CSE312 - Operating Systems - HW#2

Fatih Kaan Salgır - 171044009

Design Explanation

- In some cases `switch_thread()` needs to be triggered regardless of the timer interrupt. (Like join, mutex lock or thread exit system calls.) In these cases the program counter is incremented 2 times. To avoid this when `switch_thread` manually triggered, the program counter decrmeneted by 4 (`BYTES_PER_WORD`).

Thread Table

Thread tabel are going to keep track of:

- Thread ID
- Program Counter
- Registers
- Stack
- State

of each thread.

These data corresponds the struct defined in `syscall.h` in case of `spim`:

```
enum thread_state { READY=0, RUNNING, BLOCKED, TERMINATED };

typedef struct thread {
  int thread_id;
  reg_word R[R_LENGTH];
  reg_word HI, LO;
  mem_addr PC, nPC;
  double *FPR;
  float *FGR;
  int *FWR;
  reg_word CCR[4][32], CPR[4][32];
  mem_word *stack_seg;
  short *stack_seg_h;
  BYTE_TYPE *stack_seg_b;
  mem_addr stack_bot;
  thread_state state;
}thread;
```

- In spim source code it indicates that `stack_seg` represents the stack segment and boundaries. `stack_seg_h` and `stack_seg_b` points to same vector as `stack_seg`. This data will be backed & restored.

The process table implemented as global `std::vector`. The first element of the vector is always going to be the main thread.

```
static vector<struct thread *> thread_table;
```

Thread ID's must be uniqe threfore global static `next_id` variable initialized with 1 (0 belongs to main thread). And incremented every time when a thread created.

`current_thread` is the pointer that points to currently running thread. On switch threads, this pointer is going to point the thread should run next.

System Calls

Thread Create

```
    ...
    la $a0, arg0        # args goes to $a0, $a1, $a2
    li $v0, 18          # thread create system call
    la $s0, thread1     # s0: label of the thread
    syscall
    move $1, $v0        # save thread id
    ....
thread1:
    ...
```

This will trigger `new_thread()` function in `syscall.cpp`. This function are going to initialize thread table, only on first call. Then it will set the *program counter* as the contents of the register `$s0`. In case of the code shown above, program counter is going to be set the address of `thread1` label.

All other variables which needed to be allocated dynamically (stack etc.) allocated, stack pointers set accordingly, and the thread added to thread table.

After the thread create system call, register `$v0` is going to gets the value of thread id.

Thread Join

```
    ...
    li $a0, 1       # a0: thread id
    li $v0, 19      # join the thread with id 1
    syscall
```

The state of the currently running thread become blocked, until the thread with specified id become `TERMINATED`. When a thread is terminated with thread exit system call, checks if any thread is waiting for itself. If it is the case changes it state to `READY`.

Thread Exit

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

Exits from the currently running thread, `free`'s its resources, unlock its mutexes if there is any. However it doesn't remove from the thread table until the program has finished.

Mutex Init

```
.data
mutex1: .space 1
...
    ...
    li $v0, 21          # mutex init system call
    la $a0, mutex1
    syscall
```

Initializes the mutex on the given address. A mutex table stored in the `syscall.cpp` which contains the address of the mutex, state of the mutex (either locked or unlocked), owner id and waiting threads.

```
static vector<struct mutex> mutex_table;

enum mutex_state { LOCKED, UNLOCKED };

typedef struct mutex {
  reg_word addr;
        mutex_state state;
        int owner_id;
        thread* waiting_threads[64];
}mutex;
```

Mutex Lock

```
.data
mutex1: .space 1
...
    ...
    li $v0 22          # mutex lock system call
    la $a0, mutex1     # lock mutex1
    syscall
    ...
```

If the mutex can be obtained (if the state of the mutex is unlocked) changes the state of the mutex to locked. `owner_id` set to the id of the thread that locks the mutex.

If a thread makes a mutex lock system call when the mutex is already locked, then the thread added to the `waiting_threads` queue. And the thread will be blocked. To eliminate race condition, *program counter* will be decremented by 4 so that when the mutex unlocked, any thread can obtain the mutex.

Mutex Unlock

```
.data
mutex1: .space 1
...
    ...
    li $v0 23          # mutex unlock system call
    la $a0, mutex1     # unlock mutex1
    syscall
    ...
```

Unlocks the mutex, and all other threads waiting on this thread becomes `READY`.

Merge Sort

In main label of assembly code threads are created on different parts of array. Threads are responsible of sorting sub-arrays. After waiting for all threads to exit (with join system call), array is merged and printed on the screen.

The code is tested with different number of threads and different array sizes;

- 2 threads and array size with 16
- 4 threads and array size with 16
- 2 threads and array size with 32
- 4 threads and array size with 32

To change the array size or number of threads, parts of the array must be given manually as a parameter before thread create system call.

An example output of running 2 threads with array size 32:

```
switching: 0->1
| thread id |      PC | stack pointer | thread state |
|-----------|---------|---------------|--------------|
|          0|  4194412|       7fffee7c|       BLOCKED|
|          1|   4000a0|       7fffeffc|       RUNNING|
|          2|   4000a0|       7fffeffc|         READY|
switching: 1->2
| thread id |      PC | stack pointer | thread state |
|-----------|---------|---------------|--------------|
|          0|   40006c|       7fffee7c|       BLOCKED|
|          1|   400190|       7fffefbc|         READY|
|          2|   4000a0|       7fffeffc|       RUNNING|
switching: 2->1
| thread id |      PC | stack pointer | thread state |
|-----------|---------|---------------|--------------|
|          0|   40006c|       7fffee7c|       BLOCKED|
|          1|   400190|       7fffefbc|       RUNNING|
|          2|   400188|       7fffefcc|         READY|
switching: 1->2
| thread id |      PC | stack pointer | thread state |
|-----------|---------|---------------|--------------|
|          0|   40006c|       7fffee7c|         READY|
|          1|   4000a8|       7fffeffc|    TERMINATED|
|          2|   400188|       7fffefcc|       RUNNING|
switching: 2->0
| thread id |      PC | stack pointer | thread state |
|-----------|---------|---------------|--------------|
|          0|   40006c|       7fffee7c|       RUNNING|
|          1|   4000a8|       7fffeffc|    TERMINATED|
|          2|   400188|       7fffefcc|         READY|
switching: 0->2
| thread id |      PC | stack pointer | thread state |
|-----------|---------|---------------|--------------|
|          0|   400078|       7fffee7c|       BLOCKED|
|          1|   4000a8|       7fffeffc|    TERMINATED|
|          2|   400188|       7fffefcc|       RUNNING|
switching: 2->0
| thread id |      PC | stack pointer | thread state |
|-----------|---------|---------------|--------------|

|          0|   400078|       7fffee7c|       RUNNING|
|          1|   4000a8|       7fffeffc|    TERMINATED|
|          2|   4000a8|       7fffeffc|    TERMINATED|
switching: 0->0
| thread id |      PC | stack pointer | thread state |
|-----------|---------|---------------|--------------|
|          0|   400078|       7fffee7c|       RUNNING|
|          1|   4000a8|       7fffeffc|    TERMINATED|
|          2|   4000a8|       7fffeffc|    TERMINATED|
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
switching: 0->0
| thread id |      PC | stack pointer | thread state |
|-----------|---------|---------------|--------------|
|          0|   400264|       7fffee7c|       RUNNING|
|          1|   4000a8|       7fffeffc|    TERMINATED|
|          2|   4000a8|       7fffeffc|    TERMINATED|
18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
```

Output of running 4 threads with array size 32:

```
switching: 0->1
| thread id |      PC | stack pointer | thread state |
|-----------|---------|---------------|--------------|
|          0|  4194468|       7fffee7c|       BLOCKED|
|          1|   400110|       7fffeffc|       RUNNING|
|          2|   400110|       7fffeffc|         READY|
|          3|   400110|       7fffeffc|         READY|
|          4|   400110|       7fffeffc|         READY|
switching: 1->2
| thread id |      PC | stack pointer | thread state |
|-----------|---------|---------------|--------------|
|          0|   4000a4|       7fffee7c|         READY|
|          1|   400118|       7fffeffc|    TERMINATED|
|          2|   400110|       7fffeffc|       RUNNING|
|          3|   400110|       7fffeffc|         READY|
|          4|   400110|       7fffeffc|         READY|
switching: 2->3
| thread id |      PC | stack pointer | thread state |
|-----------|---------|---------------|--------------|
|          0|   4000a4|       7fffee7c|         READY|
|          1|   400118|       7fffeffc|    TERMINATED|
|          2|   400118|       7fffeffc|    TERMINATED|
|          3|   400110|       7fffeffc|       RUNNING|
|          4|   400110|       7fffeffc|         READY|
switching: 3->4
| thread id |      PC | stack pointer | thread state |
|-----------|---------|---------------|--------------|
|          0|   4000a4|       7fffee7c|         READY|
|          1|   400118|       7fffeffc|    TERMINATED|
|          2|   400118|       7fffeffc|    TERMINATED|
|          3|   400264|       7fffefcc|         READY|
|          4|   400110|       7fffeffc|       RUNNING|
switching: 4->0
| thread id |      PC | stack pointer | thread state |
|-----------|---------|---------------|--------------|
|          0|   4000a4|       7fffee7c|       RUNNING|
|          1|   400118|       7fffeffc|    TERMINATED|
|          2|   400118|       7fffeffc|    TERMINATED|
|          3|   400264|       7fffefcc|         READY|
|          4|   400118|       7fffeffc|    TERMINATED|
switching: 0->3
| thread id |      PC | stack pointer | thread state |
|-----------|---------|---------------|--------------|
|          0|   4000bc|       7fffee7c|       BLOCKED|
|          1|   400118|       7fffeffc|    TERMINATED|
|          2|   400118|       7fffeffc|    TERMINATED|
|          3|   400264|       7fffefcc|       RUNNING|
|          4|   400118|       7fffeffc|    TERMINATED|
switching: 3->0
| thread id |      PC | stack pointer | thread state |
|-----------|---------|---------------|--------------|
|          0|   4000bc|       7fffee7c|       RUNNING|
|          1|   400118|       7fffeffc|    TERMINATED|
|          2|   400118|       7fffeffc|    TERMINATED|
|          3|   400118|       7fffeffc|    TERMINATED|
|          4|   400118|       7fffeffc|    TERMINATED|
switching: 0->0
| thread id |      PC | stack pointer | thread state |
|-----------|---------|---------------|--------------|
|          0|   400248|       7fffee7c|       RUNNING|
|          1|   400118|       7fffeffc|    TERMINATED|
|          2|   400118|       7fffeffc|    TERMINATED|
|          3|   400118|       7fffeffc|    TERMINATED|
|          4|   400118|       7fffeffc|    TERMINATED|
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
```

Output of running 2 threads with array size 16:

```
switching: 0->1
| thread id |      PC | stack pointer | thread state |
|-----------|---------|---------------|--------------|
|          0|  4194412|       7fffee9c|       BLOCKED|
|          1|   4000a0|       7fffeffc|       RUNNING|
|          2|   4000a0|       7fffeffc|         READY|
switching: 1->2
| thread id |      PC | stack pointer | thread state |
|-----------|---------|---------------|--------------|
|          0|   40006c|       7fffee9c|         READY|
|          1|   4000a8|       7fffeffc|    TERMINATED|
|          2|   4000a0|       7fffeffc|       RUNNING|
switching: 2->0
| thread id |      PC | stack pointer | thread state |
|-----------|---------|---------------|--------------|
|          0|   40006c|       7fffee9c|       RUNNING|
|          1|   4000a8|       7fffeffc|    TERMINATED|
|          2|   400114|       7fffefec|         READY|
switching: 0->2
| thread id |      PC | stack pointer | thread state |
|-----------|---------|---------------|--------------|
|          0|   400078|       7fffee9c|       BLOCKED|
|          1|   4000a8|       7fffeffc|    TERMINATED|
|          2|   400114|       7fffefec|       RUNNING|
switching: 2->0
| thread id |      PC | stack pointer | thread state |
|-----------|---------|---------------|--------------|
|          0|   400078|       7fffee9c|       RUNNING|
|          1|   4000a8|       7fffeffc|    TERMINATED|
|          2|   4000a8|       7fffeffc|    TERMINATED|
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
```

Producer & Consumer

The shared variable will be a `word` which its value starts with 0. Producer will increment it 1000 times and consumer will decrement it 1000 times. Queue size will be 50, the producer will wait when the queue is full, similarly the consumer will wait when the queue is empty. Waiting is implemented as busy waiting in assembly code.

```
producer:
    li $s1, 0
produce_loop:
    jal wait_empty
    jal mutex_lock
    jal produce     # critical section
    jal print_newline
    jal mutex_unlock
    addi $s1, $s1, 1
    blt $s1, 1000, produce_loop
    li $v0, 20       # thread exit
    syscall
```

```
consumer:
    li $s1, 0
consume_loop:
    jal wait_full
    jal mutex_lock
    jal consume     # critical section
    jal print_newline
    jal mutex_unlock
    addi $s1, $s1, 1
    blt $s1, 1000, consume_loop
    li $v0, 20       # thread exit
    syscall
```

The only difference between two version is waiting and mutex calls are commented in the first case (Highlighted in the code above).

Race conditions without mutexes

Because of the race condition, the shared variable reperesents the size of the queue, most probably ends up a value different than 0.

```
...(truncated)
consumer -236
consumer -237
consumer -238
| thread id |      PC | stack pointer | thread state |
|-----------|---------|---------------|--------------|
|          0|   400064|       7fffee7c|       RUNNING|
|          1|   400104|       7fffeffc|    TERMINATED|
|          2|   400138|       7fffeffc|    TERMINATED|
total produced: 1000
total consumed: 1000
```

No race conditions with mutexes

The program ends with consumer thread consuming the last element which is 0.

```
| thread id |      PC | stack pointer | thread state |
|-----------|---------|---------------|--------------|
|          0|  4194404|       7fffee7c|       BLOCKED|
|          1|   4000d4|       7fffeffc|       RUNNING|
|          2|   400114|       7fffeffc|         READY|
producer 1
producer 2
producer 3
...(truncated)
consumer 2
consumer 1
consumer 0
| thread id |      PC | stack pointer | thread state |
|-----------|---------|---------------|--------------|
|          0|   400064|       7fffee7c|       RUNNING|
|          1|   400110|       7fffeffc|    TERMINATED|
|          2|   400150|       7fffeffc|    TERMINATED|
total produced: 1000
total consumed: 1000
```

In the first case, `size` is the shared variable but it is not protected, therefore it does not end up with 0. In the second case it does. Total produced and total consumed didn't change, because threads are using the different variables, not shared variables.