Akshala Bhatnagar
2018012

OS Optional Assignment-1

1. Paste your struct thread structure.

```
struct thread {
    void *esp;
    void *start_address_stack;
    struct thread *next;
    struct thread *prev;
};
```

2. Paste any new global variables or struct that you have added to the existing code.

```
struct thread *garbage_thread = NULL;

struct lock_node{
    struct lock *lock_val;
    struct lock_node *next;
};

struct lock_node *lock_list = NULL;
```

3. Paste your code corresponding to sleep.

```
void add_lock_to_lock_list(struct lock_node *new_lock){
    // printf("Entering add_lock_to_lock_list\n");
    struct lock_node *lock_ptr = lock_list;
    struct lock_node *prev_lock_ptr = NULL;
    int flag = 1;
    while(lock_ptr != NULL){
        if(lock_ptr->lock_val == new_lock->lock_val){
            flag = 0;
            break;
        }
```

```
            prev_lock_ptr = lock_ptr;
            lock_ptr = lock_ptr->next;
        }
        if(flag){
            if (prev_lock_ptr!=NULL) {
                prev_lock_ptr->next = new_lock;
            }
            else {
                lock_list = new_lock;
            }
        }
    }

    void sleep(struct lock *lock)
    {
        struct lock_node *new_lock = malloc(sizeof(struct
    lock_node));
        new_lock->lock_val = lock;
        add_lock_to_lock_list(new_lock);

        if((struct thread*)(lock->wait_list) == NULL){
            // printf("Push: Adding to NULL wait list\n");
            lock->wait_list = (void*)cur_thread;
            cur_thread->next = (struct thread*)lock->wait_list;
            cur_thread->prev = (struct thread*)lock->wait_list;
        }
        else{
            struct thread *ptr = (struct thread*)(lock->wait_list);
            while(ptr->next != (struct thread*)lock->wait_list){
                // printf("Push: Iterating wait list\n");
                ptr = ptr->next;
            }
            ptr->next = cur_thread;
            cur_thread->prev = ptr;
            ((struct thread*)(lock->wait_list))->prev = cur_thread;
            cur_thread->next = (struct thread*)(lock->wait_list);
            // printf("Push done\n");
        }
        schedule();
    }
```

4. Paste your code corresponding to wakeup.

```
    void wakeup(struct lock *lock)
    {
        // printf("wakeup\n");
        struct thread *returned_thread;
        if((struct thread*)lock->wait_list == NULL){
            // printf("Popping: NULL wait list\n");
            returned_thread = NULL;
        }
        else if(((struct thread*)(lock->wait_list))->next == (struct
    thread*)(lock->wait_list)){
            // printf("Popping: wait list has only 1 element\n");
            returned_thread = (struct thread*)(lock->wait_list);
            lock->wait_list = NULL;
            push_back(returned_thread);
        }
        else{
            returned_thread = (struct thread*)(lock->wait_list);
            struct thread *wait_list_prev = ((struct
    thread*)(lock->wait_list))->prev;
            struct thread *wait_list_next = ((struct
    thread*)(lock->wait_list))->next;
            // printf("Popping: From head of wait list\n");
            lock->wait_list = (void*)wait_list_next;
            ((struct thread*)(lock->wait_list))->prev = wait_list_prev;
            wait_list_prev->next = (struct thread*)(lock->wait_list);
            push_back(returned_thread);
        }
    }
```

5. Paste your code corresponding to the foo routine in race1.c.

```
void foo(void *ptr)
{
    struct lock *l = (struct lock*)ptr;
    int val;

    acquire();
    val = counter;
    val++;
    counter = val;
    thread_yield();
    release();

    thread_exit();
}
```

6. Dump the output of ‘‘make test2’’.

```
/usr/bin/time -v ./leak 1024000 2>&1 |egrep "kbytes|counter"
main thread exiting : counter:1024000
    Average shared text size (kbytes): 0
    Average unshared data size (kbytes): 0
    Average stack size (kbytes): 0
    Average total size (kbytes): 0
    Maximum resident set size (kbytes): 5204
    Average resident set size (kbytes): 0
```

Akshala Bhatnagar
2018012

7. Does running race2 cause deadlock in your submission?

    Yes, running race2 causes a deadlock in my submission. This is because the lock l is being acquired in foo but it is not released. This causes the program to run without terminating. Bar keeps waiting for the lock but the lock is never released by foo.

8. Does your strategy for eliminating memory leak is different from what you suggested in the assignment-2 design documentation. If yes, please highlight the changes.

    Yes, it is a little different. In the design documentation of assignment-2 I suggested that we make a garbage_list.
    There is no need to create a garbage list as there would only be one element in that list. Thus I just created a variable called garbage_thread. Also the structure for thread has been changed to incorporate the start address of the stack. When thread_exit is called we check if the garbage_thread is null. If it is not then we free the stack of this garbage thread, whose pointer is stored in garbage_thread->start_address_stack and we then free the garbage thread. Afterwards the garbage_thread is set to the cur_thread.

```
void thread_exit()
{
   struct thread *exit_thread = garbage_thread;
   if (garbage_thread != NULL) {
        free(exit_thread->start_address_stack);
        free(exit_thread);
   }
   garbage_thread = cur_thread;
   schedule();
}
```

Akshala Bhatnagar
2018012