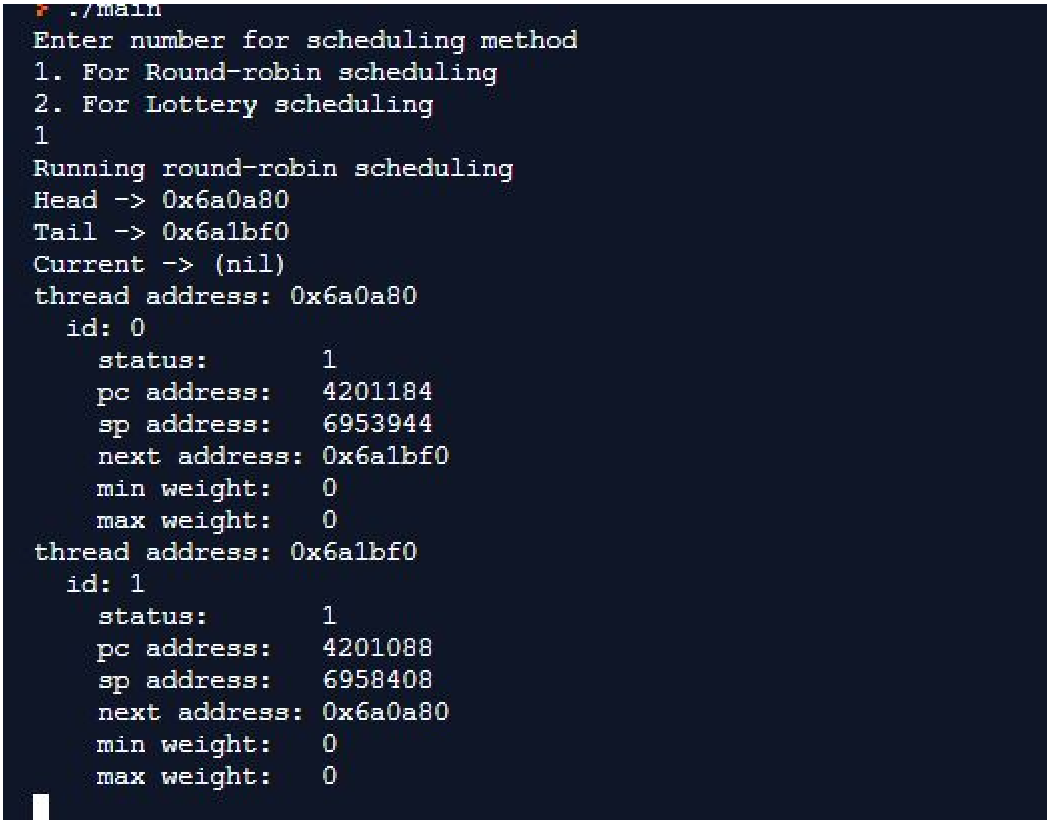
Assignment #3 - User Level Thread

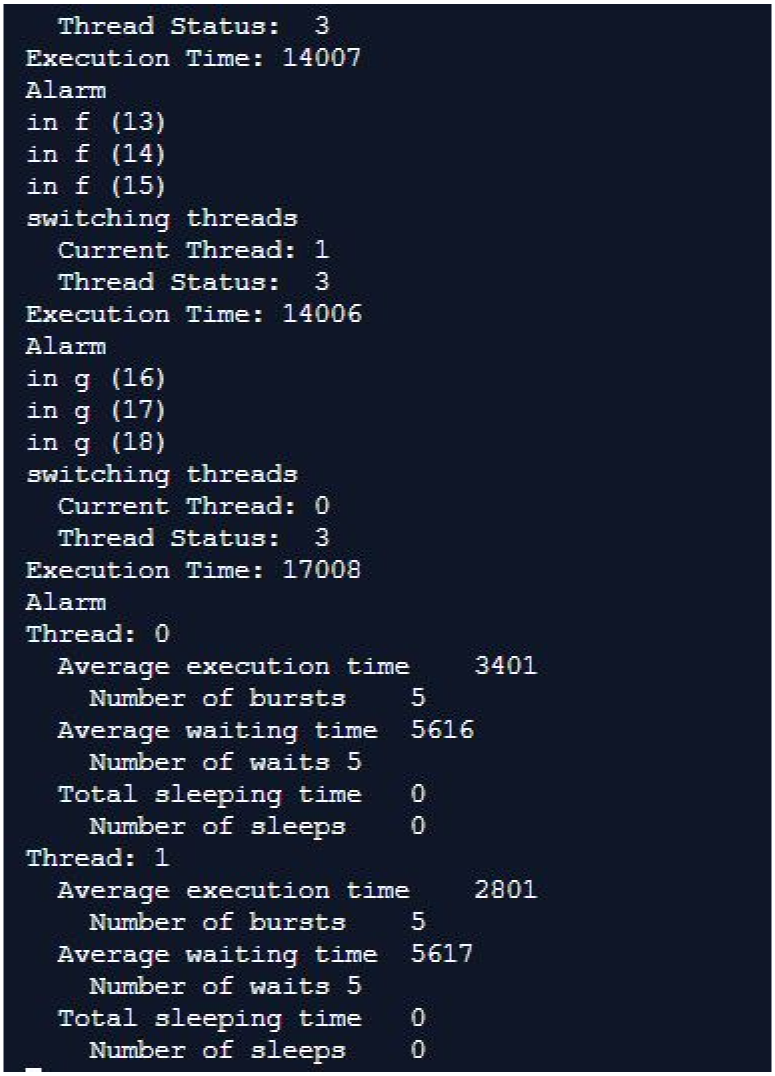
Rongguang Ou
CSC139 Fall 2019 A3

Thread Summary for Round Robin scheduling :



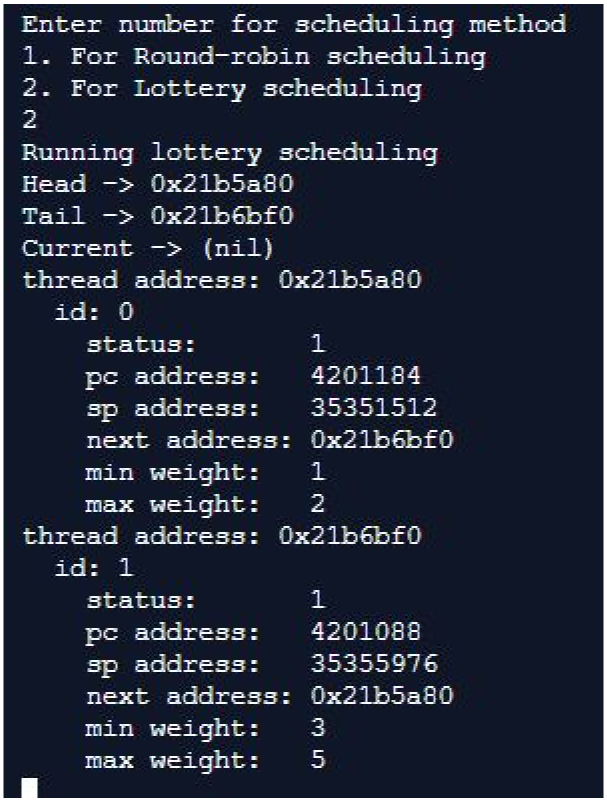
```
❯ ./main
Enter number for scheduling method
1. For Round-robin scheduling
2. For Lottery scheduling
1
Running round-robin scheduling
Head -> 0x6a0a80
Tail -> 0x6a1bf0
Current -> (nil)
thread address: 0x6a0a80
  id: 0
    status:       1
    pc address:   4201184
    sp address:   6953944
    next address: 0x6a1bf0
    min weight:   0
    max weight:   0
thread address: 0x6a1bf0
  id: 1
    status:       1
    pc address:   4201088
    sp address:   6958408
    next address: 0x6a0a80
    min weight:   0
    max weight:   0
```

Output for Round Robin Scheduling:



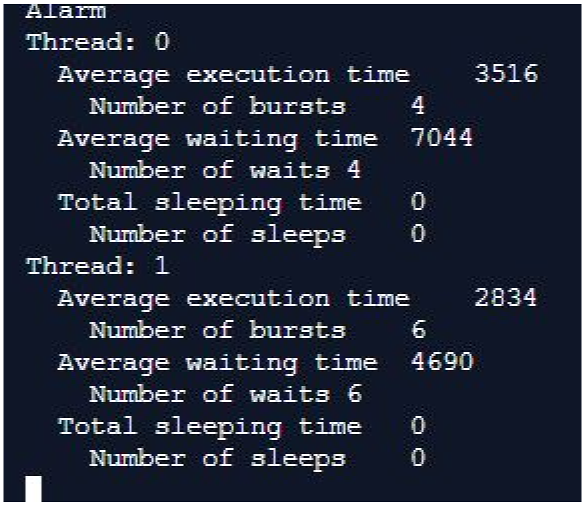
```
  Thread Status:  3
Execution Time: 14007
Alarm
in f (13)
in f (14)
in f (15)
switching threads
  Current Thread: 1
  Thread Status:  3
Execution Time: 14006
Alarm
in g (16)
in g (17)
in g (18)
switching threads
  Current Thread: 0
  Thread Status:  3
Execution Time: 17008
Alarm
Thread: 0
  Average execution time    3401
    Number of bursts    5
  Average waiting time  5616
    Number of waits 5
  Total sleeping time   0
    Number of sleeps    0
Thread: 1
  Average execution time    2801
    Number of bursts    5
  Average waiting time  5617
    Number of waits 5
  Total sleeping time   0
    Number of sleeps    0
```

Lottery based scheduling setup:



```
Enter number for scheduling method
1. For Round-robin scheduling
2. For Lottery scheduling
2
Running lottery scheduling
Head -> 0x21b5a80
Tail -> 0x21b6bf0
Current -> (nil)
thread address: 0x21b5a80
  id: 0
    status:       1
    pc address:   4201184
    sp address:   35351512
    next address: 0x21b6bf0
    min weight:   1
    max weight:   2
thread address: 0x21b6bf0
  id: 1
    status:       1
    pc address:   4201088
    sp address:   35355976
    next address: 0x21b5a80
    min weight:   3
    max weight:   5
```

Lottery based scheduling summary:



```
Alarm
Thread: 0
  Average execution time    3516
    Number of bursts    4
  Average waiting time  7044
    Number of waits 4
  Total sleeping time   0
    Number of sleeps    0
Thread: 1
  Average execution time    2834
    Number of bursts    6
  Average waiting time  4690
    Number of waits 6
  Total sleeping time   0
    Number of sleeps    0
```

SourceCode:

```
/*
        CSC139 Assignment 3 - User Level Kernel Library package
        Author : Rongguang Ou
*/
#include "stdio.h"
#include <sys/timeb.h> /* for timeb */
#include <setjmp.h> /* sigjmp , siglongjmp */
#include <signal.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <math.h>
#include <time.h>
#include <sys/time.h>
/*  OVERVIEW

                struct _weight_struct
                struct _sleep_struct
                struct _wait_struct
                struct TCB

                void go()
                void wakeSleepingThread()
                void insert_thread_to_list(TCB*  newTCB);
                int createThread(FUNC_PTR f);
                void CleanUp();
                void printEndStatus(tcbPtr curr)
                void printQueue();
                tcbPtr findTcb(int num);
                yieldCPU();
                GetId();
                void SleepThread(int sec);
                void dispatch(int sig)
*/





/**************
  BLACK BOX  *
***************/
```

```
typedef unsigned long address_t;

#ifdef __x86_64__


#define JB_SP 6
#define JB_PC 7


unsigned long tr_address(unsigned long addr)
{
    unsigned long ret;
    asm volatile("xor    %%fs:0x30,%0\n"
                 "rol    $0x11,%0\n"
                 : "=g" (ret)
                 : "0" (addr));
    return ret;
}

#else

#define JB_SP 4
#define JB_PC 5


unsigned long tr_address(unsigned long addr)
{
    unsigned long ret;
    asm volatile("xor    %%gs:0x18,%0\n"
                 "rol    $0x9,%0\n"
                 : "=g" (ret)
                 : "0" (addr));
    return ret;
}

#endif
```

```
/*************
	STRUCTS  *
**************/

/* Thread weight */
typedef struct _weight_struct
{
	int min_weight;
	int max_weight;
}_thread_weight;

/* Sleep  */
typedef struct _sleep_struct
{
	int start_sleeping;
	int sleep_to;
	struct timeb start_s;
	int total;
}_thread_sleep;

/* Wait */
typedef struct _wait_struct
{
	int start_waiting;
	int stop_waiting;
	struct timeb start_w, stop_w;
	int total;
}_thread_wait;

/* Thread Control Block */
typedef struct TCB
{
	unsigned long pc;
	unsigned long sp;
	_thread_weight weight;
	_thread_sleep sleep_time;
	_thread_wait wait_time;
	sigjmp_buf jbuf;
	int thread_id;
	int thread_status;
	int num_bursts;
	int num_waits;
	int num_sleeps;
	int exec_time;
	struct TCB *next;
}TCB;




/*****************
	CONSTANTS    *
******************/
#define MAX_THREAD_SIZE  100
#define SECOND  1000000
#define STACK_SIZE 4096
#define TIME_QUANTUM 1*SECOND
#define STATUS_READY 1
#define STATUS_SLEEPING 2
#define STATUS_RUNNING 3
#define STATUS_SUSPENDED 4
#define RoundRobin 1
#define Lottery 2

static int RUN_TIME = 15000;
static int current_schedule = 1; /* Default to RR = 1   ,  Lottery = 2 */

static int num_Thread = 0;
static int curr_thread_count = 0;
static int weight_total = 1;
struct timeb t_start, t_stop;

typedef TCB* tcbPtr;
tcbPtr head = NULL;
tcbPtr tail = NULL;
tcbPtr curr_thread = NULL;

typedef void (*FUNC_PTR)(void);





/* Prototypes */
void go();
void wakeSleepingThread();
void insert_thread_to_list(TCB*  newTCB);
int createThread(FUNC_PTR f);
void CleanUp();
void printEndStatus(tcbPtr curr);
void printQueue();
tcbPtr findTcb(int num);
void yieldCPU();
int GetId();
void SleepThread(int sec);
void f();
void g();
void dispatch(int sig);


/*******************
  Member functions *
 *******************/

/* Create thread */
int createThread(FUNC_PTR f){

        /* Allocate Memory for TCB  */
        tcbPtr newTCB = malloc(sizeof(TCB));

        if(newTCB == NULL){
                newTCB->thread_id = -1;
                num_Thread++;
        }else{
                /* Initialize/Populate  TCB */
                newTCB->thread_id = curr_thread_count++;
                newTCB->pc = (address_t)f;
                newTCB->sp = (address_t)malloc(STACK_SIZE);
                newTCB->sp = newTCB->sp + STACK_SIZE - sizeof(address_t); /* Move SP to correct region */
                newTCB->num_bursts = 0;
                newTCB->exec_time = 0;
                newTCB->num_waits = 0;
                newTCB->num_sleeps = 0;
                newTCB->sleep_time.sleep_to = 0;
                newTCB->sleep_time.start_sleeping = 0;
                newTCB->sleep_time.total = 0;
                newTCB->wait_time.start_w.millitm = 0;
                newTCB->wait_time.stop_w.millitm = 0;
```

```
			newTCB->wait_time.total = 0;
			newTCB->next = NULL;
			struct timeb t;
			ftime(&t);
			newTCB->wait_time.start_w = t;
			newTCB->thread_status = STATUS_READY;
			num_Thread++;
			if(current_schedule == Lottery){
				newTCB->weight.min_weight = weight_total;
				newTCB->weight.max_weight = weight_total +
pow(2,newTCB->thread_id);
				weight_total = newTCB->weight.max_weight +  1;
			}else if(current_schedule == Lottery){
				newTCB->weight.min_weight = 0;
				newTCB->weight.max_weight = 0;
			}else{
				//ERROR , Unknown schedule
			}

			/* If exceed max thread count */
			if(num_Thread >= MAX_THREAD_SIZE){
				CleanUp();
			}
		}

		/* Save State , set SP && PC */
		sigsetjmp(newTCB->jbuf,1);
   (newTCB->jbuf->__jmpbuf)[JB_SP] = tr_address(newTCB->sp);
		(newTCB->jbuf->__jmpbuf)[JB_PC] = tr_address(newTCB->pc);
		sigemptyset(&newTCB->jbuf->__saved_mask);

		/* Add thread to queue */
		insert_thread_to_list(newTCB);

		return newTCB->thread_id;
}

/* Add TCB to circular linked-list  */
void insert_thread_to_list(tcbPtr  newTCB){

		/* First ever TCB  */
		if(head == NULL){
			head = tail = newTCB;
```

```
        head->next = head;
    }else{ /* Not first one , add to next slot */
        newTCB->next = head;
        tail->next = newTCB;
        tail = newTCB;
    }
}


void printEndStatus(tcbPtr curr){

    int avg_run_time = 0;
    int avg_wait_time = 0;
    int total_sleep_time = 0;

    printf("Thread: %d\n" , curr->thread_id);

    if(curr->num_bursts > 0){
        avg_run_time = (curr->exec_time)/(curr->num_bursts);
    }
    if(curr->num_waits > 0){
        avg_wait_time = (curr->wait_time.total)/(curr->num_waits);
    }
    if(curr->num_sleeps > 0){
        total_sleep_time = curr->sleep_time.total;
    }

    printf("  Average execution time\t%d\n", avg_run_time);
    printf("    Number of bursts\t%d\n", curr->num_bursts);
    printf("  Average waiting time\t%d\n", avg_wait_time);
    printf("    Number of waits\t%d\n", curr->num_waits);
    printf("  Total sleeping time\t%d\n", total_sleep_time);
    printf("    Number of sleeps\t%d\n", curr->num_sleeps);

}


void CleanUp(){
    tcbPtr curr = head;
    tcbPtr trash = NULL;
    int i;
    /* Print All Thread Summary */
    for(i = 0 ; i < num_Thread; i++){
        curr->thread_status = STATUS_SUSPENDED;
        printEndStatus(curr);
        curr = curr->next;
    }
    /* Free memory allocated for threads */
    while(curr != tail){
        trash = curr;
        curr = curr->next;
        free(trash);
    }
    exit(0);
}

void printQueue(){
    tcbPtr curr;
    int counter = 0;
    if(head != NULL){
        printf("Head -> %p\n" , head);
        printf("Tail -> %p\n" , tail);
        printf("Current -> %p\n", curr_thread);
    }
    curr = head;
    do{
        printf("thread address: %p\n", curr);
        printf("  id: %d\n", curr->thread_id);
        printf("    status:       %d\n", curr->thread_status);
        printf("    pc address:   %lu\n", curr->pc);
        printf("    sp address:   %lu\n", curr->sp);
        printf("    next address: %p\n", curr->next);
        printf("    min weight:   %d\n", curr->weight.min_weight);
        printf("    max weight:   %d\n", curr->weight.max_weight);
        curr = curr->next;
        counter++;
    }while(counter != num_Thread);
}

void go(){
    signal(SIGVTALRM, dispatch); /* Assign  dispatch() as the handler for signal:
SIGVTALRM */
    srand(time(NULL));

    struct itimerval tv;
    tv.it_value.tv_sec = 2;
    tv.it_value.tv_usec = 0;
    tv.it_interval.tv_sec = 2;
    tv.it_interval.tv_usec = 0;

    setitimer(ITIMER_VIRTUAL, &tv, NULL);

    createThread(g);
    createThread(f);

    printQueue();

    while(1);
}

void SleepThread(int sec){
    printf(" SLEEPING\n");

    struct timeb t;
    ftime(&t);

    curr_thread->num_sleeps++;

    curr_thread->sleep_time.start_s = t;
    curr_thread->sleep_time.sleep_to = t.millitm + sec;

    curr_thread->thread_status = STATUS_SLEEPING;

    yieldCPU();
}

void wakeSleepingThread(){
    tcbPtr curr = head;
    int i;
    for(i = 0 ; i < num_Thread; i++){
        struct timeb t;
        ftime(&t);

        if((curr->thread_status == STATUS_SLEEPING) &&
(t.time>curr_thread->sleep_time.sleep_to)){
            curr_thread->thread_status = STATUS_READY;
            curr_thread->wait_time.start_w = t;
            curr_thread->sleep_time.total +=  ( 1000.0 * (t.time -
curr_thread->sleep_time.start_s.time ) + (t.millitm - curr_thread->sleep_time.start_s.millitm));
```

```
        }

        curr = curr->next;
    }
}

tcbPtr findTcb(int num){
    tcbPtr curr = head;
    int i;
    for(i = 0 ; i < num_Thread; i++){
        if((num >= curr->weight.min_weight) && (num <= curr->weight.max_weight)){
            break;
        }else{
            curr = curr->next;
        }
    }

    if(i == num_Thread) return NULL;

    return curr;
}



void yieldCPU(){
    printf("switching threads\n");

    ftime(&t_stop);
    curr_thread->exec_time += ( 1000.0 * (t_stop.time - t_start.time) + (t_stop.millitm -
t_start.millitm));

    printf("  Current Thread: %d\n", curr_thread->thread_id);
    printf("  Thread Status:  %d\n", curr_thread->thread_status);
    printf("Execution Time: %d\n", curr_thread->exec_time);

    usleep(2*SECOND);

    raise(SIGVTALRM);
}

void dispatch(int sig){
        wakeSleepingThread();
    printf("Alarm\n");
    //Round-robin scheduling
    if(current_schedule == 1)
    {
        if(curr_thread == NULL)
        {
            curr_thread = head;
            head->thread_status = STATUS_RUNNING;
            ftime(&t_start);
            curr_thread->wait_time.stop_w = t_start;
            curr_thread->wait_time.total += ( 1000.0 *
(curr_thread->wait_time.stop_w.time - curr_thread->wait_time.start_w.time) +
(curr_thread->wait_time.stop_w.millitm - curr_thread->wait_time.start_w.millitm));
            siglongjmp(head->jbuf, 1);
        }
        else
        {
            if( (curr_thread->exec_time) > RUN_TIME )
            {
                CleanUp();
            }

            if(sigsetjmp(curr_thread->jbuf, 1) == 1)
            {
                ftime(&t_start);
                return;
            }

            struct timeb temp_time;
            ftime(&temp_time);
            curr_thread->thread_status = STATUS_READY;
            curr_thread->wait_time.start_w = temp_time;
            curr_thread = curr_thread->next;

            while(curr_thread->thread_status != STATUS_READY)
                curr_thread= curr_thread->next;
            curr_thread->thread_status = STATUS_RUNNING;
            ftime(&t_start);
            curr_thread->wait_time.stop_w = t_start;

            if(curr_thread->wait_time.stop_w.millitm != 0)
            {
                curr_thread->wait_time.total+= ( 1000.0 *
(curr_thread->wait_time.stop_w.time - curr_thread->wait_time.start_w.time) +
(curr_thread->wait_time.stop_w.millitm - curr_thread->wait_time.start_w.millitm));
                curr_thread->num_waits++;
            }
            curr_thread->num_bursts++;
            siglongjmp(curr_thread->jbuf, 1);
        }
    } // Lottery scheduling
    else if (current_schedule == 2)
    {
        if(curr_thread == NULL)
        {
            curr_thread = head;
            head->thread_status = STATUS_RUNNING;
            ftime(&t_start);
            curr_thread->wait_time.stop_w = t_start;
            curr_thread->wait_time.total += ( 1000.0 *
(curr_thread->wait_time.stop_w.time - curr_thread->wait_time.start_w.time) +
(curr_thread->wait_time.stop_w.millitm - curr_thread->wait_time.start_w.millitm));
            siglongjmp(head->jbuf, 1);
        }
        else
        {
            if( (curr_thread->exec_time) > RUN_TIME )
            {
                CleanUp();
            }

            if(sigsetjmp(curr_thread->jbuf, 1) == 1)
            {
                return;
            }
            curr_thread->thread_status = STATUS_READY;
            struct timeb start_wait_time;
            ftime(&start_wait_time);

            curr_thread->wait_time.start_w = start_wait_time;
            int mod_value = weight_total - 1;
            int chosen_number = ( rand() % mod_value )+ 1;

            TCB *selected_thread = NULL;
            do
            {
                selected_thread = findTcb(chosen_number);
                chosen_number = ( rand() % mod_value ) + 1;
            }while(selected_thread->thread_status != STATUS_READY);

            curr_thread = selected_thread;
            ftime(&t_start);
            curr_thread->wait_time.stop_w = t_start;

            if(curr_thread->wait_time.stop_w.millitm != 0)
            {
                curr_thread->wait_time.total += ( 1000.0 *
(curr_thread->wait_time.stop_w.time - curr_thread->wait_time.start_w.time) +
(curr_thread->wait_time.stop_w.millitm - curr_thread->wait_time.start_w.millitm));
                curr_thread->num_waits++;
            }

            curr_thread->thread_status = STATUS_RUNNING;
            curr_thread->num_bursts++;

            siglongjmp(curr_thread->jbuf, 1);
        }
    }
}


//function represents thread f
void f(void){
    int i=0;
    while(1)
    {
        ++i;
        printf("in f (%d)\n",i);
        if (i % 3 == 0)
        {
            yieldCPU();
        }
        usleep(SECOND);
    }
}
```

```
//function represents thread g
void g( void ){
    int i = 0;
    while(1)
    {
        ++i;
        printf("in g (%d)\n",i);
        if (i % 3 == 0)
        {
            yieldCPU();
        }
        usleep(SECOND);
    }
}

int main()
{
    printf("Enter number for scheduling method\n");
    printf("1. For Round-robin scheduling\n");
    printf("2. For Lottery scheduling\n");
    scanf("%d",&current_schedule);
    if(current_schedule == 1){
        printf("Running round-robin scheduling\n");
    } else if(current_schedule == 2){
        printf("Running lottery scheduling\n");
    } else {
        current_schedule = 1;
        printf("Running round-robin scheduling\n");
    }

    go();
    return 0;
}
```