seqgen2.c

```
/* ========================================================================== */
/*                                                                            */
/*   seqgen2x.c                                                               */
// Sam Siewert, December 2017
//
// Sequencer Generic @ 2x Rate for 10Hz Capture
//
// The purpose of this code is to provide an example for how to best
// sequence a set of periodic services for problems similar to and including
// the final project in real-time systems.
//
// For example: Service_1 for camera frame aquisition
//              Service_2 for image analysis and timestamping
//              Service_3 for image processing (difference images)
//              Service_4 for save time-stamped image to file service
//              Service_5 for save processed image to file service
//              Service_6 for send image to remote server to save copy
//              Service_7 for elapsed time in syslog each minute for debug
//
// At least two of the services need to be real-time and need to run on a single
// core or run without affinity on the SMP cores available to the Linux
// scheduler as a group.  All services can be real-time, but you could choose
// to make just the first 2 real-time and the others best effort.
//
// For the standard project, to time-stamp images at the 1 Hz rate with unique
// clock images (unique second hand / seconds) per image, you might use the
// following rates for each service:
//
// Sequencer - 60 Hz
//                   [gives semaphores to all other services]
// Service_1 - 30 Hz, every other Sequencer loop
//                   [buffers 3 images per second]
// Service_2 - 10 Hz, every 6th Sequencer loop
//                   [time-stamp middle sample image with cvPutText or header]
// Service_3 - 5 Hz , every 12th Sequencer loop
//                   [difference current and previous time stamped images]
// Service_4 - 10 Hz, every 6th Sequencer loop
//                   [save time stamped image with cvSaveImage or write()]
// Service_5 - 5 Hz , every 12th Sequencer loop
//                   [save difference image with cvSaveImage or write()]
// Service_6 - 10 Hz, every 6th Sequencer loop
//                   [write current time-stamped image to TCP socket server]
// Service_7 - 1 Hz , every 60th Sequencer loop
//                   [syslog the time for debug]
//
// With the above, priorities by RM policy would be:
//
// Sequencer = RT_MAX   @ 60 Hz
// Servcie_1 = RT_MAX-1 @ 30 Hz
// Service_2 = RT_MAX-2 @ 10 Hz
// Service_3 = RT_MAX-3 @ 5  Hz
// Service_4 = RT_MAX-2 @ 10 Hz
// Service_5 = RT_MAX-3 @ 5  Hz
```

```c
// Service_6 = RT_MAX-2 @ 10 Hz
// Service_7 = RT_MIN   @ 1  Hz
//
// Here are a few hardware/platform configuration settings on your Jetson
// that you should also check before running this code:
//
// 1) Check to ensure all your CPU cores on in an online state.
//
// 2) Check /sys/devices/system/cpu or do lscpu.
//
//    Tegra is normally configured to hot-plug CPU cores, so to make all
//    available, as root do:
//
//    echo 0 > /sys/devices/system/cpu/cpuquiet/tegra_cpuquiet/enable
//    echo 1 > /sys/devices/system/cpu/cpu1/online
//    echo 1 > /sys/devices/system/cpu/cpu2/online
//    echo 1 > /sys/devices/system/cpu/cpu3/online
//
// 3) Check for precision time resolution and support with cat /proc/timer_list
//
// 4) Ideally all printf calls should be eliminated as they can interfere with
//    timing.  They should be replaced with an in-memory event logger or at
//    least calls to syslog.
//
// 5) For simplicity, you can just allow Linux to dynamically load balance
//    threads to CPU cores (not set affinity) and as long as you have more
//    threads than you have cores, this is still an over-subscribed system
//    where RM policy is required over the set of cores.

// This is necessary for CPU affinity macros in Linux
#define _GNU_SOURCE

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

#include <pthread.h>
#include <sched.h>
#include <time.h>
#include <semaphore.h>

#include <syslog.h>
#include <sys/time.h>

#include <errno.h>

#define USEC_PER_MSEC (1000)
#define MS_PER_SEC (1000)
#define NANOSEC_PER_SEC (1000000000)
#define NUM_CPU_CORES (1)
#define TRUE (1)
#define FALSE (0)
#define ITERATION_COUNT 53800 // 10 ms load

#define NUM_THREADS (7 + 1)

```

```c
int abortTest = FALSE;
int abortS1 = FALSE, abortS2 = FALSE, abortS3 = FALSE, abortS4 = FALSE, abortS5 = FALSE, abortS6 = FALSE, abortS7 = FALSE;
sem_t semS1, semS2, semS3, semS4, semS5, semS6, semS7;
struct timeval start_time_val;

double wcet[7];
double execution_time[7];
int execution_cycle[7];

typedef struct
{
    int threadIdx;
    unsigned long long sequencePeriods;
} threadParams_t;

void *Sequencer(void *threadp);

void *Service_1(void *threadp);
void *Service_2(void *threadp);
void *Service_3(void *threadp);
void *Service_4(void *threadp);
void *Service_5(void *threadp);
void *Service_6(void *threadp);
void *Service_7(void *threadp);
double getTimeMsec(void);
void print_scheduler(void);

#define FIB_LIMIT_FOR_32_BIT 47
#define ITERATION_COUNT_FIB 15000

void fibTest(int interation_count)
{
    int fib, fib0, fib1;
    int jdx = 0;
    for (int idx = 0; idx < interation_count; idx++)
    {
        fib = fib0 + fib1;
        while (jdx < FIB_LIMIT_FOR_32_BIT)
        {
            fib0 = fib1;
            fib1 = fib;
            fib = fib0 + fib1;
            jdx++;
        }
        jdx = 0;
    }
}


void print_data(){
    for(int i=0; i<7; i++){
        syslog(LOG_CRIT, "**** Task %d):  WCET: %f, total_execution time : %f, execution cycles : %d, average execution time : %f **** \n ", i+1, wcet[i], execution_time[i], execution_cycle[i], execution_time[i]/execution_cycle[i]);
        printf("**** Task %d):  WCET: %f, total_execution time : %f, execution cycles : %d, average execution time : %f **** \n ", i+1, wcet[i], execution_time[i],
execution_cycle[i], execution_time[i]/execution_cycle[i]);
    }

}

double read_time(double *var)
{
    struct timeval tv;
    if (gettimeofday(&tv, NULL) != 0)
    {
        perror("readTOD");
        return 0.0;
    }
    else
    {
        *var = ((double)(((double)tv.tv_sec * 1000) + (((double)tv.tv_usec) / 1000.0)));
    }
    return (*var);
}


void main(void)
{
    struct timeval current_time_val;
    int i, rc, scope;
    cpu_set_t threadcpu;
    pthread_t threads[NUM_THREADS];
    threadParams_t threadParams[NUM_THREADS];
    pthread_attr_t rt_sched_attr[NUM_THREADS];
    int rt_max_prio, rt_min_prio;
    struct sched_param rt_param[NUM_THREADS];
    struct sched_param main_param;
    pthread_attr_t main_attr;
    pid_t mainpid;
    cpu_set_t allcpuset;

    printf("Starting Sequencer Demo\n");
    syslog(LOG_CRIT, "Starting Sequencer Demo\n");

    printf("testing Fib load with iterations :%d\n", ITERATION_COUNT);
    double avg_time = 0;
    for(int i=0;i<10;i++){
        double start, end;
        read_time(&start);
        fibTest(ITERATION_COUNT);
        read_time(&end);
        double total_ex = end - start;
        avg_time += total_ex;
        printf("iteration %d) Start time: %f ms , end time: %f ms , execution time: %f ms\n\n",i, start, end, total_ex);
        syslog(LOG_CRIT, "iteration %d) Start time: %f ms , end time: %f ms , execution time: %f ms\n\n",i, start, end, total_ex);

    }
    printf("***** Average time %f *****\n", avg_time / 10);
    syslog(LOG_CRIT, "***** Average time %f *****\n", avg_time / 10);


    gettimeofday(&start_time_val, (struct timezone *)0);
    gettimeofday(&current_time_val, (struct timezone *)0);
    syslog(LOG_CRIT, "Sequencer @ sec=%d, msec=%d\n", (int)(current_time_val.tv_sec - start_time_val.tv_sec), (int)current_time_val.tv_usec / USEC_PER_MSEC);
    printf("Sequencer @ sec=%d, msec=%d\n", (int)(current_time_val.tv_sec - start_time_val.tv_sec), (int)current_time_val.tv_usec / USEC_PER_MSEC);

    printf("System has %d processors configured and %d available.\n", get_nprocs_conf(), get_nprocs());
    syslog(LOG_CRIT, "System has %d processors configured and %d available.\n", get_nprocs_conf(), get_nprocs());

    CPU_ZERO(&allcpuset);

    for (i = 0; i < NUM_CPU_CORES; i++)
        CPU_SET(i, &allcpuset);

    printf("Using CPUS=%d from total available.\n", CPU_COUNT(&allcpuset));

    // initialize the sequencer semaphores
    //
    if (sem_init(&semS1, 0, 0))
    {
        printf("Failed to initialize S1 semaphore\n");
        exit(-1);
    }
    if (sem_init(&semS2, 0, 0))
    {
        printf("Failed to initialize S2 semaphore\n");
        exit(-1);
    }
    if (sem_init(&semS3, 0, 0))
    {
        printf("Failed to initialize S3 semaphore\n");
        exit(-1);
    }
    if (sem_init(&semS4, 0, 0))
    {
        printf("Failed to initialize S4 semaphore\n");
        exit(-1);
    }
    if (sem_init(&semS5, 0, 0))
    {
        printf("Failed to initialize S5 semaphore\n");
        exit(-1);
    }
    if (sem_init(&semS6, 0, 0))
    {
        printf("Failed to initialize S6 semaphore\n");
        exit(-1);
    }
    if (sem_init(&semS7, 0, 0))
    {
        printf("Failed to initialize S7 semaphore\n");
        exit(-1);
    }

    mainpid = getpid();

    rt_max_prio = sched_get_priority_max(SCHED_FIFO);
    rt_min_prio = sched_get_priority_min(SCHED_FIFO);

    rc = sched_getparam(mainpid, &main_param);
    main_param.sched_priority = rt_max_prio;
    rc = sched_setscheduler(getpid(), SCHED_FIFO, &main_param);
    if (rc < 0)
        perror("main_param");
    print_scheduler();

    pthread_attr_getscope(&main_attr, &scope);

    if (scope == PTHREAD_SCOPE_SYSTEM)
        printf("PTHREAD SCOPE SYSTEM\n");
    else if (scope == PTHREAD_SCOPE_PROCESS)
        printf("PTHREAD SCOPE PROCESS\n");
    else
        printf("PTHREAD SCOPE UNKNOWN\n");

    printf("rt_max_prio=%d\n", rt_max_prio);
    printf("rt_min_prio=%d\n", rt_min_prio);

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

        CPU_ZERO(&threadcpu);
        CPU_SET(3, &threadcpu);

        rc = pthread_attr_init(&rt_sched_attr[i]);
        rc = pthread_attr_setinheritsched(&rt_sched_attr[i], PTHREAD_EXPLICIT_SCHED);
        rc = pthread_attr_setschedpolicy(&rt_sched_attr[i], SCHED_FIFO);
        rc=pthread_attr_setaffinity_np(&rt_sched_attr[i], sizeof(cpu_set_t), &threadcpu);

        rt_param[i].sched_priority = rt_max_prio - i;
        pthread_attr_setschedparam(&rt_sched_attr[i], &rt_param[i]);

        threadParams[i].threadIdx = i;
    }

    printf("Service threads will run on %d CPU cores\n", CPU_COUNT(&threadcpu));
    syslog(LOG_CRIT, "Service threads will run on %d CPU cores\n", CPU_COUNT(&threadcpu));

    // Create Service threads which will block awaiting release for:
    //

    // Servcie_1 = RT_MAX-1 @ 3 Hz
    //
    rt_param[1].sched_priority = rt_max_prio - 1;
    pthread_attr_setschedparam(&rt_sched_attr[1], &rt_param[1]);
    rc = pthread_create(&threads[1],       // pointer to thread descriptor
                        &rt_sched_attr[1], // use specific attributes
                        //(void *)0,               // default attributes
                        Service_1,                 // thread function entry point
                        (void *)&(threadParams[1]) // parameters to pass in
    );
    if (rc < 0)
        perror("pthread_create for service 1");
    else{
        printf("pthread_create successful for service 1\n");
        syslog(LOG_CRIT, "pthread_create successful for service 1\n");
    }

    // Service_2 = RT_MAX-2 @ 1 Hz
    //
    rt_param[2].sched_priority = rt_max_prio - 2;
    pthread_attr_setschedparam(&rt_sched_attr[2], &rt_param[2]);
    rc = pthread_create(&threads[2], &rt_sched_attr[2], Service_2, (void *)&(threadParams[2]));
    if (rc < 0)
        perror("pthread_create for service 2");
    else{
        printf("pthread_create successful for service 2\n");
        syslog(LOG_CRIT, "pthread_create successful for service 2\n");
    }

    // Service_3 = RT_MAX-3 @ 0.5 Hz
    //
    rt_param[3].sched_priority = rt_max_prio - 3;
    pthread_attr_setschedparam(&rt_sched_attr[3], &rt_param[3]);
    rc = pthread_create(&threads[3], &rt_sched_attr[3], Service_3, (void *)&(threadParams[3]));
    if (rc < 0)
        perror("pthread_create for service 3");
    else{
        printf("pthread_create successful for service 3\n");
        syslog(LOG_CRIT, "pthread_create successful for service 3\n");
    }

    // Service_4 = RT_MAX-2 @ 1 Hz
    //
    rt_param[4].sched_priority = rt_max_prio - 2;
    pthread_attr_setschedparam(&rt_sched_attr[4], &rt_param[4]);
    rc = pthread_create(&threads[4], &rt_sched_attr[4], Service_4, (void *)&(threadParams[4]));
    if (rc < 0)
        perror("pthread_create for service 4");
    else{
        printf("pthread_create successful for service 4\n");
        syslog(LOG_CRIT, "pthread_create successful for service 4\n");
    }

    // Service_5 = RT_MAX-3 @ 0.5 Hz
    //
    rt_param[5].sched_priority = rt_max_prio - 3;
    pthread_attr_setschedparam(&rt_sched_attr[5], &rt_param[5]);
    rc = pthread_create(&threads[5], &rt_sched_attr[5], Service_5, (void *)&(threadParams[5]));
    if (rc < 0)
```

```
        perror("pthread_create for service 5");
    else{

        printf("pthread_create successful for service 5\n");
        syslog(LOG_CRIT, "pthread_create successful for service 5\n");
    }

    // Service_6 = RT_MAX-2 @ 1 Hz
    //
    rt_param[6].sched_priority = rt_max_prio - 2;
    pthread_attr_setschedparam(&rt_sched_attr[6], &rt_param[6]);
    rc = pthread_create(&threads[6], &rt_sched_attr[6], Service_6, (void *)&(threadParams[6]));
    if (rc < 0)
        perror("pthread_create for service 6");
    else{

        syslog(LOG_CRIT, "pthread_create successful for service 6\n");
    }

    // Service_7 = RT_MIN   0.1 Hz
    //
    rt_param[7].sched_priority = rt_min_prio;
    pthread_attr_setschedparam(&rt_sched_attr[7], &rt_param[7]);
    rc = pthread_create(&threads[7], &rt_sched_attr[7], Service_7, (void *)&(threadParams[7]));
    if (rc < 0)
        perror("pthread_create for service 7");
    else{

        printf("pthread_create successful for service 7\n");
        syslog(LOG_CRIT, "pthread_create successful for service 7\n");
    }

    // Wait for service threads to initialize and await release by sequencer.
    //
    // Note that the sleep is not necessary of RT service threads are created wtih
    // correct POSIX SCHED_FIFO priorities compared to non-RT priority of this main
    // program.
    //
    // usleep(1000000);

    // Create Sequencer thread, which like a cyclic executive, is highest prio
    printf("Start sequencer\n");
    syslog(LOG_CRIT, "Start sequencer\n");
    threadParams[0].sequencePeriods=900;

    // Sequencer = RT_MAX   @ 30 Hz
    //
    rt_param[0].sched_priority = rt_max_prio;
    pthread_attr_setschedparam(&rt_sched_attr[0], &rt_param[0]);
    rc = pthread_create(&threads[0], &rt_sched_attr[0], Sequencer, (void *)&(threadParams[0]));
    if (rc < 0)
        perror("pthread_create for sequencer service 0");
    else{

```

```c
        printf("pthread_create successful for sequeencer service 0\n");
        syslog(LOG_CRIT, "pthread_create successful for sequeencer service 0\n");
    }

    for (i = 0; i < NUM_THREADS; i++)
        pthread_join(threads[i], NULL);

    printf("\nTEST COMPLETE\n");
    syslog(LOG_CRIT, "\nTEST COMPLETE\n");
}

void *Sequencer(void *threadp)
{
    struct timeval current_time_val;
    struct timespec delay_time = {0,16666666}; // delay for 16.67 msec, 60 Hz
    struct timespec remaining_time;
    double current_time;
    double residual;
    int rc, delay_cnt=0;
    unsigned long long seqCnt=0;
    threadParams_t *threadParams = (threadParams_t *)threadp;

    gettimeofday(&current_time_val, (struct timezone *)0);
    syslog(LOG_CRIT, "Sequencer thread @ sec=%d, msec=%d\n", (int)(current_time_val.tv_sec - start_time_val.tv_sec), (int)current_time_val.tv_usec / USEC_PER_MSEC);
    printf("Sequencer thread @ sec=%d, msec=%d\n", (int)(current_time_val.tv_sec - start_time_val.tv_sec), (int)current_time_val.tv_usec / USEC_PER_MSEC);

    do
    {
        delay_cnt = 0;
        residual = 0.0;

        gettimeofday(&current_time_val, (struct timezone *)0);
        syslog(LOG_CRIT, "Sequencer thread prior to delay @ sec=%d, msec=%d\n", (int)(current_time_val.tv_sec - start_time_val.tv_sec), (int)current_time_val.tv_usec / USEC_PER_MSEC);

        do
        {
            rc = nanosleep(&delay_time, &remaining_time);

            if (rc == EINTR)
            {
                residual = remaining_time.tv_sec + ((double)remaining_time.tv_nsec / (double)NANOSEC_PER_SEC);

                if (residual > 0.0)
                    printf("residual=%lf, sec=%d, nsec=%d\n", residual, (int)remaining_time.tv_sec, (int)remaining_time.tv_nsec);

                delay_cnt++;
            }
            else if (rc < 0)
            {
                perror("Sequencer nanosleep");
                exit(-1);
            }

        } while ((residual > 0.0) && (delay_cnt < 100));

        seqCnt++;
        gettimeofday(&current_time_val, (struct timezone *)0);
        syslog(LOG_CRIT, "Sequencer cycle %llu @ sec=%d, msec=%d\n", seqCnt, (int)(current_time_val.tv_sec - start_time_val.tv_sec), (int)current_time_val.tv_usec / USEC_PER_MSEC);

        if (delay_cnt > 1)
            printf("Sequencer looping delay %d\n", delay_cnt);

        // Release each service at a sub-rate of the generic sequencer rate

        // Servcie_1 = RT_MAX-1 @ 30 Hz
        if((seqCnt % 2) == 0)
        {
            syslog(LOG_CRIT, "Task 1 (Frame Sampler thread) Released \n");
            sem_post(&semS1); // Frame Sampler thread
        }

        // Service_2 = RT_MAX-2 @ 10 Hz
        if((seqCnt % 6) == 0)
        {
            syslog(LOG_CRIT, "Task 2 (Time-stamp with Image Analysis thread) Released \n");
            sem_post(&semS2); // Time-stamp with Image Analysis thread
        }

        // Service_3 = RT_MAX-3 @ 5 Hz
        if((seqCnt % 12) == 0)
        {
            syslog(LOG_CRIT, "Task 3 ( Difference Image Proc thread) Released \n");
            sem_post(&semS3); // Difference Image Proc thread
        }

        // Service_4 = RT_MAX-2 @ 10 Hz
        if((seqCnt % 6) == 0)
        {
            syslog(LOG_CRIT, "Task 4 (Time-stamp Image Save to File thread) Released \n");
            sem_post(&semS4); // Time-stamp Image Save to File thread
        }

        // Service_5 = RT_MAX-3 @ 5 Hz
        if((seqCnt % 12) == 0)
        {
            syslog(LOG_CRIT, "Task 5 (Processed Image Save to File thread) Released \n");
            sem_post(&semS5); // Processed Image Save to File thread
        }

        // Service_6 = RT_MAX-2 @ 10 Hz
        if((seqCnt % 6) == 0)
        {
            syslog(LOG_CRIT, "Task 6 (Send Time-stamped Image to Remote thread)
```

```c
Released \n");
            sem_post(&semS6); // Send Time-stamped Image to Remote thread
        }

        // Service_7 = RT_MIN   1 Hz
        if((seqCnt % 60) == 0)
        {
            syslog(LOG_CRIT, "Task 7 (10 sec Tick Debug thread) Released \n");
            sem_post(&semS7); // 10 sec Tick Debug thread
        }

        gettimeofday(&current_time_val, NULL);
        syslog(LOG_CRIT, "Sequencer release all sub-services @ sec=%d, msec=%d\n", (int)(current_time_val.tv_sec - start_time_val.tv_sec), (int)current_time_val.tv_usec / USEC_PER_MSEC);

    } while (!abortTest && (seqCnt < threadParams->sequencePeriods));

    sem_post(&semS1);
    sem_post(&semS2);
    sem_post(&semS3);
    sem_post(&semS4);
    sem_post(&semS5);
    sem_post(&semS6);
    sem_post(&semS7);
    abortS1 = TRUE;
    abortS2 = TRUE;
    abortS3 = TRUE;
    abortS4 = TRUE;
    abortS5 = TRUE;
    abortS6 = TRUE;
    abortS7 = TRUE;
    print_data();

    pthread_exit((void *)0);
}

void *Service_1(void *threadp)
{
    double start, end, total;
    threadParams_t *threadParams = (threadParams_t *)threadp;

    read_time(&start);
    syslog(LOG_CRIT, "Task 1, Frame Sampler thread @ msec=%f \n", start);
    printf("Task 1, Frame Sampler thread @ msec=%f \n", start);

    while (!abortS1)
    {
        sem_wait(&semS1);

        execution_cycle[0]++;
        read_time(&start);
        syslog(LOG_CRIT, "Task 1, Frame Sampler start %d @ msec=%f", execution_cycle[0], start);
        fibTest(ITERATION_COUNT);
        read_time(&end);
        total = end - start;
        if(total > wcet[0]) wcet[0] = total;
        execution_time[0] += total;
        syslog(LOG_CRIT, "Task 1, Frame Sampler Execution complete @ msec=%f, execution time : %f ms\n", end, total);
    }

    pthread_exit((void *)0);
}

void *Service_2(void *threadp)
{

    double start, end, total;
    threadParams_t *threadParams = (threadParams_t *)threadp;

    read_time(&start);
    syslog(LOG_CRIT, "Task 2, Time-stamp with Image Analysis thread @ msec=%f \n", start);
    printf("Task 2, Time-stamp with Image Analysis thread @ msec=%f \n", start);

    while (!abortS2)
    {
        sem_wait(&semS2);

        execution_cycle[1]++;
        read_time(&start);
        syslog(LOG_CRIT, "Task 2, Time-stamp with Image Analysis thread start %d @ msec=%f", execution_cycle[1], start);
        fibTest(ITERATION_COUNT);
        read_time(&end);
        total = end - start;
        if(total > wcet[1]) wcet[1] = total;
        execution_time[1] += total;
        syslog(LOG_CRIT, "Task 2, Time-stamp with Image Analysis thread Execution complete @ msec=%f, execution time : %f ms\n", end, total);
    }

    pthread_exit((void *)0);

}

void *Service_3(void *threadp)
{

    double start, end, total;
    threadParams_t *threadParams = (threadParams_t *)threadp;

    read_time(&start);
    syslog(LOG_CRIT, "Task 3, Difference Image Proc thread @ msec=%f \n", start);
    printf("Task 3, Difference Image Proc thread @ msec=%f \n", start);

    while (!abortS3)
    {
        sem_wait(&semS3);

        execution_cycle[2]++;
        read_time(&start);
        syslog(LOG_CRIT, "Task 3, Difference Image Proc  start %d @ msec=%f", execution_cycle[2], start);
        fibTest(ITERATION_COUNT);
        read_time(&end);
        total = end - start;
        if(total > wcet[2]) wcet[2] = total;
        execution_time[2] += total;
        syslog(LOG_CRIT, "Task 3, Difference Image Proc Execution complete @ msec=%f, execution time : %f ms\n", end, total);
    }

    pthread_exit((void *)0);
}

void *Service_4(void *threadp)
{

    double start, end, total;
    threadParams_t *threadParams = (threadParams_t *)threadp;

    read_time(&start);
    syslog(LOG_CRIT, "Task 4, Time-stamp Image Save to File thread @ msec=%f \n", start);
    printf("Task 4, Time-stamp Image Save to File thread @ msec=%f \n", start);

    while (!abortS4)
    {
        sem_wait(&semS4);

        execution_cycle[3]++;
        read_time(&start);
        syslog(LOG_CRIT, "Task 4, Time-stamp Image Save to File start %d @ msec=%f", execution_cycle[3], start);
        fibTest(ITERATION_COUNT);
        read_time(&end);
        total = end - start;
        if(total > wcet[3]) wcet[3] = total;
        execution_time[3] += total;
        syslog(LOG_CRIT, "Task 4, Time-stamp Image Save to File Execution complete @ msec=%f, execution time : %f ms\n", end, total);
    }

    pthread_exit((void *)0);

}

void *Service_5(void *threadp)
{

    double start, end, total;
    threadParams_t *threadParams = (threadParams_t *)threadp;

    read_time(&start);
    syslog(LOG_CRIT, "Task 5, Processed Image Save to File thread @ msec=%f \n", start);
    printf("Task 5, Processed Image Save to File thread @ msec=%f \n", start);

    while (!abortS5)
    {
        sem_wait(&semS5);

        execution_cycle[4]++;
        read_time(&start);
        syslog(LOG_CRIT, "Task 5, Processed Image Save to File start %d @ msec=%f", execution_cycle[4], start);
        fibTest(ITERATION_COUNT);
        read_time(&end);
        total = end - start;
        if(total > wcet[4]) wcet[4] = total;
        execution_time[4] += total;
        syslog(LOG_CRIT, "Task 5, Processed Image Save to File Execution complete @ msec=%f, execution time : %f ms\n", end, total);
    }

    pthread_exit((void *)0);


}

void *Service_6(void *threadp)
{

    double start, end, total;
    threadParams_t *threadParams = (threadParams_t *)threadp;

    read_time(&start);
    syslog(LOG_CRIT, "Task 6, Send Time-stamped Image to Remote thread @ msec=%f \n", start);
    printf("Task 6, Send Time-stamped Image to Remote thread @ msec=%f \n", start);

    while (!abortS6)
    {
        sem_wait(&semS6);

        execution_cycle[5]++;
        read_time(&start);
        syslog(LOG_CRIT, "Task 6, Send Time-stamped Image to Remote start %d @ msec=%f", execution_cycle[5], start);
        fibTest(ITERATION_COUNT);
        read_time(&end);
        total = end - start;
        if(total > wcet[5]) wcet[5] = total;
        execution_time[5] += total;
        syslog(LOG_CRIT, "Task 6, Send Time-stamped Image to Remote Execution complete @ msec=%f, execution time : %f ms\n", end, total);
    }

    pthread_exit((void *)0);

}

void *Service_7(void *threadp)
{
    double start, end, total;
    threadParams_t *threadParams = (threadParams_t *)threadp;

    read_time(&start);
    syslog(LOG_CRIT, "Task 7, 10 sec Tick Debug thread @ msec=%f \n", start);
    printf("Task 7, 10 sec Tick Debug Thread @ msec=%f \n", start);

    while (!abortS7)
    {
        sem_wait(&semS7);

        execution_cycle[6]++;
        read_time(&start);
        syslog(LOG_CRIT, "Task 7, 10 sec Tick Debug start %d @ msec=%f", execution_cycle[6], start);
        fibTest(ITERATION_COUNT);
        read_time(&end);
        total = end - start;
        if(total > wcet[6]) wcet[6] = total;
        execution_time[6] += total;
        syslog(LOG_CRIT, "Task 7, 10 sec Tick Debug Execution complete @ msec=%f, execution time : %f ms\n", end, total);
    }

    pthread_exit((void *)0);
}

double getTimeMsec(void)
{
    struct timespec event_ts = {0, 0};

    clock_gettime(CLOCK_MONOTONIC, &event_ts);
    return ((event_ts.tv_sec) * 1000.0) + ((event_ts.tv_nsec) / 1000000.0);
}

void print_scheduler(void)
{
    int schedType;

    schedType = sched_getscheduler(getpid());

    switch (schedType)
    {
    case SCHED_FIFO:
        printf("Pthread Policy is SCHED_FIFO\n");
        break;
    case SCHED_OTHER:
        printf("Pthread Policy is SCHED_OTHER\n");
        exit(-1);
        break;
    case SCHED_RR:
        printf("Pthread Policy is SCHED_RR\n");
        exit(-1);
        break;
    default:
        printf("Pthread Policy is UNKNOWN\n");
        exit(-1);
    }
```

```c
}

```