

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;

```
1      .data
2      msg:  .ascii "thrFunc"
3      msg3: .ascii "All Threads are finished!\n"
4      msg2: .ascii "Waiting to join all threads...\n"
5      msg1: .ascii "Hello from Thread\n"
6      msg4: .ascii "\n\n"
7      msg5: .ascii "Global Variable: "
8      .text
9      .globl thrFunc
10     main:
11         li $t0,1 #global variable initial value is 1
12
13         li $v0, 18 #create syscall
14         la $a0, msg
15         syscall
16
17         li $v0, 18 #create syscall
18         la $a0, msg
19         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;

```

20  join:
21      li $v0, 4
22      la $a0, msg2
23      syscall
24
25      li $v0, 19 # join syscall
26      syscall
27      bnez $v0, join # if result is not zero wait
28
29      li $v0, 4
30      la $a0, msg3 #threads are finished message
31      syscall

```

Figure 3

2.1.3 Exit Thread Syscall

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

Assembly Test File;

```

45      li $v0, 20 # exit thread syscall
46      syscall

```

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;

```
2  msg: .asciiz "thrFunc"
3  msg3: .asciiz "All Threads are finished!\n"
4  msg2: .asciiz "Waiting to join all threads...\n"
5  msg1: .asciiz "Hello from Thread\n"
6  msg4: .asciiz "\n\n"
7  msg5: .asciiz "Global Variable: "
8  .text
9  .globl thrFunc
10 main:
11     li $t0,1 #global variable initial value is 1
12
13     li $v0, 18 #create syscall
14     la $a0, msg
15     syscall
16
17     li $v0, 18 #create syscall
18     la $a0, msg
19     syscall
20
21 join:
22     li $v0, 4
23     la $a0, msg2
24     syscall
25
26     li $v0, 19 # join syscall
27     syscall
28     bnez $v0, join # if result is not zero wait
29
30     li $v0, 4
31     la $a0, msg3 #threads are finished message
32     syscall
33
34 exitMain:
35     li $v0, 4
36     la $a0, msg5
37     syscall
38
39     li $v0, 1
40     move $a0,$t0
41     syscall
42
43     li $v0, 4
44     la $a0, msg4
45     syscall
46
47     li $v0, 20 # exit thread syscall
48     syscall
```

(a) first half of code

```
47 thrFunc:
48     #thread function for both thread
49     li $v0, 4
50     la $a0, msg1 # give message
51     syscall
52
53     addi $t0,$t0,1 # increment global variable
54
55     li $v0, 4
56     la $a0, msg5
57     syscall
58
59     li $v0, 1 #print gloabal variable
60     move $a0,$t0
61     syscall
62
63     li $v0, 4
64     la $a0, msg4
65     syscall
66
67 exitThr:
68     li $v0, 20 # exit thread syscall
69     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;

```
1  .data
2  msg:  .ascii "producer1"
3  msg1: .ascii "consumer1"
4  msg2: .ascii "Buffer in Main: "
5  msg3: .ascii "Buffer in Producer: "
6  msg9: .ascii "Buffer in Consumer: "
7  msg4: .ascii "Waiting to join all threads...\n"
8  msg5: .ascii "All threads are finished!\n"
9  msg7: .ascii " "
10 msg8: .ascii "Waiting for mutex...\n"
11
12 newLine: .ascii "\n"
13 .text
14 .globl producer1
15 .globl consumer1
16 main:
17
18     li $t0, 0 #global buffer between threads
19     li $t1, 10 #global variable
20
21     #create producer thread
22     li $v0, 18
23     la $a0, msg
24     syscall
25
26     #create consumer thread
27     li $v0, 18
28     la $a0, msg1
29     syscall
30
31 joinStart:
32     #join threads(pthread_join())
33     li $v0, 4
34     la $a0, msg4 #print message
35     syscall
36
37     li $v0, 4
38     la $a0, msg2 #print message
39     syscall
```

(e) first half of code

```
39
40     li $v0, 1 #print buffer
41     move $a0, $t0
42     syscall
43
44     li $v0, 4
45     la $a0, newLine #newLine
46     syscall
47
48     li $v0, 19 #join syscall
49     syscall
50
51     #if result is not zero
52     #keep waiting for all threads
53     bnez $v0, joinStart
54
55     #if all threads are joined...
56     li $v0, 4
57     la $a0, msg5
58     syscall
59
60 exit:
61     li $v0, 20 # Main exit
62     syscall
```

(f) other half

Other Threads assembly Code(no mutex yet!);

```
62 #-----PART1(no mutex)-----
63
64 producer1:
65     li $t2,0 # int i = 0
66     #in random time make context switch
67     #generate 3 random number
68
69 producerLoop1:
70     bgt $t2,$t1,exitProducerThread1 # if i > 10 then exit
71
72     #add element to buffer
73     addi $t0,$t0,1
74     li $v0, 4
75     la $a0, msg3 #print message
76     syscall
77
78     li $v0, 1 #print buffer
79     move $a0, $t0
80     syscall
81
82     li $v0, 4
83     la $a0, newLine #newline
84     syscall
85
86     beq $t2,2,contextSwitch1
87     beq $t2,7,contextSwitch1
88
89     addi $t2,$t2,1 # ++i
90     j producerLoop1
91 contextSwitch1:
92     li $v0, 24 # context Switch syscall
93     syscall
94     j producerLoop1
95
96 exitProducerThread1:
97     li $v0, 20 # exit thread syscall
98     syscall
```

(g) Producer Thread

```
100 consumer1:
101     li $t3,0 # int i = 0
102
103     #give waiting mutex message
104     #li $v0, 4
105     #la $a0, msg8 #print message
106     #syscall
107
108 consumerLoop1:
109     bgt $t3,$t1,exitConsumerThread1 # if i > 10 then exit
110
111     #remove element to buffer
112     addi $t0,$t0,-1
113
114     li $v0, 4
115     la $a0, msg9 #print message
116     syscall
117
118     li $v0, 1 #print buffer
119     move $a0, $t0
120     syscall
121
122     li $v0, 4
123     la $a0, newLine #newline
124     syscall
125
126     beq $t3,4,contextSwitch2
127     beq $t3,9,contextSwitch2
128
129     addi $t3,$t3,1 # ++i
130     j consumerLoop1
131
132 contextSwitch2:
133     li $v0, 24 # context Switch syscall
134     syscall
135     j consumerLoop1
136
137 exitConsumerThread1:
138     li $v0, 20 # exit thread syscall
139     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.

```

-----
***Thread Was Switched***
ThreadID:1
Thread Name:thread1
Thread Function Name:producer1
Program Counter:0x4000ac
Stack Pointer address:3e2c33a0
State:Running

Buffer in Producer: 1
Buffer in Producer: 2
Buffer in Producer: 3
***Thread Was Switched***
ThreadID:2
Thread Name:thread2
Thread Function Name:consumer1
Program Counter:0x400114
Stack Pointer address:3e2c34a0
State:Running

Buffer in Consumer: 2
Buffer in Consumer: 1
Buffer in Consumer: 0
Buffer in Consumer: -1
Buffer in Consumer: -2
***Thread Was Switched***
ThreadID:0
Thread Name:main
Thread Function Name:main
Program Counter:0x40008c
Stack Pointer address:3e2c32a0
State:Running

Waiting to join all threads...
Buffer in Main: -2

```

(a) Half of test result

```

***Thread Was Switched***
ThreadID:2
Thread Name:thread2
Thread Function Name:consumer1
Program Counter:0x40016c
Stack Pointer address:3e2c34a0
State:Running

Buffer in Consumer: -46
***Thread Was Switched***
ThreadID:0
Thread Name:main
Thread Function Name:main
Program Counter:0x40008c
Stack Pointer address:3e2c32a0
State:Running

***Thread Was Switched***
ThreadID:2
Thread Name:thread2
Thread Function Name:consumer1
Program Counter:0x40016c
Stack Pointer address:3e2c34a0
State:Running

Buffer in Consumer: -47
***Thread Was Switched***
ThreadID:0
Thread Name:main
Thread Function Name:main
Program Counter:0x400090
Stack Pointer address:3e2c32a0
State:Running

All threads are finished!
akif@ubuntu:~/Desktop/operating-system/HW2/spim/spim$

```

(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**. Let's 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

```
1  .data
2  msg:  .ascii "producer1"
3  msg1: .ascii "consumer1"
4  msg11: .ascii "producer"
5  msg12: .ascii "consumer"
6  msg2:  .ascii "Buffer in Main: "
7  msg3:  .ascii "Buffer in Producer: "
8  msg9:  .ascii "Buffer in Consumer: "
9  msg4:  .ascii "Waiting to join all threads...\n"
10 msg5:  .ascii "All threads are finished!\n"
11 msg7:  .ascii " "
12 msg8:  .ascii "Waiting for mutex...\n"
13 msg10: .ascii "m1"
14
15 newline: .ascii "\n"
16 .text
17 .globl producer1
18 .globl consumer1
19 .globl producer
20 .globl consumer
21 main:
22
23     li $t0,0 #global buffer between threads
24     li $t1,10 #global variable
25     #create producer thread
26     li $v0, 18
27     la $a0, msg
28     syscall
29
30     #create consumer thread
31     li $v0, 18
32     la $a0, msg1
33     syscall
34
35     #create mutex
36     li $v0, 23
37     la $a0, msg10 #mutex name
38     syscall
```

(c) first half of code

```
39 joinStart:
40     #join threads(pthread_join())
41     li $v0, 4
42     la $a0, msg4 #print message
43     syscall
44
45     li $v0, 4
46     la $a0, msg2 #print message
47     syscall
48
49     li $v0, 1 #print buffer
50     move $a0, $t0
51     syscall
52
53     li $v0, 4
54     la $a0, newline #newline
55     syscall
56
57     li $v0, 19 #join syscall
58     syscall
59     #if result is not zero
60     #keep waiting for all threads
61     bnez $v0, joinStart
62
63     #if all threads are joined...
64     li $v0, 4
65     la $a0, msg5
66     syscall
67 exit:
68     li $v0, 20 # Main exit
69     syscall
70
```

(d) other half

Other Threads assembly Code

```

132 #-----PART2(with Mutex)-----
133 producer:
134     li $t2,0 # int i = 0
135
136     #give waiting mutex message
137     #li $v0, 4
138     #la $a0, msg8 #print message
139     #syscall
140 producerLoop:
141     bgt $t2,$t1,exitProducerThread # if i > 10 then exit
142 producerMutex:
143     li $v0, 21 #mutex lock syscall
144     syscall
145     #if result is not zero
146     #keep waiting for mutex
147     bnez $v0, producerMutex
148
149     #mutex locked
150     #add element to buffer
151     addi $t0,$t0,1
152
153     bne $t2,2,cont5
154     li $v0, 24 # context Switch syscall
155     syscall
156 cont5:
157     bne $t2,7,cont6
158     li $v0, 24 # context Switch syscall
159     syscall
160 cont6:
161
162     li $v0, 22 #mutex unlock syscall
163     syscall
164
165     addi $t2,$t2,1 # ++i
166     j producerLoop
167
168 exitProducerThread:
169     li $v0, 20 # exit thread syscall
170     syscall

```

(e) Producer Thread

```

173 consumer:
174     li $t3,0 # int i = 0
175
176     #give waiting mutex message
177     #li $v0, 4
178     #la $a0, msg8 #print message
179     #syscall
180 consumerLoop:
181     bgt $t3,$t1,exitConsumerThread # if i > 10 then exit
182 consumerMutex:
183     li $v0, 21 #mutex lock syscall
184     syscall
185     #keep waiting for mutex
186     #if result is not zero
187     bnez $v0, consumerMutex
188
189     #mutex locked
190     #remove element to buffer
191     addi $t0,$t0,-1
192
193     bne $t3,4,cont7
194     li $v0, 24 # context Switch syscall
195     syscall
196 cont7:
197     bne $t3,9,cont8
198     li $v0, 24 # context Switch syscall
199     syscall
200 cont8:
201
202     li $v0, 22 #mutex unlock syscall
203     syscall
204
205     addi $t3,$t3,1 # ++i
206     j consumerLoop
207
208 exitConsumerThread:
209     li $v0, 20 # exit thread syscall
210     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/spin/spin$ ./spin -file "SPIMOS_GTU_2.s"
Loaded: /usr/share/spin/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.

```

-----
***Thread Was Switched***
ThreadID:1
Thread Name:thread1
Thread Function Name:producer
Program Counter:0x40013c
Stack Pointer address:527c83a0
State:Running

***Thread Was Switched***
ThreadID:2
Thread Name:thread2
Thread Function Name:consumer
Program Counter:0x400190
Stack Pointer address:527c84a0
State:Running

***Thread Was Switched***
ThreadID:0
Thread Name:main
Thread Function Name:main
Program Counter:0x40009c
Stack Pointer address:527c82a0
State:Running

Waiting to join all threads...
Buffer in Main: -2
***Thread Was Switched***
ThreadID:1
Thread Name:thread1
Thread Function Name:producer
Program Counter:0x400164
Stack Pointer address:527c83a0
State:Running

***Thread Was Switched***
ThreadID:2
Thread Name:thread2
Thread Function Name:consumer
Program Counter:0x4001b8
Stack Pointer address:527c84a0
State:Running

***Thread Was Switched***
ThreadID:0
Thread Name:main
Thread Function Name:main
Program Counter:0x40009c
Stack Pointer address:527c82a0
State:Running

Waiting to join all threads...

```

(a) Half of test result

```

***Thread Was Switched***
ThreadID:1
Thread Name:thread1
Thread Function Name:producer
Program Counter:0x400174
Stack Pointer address:527c83a0
State:Running

***Thread Was Switched***
ThreadID:2
Thread Name:thread2
Thread Function Name:consumer
Program Counter:0x4001c8
Stack Pointer address:527c84a0
State:Running

***Thread Was Switched***
ThreadID:0
Thread Name:main
Thread Function Name:main
Program Counter:0x40009c
Stack Pointer address:527c82a0
State:Running

Waiting to join all threads...
Buffer in Main: 0
All threads are finished!
akif@ubuntu:~/Desktop/operating-system/HW2/spin/spin$

```

(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 be 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

```
//my syscalls
#define CREATE_THREAD_SYSCALL 18
#define JOIN_THREAD_SYSCALL 19
#define EXIT_THREAD_SYSCALL 20
#define LOCK_MUTEX_SYSCALL 21
#define UNLOCK_MUTEX_SYSCALL 22
#define CREATE_MUTEX_SYSCALL 23
#define CONTEXT_SWITCH_SYSCALL 24
```

(a) My syscalls in SPIM

```
private:
    initProcess process; //init process
    vector<HandleThread*> threadTable; //thread table
    vector<mutex> mutexTable; // mutex table
    vector<HandleThread*> finishedThreads;
    queue<int> readyQueue; //round robing scheduling thread queue.
    int tid; //counter for unique tid
    int curTid; // current thread id
    bool runned; // is already runned main
};
```

(b) My Data in SPIM