GTU Department of Computer Engineering
CSE312-Spring 2022
Homework 1 Report

AHMET OKUR
1801042655

Problem Defination

We have to implement a multithreading library for this simple OS. Our library have to consist of functions for creating, terminating, yielding, and joining the threads.

We should Write a number of threads that communicate with each other in a producer consumer fashion. We should include the Peterson algorithm in these threads.

Plan & Design

- I made my design inspired by the operating system written by Viktor Engelmann.

- I used the Round-Robin scheduling algorithm written by Viktor Engelmann. The reason I use Round-Robin is that my threads don't have any priority.

- The main thread structure requires us to define the start address of the thread, for the program counter, registers for keeping and changing value of items in thread, Stack and CPU State.

- We have a thread that performs one producer and one consumer transaction.

- If the buffer is full, producer algorithm becomes busy waiting. If the buffer is empty, the consumer algorithm becomes busy waiting.

- Producer-consumer algorithms can become race conditions within our means. I tried to solve this with the Peterson algorithm.

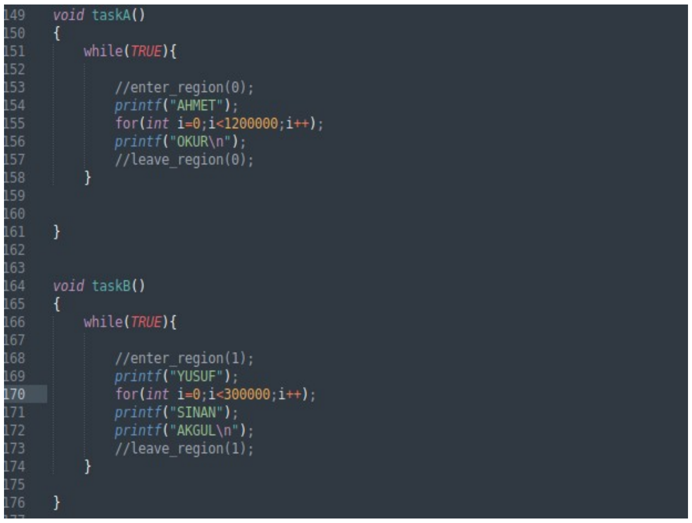
Failed Missions

- I couldn't do the yield and join tasks because I couldn't figure out the logic of the interrupt operation in the operating system we are working on.

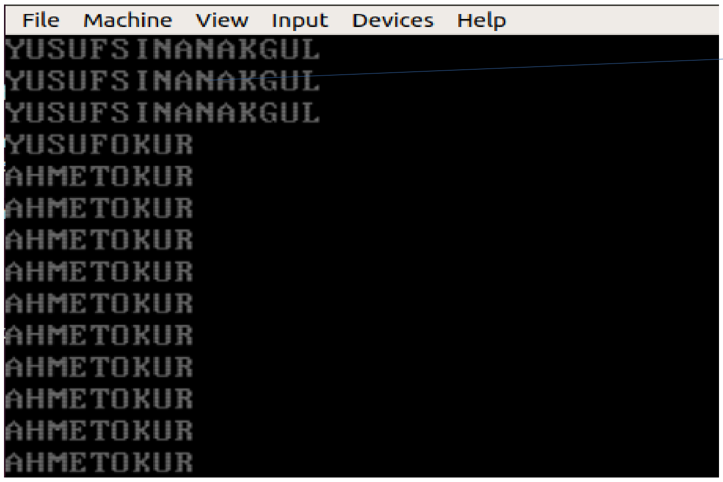
Test Case and Codes

Simple implementation of the Peterson algorithm

1)Without the Peterson algorithm (race condition)



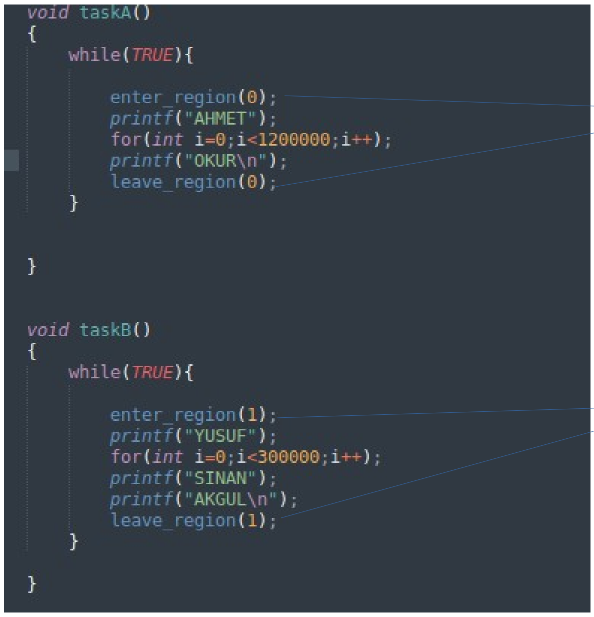
```c
void taskA()
{
    while(TRUE){

        //enter_region(0);
        printf("AHMET");
        for(int i=0;i<1200000;i++);
        printf("OKUR\n");
        //leave_region(0);
    }


}


void taskB()
{
    while(TRUE){

        //enter_region(1);
        printf("YUSUF");
        for(int i=0;i<300000;i++);
        printf("SINAN");
        printf("AKGUL\n");
        //leave_region(1);
    }

}
```



```
YUSUFSINANAKGUL
YUSUFSINANAKGUL
YUSUFSINANAKGUL
YUSUFOKUR
AHMETOKUR
AHMETOKUR
AHMETOKUR
AHMETOKUR
AHMETOKUR
AHMETOKUR
AHMETOKUR
AHMETOKUR
AHMETOKUR
AHMETOKUR
```

Race condition

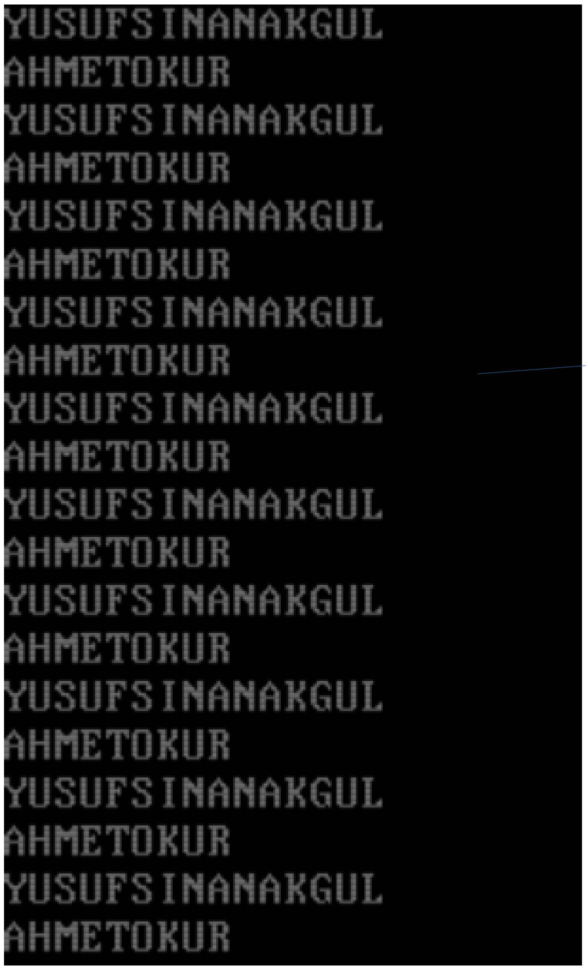
2) Peterson algorithm



```
void taskA()
{
    while(TRUE){

        enter_region(0);
        printf("AHMET");
        for(int i=0;i<1200000;i++);
        printf("OKUR\n");
        leave_region(0);
    }


}


void taskB()
{
    while(TRUE){

        enter_region(1);
        printf("YUSUF");
        for(int i=0;i<300000;i++);
        printf("SINAN");
        printf("AKGUL\n");
        leave_region(1);
    }

}
```

Critical Region

Critical Region

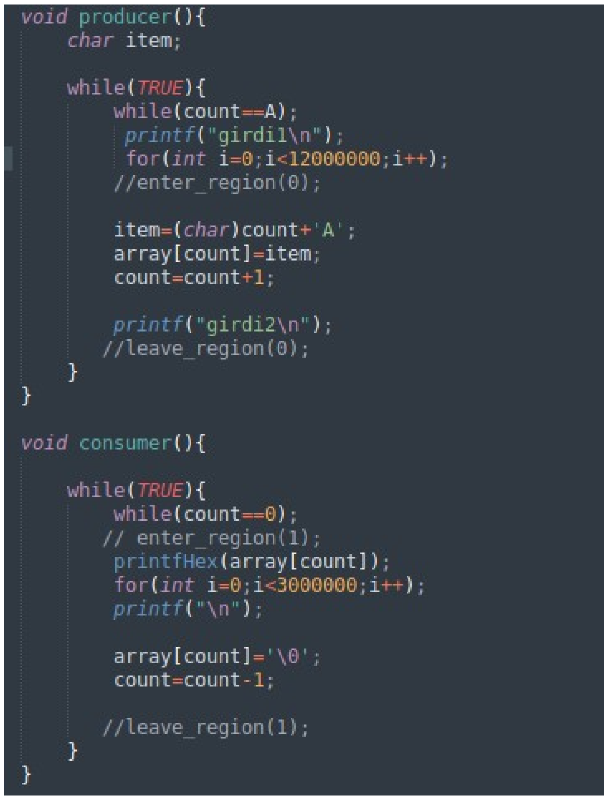


```
YUSUFSINANAKGUL
AHMETOKUR
YUSUFSINANAKGUL
AHMETOKUR
YUSUFSINANAKGUL
AHMETOKUR
YUSUFSINANAKGUL
AHMETOKUR
YUSUFSINANAKGUL
AHMETOKUR
YUSUFSINANAKGUL
AHMETOKUR
YUSUFSINANAKGUL
AHMETOKUR
YUSUFSINANAKGUL
AHMETOKUR
YUSUFSINANAKGUL
AHMETOKUR
YUSUFSINANAKGUL
AHMETOKUR
```

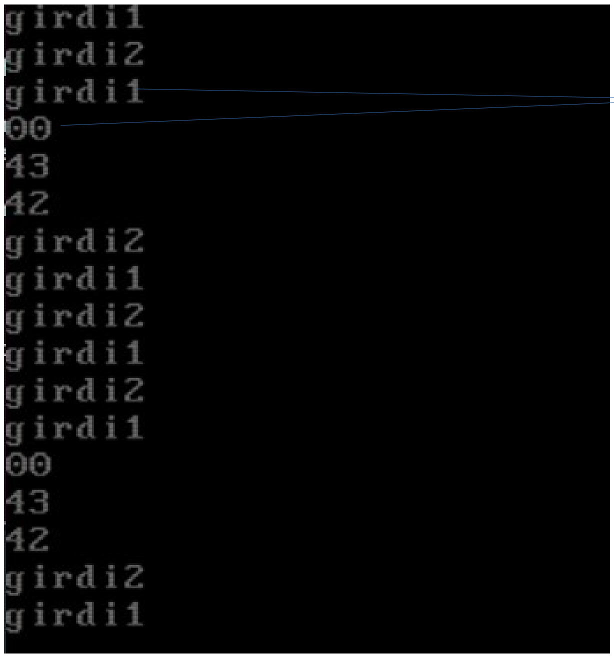
You cant see race condition

Producer consumer threads and peterson algorithm

1)Without the Peterson algorithm (race condition)

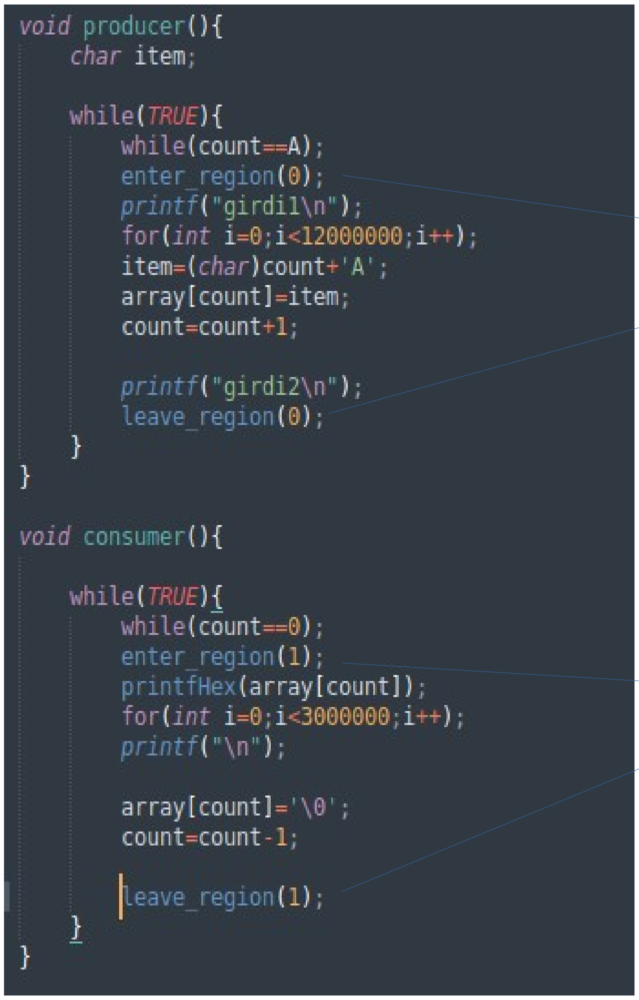


```
void producer(){
    char item;

    while(TRUE){
        while(count==A);
         printf("girdi1\n");
         for(int i=0;i<12000000;i++);
        //enter_region(0);

        item=(char)count+'A';
        array[count]=item;
        count=count+1;

        printf("girdi2\n");
       //leave_region(0);
    }
}

void consumer(){

    while(TRUE){
        while(count==0);
       // enter_region(1);
        printfHex(array[count]);
        for(int i=0;i<3000000;i++);
        printf("\n");

        array[count]='\0';
        count=count-1;

       //leave_region(1);
    }
}
```



```
girdi1
girdi2
girdi1
00
43
42
girdi2
girdi1
girdi2
girdi1
girdi2
girdi1
00
43
42
girdi2
girdi1
```

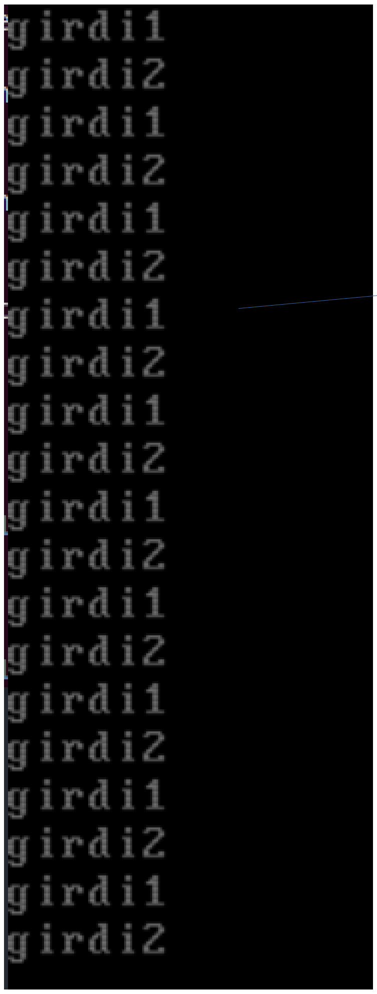
this part should be girdi1 and girdi2
(this code is not working well)



```
void producer(){
    char item;

    while(TRUE){
        while(count==A);
        enter_region(0);
        printf("girdi1\n");
        for(int i=0;i<12000000;i++);
        item=(char)count+'A';
        array[count]=item;
        count=count+1;

        printf("girdi2\n");
        leave_region(0);
    }
}

void consumer(){

    while(TRUE){
        while(count==0);
        enter_region(1);
        printfHex(array[count]);
        for(int i=0;i<3000000;i++);
        printf("\n");

        array[count]='\0';
        count=count-1;

        leave_region(1);
    }
}
```

Critical Region

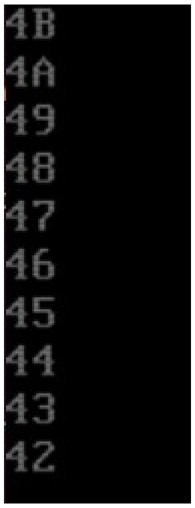
Critical Region



```
girdi1
girdi2
girdi1
girdi2
girdi1
girdi2
girdi1
girdi2
girdi1
girdi2
girdi1
girdi2
girdi1
girdi2
girdi1
girdi2
girdi1
girdi2
girdi1
girdi2
```

Producer

consumer



```
4B
4A
49
48
47
46
45
44
43
42
```