Operating Systems Assignment 3
Petersons Algorithm for N Processes

Team:

1- Srinivas Piskala Ganesh Babu(spg349) – N13138339
2- Nanda Kishore Kalidindi (nkk263) – N16138926

Petersons Solution shares two data items between processes:

1. Int Turn - *Whose turn to enter the Critical Section ?*
2. Boolean flag [n] - *Whose is READY to enter the Critical Section ?*

----------- Approach 1:

Language Used: Ruby

Using the **FLAG or READY array** to check if any other process in the array is READY before execution of the critical section. If no other process is Ready at a given time, Execute the critical section else wait.

Flow:

- Once the Process is ready to execute the critical section, it sets the ***TURN = [self]*** and ***FLAG = [True]*** variable.
- Check the other processes by iteration if their ***FLAG = [TRUE]***
  - ***If True:*** *Wait*
  - ***Else:*** *Execute the Critical Section*
- *Therefore, **Bound Wait and Progress** Achieved*

Code:

```
@n = 9

@turn = 0
@interested = @n.times.map { |x| false }

# Counter to validate if the operations in critial region are executing as
# expected
@count = 0

def enter_region(process)
  @interested[process] = true
  @turn = process
  while (@turn == process) && are_other_processes_interested?(process)
  end
  @count = @count + 1
  leave_region(process)
end

def leave_region(process)
```

```
  @interested[process] = false
end

# Method that returns true if any other process is in the interested state
def are_other_processes_interested?(process)
  @interested.each_with_index do |x, index|
    if index != process && x == true
      return true
    end
  end
  false
end

def process_execution(process)
  enter_region(process)
end

t = []

(0..@n).each do |x|
  t[x] = Thread.new{process_execution(x)}
end

(0..@n).each do |x|
  t[x].join
end

p @count
```

----------- Approach 2:

Language Used: C

Non Greedy Approach similar to 2 Process Peterson Method. Every process gives the turn to the next process (turn -> Next) and puts itself in the ready state (Flag[self] = True).

- Turn being an INT variable has only one value at a given point of time and hence the race condition can be avoided.

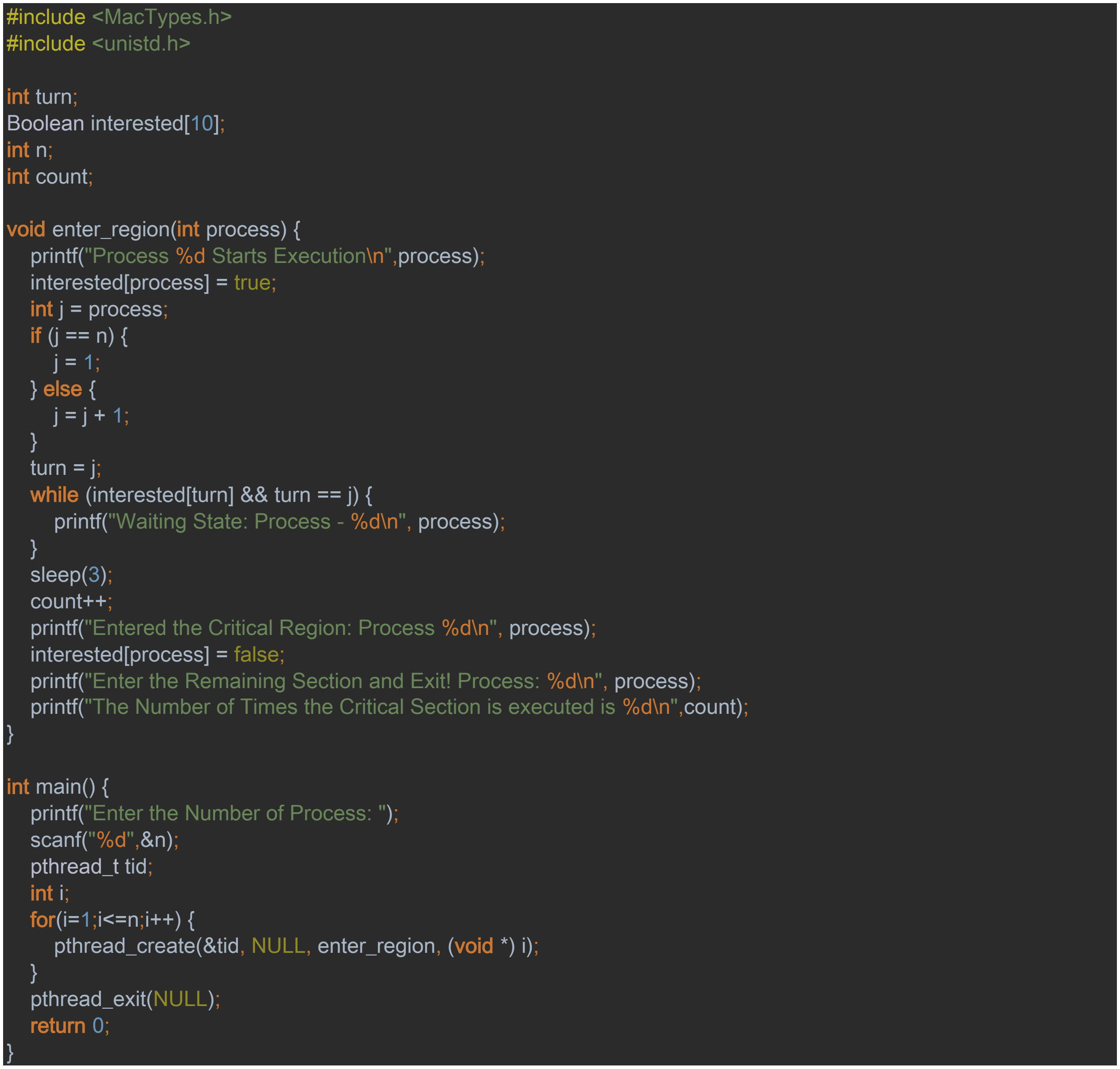
Flow:

- Once the Process is ready to execute the critical section, it sets the ***FLAG = [True]*** and ***TURN = [self + 1]*** variable.
- Check the next processes ***(self + 1)*** if its ***FLAG = [TRUE] and TURN = [self + 1]***
    - ***If True:*** *Wait*
    - ***Else:*** *Execute the Critical Section*
- *Therefore,* ***Bound Wait and Progress*** *Achieved*

Code:

```
#include <stdio.h>
#include <pthread.h>
```

Operating Systems Assignment 3
Petersons Algorithm for N Processes



```
#include <MacTypes.h>
#include <unistd.h>

int turn;
Boolean interested[10];
int n;
int count;

void enter_region(int process) {
    printf("Process %d Starts Execution\n",process);
    interested[process] = true;
    int j = process;
    if (j == n) {
        j = 1;
    } else {
        j = j + 1;
    }
    turn = j;
    while (interested[turn] && turn == j) {
        printf("Waiting State: Process - %d\n", process);
    }
    sleep(3);
    count++;
    printf("Entered the Critical Region: Process %d\n", process);
    interested[process] = false;
    printf("Enter the Remaining Section and Exit! Process: %d\n", process);
    printf("The Number of Times the Critical Section is executed is %d\n",count);
}

int main() {
    printf("Enter the Number of Process: ");
    scanf("%d",&n);
    pthread_t tid;
    int i;
    for(i=1;i<=n;i++) {
        pthread_create(&tid, NULL, enter_region, (void *) i);
    }
    pthread_exit(NULL);
    return 0;
}
```

Operating Systems Assignment 3

Petersons Algorithm for N Processes