LAB 2 XV6 Threads

Part 1) Clone() system call

As done in Lab 1 to make a system call in XV-6, we have to make changes in
1) usys.S : An interface for the system call is defined here so that the user can use it.
2) syscall.h : System call number is added here.
3) syscall.c : Function is defined here and function pointer to the syscall function is also added here.
4) defs.h : The figure shows the changes made in the defs.h file.
5) user.h : The function called by the user is defined here.

Apart from this I have made changes in proc.c and sysproc.c.

6) proc.c :- The clone system call create threads, it is done by editing the fork system call.

Allowing threads to use the same page table.

```
//CS202 same page table for the thread
np->pgdir = currproc->pgdir;
```

Allocating stack for the threads which consists of the parent stack on the top.

Other things are same as the fork system call.

```
//allocating stack for the thread
uint parent_size = *(uint *)currproc->tf->ebp - currproc->tf->esp;
uint base = *(uint *)currproc->tf->ebp - currproc->tf->ebp;

np->tf->esp = (uint)stack + size - parent_size;
np->tf->ebp = (uint)stack + size - base;

// copy the parents stack memory
memmove((void*) np->tf->esp, (const void *) currproc->tf->esp, parent_size);
```

Change in wait system call so that threads do not clear the page table and thus preventing segmentation fault. Only the creator process can clear the page table by this change.

```
// CS 202
if(p->pgdir != curproc->pgdir){
        freevm(p->pgdir);
}
```

Part 2 Thread library

thread_create :- The thread_create creates the thread using clone if clone is successful it will return 0 to the thread and the thread will start the routine while the caller will exit returning

the tid.(in thread.c) The memory for the thread is allocated using malloc which is of two page size.

```
int thread_create(void *(*start_routine)(void*), void *arg)
{
        int size = PGSIZE * 2;
        void *stack = malloc(PGSIZE * 2);
        int tid;

        tid = clone(stack, size);

        if(tid == 0)
        {

                (*start_routine)(arg);
                exit();
        }

        return tid;
}
```

Spin lock Implementation :-

thread.h :- The lock structure which is lock_t has one member named flag to indicate locked or not locked.

```
//lock struct
typedef struct lock_t lock_t;
struct lock_t{
        unsigned int flag;
};
```

thread.c:-

The lock_init function is shown below, used to initialize the lock.

```
int lock_init(lock_t *lock){
        lock->flag = 0;
        return 0;
}
```

lock_acquire and lock release is shown below, lock acquire test and sets the flag. It uses xchg for the purpose, if old value is 1 it will wait else set the flag to 0 and move on. lock_release sets the flag to 0 simply.

```
void lock_acquire(lock_t *lock){
        while(xchg(&lock->flag, 1) != 0)
                ;
}

void lock_release(lock_t *lock){
        xchg(&lock->flag, 0);
}
```

Part 3) Testing using frisbee.c

Main function:- It will create threads and the routine the threads will followed by the thread will be pass function which is basically simulating the frisbee scenario, where the threads will try to acquire the lock using lock_acquire and will check if its turn to pass the token. If the passes have gone past the max pass number than it will exit the function returning 0.

After the threads have finished in the main function the there will be waiting for each thread to finish.

Main function :-

```
lock_init(slock);
int i = 0;

for(i = 0 ; i < total_thread; i++){

        thread_create(pass,(void *)i);

}

for(i=0; i<total_thread; i++){
  wait();
}
```

Pass function which is the routine for the thread :-

```
void* pass(void *arg){

        int id;
        id = (int)arg;

        for(;;){

                lock_acquire(slock);

                if(id == turn && cpass < maxp ){
                        cpass = cpass + 1;
                        turn =  id + 1;

                        if(turn == total_thread){
                                turn = 0;
                        }

                        printf(1,"Pass No:%d Thread %d passes the token to Thread %d\n",cpass,id,turn);
                }
                else if(cpass >= maxp)
                {
                //      printf(2,"wrong acquire by %d\n",id);
                        lock_release(slock);
                        return 0;
                }
                lock_release(slock);
        }
}
```

In the make file thread_lib have been added as shown below.

```
ULIB = ulib.o usys.o printf.o umalloc.o thread_lib.o
```

Testing :- Testing for 20 threads and passes is 40

$ Frisbee 20 40

```
$ frisbee 20 40
Pass No:1 Thread 0 passes the token to Thread 1
Pass No:2 Thread 1 passes the token to Thread 2
Pass No:3 Thread 2 passes the token to Thread 3
Pass No:4 Thread 3 passes the token to Thread 4
Pass No:5 Thread 4 passes the token to Thread 5
Pass No:6 Thread 5 passes the token to Thread 6
Pass No:7 Thread 6 passes the token to Thread 7
Pass No:8 Thread 7 passes the token to Thread 8
Pass No:9 Thread 8 passes the token to Thread 9
Pass No:10 Thread 9 passes the token to Thread 10
Pass No:11 Thread 10 passes the token to Thread 11
Pass No:12 Thread 11 passes the token to Thread 12
Pass No:13 Thread 12 passes the token to Thread 13
Pass No:14 Thread 13 passes the token to Thread 14
Pass No:15 Thread 14 passes the token to Thread 15
Pass No:16 Thread 15 passes the token to Thread 16
Pass No:17 Thread 16 passes the token to Thread 17
Pass No:18 Thread 17 passes the token to Thread 18
Pass No:19 Thread 18 passes the token to Thread 19
Pass No:20 Thread 19 passes the token to Thread 0
Pass No:21 Thread 0 passes the token to Thread 1
Pass No:22 Thread 1 passes the token to Thread 2
Pass No:23 Thread 2 passes the token to Thread 3
Pass No:24 Thread 3 passes the token to Thread 4
Pass No:25 Thread 4 passes the token to Thread 5
Pass No:26 Thread 5 passes the token to Thread 6
Pass No:27 Thread 6 passes the token to Thread 7
Pass No:28 Thread 7 passes the token to Thread 8
Pass No:29 Thread 8 passes the token to Thread 9
Pass No:30 Thread 9 passes the token to Thread 10
Pass No:31 Thread 10 passes the token to Thread 11
Pass No:32 Thread 11 passes the token to Thread 12
Pass No:33 Thread 12 passes the token to Thread 13
Pass No:34 Thread 13 passes the token to Thread 14
Pass No:35 Thread 14 passes the token to Thread 15
Pass No:36 Thread 15 passes the token to Thread 16
Pass No:37 Thread 16 passes the token to Thread 17
Pass No:38 Thread 17 passes the token to Thread 18
Pass No:39 Thread 18 passes the token to Thread 19
Pass No:40 Thread 19 passes the token to Thread 0
Simulation of Frisbee game has finished, 40 rounds were played in total!
```