Assingment 5
Nama : Kosmas Rio Legowo
NIM : 23/512012/PA/21863

7.1 Develop a Simple Thread Program

Awalnya terdapat fungsi pthread_join

```
int main() {

    pthread_t a_thread;
    pthread_create(newthread: &a_thread, attr: NULL, start_routine: thread_function, arg: NULL);
    pthread_join(th: a_thread, thread_return: NULL);
    printf(format: "Inside Main Program\n");
    for(j=4;j>=0;j--)
    {
        printf(format: "%d\n",j);
        sleep(seconds: 1);
    }
}
```

```
kosmasrio_legowo@cloudshell:~$ gcc -o simple_thread simple_thread.c -lpthread
kosmasrio_legowo@cloudshell:~$ ./simple_thread
Inside Thread
0
1
2
3
4
Inside Main Program
4
3
2
1
0
kosmasrio_legowo@cloudshell:~$
```

Kemudian setelah diberi comment di line yang berisi “pthread_join”

```
int main() {

    pthread_t a_thread;
    pthread_create(newthread: &a_thread, attr: NULL, start_routine: thread_function, arg: NULL);
    //pthread_join(a_thread, NULL);
    printf(format: "Inside Main Program\n");
    for(j=4;j>=0;j--)
    {
        printf(format: "%d\n",j);
        sleep(seconds: 1);
    }
}
```

```
kosmasrio_legowo@cloudshell:~$ gcc -o simple_thread simple_thread.c -lpthread
kosmasrio_legowo@cloudshell:~$ ./simple_thread
Inside Main Program
4
Inside Thread
0
3
1
2
2
1
3
0
4
```

Hal yang terjadi akibat dari line yang berisi “pthread_join” diberi comment adalah program utama berjalan bersamaan secara paralel dengan Thread karena fungsi dari “pthread_join” yang akan mengharuskan program untuk menunggu Thread dari parameter fungsi “pthread_join” selesai justru kita comment, sehingga program berjalan secara bersamaan dengan Thread tanpa menunggu Thread tersebut selesai dijalankan.

7.2 Passing and Receiving Values to and from a Thread

inout_thread.c > main

```c
#include<stdlib.h>
#include<unistd.h>
#include<pthread.h>
struct arg_struct
{
    int a;
    int b;
    int sum;
};
void *addition(void *arguments){
    struct arg_struct *args = arguments;
    args -> sum = args -> a + args -> b;
    pthread_exit(retval: NULL);
}
int main(){
    pthread_t t;
    struct arg_struct *args =malloc(size: sizeof *args);
    args -> a = 10;
    args -> b = 5;
    pthread_create(newthread: &t, attr: NULL, start_routine: addition, arg: args);
    pthread_join(th: t, thread_return: NULL);
    printf(format: "%d + %d = %d\n", args -> a, args -> b, args -> sum);
}
```

```
kosmasrio_legowo@cloudshell:~$ gcc -o inout_thread.out inout_thread.c -lpthread
kosmasrio_legowo@cloudshell:~$ ./inout_thread.out
10 + 5 = 15
```

Kode tersebut menggunakan pthreads untuk melakukan penjumlahan dua angka secara paralel. Struct arg_struct digunakan untuk menyimpan nilai a, b, dan sum. Fungsi addition() yang dijalankan dalam thread menambahkan a dan b, lalu menyimpan hasilnya di sum. Dalam fungsi main(), thread dibuat menggunakan pthread_create(), dan pthread_join() digunakan untuk menunggu thread selesai sebelum mencetak hasil penjumlahan. Nilai yang dijumlahkan (10 + 5) kemudian dicetak sebagai output.

7.3 Deadlock

Awalnya function2 berisi dan menghasilkan output sebagai berikut:

```
void *function2() {
    pthread_mutex_lock(mutex: &res_b);
    printf(format: "Thread TWO acquired res_b\n");
    sleep(seconds: 1);
    pthread_mutex_lock(mutex: &res_a);
    printf(format: "Thread TWO acquired res_a\n");
    pthread_mutex_unlock(mutex: &res_a);
    printf(format: "Thread TWO released res_a\n");
    pthread_mutex_unlock(mutex: &res_b);
    printf(format: "Thread TWO released res_b\n");
}
```

```
kosmasrio_legowo@cloudshell:~$ gcc -o deadlock.out deadlock.c -lpthread
kosmasrio_legowo@cloudshell:~$ ./deadlock.out
Thread ONE acquired res_a
Thread TWO acquired res_b
^C
```

Kemudian setelah function2 dimodifikasi sehingga mempunyai urutan yang sama seperti function1:

```
void *function2() {
  pthread_mutex_lock(mutex: &res_a);
    printf(format: "Thread ONE acquired res_a\n");
    sleep(seconds: 1);
    pthread_mutex_lock(mutex: &res_b);
    printf(format: "Thread ONE acquired res_b\n");
    pthread_mutex_unlock(mutex: &res_b);
    printf(format: "Thread ONE released res_b\n");
    pthread_mutex_unlock(mutex: &res_a);
    printf(format: "Thread ONE released res_a\n");
}
```

```
kosmasrio_legowo@cloudshell:~$ gcc -o deadlock.out deadlock.c -lpthread
kosmasrio_legowo@cloudshell:~$ ./deadlock.out
Thread ONE acquired res_a
Thread ONE acquired res_b
Thread ONE released res_b
Thread ONE released res_a
Thread ONE acquired res_a
Thread ONE acquired res_b
Thread ONE released res_b
Thread ONE released res_a
Thread joined
```

Yang terjadi adalah Thread pertama dan Thread kedua masing-masing meminta resource a terlebih dahulu. Akan tetapi, karena Thread kedua berjalan lebih cepat dan berhasil mendapatkan resource a terlebih dahulu. Hal ini mengakibatkan fungsi Thread pertama untuk menunggu resource a “dilepaskan” oleh Thread lain yang menggunakannya (dalam hal ini Thread kedua). Sehingga fungsi yang menggunakan Thread pertama sepenuhnya masih menunggu fungsi dari Thread kedua untuk selesai dijalankan hingga resource a dilepaskan dan dapat diakses oleh fungsi dari Thread pertama hingga akhirnya kedua fungsi Thread berhasil dijalankan secara berurutan.