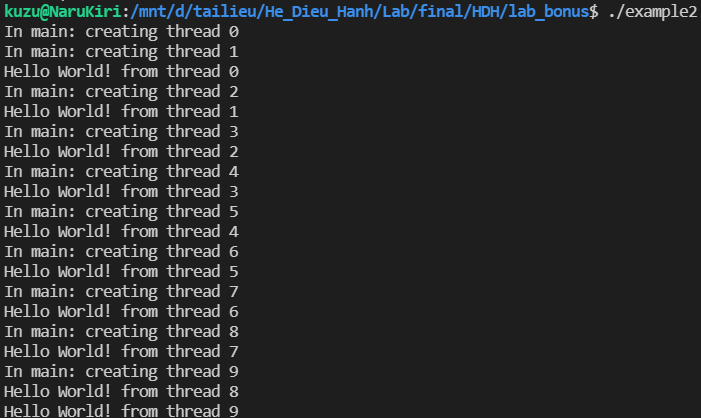
    //free thread

    pthread\_exit(NULL);

    return 0;

}

Kết quả:



Example3: A joinable state for portability purposesDemonstrates how to explicitly create pthreads in a joinable state for portability purposes. Also shows how  
to use the pthread exit status parameter.

#include <pthread.h>

#include <stdlib.h>

#include <stdio.h>

#include <math.h>

#define NUM\_THREADS     4

#define NUM\_LOOPS    10000000

void \* user\_def\_func(void \* threadID) {

    long TID;

    TID = (long) threadID;

    int i;

    double result = 0.0;

    printf("Thread %ld starting...\n", TID);

    for(int i = 0; i < NUM\_LOOPS; i++) {

        result = result + sin(i) + tan(i);

    }

    printf("Thread %ld done. Result = %e\n", TID, result);

    pthread\_exit((void \*)threadID);

}

long taskids[NUM\_THREADS];

int main(int argc, char\* argv) {

    pthread\_t threads[NUM\_THREADS];

    pthread\_attr\_t attr; //attribute of threads

    int creation\_flag, join\_flag;

    long i;

    void \*status;   //status of threads

    // Initialisze and set thread detached attribute

    pthread\_attr\_init(&attr);

    pthread\_attr\_setdetachstate(&attr, PTHREAD\_CREATE\_JOINABLE);

    for(i = 0; i < NUM\_THREADS; i++) {

        printf("In main: creating thread %ld\n", i);

        creation\_flag = pthread\_create(&threads[i], &attr, user\_def\_func, (void\*)i);

        if (creation\_flag) {

            printf("ERROR: return code from pthread\_create(0 is %d\n",  creation\_flag);

            exit(-1);

        }

    }

    pthread\_attr\_destroy(&attr);

    for(i = 0; i < NUM\_THREADS; i++) {

        join\_flag = pthread\_join(threads[i], &status);

        if (join\_flag) {

            printf("ERROR: return code from pthread\_join is %d\n", join\_flag);

            exit(-1);

        }

        printf("Main: completed join with thread %ld having a status of %ld\n", i, (long)status);

    }

    printf("Main: program completed. Exiting.\n");

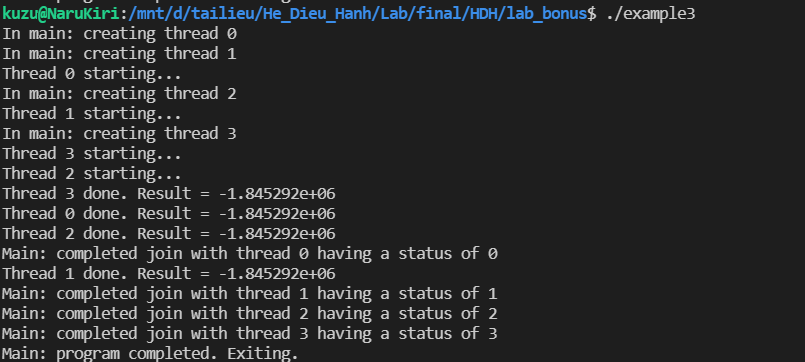
    //free thread

    pthread\_exit(NULL);

    return 0;

}

Kết quả



Example4: Race conditionThis example uses a mutex variable to protect the global sum while each thread updates it. Race condition  
is an important problem in parallel programming

#include <pthread.h>

#include <stdio.h>

#include <stdlib.h>

/\* Define global data where everyone can see them \*/

#define NUMTHRDS 8

#define VECLEN 100000

pthread\_mutex\_t mutexsum;

int \*a, \*b;

long sum=0.0;

void \*dotprod(void \*arg)

{

    /\* Each thread works on a different set of data.

    \* The offset is specified by the arg parameter. The size of

    \* the data for each thread is indicated by VECLEN.

    \*/

    int i, start, end, offset, len;

    long tid;

    tid = (long)arg;

    offset = tid;

    len = VECLEN;

    start = offset\*len;

    end = start + len;

    /\* Perform my section of the dot product \*/

    printf("thread: %ld starting. start=%d end=%d\n",tid,start,end-1);

    for (i=start; i<end ; i++) {

        pthread\_mutex\_lock(&mutexsum);

        sum += (a[i] \* b[i]);

        pthread\_mutex\_unlock(&mutexsum);

    }

    printf("thread: %ld done. Global sum now is=%li\n",tid,sum);

    pthread\_exit((void\*) 0);

}

int main (int argc, char \*argv[])

{

    long i;

    void \*status;

    pthread\_t threads[NUMTHRDS];

    pthread\_attr\_t attr;

    /\* Assign storage and initialize values \*/

    a = (int\*) malloc (NUMTHRDS\*VECLEN\*sizeof(int));

    b = (int\*) malloc (NUMTHRDS\*VECLEN\*sizeof(int));

    for (i=0; i<VECLEN\*NUMTHRDS; i++)

    a[i]=b[i]=1;

    /\* Initialize mutex variable \*/

    pthread\_mutex\_init(&mutexsum, NULL);

    /\* Create threads as joinable, each of which will execute the dot product

    \* routine. Their offset into the global vectors is specified by passing

    \* the "i" argument in pthread\_create().

    \*/

    pthread\_attr\_init(&attr);

    pthread\_attr\_setdetachstate(&attr, PTHREAD\_CREATE\_JOINABLE);

    for(i=0;i<NUMTHRDS;i++)

    pthread\_create(&threads[i], &attr, dotprod, (void \*)i);

    pthread\_attr\_destroy(&attr);

    /\* Wait on the other threads for final result \*/

    for(i=0;i<NUMTHRDS;i++) {

        pthread\_join(threads[i], &status);

    }

    /\* After joining, print out the results and cleanup \*/

    printf ("Final Global Sum=%li\n",sum);

    free (a);

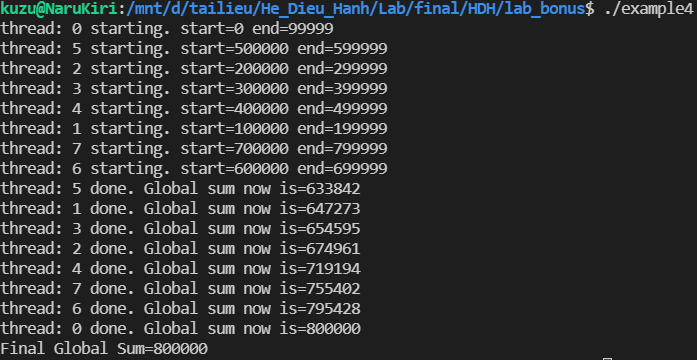
    free (b);

    pthread\_mutex\_destroy(&mutexsum);

    pthread\_exit(NULL);

}

Kết quả



**2 Multithread Programming with OpenMP**

Example1: Simple "Hello World" program. Every thread executes all code enclosed in the parallel region.  
OpenMP library routines are used to obtain thread identifiers and total number of threads.

#include <omp.h>

#include <stdio.h>

#include <stdlib.h>

int main(int argc, char \*argv[]) {

    int nthreads, tid;

    /\* Fork a team of threads with each thread having a private tid variable \*/

    #pragma omp parallel private(tid)

    {

            /\* Obtain and print thread id \*/

        tid = omp\_get\_thread\_num();

        printf("Hello World from thread = %d\n", tid);

        /\* Only master thread does this \*/

        if (tid == 0)

        {

            nthreads = omp\_get\_num\_threads();

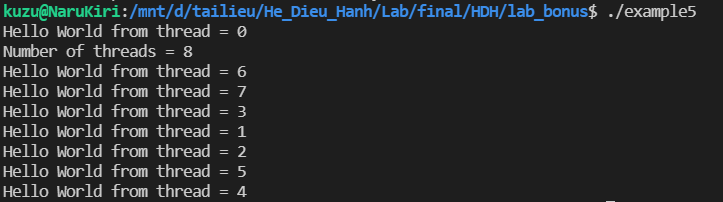
            printf("Number of threads = %d\n", nthreads);

        }

    } /\* All threads join master thread and terminate \*/

}

Kết quả



Example2: Work-Sharing Constructs - DO / for Directive. The DO / for directive specifies that the  
iterations of the loop immediately following it must be executed in parallel by the team. This assumes a  
parallel region has already been initiated, otherwise it executes in serial on a single processor

#include <omp.h>

#include <stdio.h>

#include <stdlib.h>

/\* Define some values \*/

#define N 1000

#define CHUNKSIZE 100

#define OMP\_NUM\_THREADS 10

#define MAX\_THREADS 48

/\* Global variables \*/

int count[MAX\_THREADS];

int main(int argc, char \*\*argv){

    int i, chunk;

    float a[N], b[N], c[N];

    /\* Some initializations \*/

    for(i = 0; i < N; i++){

        a[i] = b[i] = i \* 1.0; // values = i with float type

    }

    //for(i = 0; i < OMP\_NUM\_THREADS; i++){

    // count[i] = 0;

    //}

    chunk = CHUNKSIZE;

    #pragma omp parallel shared(a,b,c,chunk) private(i)

    {

        omp\_set\_num\_threads(OMP\_NUM\_THREADS);

        #pragma omp for schedule(dynamic,chunk) nowait

        for(i = 0; i < N; i++){

            int tid = omp\_get\_thread\_num();

            printf("Iter %d running from thread %d\n", i, tid);

            c[i] = a[i] + b[i];

            // Increase count[tid]

            count[tid]++;

        }

    }

    /\* Validation \*/

    printf("Vector c: \n");

    for(i = 0; i < 10; i++){

        printf("%f ", c[i]);

    }

    printf("...\n");

    /\* Statistic \*/

    // printf("Num of iter with thread:\n");

    // for(i = 0; i < MAX\_THREADS; i++){

    // if(count[i] != 0)

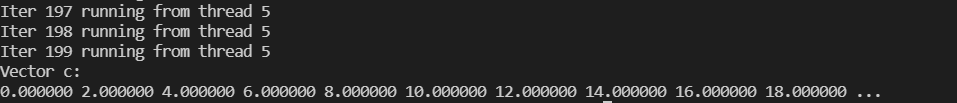
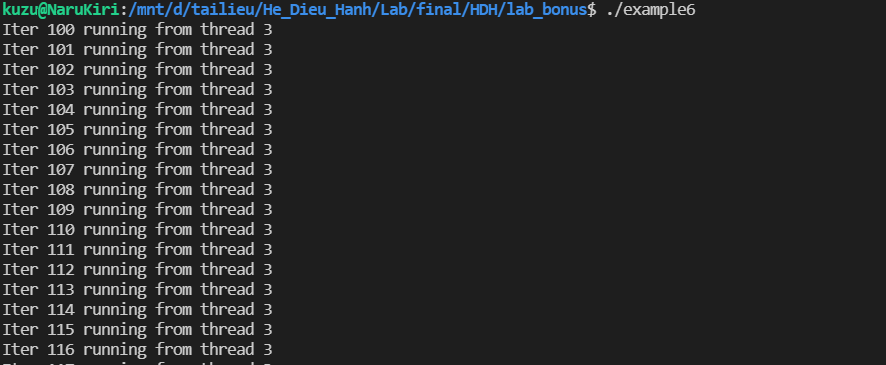
    // printf("\tThread %d run %d iter.\n", i, count[i]);

    // }

    return 0;

}

Kết quả



Example3: Work-Sharing Constructs - SECTIONS Directive

#include <omp.h>

#include <stdio.h>

#include <stdlib.h>

/\* Define some values \*/

#define N 1000

#define CHUNKSIZE 100

#define OMP\_NUM\_THREADS 12

#define MAX\_THREADS 48

/\* Global variables \*/

int count[MAX\_THREADS];

int main(int argc, char \*\*argv){

    int i, chunk;

    float a[N], b[N], c[N], d[N];

    /\* Some initializations \*/

    for(i = 0; i < N; i++){

        a[i] = i \* 1.0;

        b[i] = i + 2.0;

    }

    for(i = 0; i < OMP\_NUM\_THREADS; i++){

        count[i] = 0;

    }

    chunk = CHUNKSIZE;

    #pragma omp parallel shared(a,b,c,d) private(i)

    {

        omp\_set\_num\_threads(OMP\_NUM\_THREADS);

        #pragma omp sections nowait

        {

            #pragma omp section

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

                    int tid\_s1 = omp\_get\_thread\_num();

                    printf("\tIter %d running from thread %d\n", i, tid\_s1);

                    c[i] = a[i] + b[i];

                    // Increase count

                    count[tid\_s1]++;

                }

            #pragma omp section

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

                    int tid\_s2 = omp\_get\_thread\_num();

                    printf("\tIter %d running from thread %d\n", i, tid\_s2);

                    d[i] = a[i] \* b[i];

                    // Increase count

                    count[tid\_s2]++;

                }

        }

    }

    /\* Validation \*/

    printf("Vector c: \n\t");

    for(i = 0; i < 10; i++){

        printf("%f ", c[i]);

    }

    printf("...\n");

    printf("Vector d: \n\t");

    for(i = 0; i < 10; i++){

        printf("%f ", d[i]);

    }

    printf("...\n");

    /\* Statistic \*/

    printf("Num of iter with thread:\n");

    for(i = 0; i < MAX\_THREADS; i++){

        if (count[i] != 0)

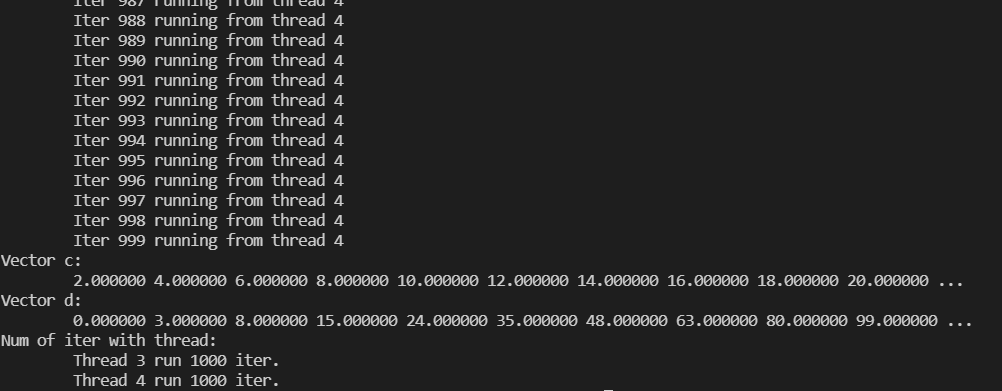
            printf("\tThread %d run %d iter.\n", i, count[i]);

    }

    return 0;

}

Kết quả



Example4: THREADPRIVATE Directive The THREADPRIVATE directive is used to make global file  
scope variables (C/C++) or common blocks (Fortran) local and persistent to a thread through the execution  
of multiple parallel regions.

#include <omp.h>

#include <stdio.h>

#include <stdlib.h>

/\* Define some values \*/

#define N 1000

#define CHUNKSIZE 10

#define MAX\_THREADS 48

#define NUM\_THREADS 4

/\* Global variables \*/

int count[MAX\_THREADS];

int a, b, i, tid;

float x;

#pragma omp threadprivate(a, x)

    int main(int argc, char \*\*argv){

    /\* Explicitly turn off dynamic threads \*/

        omp\_set\_dynamic(0);

        omp\_set\_num\_threads(NUM\_THREADS);

        printf("1st Parallel Region:\n");

        #pragma omp parallel private(b,tid)

        {

            tid = omp\_get\_thread\_num();

            a = tid;

            b = tid;

            x = 1.1 \* tid + 1.0;

            printf("Thread %d: a, b, x = %d, %d, %f\n", tid, a, b, x);

        }

        printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

        printf("Master thread doing serial work here\n");

        printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

        printf("2nd Parallel Region:\n");

        #pragma omp parallel private(tid)

        {

            tid = omp\_get\_thread\_num();

            printf("Thread %d: a, b, x = %d, %d, %f\n", tid, a, b, x);

        }

        return 0;

    }

Kết quả

