

# USING PTHREAD TO SOLVE THE BIG PROBLEM OF ADDING NUMBERS

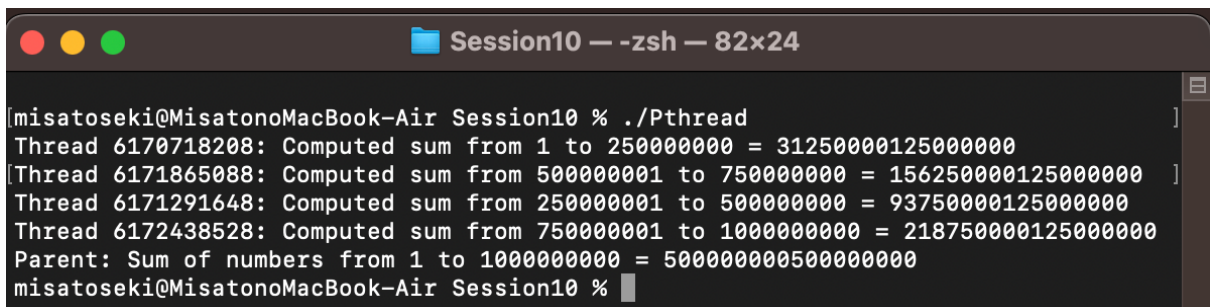
## 1. INTRODUCTION

In this report, we will observe what the code looks like when using Pthread for the process we did in the previous assignment.

## 2. C CODE FOR MULTI-THREADING USING PTHREAD

First, this code defines the adding-up method done by each thread. Each thread determines the range to calculate using passed thread ID. Then each thread sums the numbers in that range and adds the result to *sum*. When threads access *sum*, the *pthread\_mutex\_lock()* function is used to avoid race conditions.

Next, the main method is defined. The first *for-loop* sets the ID of each thread and creates the thread. *pthread\_create* function creates a new thread and specifies the function the thread will execute. In next *for-loop*, *pthread\_join* function waits for the parent process until the thread completes. Finally, the computed sum is displayed.



```
Session10 — -zsh — 82x24
[ misatoseki@MisatonoMacBook-Air Session10 % ./Pthread
Thread 6170718208: Computed sum from 1 to 250000000 = 31250000125000000
Thread 6171865088: Computed sum from 500000001 to 750000000 = 156250000125000000
Thread 6171291648: Computed sum from 250000001 to 500000000 = 93750000125000000
Thread 6172438528: Computed sum from 750000001 to 1000000000 = 218750000125000000
Parent: Sum of numbers from 1 to 1000000000 = 500000000500000000
misatoseki@MisatonoMacBook-Air Session10 % ]
```

Picture1. Executed result.

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <pthread.h>
4
5  #define THREAD_COUNT 4
6  #define TASK_SIZE 250000000
7  #define N (THREAD_COUNT * TASK_SIZE)
8
9  long sum = 0;
10 pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
11
12 void* task(void* arg)
13 {
14     int id = *(int*)arg;
15     int start = id * TASK_SIZE + 1;
16     int end = (id + 1) * TASK_SIZE;
17     long threadSum = 0;
18
19     for (int i = start; i <= end; i++)
20     {
21         threadSum += i;
22     }
23
24     pthread_mutex_lock(&mutex);
25     sum += threadSum;
26     printf("Thread %ld: Computed sum from %d to %d = %ld\n", (long)pthread_self(), start, end, threadSum);
27     pthread_mutex_unlock(&mutex);
28
29     return NULL;
30 }
31
32 int main()
33 {
34     pthread_t threads[THREAD_COUNT];
35     int thread_ids[THREAD_COUNT];
36
37     for (int i = 0; i < THREAD_COUNT; i++)
38     {
39         thread_ids[i] = i;
40         pthread_create(&threads[i], NULL, task, (void*)&thread_ids[i]);
41     }
42
43     for (int i = 0; i < THREAD_COUNT; i++)
44     {
45         pthread_join(threads[i], NULL);
46     }
47
48     printf("Parent: Sum of numbers from 1 to %d = %ld\n", N, sum);
49
50     return 0;

```

Picture2. C code for multi-threading using Pthread

### 3. WHAT I LEARNED

In this assignment, we used Pthreads to execute the process of the previous assignment. As a result, differences were found in the code in terms of thread creation, synchronization, and thread waiting.

The multi-threading process completed successfully with or without the use of Pthreads. Then what is the advantage to use Pthread? It is their compatibility. Programs using Pthreads have compatibility with any POSIX-compliant system because Pthreads are a type of POSIX thread.