Day2

Table of Contents

1. Agenda

- Passing arguments to the function
- Race condition
- Mutex

2. Creating N number of threads

In this code you can see the issue while passing address of iteration for thread ID.

2.1. code

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

void* hello(void* threadId){
    int tid = *(int*)threadId;
    printf("Hello World %d of %d\n", tid, N);
}

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

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

2.2. compile

```
file=pth3
gcc $file.c -o $file.out -lpthread
```

2.3. run

```
file=pth3
./$file.out
```

```
Hello World 3 of 20
Hello World 4 of 20
Hello World 4 of 20
Hello World 4 of 20
Hello World 7 of 20
Hello World 7 of 20
Hello World 8 of 20
Hello World 9 of 20
Hello World 10 of 20
Hello World 11 of 20
Hello World 11 of 20
Hello World 12 of 20
Hello World 13 of 20
Hello World 14 of 20
Hello World 15 of 20
Hello World 16 of 20
Hello World 17 of 20
Hello World 18 of 20
Hello World 19 of 20
Hello World 20 of 20
```

3. Program to check address of each iteration

```c
#include<stdio.h>
int main(){
    for(int i = 0; i < 20; i++){
        printf("%d is located at %p\n", i, &i);
    }
    return 0;
}
```

```
0 is located at 0x7ffe03362874
1 is located at 0x7ffe03362874
2 is located at 0x7ffe03362874
3 is located at 0x7ffe03362874
4 is located at 0x7ffe03362874
5 is located at 0x7ffe03362874
6 is located at 0x7ffe03362874
7 is located at 0x7ffe03362874
8 is located at 0x7ffe03362874
9 is located at 0x7ffe03362874
10 is located at 0x7ffe03362874
11 is located at 0x7ffe03362874
12 is located at 0x7ffe03362874
13 is located at 0x7ffe03362874
14 is located at 0x7ffe03362874
15 is located at 0x7ffe03362874
16 is located at 0x7ffe03362874
17 is located at 0x7ffe03362874
18 is located at 0x7ffe03362874
```

```
19 is located at 0x7ffe03362874
```

4. Create array of size N using N threads

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

int* arr;
void* hello(void* threadId){
    int tid = *(int*)threadId;
    arr[tid] = tid;
    free(threadId);
}
int main(){
    arr = malloc(sizeof(int) * N);
    pthread_t* t;
    t = malloc(sizeof(pthread_t) * N);
    for(int i = 0; i < N; i++){
        int* a;
        a = malloc(sizeof(int));
        *a = i;
        pthread_create(&t[i], NULL, hello, (void*)a);
    }
    for(int i = 0; i < N; i++)
        pthread_join(t[i], NULL);
    free(t);
    for(int i = 0; i < N; i++) printf("%d ", arr[i]);
    printf("\n");
    return 0;
}
```

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

5. Sum of elements of an array of size N using N threads

In below code you'll experience race condition.

```c
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<pthread.h>
#define N 20000

int sum = 0;
void* hello(void* threadId){
    int tid = *(int*)threadId;
    sum+= tid;
    free(threadId);
}

int main(){
    pthread_t* t;
    t = malloc(sizeof(pthread_t) * N);
    for(int i = 0; i < N; i++){
        int* a;
```

```
        a = malloc(sizeof(int));
        *a = i;
        pthread_create(&t[i], NULL, hello, (void*)a);
    }
    for(int i = 0; i < N; i++)
        pthread_join(t[i], NULL);
    free(t);

    printf("Sum = %d\n", sum);
    if(sum + N == (N * (N + 1) / 2)){
        printf("_____Passed_____\n");
    }
    else{
        printf("_____Failed_____\n");
    }
    return 0;
}
```

```
gcc sumWithRaceCondition.c -o sumWithRaceCondition.out -lpthread
```

```
./sumWithRaceCondition.out
```

```
Sum = 199983649
_____Failed_____
```

```
./sumWithRaceCondition.out
```

```
Sum = 199973140
_____Failed_____
```

```
./sumWithRaceCondition.out
```

```
Sum = 199990000
_____Passed_____
```

6. Mutex: Sum of elements of an array of size N using N threads

```c
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<pthread.h>
#define N 10000

pthread_mutex_t mutex;

int sum = 0;
void* hello(void* threadId){
    int tid = *(int*)threadId;

    pthread_mutex_lock(&mutex);
    sum+= tid;
    pthread_mutex_unlock(&mutex);
```

```
    free(threadId);
}
int main(){
    pthread_t* t;
    pthread_mutex_init(&mutex, NULL);
    t = malloc(sizeof(pthread_t) * N);
    for(int i = 0; i < N; i++){
        int* a;
        a = malloc(sizeof(int));
        *a = i;
        pthread_create(&t[i], NULL, hello, (void*)a);
    }
    for(int i = 0; i < N; i++)
        pthread_join(t[i], NULL);
    free(t);

    pthread_mutex_destroy(&mutex);
    printf("Sum = %d\n", sum);
    if(sum + N == (N * (N + 1) / 2)){
        printf("_____Passed_____\n");
    }
    else{
        printf("_____Failed_____\n");
    }
    return 0;
}
```

```
Sum = 49995000
_____Passed_____
```

Author: Abhishek Raj
Created: 2024-12-13 Fri 07:24