Design Documentation

Anunay Yadav
2018021

1 : push_back(t):

pushes thread at the end of the ready list while maintaining a circular doubly linked list.

Code :

```
static void push_back(struct thread *t)
{
    struct thread *temp = ready_list;
    if(temp == NULL){
        ready_list = t;
        t -> prev = ready_list;
        t -> next = ready_list;
        return;
    }
    struct thread* last = temp -> prev;
    last -> next = t;
    t -> prev = last;
    t -> next = ready_list;
    ready_list -> prev = t;
}
```

2 : pop_front()

removes the first thread from the ready list while maintaining circular doubly linked list and returns to caller.

Code :

```
static struct thread *pop_front()
{
    if(ready_list == NULL)
        return NULL;
    struct thread *ret = ready_list;
    if(ready_list -> next == ready_list)
    {
        ready_list = NULL;
        return ret;
    }
    struct thread *temp = ready_list -> prev;
    temp -> next = ready_list -> next;
    ready_list -> next -> prev = temp;
    ready_list = ready_list -> next;
```

```
    return ret;
}
```

3 : create_thread(func,param)
    creates new thread and allocates stack and pushes elements in stack to correctly allign it with context_switch.

code :

```
void create_thread(func_t func, void *param)
{
    struct thread* new_thread = (struct thread*) malloc(sizeof(struct thread));
    unsigned *stack = (unsigned*) malloc(SIZE);
    stack += 1024;
    if(param != NULL){
        stack--;
        *stack = ((unsigned) param);
    }
    stack--;
    *stack = 0;
    stack--;
    *stack = ((unsigned) func);
    stack -= 4;
    new_thread -> esp = stack;
    push_back(new_thread);
}
```

4 : make test
output :

```
./app 1024
starting main thread: num_threads: 1024
main thread exiting: counter:30768239300147200
./app 1024 1
starting main thread: num_threads: 1024
bar1: (nil)
bar2: (nil)
bar1: 0x1
main thread exiting: counter:0
```

5 : stratergy to free stack and thread
case 1 : assuming we can change context switch
- > we will implement a list which contains the starting address of stack and thread address and a boolean which tells us wether the current thread is killed or not.

- > if it is killed we will free its stack and thread after assigning stack of next thread in context switch which ensures that prev stack is killed only, above information is maintaing in the list mentioned above.
- > as ready list and threads are global. we can find the address of each and every thread and starting address of stack. And hence free it in context_switch.
- > hence at the end only main thread remains.

case 2 : assuming we cannot change context switch
- > first implement a linked list storing the thread addresses and starting address of thier stack.
- > implement a dead_list which contains the threads which are dead and are need to be freed.
- > whenever a thread exits in thread_exit. It first frees all the threads and there stacks in dead_list with the help of linked list storing address of stack and thread, after cleaning it adds itself to the dead_list.