# 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:

```
kuzu@NaruKiri:/mnt/d/tailieu/He_Dieu_Hanh/Lab/final/HDH/lab_bonus$ ./example1
In main: creating thread 0
In main: creating thread 1
Hello World! from thread 0
In main: creating thread 2
Hello World! from thread 1
In main: creating thread 3
Hello World! from thread 2
In main: creating thread 4
Hello World! from thread 3
In main: creating thread 5
Hello World! from thread 4
In main: creating thread 6
Hello World! from thread 5
In main: creating thread 7
Hello World! from thread 6
Hello World! from thread 7
In main: creating thread 8
In main: creating thread 9
Hello World! from thread 8
Hello World! from thread 9
```

**Example2: Thread Argument Passing**

This 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.

```c
#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:

```
kuzu@NaruKiri:/mnt/d/tailieu/He_Dieu_Hanh/Lab/final/HDH/lab_bonus$ ./example2
In main: creating thread 0
In main: creating thread 1
Hello World! from thread 0
In main: creating thread 2
Hello World! from thread 1
In main: creating thread 3
Hello World! from thread 2
In main: creating thread 4
Hello World! from thread 3
In main: creating thread 5
Hello World! from thread 4
In main: creating thread 6
Hello World! from thread 5
In main: creating thread 7
Hello World! from thread 6
In main: creating thread 8
Hello World! from thread 7
In main: creating thread 9
Hello World! from thread 8
Hello World! from thread 9
```

**Example3: A joinable state for portability purposes**

Demonstrates 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
```

```c
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

```
kuzu@NaruKiri:/mnt/d/tailieu/He_Dieu_Hanh/Lab/final/HDH/lab_bonus$ ./example3
In main: creating thread 0
In main: creating thread 1
Thread 0 starting...
In main: creating thread 2
Thread 1 starting...
In main: creating thread 3
Thread 3 starting...
Thread 2 starting...
Thread 3 done. Result = -1.845292e+06
Thread 0 done. Result = -1.845292e+06
Thread 2 done. Result = -1.845292e+06
Main: completed join with thread 0 having a status of 0
Thread 1 done. Result = -1.845292e+06
Main: completed join with thread 1 having a status of 1
Main: completed join with thread 2 having a status of 2
Main: completed join with thread 3 having a status of 3
Main: program completed. Exiting.
```

**Example4: Race condition**

This 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

```
kuzu@NaruKiri:/mnt/d/tailieu/He_Dieu_Hanh/Lab/final/HDH/lab_bonus$ ./example4
thread: 0 starting. start=0 end=99999
thread: 5 starting. start=500000 end=599999
thread: 2 starting. start=200000 end=299999
thread: 3 starting. start=300000 end=399999
thread: 4 starting. start=400000 end=499999
thread: 1 starting. start=100000 end=199999
thread: 7 starting. start=700000 end=799999
thread: 6 starting. start=600000 end=699999
thread: 5 done. Global sum now is=633842
thread: 1 done. Global sum now is=647273
thread: 3 done. Global sum now is=654595
thread: 2 done. Global sum now is=674961
thread: 4 done. Global sum now is=719194
thread: 7 done. Global sum now is=755402
thread: 6 done. Global sum now is=795428
thread: 0 done. Global sum now is=800000
Final Global Sum=800000
```

## 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

```
kuzu@NaruKiri:/mnt/d/tailieu/He_Dieu_Hanh/Lab/final/HDH/lab_bonus$ ./example5
Hello World from thread = 0
Number of threads = 8
Hello World from thread = 6
Hello World from thread = 7
Hello World from thread = 3
Hello World from thread = 1
Hello World from thread = 2
Hello World from thread = 5
Hello World from thread = 4
```

**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

```
kuzu@NaruKiri:/mnt/d/tailieu/He_Dieu_Hanh/Lab/final/HDH/lab_bonus$ ./example6
Iter 100 running from thread 3
Iter 101 running from thread 3
Iter 102 running from thread 3
Iter 103 running from thread 3
Iter 104 running from thread 3
Iter 105 running from thread 3
Iter 106 running from thread 3
Iter 107 running from thread 3
Iter 108 running from thread 3
Iter 109 running from thread 3
Iter 110 running from thread 3
Iter 111 running from thread 3
Iter 112 running from thread 3
Iter 113 running from thread 3
Iter 114 running from thread 3
Iter 115 running from thread 3
Iter 116 running from thread 3
Iter 197 running from thread 5
Iter 198 running from thread 5
Iter 199 running from thread 5
Vector c:
0.000000 2.000000 4.000000 6.000000 8.000000 10.000000 12.000000 14.000000 16.000000 18.000000 ...
```

**Example3:** Work-Sharing Constructs - SECTIONS Directive

```c
#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

```
        Iter 988 running from thread 4
        Iter 989 running from thread 4
        Iter 990 running from thread 4
        Iter 991 running from thread 4
        Iter 992 running from thread 4
        Iter 993 running from thread 4
        Iter 994 running from thread 4
        Iter 995 running from thread 4
        Iter 996 running from thread 4
        Iter 997 running from thread 4
        Iter 998 running from thread 4
        Iter 999 running from thread 4
Vector c:
        2.000000 4.000000 6.000000 8.000000 10.000000 12.000000 14.000000 16.000000 18.000000 20.000000 ...
Vector d:
        0.000000 3.000000 8.000000 15.000000 24.000000 35.000000 48.000000 63.000000 80.000000 99.000000 ...
Num of iter with thread:
        Thread 3 run 1000 iter.
        Thread 4 run 1000 iter.
```

**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.

```c
#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

```
kuzu@NaruKiri:/mnt/d/tailieu/He_Dieu_Hanh/Lab/final/HDH/lab_bonus$ ./example8
1st Parallel Region:
Thread 0: a, b, x = 0, 0, 1.000000
Thread 2: a, b, x = 2, 2, 3.200000
Thread 3: a, b, x = 3, 3, 4.300000
Thread 1: a, b, x = 1, 1, 2.100000
**********************************
Master thread doing serial work here
**********************************
2nd Parallel Region:
Thread 2: a, b, x = 2, 0, 3.200000
Thread 0: a, b, x = 0, 0, 1.000000
Thread 1: a, b, x = 1, 0, 2.100000
Thread 3: a, b, x = 3, 0, 4.300000
```