**Lab Report:2** **Abhishek Srivastava & Ravdeep Pasricha**
CS202 Advanced Operating Systems Student Id: 861307778 & 861307836
Prof. Nael B. Abu-Ghazaleh March 14, 2017

## Problem

- **clone()** system call.
- **thread_create()** function implementation.
- **Spin Lock** initialization, acquire and release implementation.
- **Array based queue Lock** initialization, acquire and release implementation.
- **Sequential Lock** initialization, acquire and release implementation.
- **MCS Lock** initialization, acquire and release implementation.
- **Frisbee** program implementation.

## Solution

- Files Modified
  - **defs.h**: Modified for adding clone system call *int clone(void *, int);*.
  - **proc.c**: Added the definition of *int clone(void* stack, int size)* to create a new process, copy the pagedir of the parent and create a local stack of thread for each thread and copy the content of the parent process. *wait()* and *exit()* system call is also modified to handle the shared data between the threads. It is done by maintaining a thread count and freeing the memory only when a single process/thread is left.
  - **proc.h**: Modified *proc* data structure to add *ppid* variable which has the parent process id. It is used to identify the thread who share the same parents. For calculation of the execution time few more parameters are added to the proc structure.
  - **syscall.h**, **syscall.c**, **sysproc.c**, s**usys.S**, **user.h**: Modified to add Clone System call.
  - **Makefile**: Modified for the compilation purposes.
- File added
  - **frisbee.c** : Implementation of frisbee program using simple spin lock.
  - **frisbee_arr.c**: Implementation of frisbee program using array based queuing lock.
  - **frisbee_seq.c**: Implementation of frisbee program using sequential lock.
  - **frisbee_mcs.c**: Implementation of frisbee program using mcs lock.
  - **libthread.h**: Contains the definition of all the locks and thread calls.
  - **libthread.c**: Contains the implementation of all the lock calls for initialization, acquire and release.
- Functions and System calls added
  **Clone System call in proc.c**

```
int
clone(void* stack, int size)
{
    int i, pid;
    struct proc *np;

    // Allocate process.
    if((np = allocproc()) == 0){
        return -1;
    }

    //cs202: use the same address space as parent
    np->pgdir = proc->pgdir;
    np->sz = proc->sz;
    np->parent = proc;
    *np->tf = *proc->tf;
    //cs202: assigning parent id
    np->ppid = proc->pid;
    proc->thread_count++;
    // Clear %eax so that fork returns 0 in the child.
    np->tf->eax = 0;
    //cs202: calculating esp location in current stack
    //which is also the stack size
    uint esp_addr = *(uint*)proc->tf->ebp - proc->tf->esp;
    //cs202: calculating ebp locatoin in current stack
    uint ebp_addr = *(uint*)proc->tf->ebp - proc->tf->ebp;
    //cs202: set esp value of created thread
    np->tf->esp = (uint)stack + size - esp_addr;
    //cs202: set ebp value of created thread
    np->tf->ebp = (uint)stack + size - ebp_addr;

    //cs202: copying the stack content from current process
    memmove((void *)np->tf->esp, (void *)proc->tf->esp, esp_addr);
    for(i = 0; i < NOFILE; i++)
        if(proc->ofile[i])
            np->ofile[i] = filedup(proc->ofile[i]);
    np->cwd = idup(proc->cwd);
    safestrcpy(np->name, proc->name, sizeof(proc->name));
    pid = np->pid;
    acquire(&ptable.lock);
    np->state = RUNNABLE;
    release(&ptable.lock);
    return pid;
}
```

**Locks defintion added in libthread.h**

```
//cs202
//spin lock
```

```
struct lock_t {
     //store the lock value if 0 then not locked, if 1 then locked
    uint is_locked;
};


//spin lock calls implementation
void lock_init(struct lock_t* l);
void lock_acquire(struct lock_t* l);
void lock_release(struct lock_t* l);

//array lock
struct alock_t {
    uint *lock_array;   //lock array
    uint num_th;        //number of threads
    uint next_slot;     //next slot id
    uint curr_slot;     //current slot id
};

//array lock calls implementation
void alock_init(struct alock_t* l, uint num_t);
void alock_acquire(struct alock_t* l, int tid);
void alock_release(struct alock_t* l);
int fetch_and_increment(uint* next);

//sequential lock
struct seqlock_t {
    uint seq_counter;
    struct lock_t sl;
};

//seq lock calls implementation
void seqlock_init(struct seqlock_t* l);
void write_lock(struct seqlock_t* l);
void write_unlock(struct seqlock_t* l);
uint read_counter(struct seqlock_t* l);

//mcs lock
struct mcslock_node {
    uint is_locked;
    struct mcslock_node *next;
};

//mcs lock node
struct mcslock_t {
    struct mcslock_node* node;
};
```

```
//mcs lock calls implementation
void mcslock_init(struct mcslock_t* l);
void mcslock_nodeinit(struct mcslock_node* n);
void mcslock_acquire(struct mcslock_t* l, struct mcslock_node* n);
void mcslock_release(struct mcslock_t* l, struct mcslock_node* n);
struct mcslock_node* fetch_and_store(struct mcslock_t* l,
                                struct mcslock_node* n);

//thread calls
int thread_create(void* (*start_routing)(void *) , void *);
int thread_join(void);
```

**Implementation of locks calls in libthread.c**

```
#include ”types.h”
#include ”user.h”
#include ”stat.h”
#include ”x86.h”
#include ”libthread.h”

//spin lock calls
void lock_init(struct lock_t* l) {
    l->is_locked = 0;
}

void lock_acquire(struct lock_t* l) {
    while(xchg(&l->is_locked, 1) != 0);
}

void lock_release(struct lock_t* l) {
    if(l->is_locked)
        xchg(&l->is_locked, 0);
}

//array lock calls
void alock_init(struct alock_t* l, uint num_t) {
    //memory allocation for array lock
    uint *arr = malloc(sizeof(uint) * num_t);
    //intialization
    l->lock_array = arr;
    int i;
    for(i=0; i<num_t; i++)
        l->lock_array[i] = 0;
    l->lock_array[0] = 1;

    l->num_th = num_t;
    l->next_slot = 0;
    l->curr_slot = -1;
}
```

```
void alock_acquire(struct alock_t* l, int tid) {
    while( l->lock_array[tid] != 1);
    l->curr_slot = tid;
    l->next_slot = l->curr_slot + 1;
    if(l->next_slot == l->num_th)
        l->next_slot = 0;
}

void alock_release(struct alock_t* l) {
    if( l->lock_array[l->curr_slot] ) {
        l->lock_array[l->curr_slot] = 0;
        l->lock_array[l->next_slot] = 1;
    }
}

int fetch_and_increment(uint *next) {
    xchg(next, *next+1);
    return (*next)-1;
}

//seq lock calls
void seqlock_init(struct seqlock_t* l) {
    l->seq_counter = 0;
    lock_init(&l->sl);
}

void write_lock(struct seqlock_t* l) {
    lock_acquire(&l->sl);
    l->seq_counter++;
}

void write_unlock(struct seqlock_t* l) {
    l->seq_counter++;
    lock_release(&l->sl);
}

uint read_counter(struct seqlock_t* l) {
    return l->seq_counter;
}

//mcs lock calls
void mcslock_init(struct mcslock_t* l) {
    l->node = 0;
}

void mcslock_nodeinit(struct mcslock_node* n) {
    n->is_locked = 0;
    n->next = 0;
}

void mcslock_acquire(struct mcslock_t* l, struct mcslock_node* n) {
    //Algo/pseudo code from the paper

    struct mcslock_node* pred = fetch_and_store(l,n);
    if(pred != 0) {
        n->is_locked = 1;
        pred->next = n;
        while(n->is_locked);
    }
}

void mcslock_release(struct mcslock_t* l, struct mcslock_node* n) {
    //Algo from the book
    if(n->next == 0) {
        struct mcslock_node* old_tail = fetch_and_store(l,0);
        if(old_tail == n)
            return;
        struct mcslock_node* usurper = fetch_and_store(l,old_tail);
        while(n->next == 0);
        if(usurper != 0)
            usurper->next = n->next;
        else
            n->next->is_locked = 0;
    }
    else
        n->next->is_locked = 0;
}

struct mcslock_node* fetch_and_store(struct mcslock_t* l,
                                     struct mcslock_node* n) {
    struct mcslock_node* ret_node;
    ret_node = l->node;
    l->node = n;
    return ret_node;
}

//thread interface
int thread_create(void* start_routine(void *), void* arg) {
    uint stack_size = 4096;
    void *stack = malloc(stack_size);
    int t_id = clone(stack,stack_size);
    if(t_id == 0) {
        (start_routine)(arg);
        exit();
    }
    else
        return t_id;
}

int thread_join(void) {
    return wait();
}
```

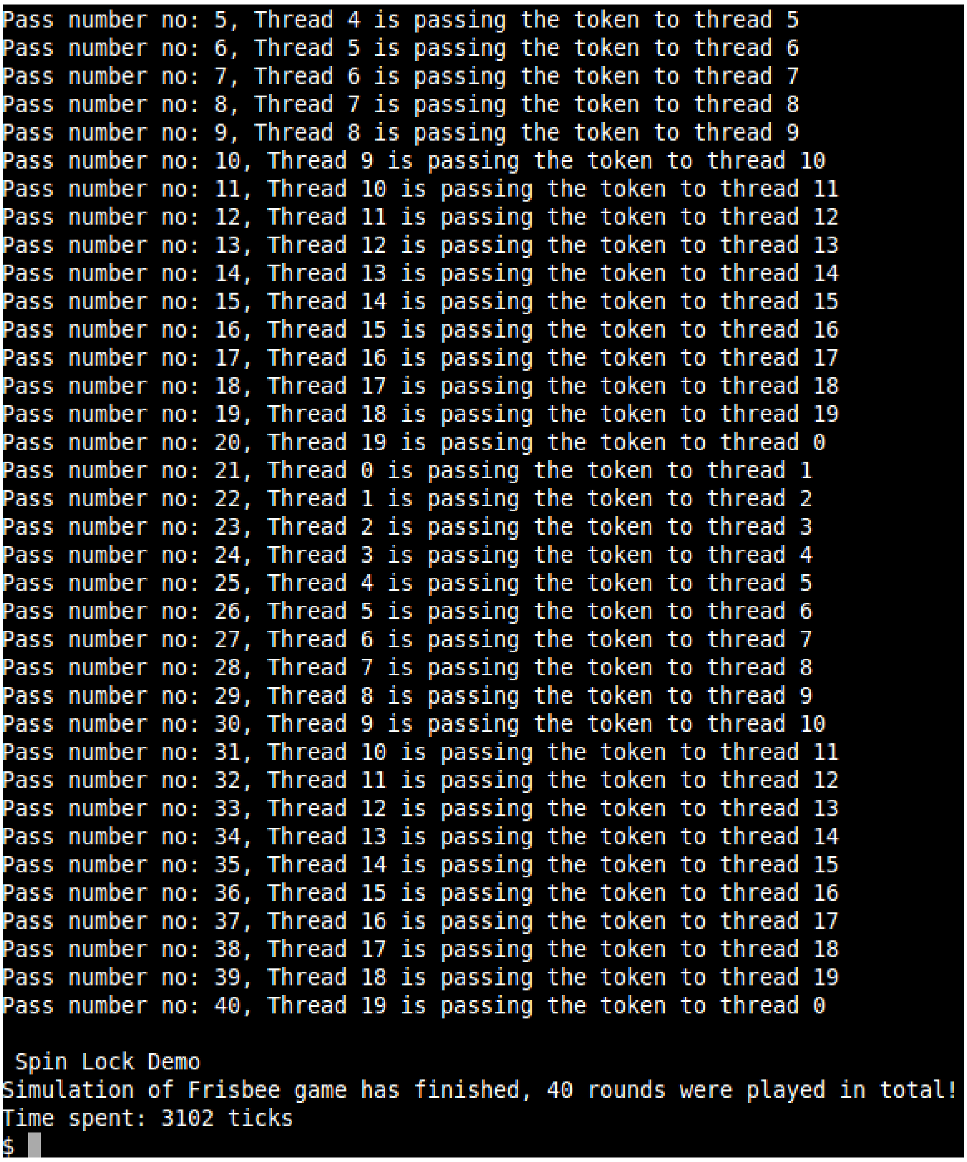
**Frisbee implementation with Spin Lock**

```
#include ”types.h”
#include ”stat.h”
#include ”user.h”
#include ”libthread.h”

//total number of threads to be created
uint num_threads;
//total number of times frisbee to be passed
uint num_pass;

//values related to frisbee
struct frisbee {
    uint token; //token value
    uint pass_value; //pass value
    struct lock_t sl; //spin lock
} spin;

struct data {
    int tid;
};

//This is the function which threads will be executing
//In this thread we check the token value and if it is same as thread i
//we increase the pass count till number of pass value is less than tot
void* pass_frisbee(void* arg)
{
    struct data *fdata = (struct data *)arg;

    int thid = fdata->tid; //thread id

    while(spin.pass_value < num_pass) {
        //acquire the lock
        lock_acquire(&spin.sl);

        //if token value is same thread id
        if(thid == spin.token && spin.pass_value < num_pass) {
            spin.pass_value++;
            spin.token = spin.token++;
            if(spin.token == num_threads)
                spin.token = 0;
            printf(1,”Pass number no: %d, Thread %d is passing the toke
            to thread %d\n”, spin.pass_value, thid, spin.token);
        }
        //release the lock
        lock_release(&spin.sl);
    }
    return 0;
}

int main(int argc, char*argv[])
{
    struct data *tdata;
    //check if parameters entered are correct
    if(argc != 3) {
        printf(1,”frisbee <num_thread> <num_pass>”);
        exit();
    }

    num_threads = atoi(argv[1]);
    num_pass = atoi(argv[2]);
    //spin lock initialization
    //initialize token value
    spin.token = 0;
    spin.pass_value = 0;
    //lock intialization
    lock_init(&spin.sl);
    //create threads
    int i;
    //thread data to pass to the threads
    tdata = malloc(sizeof(struct data) * num_threads);
    for(i=0; i<num_threads; i++)
        tdata[i].tid = i;

    for(i = 0; i<num_threads; i++) {
        thread_create((void*)pass_frisbee,(void*)&tdata[i]);
    }
    //wait for all threads to finish
    for(i = 0; i<num_threads; i++) {
        thread_join();
    }
    printf(1, ”\nSpin Lock Demo\n”);
    printf(1, ”\nSimulation of Frisbee game has finished,
            %d rounds were played in total!\n”, num_pass);
    return 0;
}
```

Similar to this we have implemented Array based queue locking, Sequential locking and MCS locking. We have provided their code *frisbee_arr.c*, *frisbee_seq.c* and *frisbee_mcs.c*.

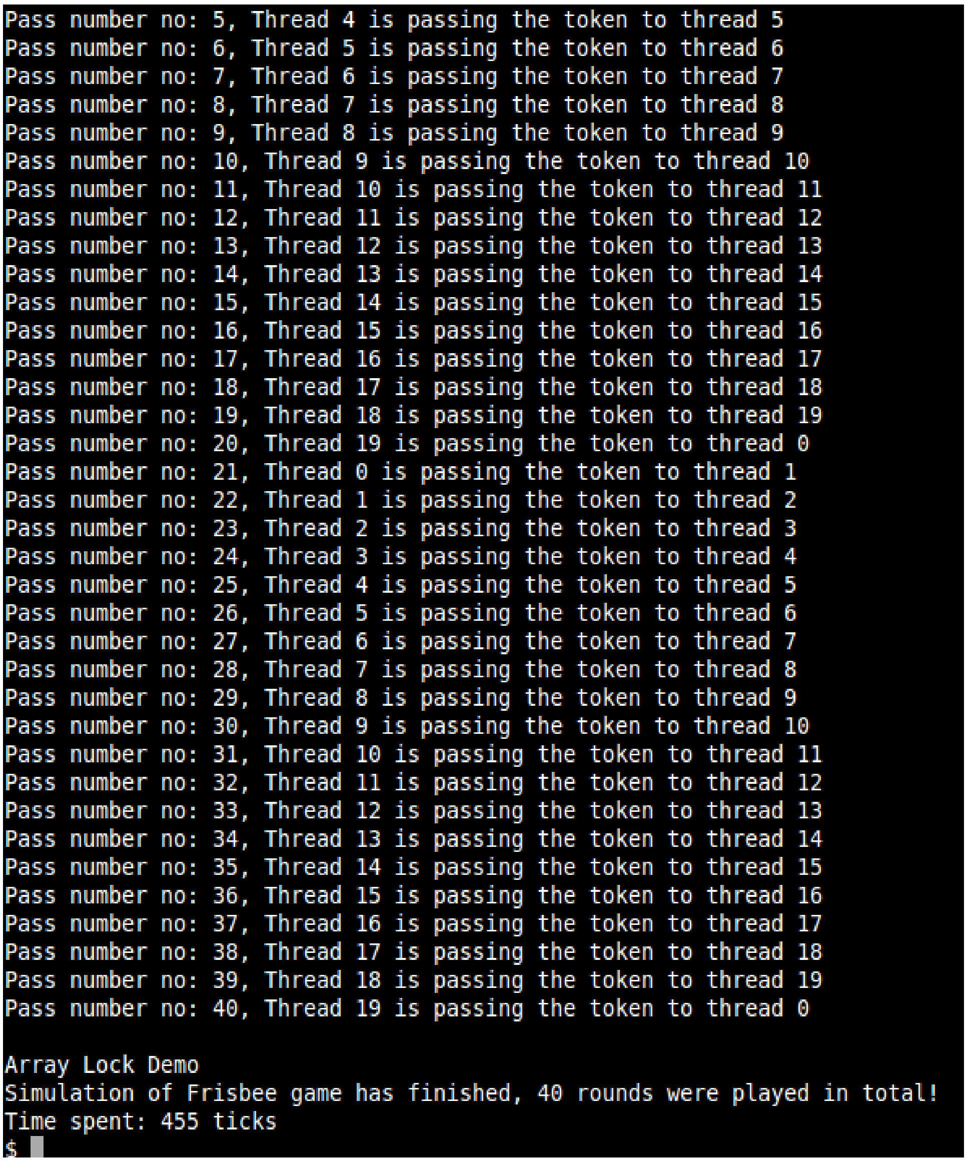
- Result

  From the execution of all the lock implementation we can see in the results that spin lock is the slowest one. Sequential lock(921 ticks) performs way better than Spin lock(3102 ticks) when thread number is in large numbers. Screen shots of the executions of all the locks implemented is attached below with execution time in ticks is also shown in them.



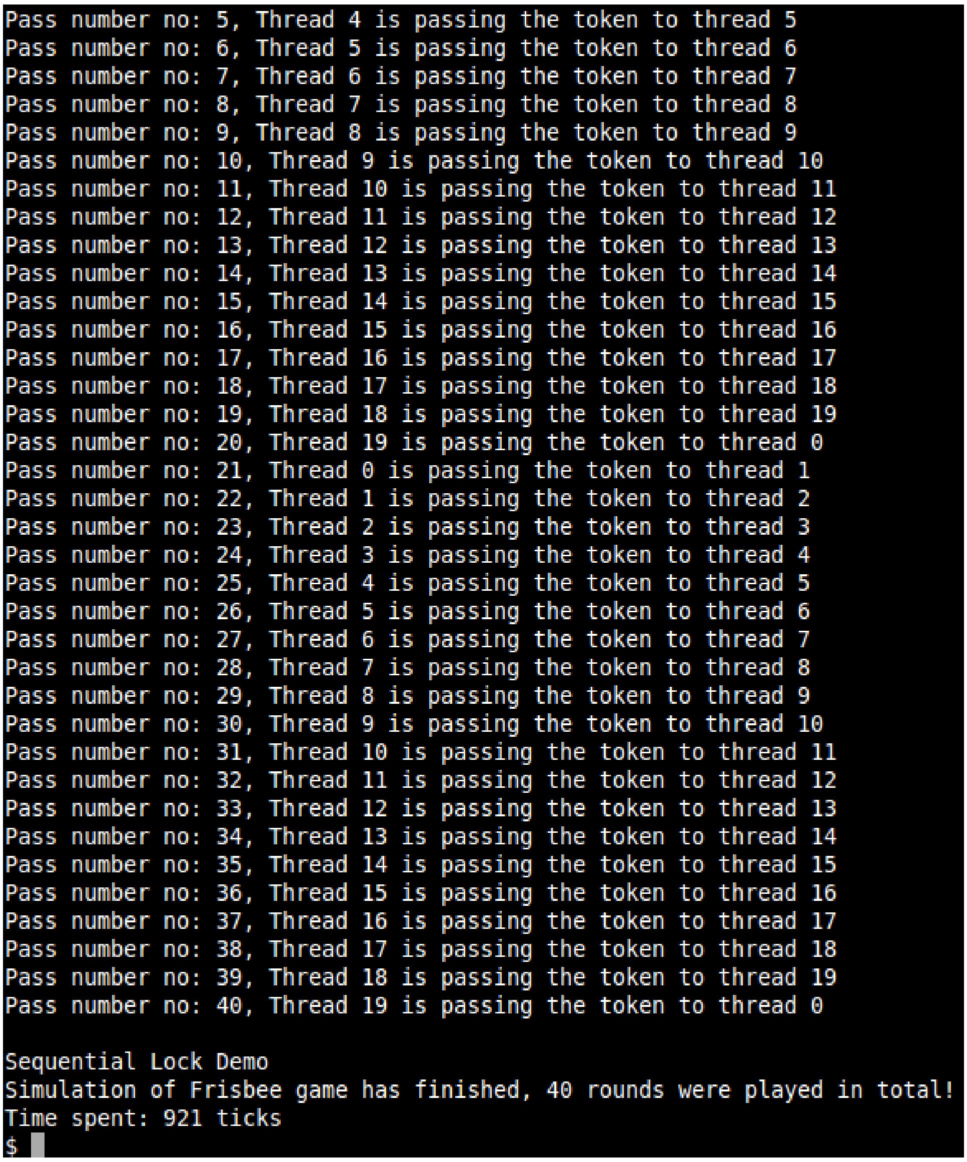
```
Pass number no: 5, Thread 4 is passing the token to thread 5
Pass number no: 6, Thread 5 is passing the token to thread 6
Pass number no: 7, Thread 6 is passing the token to thread 7
Pass number no: 8, Thread 7 is passing the token to thread 8
Pass number no: 9, Thread 8 is passing the token to thread 9
Pass number no: 10, Thread 9 is passing the token to thread 10
Pass number no: 11, Thread 10 is passing the token to thread 11
Pass number no: 12, Thread 11 is passing the token to thread 12
Pass number no: 13, Thread 12 is passing the token to thread 13
Pass number no: 14, Thread 13 is passing the token to thread 14
Pass number no: 15, Thread 14 is passing the token to thread 15
Pass number no: 16, Thread 15 is passing the token to thread 16
Pass number no: 17, Thread 16 is passing the token to thread 17
Pass number no: 18, Thread 17 is passing the token to thread 18
Pass number no: 19, Thread 18 is passing the token to thread 19
Pass number no: 20, Thread 19 is passing the token to thread 0
Pass number no: 21, Thread 0 is passing the token to thread 1
Pass number no: 22, Thread 1 is passing the token to thread 2
Pass number no: 23, Thread 2 is passing the token to thread 3
Pass number no: 24, Thread 3 is passing the token to thread 4
Pass number no: 25, Thread 4 is passing the token to thread 5
Pass number no: 26, Thread 5 is passing the token to thread 6
Pass number no: 27, Thread 6 is passing the token to thread 7
Pass number no: 28, Thread 7 is passing the token to thread 8
Pass number no: 29, Thread 8 is passing the token to thread 9
Pass number no: 30, Thread 9 is passing the token to thread 10
Pass number no: 31, Thread 10 is passing the token to thread 11
Pass number no: 32, Thread 11 is passing the token to thread 12
Pass number no: 33, Thread 12 is passing the token to thread 13
Pass number no: 34, Thread 13 is passing the token to thread 14
Pass number no: 35, Thread 14 is passing the token to thread 15
Pass number no: 36, Thread 15 is passing the token to thread 16
Pass number no: 37, Thread 16 is passing the token to thread 17
Pass number no: 38, Thread 17 is passing the token to thread 18
Pass number no: 39, Thread 18 is passing the token to thread 19
Pass number no: 40, Thread 19 is passing the token to thread 0

 Spin Lock Demo
Simulation of Frisbee game has finished, 40 rounds were played in total!
Time spent: 3102 ticks
$ 
```

Figure 1: Spin Lock Execution Result.



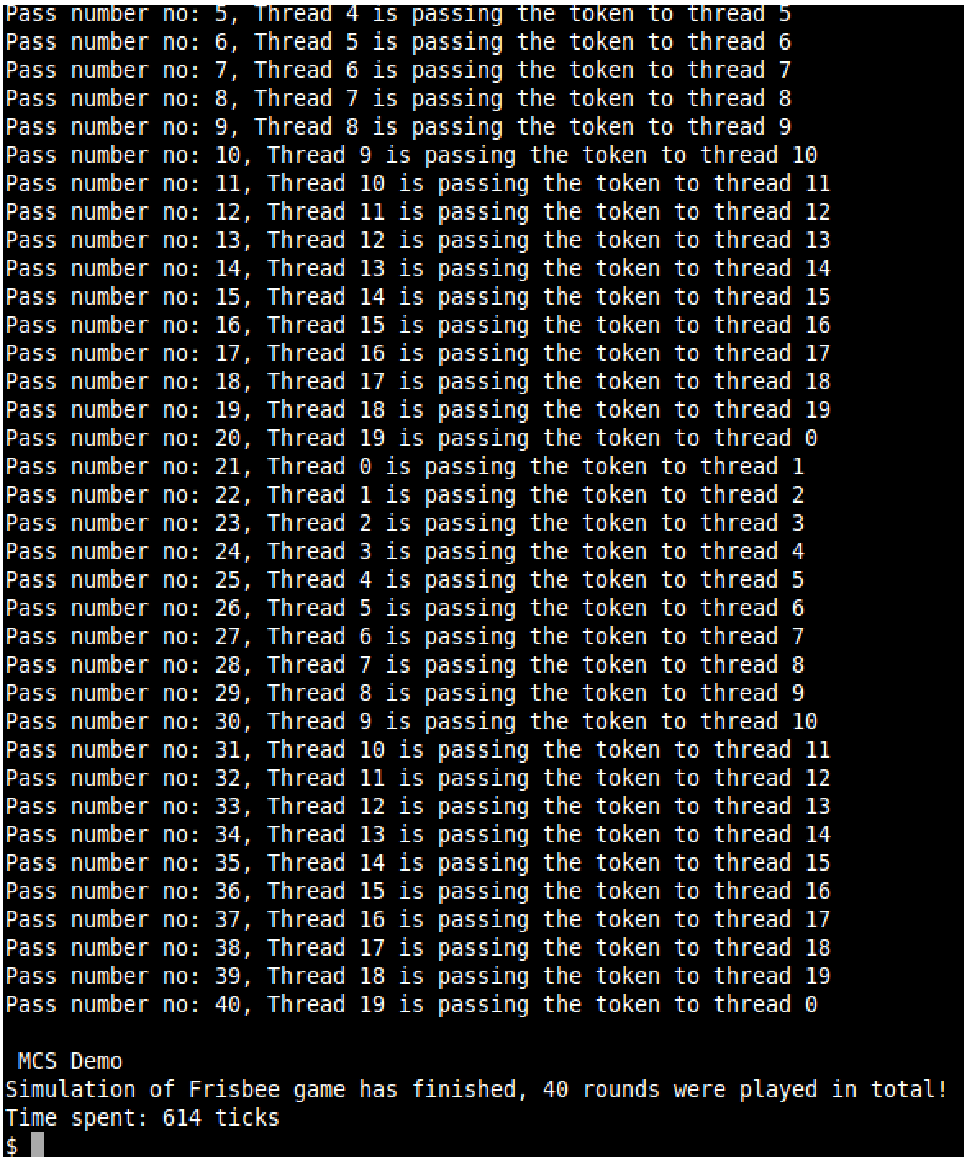
```
Pass number no: 5, Thread 4 is passing the token to thread 5
Pass number no: 6, Thread 5 is passing the token to thread 6
Pass number no: 7, Thread 6 is passing the token to thread 7
Pass number no: 8, Thread 7 is passing the token to thread 8
Pass number no: 9, Thread 8 is passing the token to thread 9
Pass number no: 10, Thread 9 is passing the token to thread 10
Pass number no: 11, Thread 10 is passing the token to thread 11
Pass number no: 12, Thread 11 is passing the token to thread 12
Pass number no: 13, Thread 12 is passing the token to thread 13
Pass number no: 14, Thread 13 is passing the token to thread 14
Pass number no: 15, Thread 14 is passing the token to thread 15
Pass number no: 16, Thread 15 is passing the token to thread 16
Pass number no: 17, Thread 16 is passing the token to thread 17
Pass number no: 18, Thread 17 is passing the token to thread 18
Pass number no: 19, Thread 18 is passing the token to thread 19
Pass number no: 20, Thread 19 is passing the token to thread 0
Pass number no: 21, Thread 0 is passing the token to thread 1
Pass number no: 22, Thread 1 is passing the token to thread 2
Pass number no: 23, Thread 2 is passing the token to thread 3
Pass number no: 24, Thread 3 is passing the token to thread 4
Pass number no: 25, Thread 4 is passing the token to thread 5
Pass number no: 26, Thread 5 is passing the token to thread 6
Pass number no: 27, Thread 6 is passing the token to thread 7
Pass number no: 28, Thread 7 is passing the token to thread 8
Pass number no: 29, Thread 8 is passing the token to thread 9
Pass number no: 30, Thread 9 is passing the token to thread 10
Pass number no: 31, Thread 10 is passing the token to thread 11
Pass number no: 32, Thread 11 is passing the token to thread 12
Pass number no: 33, Thread 12 is passing the token to thread 13
Pass number no: 34, Thread 13 is passing the token to thread 14
Pass number no: 35, Thread 14 is passing the token to thread 15
Pass number no: 36, Thread 15 is passing the token to thread 16
Pass number no: 37, Thread 16 is passing the token to thread 17
Pass number no: 38, Thread 17 is passing the token to thread 18
Pass number no: 39, Thread 18 is passing the token to thread 19
Pass number no: 40, Thread 19 is passing the token to thread 0

Array Lock Demo
Simulation of Frisbee game has finished, 40 rounds were played in total!
Time spent: 455 ticks
$
```

Figure 2: Array Based Queuing Lock Execution Result.



```
Pass number no: 5, Thread 4 is passing the token to thread 5
Pass number no: 6, Thread 5 is passing the token to thread 6
Pass number no: 7, Thread 6 is passing the token to thread 7
Pass number no: 8, Thread 7 is passing the token to thread 8
Pass number no: 9, Thread 8 is passing the token to thread 9
Pass number no: 10, Thread 9 is passing the token to thread 10
Pass number no: 11, Thread 10 is passing the token to thread 11
Pass number no: 12, Thread 11 is passing the token to thread 12
Pass number no: 13, Thread 12 is passing the token to thread 13
Pass number no: 14, Thread 13 is passing the token to thread 14
Pass number no: 15, Thread 14 is passing the token to thread 15
Pass number no: 16, Thread 15 is passing the token to thread 16
Pass number no: 17, Thread 16 is passing the token to thread 17
Pass number no: 18, Thread 17 is passing the token to thread 18
Pass number no: 19, Thread 18 is passing the token to thread 19
Pass number no: 20, Thread 19 is passing the token to thread 0
Pass number no: 21, Thread 0 is passing the token to thread 1
Pass number no: 22, Thread 1 is passing the token to thread 2
Pass number no: 23, Thread 2 is passing the token to thread 3
Pass number no: 24, Thread 3 is passing the token to thread 4
Pass number no: 25, Thread 4 is passing the token to thread 5
Pass number no: 26, Thread 5 is passing the token to thread 6
Pass number no: 27, Thread 6 is passing the token to thread 7
Pass number no: 28, Thread 7 is passing the token to thread 8
Pass number no: 29, Thread 8 is passing the token to thread 9
Pass number no: 30, Thread 9 is passing the token to thread 10
Pass number no: 31, Thread 10 is passing the token to thread 11
Pass number no: 32, Thread 11 is passing the token to thread 12
Pass number no: 33, Thread 12 is passing the token to thread 13
Pass number no: 34, Thread 13 is passing the token to thread 14
Pass number no: 35, Thread 14 is passing the token to thread 15
Pass number no: 36, Thread 15 is passing the token to thread 16
Pass number no: 37, Thread 16 is passing the token to thread 17
Pass number no: 38, Thread 17 is passing the token to thread 18
Pass number no: 39, Thread 18 is passing the token to thread 19
Pass number no: 40, Thread 19 is passing the token to thread 0

Sequential Lock Demo
Simulation of Frisbee game has finished, 40 rounds were played in total!
Time spent: 921 ticks
$
```

Figure 3: Sequential Lock Execution Result.



```
Pass number no: 5, Thread 4 is passing the token to thread 5
Pass number no: 6, Thread 5 is passing the token to thread 6
Pass number no: 7, Thread 6 is passing the token to thread 7
Pass number no: 8, Thread 7 is passing the token to thread 8
Pass number no: 9, Thread 8 is passing the token to thread 9
Pass number no: 10, Thread 9 is passing the token to thread 10
Pass number no: 11, Thread 10 is passing the token to thread 11
Pass number no: 12, Thread 11 is passing the token to thread 12
Pass number no: 13, Thread 12 is passing the token to thread 13
Pass number no: 14, Thread 13 is passing the token to thread 14
Pass number no: 15, Thread 14 is passing the token to thread 15
Pass number no: 16, Thread 15 is passing the token to thread 16
Pass number no: 17, Thread 16 is passing the token to thread 17
Pass number no: 18, Thread 17 is passing the token to thread 18
Pass number no: 19, Thread 18 is passing the token to thread 19
Pass number no: 20, Thread 19 is passing the token to thread 0
Pass number no: 21, Thread 0 is passing the token to thread 1
Pass number no: 22, Thread 1 is passing the token to thread 2
Pass number no: 23, Thread 2 is passing the token to thread 3
Pass number no: 24, Thread 3 is passing the token to thread 4
Pass number no: 25, Thread 4 is passing the token to thread 5
Pass number no: 26, Thread 5 is passing the token to thread 6
Pass number no: 27, Thread 6 is passing the token to thread 7
Pass number no: 28, Thread 7 is passing the token to thread 8
Pass number no: 29, Thread 8 is passing the token to thread 9
Pass number no: 30, Thread 9 is passing the token to thread 10
Pass number no: 31, Thread 10 is passing the token to thread 11
Pass number no: 32, Thread 11 is passing the token to thread 12
Pass number no: 33, Thread 12 is passing the token to thread 13
Pass number no: 34, Thread 13 is passing the token to thread 14
Pass number no: 35, Thread 14 is passing the token to thread 15
Pass number no: 36, Thread 15 is passing the token to thread 16
Pass number no: 37, Thread 16 is passing the token to thread 17
Pass number no: 38, Thread 17 is passing the token to thread 18
Pass number no: 39, Thread 18 is passing the token to thread 19
Pass number no: 40, Thread 19 is passing the token to thread 0

 MCS Demo
Simulation of Frisbee game has finished, 40 rounds were played in total!
Time spent: 614 ticks
$
```

Figure 4: MCS Lock Execution Result.