Day2

Table of Contents

1. Agenda

- Race condition
- Mutex
- Reduction
- Returning from function

2. Printing ID

```
#include<stdio.h>
#include<stdlib.h>
#include<pthread.h>
#define N 10
void* printId(void* args){
    int id = *(int*) args;
    printf("Hello from %d\n", id);
    free(args);

}
int main(){
    pthread_t th[N];
    for(int i = 0; i < N; i++){
        int *threadid = malloc(sizeof(int));
        *threadid = i;
        pthread_create(&th[i], NULL, printId, (void*)threadid);
    }
    for(int i = 0; i < N; i++){
        pthread_join(th[i], NULL);
    }

}
```

```
Hello from 0
Hello from 2
Hello from 1
Hello from 3
Hello from 4
Hello from 5
Hello from 6
Hello from 7
Hello from 8
Hello from 9
```

3. Task2

Create an array of size N and let each thread to initialize the array elements in specific order given below.

```
Thread 0 will insert 0 at index 0
Thread 2 will insert 1 at index 1
Thread 2 will insert 2 at index 2
.
.
.
Thread N will insert N at index N
```

The output can be in any order.

4. Solution: Task2

```c
#include<stdio.h>
#include<stdlib.h>
#include<pthread.h>
#define N 20

int arr[N];
void *hello(void* threadId){
    long tid = (long)threadId;
    arr[tid] = tid;
    return NULL;
}

int main(){
    pthread_t* t;
    t = malloc(sizeof(pthread_t) * N);

    for(long i = 0; i < N; i++)
        pthread_create(&t[i], NULL, hello, (void*)i);
    for(long i = 0; i < N; i++)
        pthread_join(t[i], NULL);
    free(t);
    for(int i = 0; i < N; i++){
        printf("%d ", arr[i]);
    }
    return 0;
}
```

```
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
```

5. Task3

Create a program that will calculate sum upto N natural numbers.

6. Solution: Task3

This program demostrates the race condition that is happening when every thread is trying to modify the shared variable sum. The result in this case shows data loss and the actual sum is different from expected. To fix this issue we will require a lock mechanism called mutexes in pthread.

```
#include<stdio.h>
#include<stdlib.h>
#include<pthread.h>
#define N 100000

long sum = 0;
void *hello(void* threadId){
    long tid = (long)threadId;
    sum += (tid);
    return NULL;
}

int main(){
    pthread_t* t;
    t = malloc(sizeof(pthread_t) * N);

    for(long i = 0; i < N; i++)
        pthread_create(&t[i], NULL, hello, (void*)i);
    for(long i = 0; i < N; i++)
        pthread_join(t[i], NULL);
    printf("Natural Number sum: %ld\n", sum + N);
    printf("Expected Natural Number sum: %ld\n", (N * (N + 1) / 2));
    free(t);
    return 0;
}
```

```
Natural Number sum: 4998968405
Expected Natural Number sum: 705082704
```

7. Mutex

A mutex (short for “mutual exclusion”) is a synchronization primitive that allows multiple threads to share the same resource but not simultaneously. When one thread locks a mutex, other threads that try to lock it will block until it is unlocked.

7.1. Importance

- Prevents race conditions by ensuring that only one thread can access a critical section of code at a time.
- Ensures data consistency and integrity when multiple threads share and modify data.

7.2. Code

```c
#include<stdio.h>
#include<stdlib.h>
#include<pthread.h>
#define N 300000

pthread_mutex_t mutex;
long sum = 0;
void *hello(void* threadId){
    long tid = (long)threadId;
    pthread_mutex_lock(&mutex);
    sum += (tid);
    pthread_mutex_unlock(&mutex);
    return NULL;
}

int main(){
    pthread_t* t;
    pthread_mutex_init(&mutex, NULL);
    t = malloc(sizeof(pthread_t) * N);

    for(long i = 0; i < N; i++)
        pthread_create(&t[i], NULL, hello, (void*)i);
    for(long i = 0; i < N; i++)
        pthread_join(t[i], NULL);
    pthread_mutex_destroy(&mutex);
    printf("Natural Number sum: %ld\n", sum);
    printf("Natural Number sum original: %ld\n", ((N * ((N - 1) * 1L)) / 2));
    free(t);
    return 0;
}
```

```
Natural Number sum: 44999850000
Natural Number sum original: 44999850000
```

8. Task4

Initialize array of size N. Calculate sum of all the elements of the array by using reduction algorithm.

9. Reduction

```c
#include<stdio.h>
#include<stdlib.h>
#include<pthread.h>
#define N 5000000
#define T 16

pthread_mutex_t mutex;
long sum = 0;
int arr[N];

void *hello(void* threadId){
    long tid = (long)threadId;
    long localSum = 0;
    int chunk_size = N / T;
    int start = tid * chunk_size;
    int end = (tid + 1) * chunk_size;
    if (tid == T - 1) {
        end = N;
    }
    for (int i = start; i < end; i++) {
        localSum += (long)arr[i];
    }
    pthread_mutex_lock(&mutex);
    sum += localSum;
    pthread_mutex_unlock(&mutex);
    return NULL;
}

int main(){
    for(int i = 0; i < N; i++){
        arr[i] = i + 1;
    }
    pthread_t* t;
    pthread_mutex_init(&mutex, NULL);
    t = malloc(sizeof(pthread_t) * N);

    for(long i = 0; i < T; i++)
        pthread_create(&t[i], NULL, hello, (void*)i);
    for(long i = 0; i < T; i++)
        pthread_join(t[i], NULL);
    pthread_mutex_destroy(&mutex);
    printf("Sum using manual reduction: %ld\n", sum);
    printf("Natural Number sum original: %ld\n", ((N * ((N + 1) * 1L)) / 2));
    free(t);
    return 0;
}
```

```
Sum using manual reduction: 12500002500000
Natural Number sum original: 12500002500000
```

10. Reduction Alter

```
#include<stdio.h>
#include<stdlib.h>
#include<pthread.h>
#define N 5000000
#define T 16

long globalArraySum[T];
int arr[N];

void *hello(void* threadId){
    long tid = (long)threadId;
    long localSum = 0;
    int chunk_size = N / T;
    int start = tid * chunk_size;
    int end = (tid + 1) * chunk_size;
    if (tid == T - 1) {
        end = N;
    }
    for (int i = start; i < end; i++) {
        localSum += (long)arr[i];
    }
    globalArraySum[tid] = localSum;
    return NULL;
}

int main(){
    for(int i = 0; i < N; i++){
        arr[i] = i + 1;
    }
    pthread_t* t;
    t = malloc(sizeof(pthread_t) * N);
    for(long i = 0; i < T; i++)
        pthread_create(&t[i], NULL, hello, (void*)i);
    for(long i = 0; i < T; i++)
        pthread_join(t[i], NULL);
    long sum = 0;
    for(int i = 0; i < T; i++){
        sum+= globalArraySum[i];
    }
    printf("Sum using manual reduction: %ld\n", sum);
    printf("Natural Number sum original: %ld\n", ((N * ((N + 1) * 1L)) / 2));
    free(t);
    return 0;
}
```

```
Sum using manual reduction: 12500002500000
Natural Number sum original: 12500002500000
```

11. Return from function

```
#include<stdio.h>
#include<stdlib.h>
#include<pthread.h>
#define N 5000000
#define T 16

int arr[N];

void *hello(void* threadId){
    long tid = (long)threadId;
    long localSum = 0;
    int chunk_size = N / T;
    int start = tid * chunk_size;
    int end = (tid + 1) * chunk_size;
    if (tid == T - 1) {
        end = N;
    }
    for (int i = start; i < end; i++) {
        localSum += (long)arr[i];
    }
    return (void*)localSum;
}

int main(){
    for(int i = 0; i < N; i++){
        arr[i] = i + 1;
    }
    pthread_t* t;
    t = malloc(sizeof(pthread_t) * N);

    long sum = 0, localSum;
    for(long i = 0; i < T; i++)
        pthread_create(&t[i], NULL, hello, (void*)i);
    for(long i = 0; i < T; i++){
        pthread_join(t[i], (void**)&localSum);
        sum+= *(long*)&localSum;
    }
    printf("Sum using manual reduction: %ld\n", sum);
    printf("Natural Number sum original: %ld\n", ((N * ((N + 1) * 1L)) / 2));
    free(t);
    return 0;
}
```

```
Sum using manual reduction: 12500002500000
```

```
Natural Number sum original: 12500002500000
```

12. Returning from function

```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#define N 30000
#define T 4

int arr[N];

void *hello(void* threadId) {
    long tid = (long)threadId;
    long *localSum = malloc(sizeof(long)); // Allocate memory for the local sum
    *localSum = 0;
    int chunk_size = N / T;
    int start = tid * chunk_size;
    int end = (tid + 1) * chunk_size;

    // Ensure the last thread processes the remaining elements
    if (tid == T - 1) {
        end = N;
    }

    for (int i = start; i < end; i++) {
        *localSum += arr[i];
    }

    return (void*)localSum;
}

int main() {
    for (int i = 0; i < N; i++) {
        arr[i] = i + 1;
    }

    pthread_t threads[T];
    void *status;
    long sum = 0;

    // Create threads
    for (long i = 0; i < T; i++) {
        pthread_create(&threads[i], NULL, hello, (void*)i);
    }

    // Join threads and aggregate the local sums
    for (long i = 0; i < T; i++) {
        pthread_join(threads[i], &status);
        sum += *(long*)status;
        free(status); // Free the allocated memory for the local sum
    }

    printf("Sum using manual reduction: %ld\n", sum);
    printf("Natural Number sum original: %ld\n", ((N * 1L * (N + 1)) / 2));

    return 0;
}
```

```
Sum using manual reduction: 450015000
Natural Number sum original: 450015000
```

13. Sum of all the elements of the array

```c
#include<stdio.h>
#include<stdlib.h>
#include<pthread.h>
#define N 300000

pthread_mutex_t mutex;
long sum = 0;
int arr[N];
void *hello(void* threadId){
    long tid = (long)threadId;
    pthread_mutex_lock(&mutex);
    sum += arr[tid];
    pthread_mutex_unlock(&mutex);
    return NULL;
}

int main(){
    for(int i = 0; i < N; i++){
        arr[i] = i + 1;
    }
    pthread_t* t;
    pthread_mutex_init(&mutex, NULL);
    t = malloc(sizeof(pthread_t) * N);

    for(long i = 0; i < N; i++)
        pthread_create(&t[i], NULL, hello, (void*)i);
    for(long i = 0; i < N; i++)
        pthread_join(t[i], NULL);
    pthread_mutex_destroy(&mutex);
    printf("Natural Number sum: %ld\n", sum);
    printf("Natural Number sum original: %ld\n", ((N * ((N + 1) * 1L)) / 2));
    free(t);
    return 0;
}
```

```
Natural Number sum: 45000150000
Natural Number sum original: 45000150000
```

Author: Abhishek Raj
Created: 2024-06-26 Wed 15:06