May 29, 2021

GTU Department of Computer Engineering
CSE312/CSE504 - Spring 2021
Homework 2 Report

Akif Kartal
171044098

1 Problem Definition

The problem is to implement a kernel that will support multi- threading, interrupt handling and inter-process communication in SPIM simulator.

2 Solution

In order to solve this problem first we need to implement **required system calls** for this problem.

2.1 System Calls

2.1.1 Create Thread Syscall

In order to create thread correct way we need to this syscall.

Assembly Test File;

```
 .data
msg:    .asciiz "thrFunc"
msg3:   .asciiz "All Threads are finished!\n"
msg2:   .asciiz "Waiting to join all threads...\n"
msg1:   .asciiz "Hello from Thread\n"
msg4:   .asciiz "\n\n"
msg5:   .asciiz "Global Variable: "
.text
.globl thrFunc
main:
        li $t0,1 #global variable initial value is 1

        li $v0, 18 #create syscall
        la $a0, msg
        syscall

        li $v0, 18 #create syscall
        la $a0, msg
        syscall
```

Figure 1

Output;

```
akif@ubuntu:~/Desktop/operating-system/HW2/spim/spim$ ./spim -file "test2.s"
Loaded: /usr/share/spim/exceptions.s
Waiting to join all threads...
------------All Threads-----------
ThreadID:0
Thread Name:main
Thread Function Name:main
Program Counter:0x4
Stack Pointer address:0
State:Running

ThreadID:1
Thread Name:thread1
Thread Function Name:thrFunc
Program Counter:0x4000a0
Stack Pointer address:1f9363a0
State:Ready

ThreadID:2
Thread Name:thread2
Thread Function Name:thrFunc
Program Counter:0x4000a0
Stack Pointer address:1f9364a0
State:Ready

----------------------------------
```

Figure 2

2.1.2 Join Thread Syscall

In order to join thread correct way we need to this syscall. My Join syscal is waiting all thread to be finished.

Assembly Test File;

Figure 3

2.1.3 Exit Thread Syscall

In order to join thread correct way we need to this syscall.

Assembly Test File;

Figure 4

2.1.4 Combine All Syscall

We have implemented required system calls. In following test we will test these by incrementing a global variable which is shared between all threads.

Following two picture contains test file assembly codes;

```
msg:   .asciiz "thrFunc"
msg3:  .asciiz "All Threads are finished!\n"
msg2:  .asciiz "Waiting to join all threads...\n"
msg1:  .asciiz "Hello from Thread\n"
msg4:  .asciiz "\n\n"
msg5:  .asciiz "Global Variable: "
.text
.globl thrFunc
main:
        li $t0,1 #global variable initial value is 1

        li $v0, 18 #create syscall
        la $a0, msg
        syscall

        li $v0, 18 #create syscall
        la $a0, msg
        syscall
join:
        li $v0, 4
        la $a0, msg2
        syscall

        li $v0, 19 # join syscall
        syscall
        bnez $v0, join # if result is not zero wait

        li $v0, 4
        la $a0, msg3 #threads are finished message
        syscall
exitMain:
        li $v0, 4
        la $a0, msg5
        syscall

        li $v0, 1
        move $a0,$t0
        syscall

        li $v0, 4
        la $a0, msg4
        syscall

        li $v0, 20 # exit thread syscall
        syscall
```

(a) first half of code

```
thrFunc:
        #thread function for both thread
        li $v0, 4
        la $a0, msg1 # give message
        syscall

        addi $t0,$t0,1 # increment global variable

        li $v0, 4
        la $a0, msg5
        syscall

        li $v0, 1 #print gloabal variable
        move $a0,$t0
        syscall

        li $v0, 4
        la $a0, msg4
        syscall
exitThr:
        li $v0, 20 # exit thread syscall
        syscall
```

(b) other half

Test Results

```
akif@ubuntu:~/Desktop/operating-system/HW2/spim/spim$ ./spim -file "test2.s"
Loaded: /usr/share/spim/exceptions.s
Waiting to join all threads...
-----------All Threads----------
ThreadID:0
Thread Name:main
Thread Function Name:main
Program Counter:0x4
Stack Pointer address:0
State:Running

ThreadID:1
Thread Name:thread1
Thread Function Name:thrFunc
Program Counter:0x4000a0
Stack Pointer address:1f9363a0
State:Ready

ThreadID:2
Thread Name:thread2
Thread Function Name:thrFunc
Program Counter:0x4000a0
Stack Pointer address:1f9364a0
State:Ready

---------------------------------
```

(c) first half of result

```
***Thread Was Switched***
ThreadID:1
Thread Name:thread1
Thread Function Name:thrFunc
Program Counter:0x4000a0
Stack Pointer address:1f9363a0
State:Running

Hello from Thread
Global Variable: 2

***Thread Was Switched***
ThreadID:2
Thread Name:thread2
Thread Function Name:thrFunc
Program Counter:0x4000a0
Stack Pointer address:1f9364a0
State:Running

Hello from Thread
Global Variable: 3

***Thread Was Switched***
ThreadID:0
Thread Name:main
Thread Function Name:main
Program Counter:0x400054
Stack Pointer address:1f9362a0
State:Running

Waiting to join all threads...
All Threads are finished!
Global Variable: 3

akif@ubuntu:~/Desktop/operating-system/HW2/spim/spim$
```

(d) other half

In this test, context switch was made by using **Round Robin scheduling algorithm** as can be seen in result.

Also, since thread function is very simple in this test there was no timer interrupt.

3 Problems

3.1 Producer Consumer Problem

In this problem we have two different version first is without mutex and second is with mutex.In this test buffer is a variable only not an array

3.1.1 Without Mutex

In this version we will not use mutex and we will show that there is a race condition.

Main Thread assembly Code;

```
.data
msg:   .asciiz "producer1"
msg1:   .asciiz "consumer1"
msg2:   .asciiz "Buffer in Main: "
msg3:   .asciiz "Buffer in Producer: "
msg9:   .asciiz "Buffer in Consumer: "
msg4:  .asciiz "Waiting to join all threads...\n"
msg5:  .asciiz "All threads are finished!\n"
msg7:  .asciiz " "
msg8:  .asciiz "Waiting for mutex...\n"

newLine:   .asciiz "\n"
.text
.globl producer1
.globl consumer1
main:

    li $t0,0 #global buffer between threads
    li $t1,10 #global variable

    #create producer thread
    li $v0, 18
    la $a0, msg
    syscall

    #create consumer thread
    li $v0, 18
    la $a0, msg1
    syscall
joinStart:
    #join threads(pthread_join())
    li $v0, 4
    la $a0, msg4 #print message
    syscall

    li $v0, 4
    la $a0, msg2 #print message
    syscall
```

(e) first half of code

```

    li $v0, 1 #print buffer
    move $a0, $t0
    syscall

    li $v0, 4
    la $a0, newLine #newLine
    syscall

    li $v0, 19  #join syscall
    syscall
    #if result is not zero
    #keep waiting for all threads
    bnez  $v0, joinStart

    #if all threads are joined...
    li $v0, 4
    la $a0, msg5
    syscall
exit:
    li $v0, 20 # Main exit
    syscall
```

(f) other half

Other Threads assembly Code(no mutex yet!);

```
#----------PART1(no mutex)----------------

producer1:
    li $t2,0 # int i = 0
    #in random time make context switch
        #generate 3 random number

producerLoop1:
    bgt $t2,$t1,exitProducerThread1 # if i > 10 then exit

    #add element to buffer
    addi $t0,$t0,1
    li $v0, 4
    la $a0, msg3 #print message
    syscall

    li $v0, 1 #print buffer
    move $a0, $t0
    syscall

    li $v0, 4
    la $a0, newLine #newLine
    syscall

    beq $t2,2,contextSwitch1
    beq $t2,7,contextSwitch1

    addi $t2,$t2,1 # ++i
    j producerLoop1
contextSwitch1:
    li $v0, 24 # context Switch syscall
    syscall
    j producerLoop1

exitProducerThread1:
    li $v0, 20 # exit thread syscall
    syscall
```

(g) Producer Thread

```
consumer1:
    li $t3,0 # int i = 0

    #give waiting mutex message
    #li $v0, 4
    #la $a0, msg8 #print message
    #syscall
consumerLoop1:
    bgt $t3,$t1,exitConsumerThread1 # if i > 10 then exit

    #remove element to buffer
    addi $t0,$t0,-1

    li $v0, 4
    la $a0, msg9 #print message
    syscall

    li $v0, 1 #print buffer
    move $a0, $t0
    syscall

    li $v0, 4
    la $a0, newLine #newLine
    syscall

    beq $t3,4,contextSwitch2
    beq $t3,9,contextSwitch2

    addi $t3,$t3,1 # ++i
    j consumerLoop1
contextSwitch2:
    li $v0, 24 # context Switch syscall
    syscall
    j consumerLoop1
exitConsumerThread1:
    li $v0, 20 # exit thread syscall
    syscall
```

(h) Consumer Thread

What are we doing in this test?

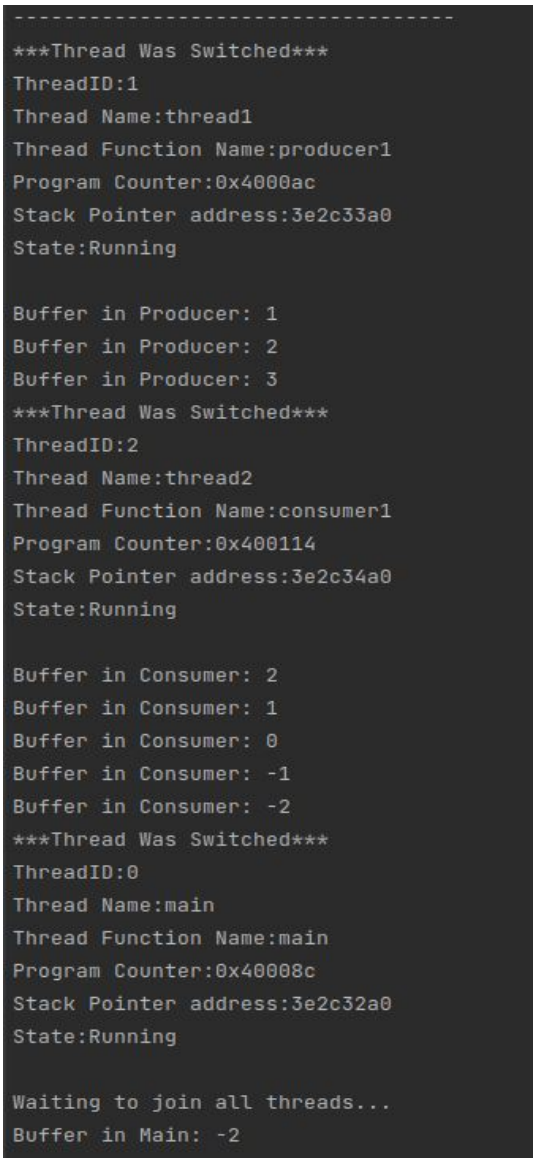
In this test we have a global buffer and producer is incrementing it, also consumer is decrementing it. But, while doing this in both producer and consumer threads we are making 2 times context switch by using a context switch syscall in order to simulate real problem. Lastly, we are printing buffer's situation in all threads(main,producer,consumer). Check the following test results to see race condition.

Test Results (no mutex yet!);

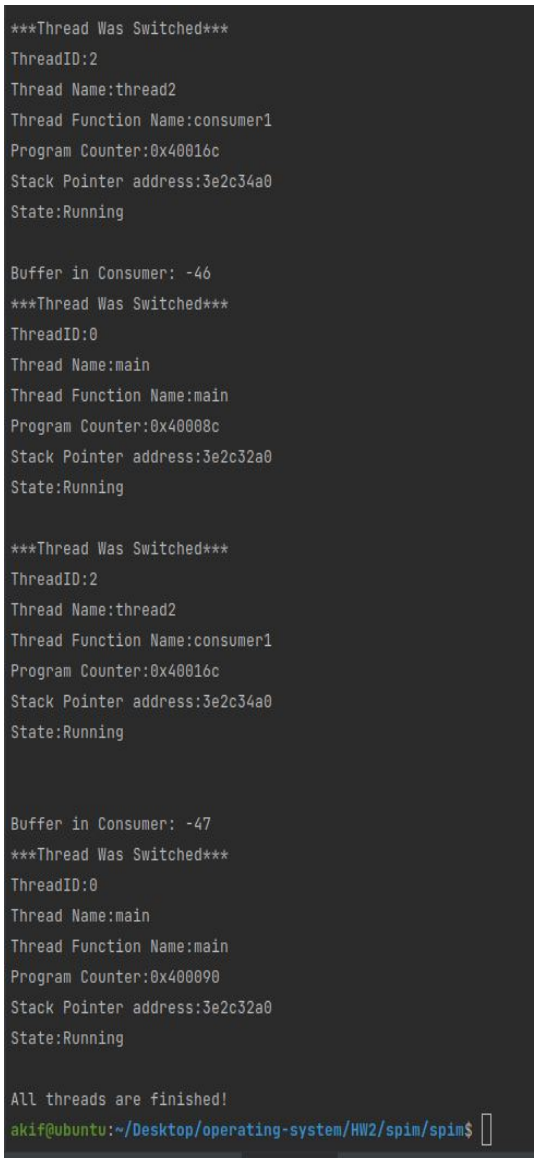
```
akif@ubuntu:~/Desktop/operating-system/HW2/spim/spim$ ./spim -file "SPIMOS_GTU_2.s"
Loaded: /usr/share/spim/exceptions.s
Waiting to join all threads...
Buffer in Main: 0
------------All Threads-----------
ThreadID:0
Thread Name:main
Thread Function Name:main
Program Counter:0x4
Stack Pointer address:0
State:Running

ThreadID:1
Thread Name:thread1
Thread Function Name:producer1
Program Counter:0x4000ac
Stack Pointer address:3e2c33a0
State:Ready

ThreadID:2
Thread Name:thread2
Thread Function Name:consumer1
Program Counter:0x400114
Stack Pointer address:3e2c34a0
State:Ready

---------------------------------
```

Figure 5: All Threads in program

As you can see buffer is 0 at the beginning of the program. Let see thread switching and race conditions.



(a) Half of test result



(b) other half after a few context switch

As you can see from test result buffer start with 0 but finished with -47. Since there is a race condition ,we don't know which thread will run how many times. Now **we have showed the race condition**. Lets look at the with mutex version.

3.1.2 With Mutex

In this version we will use mutex and we will show that there is no race condition.

Main Thread assembly Code

```
.data
msg:   .asciiz "producer1"
msg1:   .asciiz "consumer1"
msg11:   .asciiz "producer"
msg12:   .asciiz "consumer"
msg2:   .asciiz "Buffer in Main: "
msg3:   .asciiz "Buffer in Producer: "
msg9:   .asciiz "Buffer in Consumer: "
msg4:  .asciiz "Waiting to join all threads...\n"
msg5:  .asciiz "All threads are finished!\n"
msg7:  .asciiz " "
msg8:  .asciiz "Waiting for mutex...\n"
msg10:  .asciiz "m1"

newLine:   .asciiz "\n"
.text
.globl producer1
.globl consumer1
.globl producer
.globl consumer
main:

    li $t0,0 #global buffer between threads
    li $t1,10 #global variable
    #create producer thread
    li $v0, 18
    la $a0, msg
    syscall

    #create consumer thread
    li $v0, 18
    la $a0, msg1
    syscall

    #create mutex
    li $v0, 23
    la $a0, msg10 #mutex name
    syscall
```

(c) first half of code

```
joinStart:
    #join threads(pthread_join())
    li $v0, 4
    la $a0, msg4 #print message
    syscall

    li $v0, 4
    la $a0, msg2 #print message
    syscall

    li $v0, 1 #print buffer
    move $a0, $t0
    syscall

    li $v0, 4
    la $a0, newLine #newLine
    syscall

    li $v0, 19  #join syscall
    syscall
    #if result is not zero
    #keep waiting for all threads
    bnez  $v0, joinStart

    #if all threads are joined...
    li $v0, 4
    la $a0, msg5
    syscall
exit:
    li $v0, 20 # Main exit
    syscall
```

(d) other half

Other Threads assembly Code

```
#---------PART2(with Mutex)------------
producer:
    li $t2,0 # int i = 0

    #give waiting mutex message
    #li $v0, 4
    #la $a0, msg8 #print message
    #syscall
producerLoop:
    bgt $t2,$t1,exitProducerThread # if i > 10 then exit
producerMutex:
    li $v0, 21  #mutex lock syscall
    syscall
    #if result is not zero
    #keep waiting for mutex
    bnez  $v0, producerMutex

    #mutex locked
    #add element to buffer
    addi $t0,$t0,1

    bne $t2,2,cont5
    li $v0, 24 # context Switch syscall
    syscall
cont5:
    bne $t2,7,cont6
    li $v0, 24 # context Switch syscall
    syscall
cont6:

    li $v0, 22  #mutex unlock syscall
    syscall

    addi $t2,$t2,1 # ++i
    j producerLoop

exitProducerThread:
    li $v0, 20 # exit thread syscall
    syscall
```

(e) Producer Thread

```
consumer:
    li $t3,0 # int i = 0

    #give waiting mutex message
    #li $v0, 4
    #la $a0, msg8 #print message
    #syscall
consumerLoop:
    bgt $t3,$t1,exitConsumerThread # if i > 10 then exit
consumerMutex:
    li $v0, 21  #mutex lock syscall
    syscall
    #keep waiting for mutex
    #if result is not zero
    bnez  $v0, consumerMutex

    #mutex locked
    #remove element to buffer
    addi $t0,$t0,-1

    bne $t3,4,cont7
    li $v0, 24 # context Switch syscall
    syscall
cont7:
    bne $t3,9,cont8
    li $v0, 24 # context Switch syscall
    syscall
cont8:

    li $v0, 22  #mutex unlock syscall
    syscall

    addi $t3,$t3,1 # ++i
    j consumerLoop

exitConsumerThread:
    li $v0, 20 # exit thread syscall
    syscall
```

(f) Consumer Thread

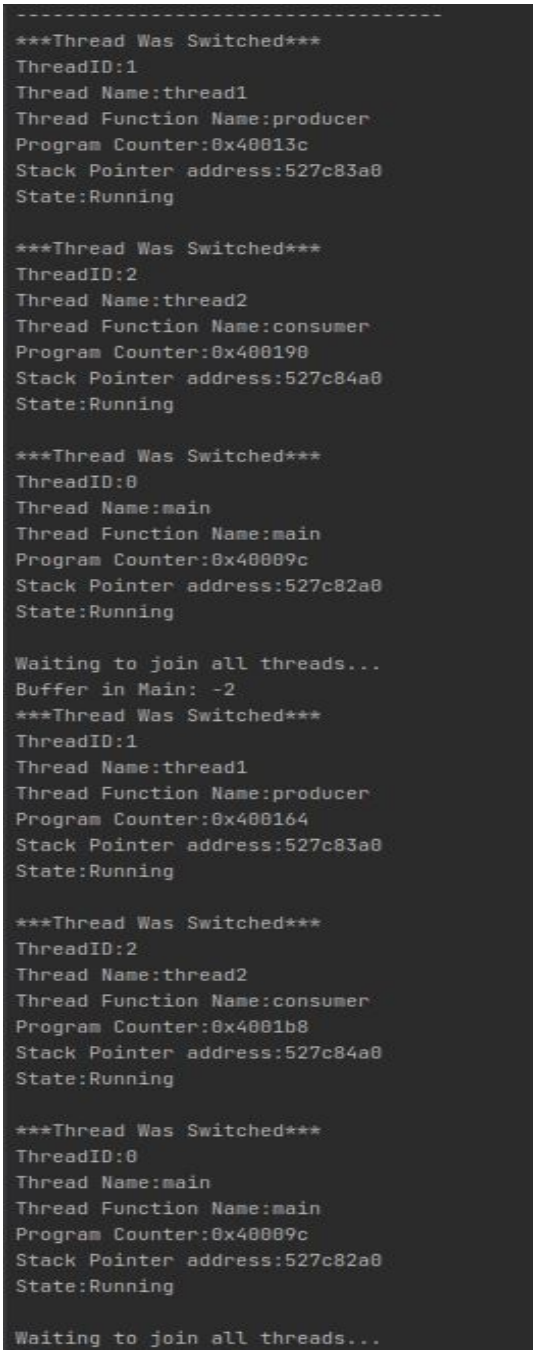
As in one before test, as a buffer we are only incrementing and decrementing a global variable in this test.

Test Results

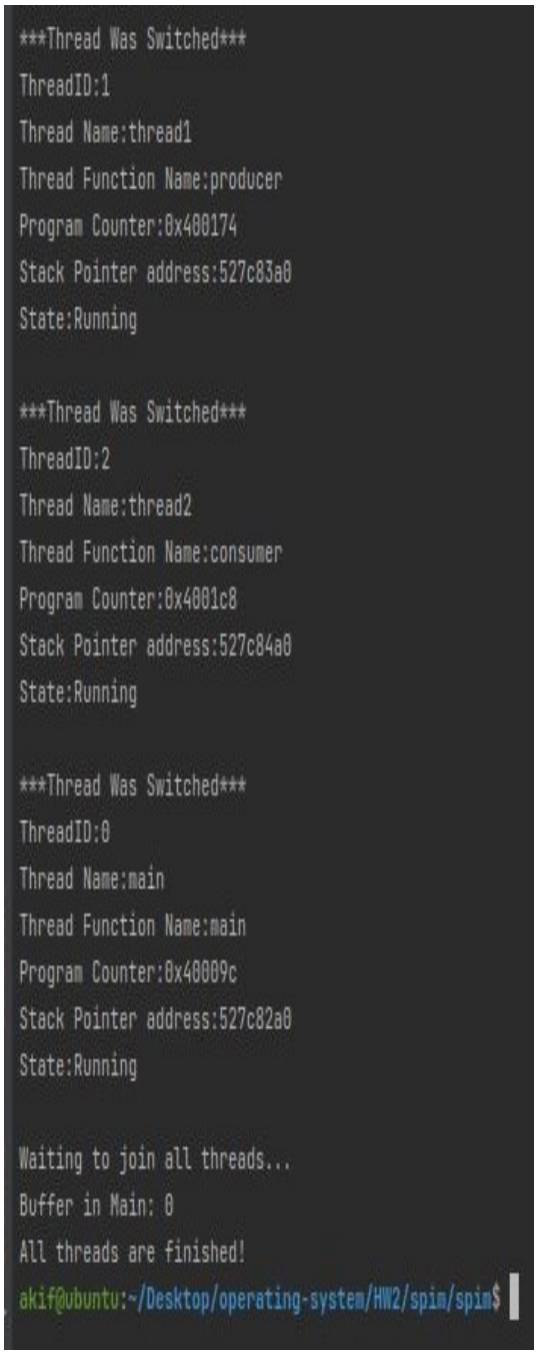
```
akif@ubuntu:~/Desktop/operating-system/HW2/spim/spim$ ./spim -file "SPIMOS_GTU_2.s"
Loaded: /usr/share/spim/exceptions.s
Waiting to join all threads...
Buffer in Main: 0
------------All Threads-----------
ThreadID:0
Thread Name:main
Thread Function Name:main
Program Counter:0x4
Stack Pointer address:0
State:Running

ThreadID:1
Thread Name:thread1
Thread Function Name:producer
Program Counter:0x40013c
Stack Pointer address:527c83a0
State:Ready

ThreadID:2
Thread Name:thread2
Thread Function Name:consumer
Program Counter:0x400190
Stack Pointer address:527c84a0
State:Ready

Mutex Name:m1
Mutex OwnerID:0
Mutex Locked:0
----------------------------------
```

Figure 6: All Threads and Mutexes in program

As you can see buffer is 0 at the beginning of the program. Let see thread switching and result.



(a) Half of test result



(b) other half after a few context switch

As a result since we have used mutex last result is 0 as expected.

3.2 Merge Sort with multi-threaded

In this problem we will implement merge sort with using more than one thread.

Solution Steps

- First, implement multi-threaded merge sort in C++ and make sure it works correctly.
- Then, write multi-threaded merge sort in mips assembly by using C++ code.
- Test assembly file in spim and make sure it works.

3.2.1 Multi-threaded merge sort in C++

In homework pdf file given merge sort code **is not work** with different array and thread size. Then I found and corrected a working multi-threaded merge sort in C++ code. Check following test results.

Simple Test Result

```
akif@ubuntu:~/Desktop$ g++ -o mergeSort last.cpp -lpthread -lrt
akif@ubuntu:~/Desktop$ ./mergeSort
Array Size: 30
Number of Thread: 2
Time taken: 0.001732
Sorting Done Successfully
Sorted array: 0 5 5 11 21 26 27 32 32 36 40 41 41 43 53 56 56 58 61 68 75 78 85 87 88 89 93 98 98 99
akif@ubuntu:~/Desktop$ g++ -o mergeSort last.cpp -lpthread -lrt
akif@ubuntu:~/Desktop$ ./mergeSort
Array Size: 25
Number of Thread: 4
Time taken: 0.001453
Sorting Done Successfully
Sorted array: 2 2 4 24 24 28 31 32 35 35 41 52 57 58 62 63 70 73 74 76 80 81 95 96 97
akif@ubuntu:~/Desktop$ g++ -o mergeSort last.cpp -lpthread -lrt
akif@ubuntu:~/Desktop$ ./mergeSort
Array Size: 28
Number of Thread: 6
Time taken: 0.006433
Sorting Done Successfully
Sorted array: 0 2 6 10 14 15 16 18 21 26 31 37 39 44 49 58 59 63 67 68 79 84 84 84 86 93 94 98
akif@ubuntu:~/Desktop$ g++ -o mergeSort last.cpp -lpthread -lrt
akif@ubuntu:~/Desktop$ ./mergeSort
Array Size: 40
Number of Thread: 10
Time taken: 0.003012
Sorting Done Successfully
Sorted array: 6 8 11 12 13 19 20 23 25 28 31 33 36 37 37 42 43 47 48 51 53 57 58 58 60 62 65 67 73 75 75 76 78 78 80 82 84 87 89 91
akif@ubuntu:~/Desktop$ g++ -o mergeSort last.cpp -lpthread -lrt
akif@ubuntu:~/Desktop$ ./mergeSort
Array Size: 20
Number of Thread: 15
Time taken: 0.010954
Sorting Done Successfully
Sorted array: 9 11 26 28 31 32 46 49 49 54 55 60 73 77 81 82 85 90 93 95
akif@ubuntu:~/Desktop$
```

Figure 7: Multi-threaded merge sort in C++

3.2.2 Multi-threaded merge sort in mips assembly

I have implemented Multi-threaded merge sort in SPIMOS_GTU_1.s file as mips assembly by using my C++ code. Check **SPIMOS_GTU_1.s** file to see code.

3.2.3 Testing in SPIM

Since, merge sort is a recursive algorithm it is difficult to implement recursive functions in assembly, you need to very careful, because of this difficulty my assembly code **didn't work** in spim. I am creating threads **correctly** but I am getting some program counter error because of **recursive functions** in merge sort algorithm.

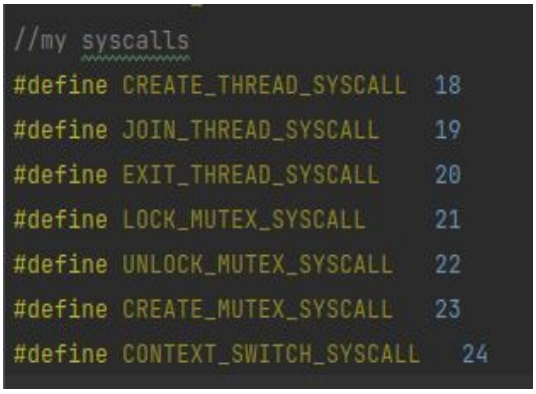
My threads in merge Sort

```
akif@ubuntu:~/Desktop/operating-system/HW2/spim/spim$ ./spim -file "SPIMOS_GTU_1.s"
Loaded: /usr/share/spim/exceptions.s
Enter size of Array(less than 100): 6
Enter number of Thread: 5
Enter array elements one by one
99
54
22
65
1
89
Waiting to join all threads...
------------All Threads-----------
ThreadID:0
Thread Name:main
Thread Function Name:main
Program Counter:0x4
Stack Pointer address:0
State:Running

ThreadID:1
Thread Name:thread1
Thread Function Name:threadFunction
Program Counter:0x4002e8
Stack Pointer address:1ae6b3a0
State:Ready

ThreadID:2
Thread Name:thread2
Thread Function Name:threadFunction
Program Counter:0x4002e8
Stack Pointer address:1ae6b4a0
State:Ready

ThreadID:3
Thread Name:thread3
Thread Function Name:threadFunction
Program Counter:0x4002e8
Stack Pointer address:1ae6b5a0
State:Ready

ThreadID:4
Thread Name:thread4
Thread Function Name:threadFunction
Program Counter:0x4002e8
Stack Pointer address:1ae6b6a0
State:Ready

ThreadID:5
Thread Name:thread5
Thread Function Name:threadFunction
Program Counter:0x4002e8
Stack Pointer address:1ae6b7a0
State:Ready
```

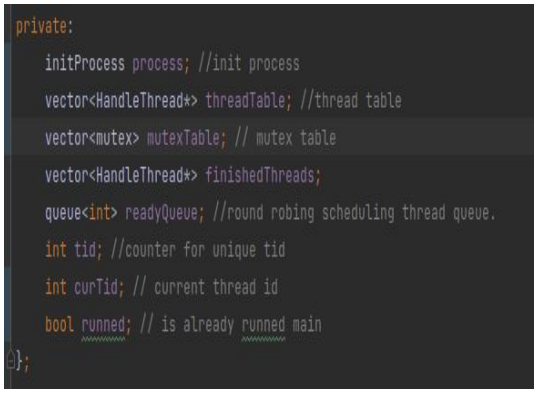
Figure 8: Threads in merge Sort

As you can see, we are creating threads correct way but after that because of recursive algorithms we are getting program counter errors which is not happen in producer consumer problem. Thread function is merge function C++ code.

4 Implementation Details in SPIM



(a) My syscalls in SPIM



(b) My Data in SPIM