Design Documentation

Anunay Yadav
2018021

1. thread structure ->

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

2. global variables and structures ->

```
struct lock_data{
    struct lock* lock;
    struct lock_data* next;
}; // for list

struct thread *ready_list = NULL;     // ready list
struct thread *cur_thread = NULL;     // current thread
struct lock_data * lock_list = NULL; // list of locks
struct thread* dead_list = NULL; // list of dead threads
```

3. sleep code : ->
pushes currrent thread to waitlist of the lock and than schedules new.

```
    struct thread* cur = (struct thread*) lock -> wait_list;
    push_unique(lock);
    struct thread *temp = cur;
    if(temp == NULL){
        lock -> wait_list = cur_thread;
        cur_thread -> prev = (struct thread*) lock -> wait_list;
        cur_thread -> next = (struct thread*) lock -> wait_list;
        return;
    }
    struct thread* last = temp -> prev;
    last -> next = cur_thread;
    cur_thread -> prev = last;
    cur_thread -> next = (struct thread*) lock -> wait_list;
    ((struct thread*) lock -> wait_list )-> prev = cur_thread;
    schedule();
```

4. wakeup code : ->

pops the first thread from the lock waitlist and pushes it back to the ready list.

```
struct thread* cur = (struct thread*) lock -> wait_list;
if(cur != NULL){
    struct thread* temp = (struct thread*) lock -> wait_list;
    struct thread* ret = temp;
    if(temp != temp -> next){
        temp -> prev -> next = temp -> next;
        temp -> next -> prev = temp -> prev;
        lock -> wait_list = temp -> next;
    }
    else
        lock -> wait_list = NULL;
    push_back(ret);
}
```

5.foo routine ->
acquires is used only in critical section which ensures that counter is not changed after getting rescheduled to another thread.

```
struct lock *l = (struct lock*)ptr;
int val;
acquire(l);
val = counter;
thread_yield();
val++;
counter = val;
release(l);
thread_exit();
```

6.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): 5208
    Average resident set size (kbytes): 0
```

7. Does race2 cause deadlock :
YES as foo routine of race2 acquires the lock but never releases the lock hence never lets anone scheduled and causes a deadlock.

8. stratergy of memory leak:

Same stratergy but with more lists and structures to help in implementation