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
```

```
// 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)

```

```
110 int abortTest = FALSE;
111 int abortS1 = FALSE, abortS2 = FALSE, abortS3 = FALSE, abortS4 = FALSE, abortS5 =
    FALSE, abortS6 = FALSE, abortS7 = FALSE;
112 sem_t semS1, semS2, semS3, semS4, semS5, semS6, semS7;
113 struct timeval start_time_val;
114
115 double wcet[7];
116 double execution_time[7];
117 int execution_cycle[7];
118
119 typedef struct
120 {
121     int threadIdx;
122     unsigned long long sequencePeriods;
123 } threadParams_t;
124
125 void *Sequencer(void *threadp);
126
127 void *Service_1(void *threadp);
128 void *Service_2(void *threadp);
129 void *Service_3(void *threadp);
130 void *Service_4(void *threadp);
131 void *Service_5(void *threadp);
132 void *Service_6(void *threadp);
133 void *Service_7(void *threadp);
134 double getTimeMsec(void);
135 void print_scheduler(void);
136
137 #define FIB_LIMIT_FOR_32_BIT 47
138 #define ITERATION_COUNT_FIB 15000
139
140 void fibTest(int interation_count)
141 {
142     int fib, fib0, fib1;
143     int jdx = 0;
144     for (int idx = 0; idx < interation_count; idx++)
145     {
146         fib = fib0 + fib1;
147         while (jdx < FIB_LIMIT_FOR_32_BIT)
148         {
149             fib0 = fib1;
150             fib1 = fib;
151             fib = fib0 + fib1;
152             jdx++;
153         }
154         jdx = 0;
155     }
156 }
157
158
159 void print_data(){
160     for(int i=0; i<7; i++){
161         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]);
162         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);
```

```
216
217
218     gettimeofday(&start_time_val, (struct timezone *)0);
219     gettimeofday(&current_time_val, (struct timezone *)0);
220     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);
221     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);
222
223     printf("System has %d processors configured and %d available.\n",
    get_nprocs_conf(), get_nprocs());
224     syslog(LOG_CRIT, "System has %d processors configured and %d available.\n",
    get_nprocs_conf(), get_nprocs());
225
226     CPU_ZERO(&allcpuset);
227
228     for (i = 0; i < NUM_CPU_CORES; i++)
229         CPU_SET(i, &allcpuset);
230
231     printf("Using CPUS=%d from total available.\n", CPU_COUNT(&allcpuset));
232
233     // initialize the sequencer semaphores
234     //
235     if (sem_init(&semS1, 0, 0))
236     {
237         printf("Failed to initialize S1 semaphore\n");
238         exit(-1);
239     }
240     if (sem_init(&semS2, 0, 0))
241     {
242         printf("Failed to initialize S2 semaphore\n");
243         exit(-1);
244     }
245     if (sem_init(&semS3, 0, 0))
246     {
247         printf("Failed to initialize S3 semaphore\n");
248         exit(-1);
249     }
250     if (sem_init(&semS4, 0, 0))
251     {
252         printf("Failed to initialize S4 semaphore\n");
253         exit(-1);
254     }
255     if (sem_init(&semS5, 0, 0))
256     {
257         printf("Failed to initialize S5 semaphore\n");
258         exit(-1);
259     }
260     if (sem_init(&semS6, 0, 0))
261     {
262         printf("Failed to initialize S6 semaphore\n");
263         exit(-1);
264     }
265     if (sem_init(&semS7, 0, 0))
266     {
267         printf("Failed to initialize S7 semaphore\n");
268         exit(-1);
```

```
269     }
270
271     mainpid = getpid();
272
273     rt_max_prio = sched_get_priority_max(SCHED_FIFO);
274     rt_min_prio = sched_get_priority_min(SCHED_FIFO);
275
276     rc = sched_getparam(mainpid, &main_param);
277     main_param.sched_priority = rt_max_prio;
278     rc = sched_setscheduler(getpid(), SCHED_FIFO, &main_param);
279     if (rc < 0)
280         perror("main_param");
281     print_scheduler();
282
283     pthread_attr_getscope(&main_attr, &scope);
284
285     if (scope == PTHREAD_SCOPE_SYSTEM)
286         printf("PTHREAD SCOPE SYSTEM\n");
287     else if (scope == PTHREAD_SCOPE_PROCESS)
288         printf("PTHREAD SCOPE PROCESS\n");
289     else
290         printf("PTHREAD SCOPE UNKNOWN\n");
291
292     printf("rt_max_prio=%d\n", rt_max_prio);
293     printf("rt_min_prio=%d\n", rt_min_prio);
294
295     for (i = 0; i < NUM_THREADS; i++)
296     {
297
298         CPU_ZERO(&threadcpu);
299         CPU_SET(3, &threadcpu);
300
301         rc = pthread_attr_init(&rt_sched_attr[i]);
302         rc = pthread_attr_setinheritsched(&rt_sched_attr[i], PTHREAD_EXPLICIT_SCHED);
303         rc = pthread_attr_setschedpolicy(&rt_sched_attr[i], SCHED_FIFO);
304         rc=pthread_attr_setaffinity_np(&rt_sched_attr[i], sizeof(cpu_set_t), &
    threadcpu);
305
306         rt_param[i].sched_priority = rt_max_prio - i;
307         pthread_attr_setschedparam(&rt_sched_attr[i], &rt_param[i]);
308
309         threadParams[i].threadIdx = i;
310     }
311
312     printf("Service threads will run on %d CPU cores\n", CPU_COUNT(&threadcpu));
313     syslog(LOG_CRIT, "Service threads will run on %d CPU cores\n", CPU_COUNT(&
    threadcpu));
314
315     // Create Service threads which will block awaiting release for:
316     //
317
318     // Servcie_1 = RT_MAX-1 @ 3 Hz
319     //
320     rt_param[1].sched_priority = rt_max_prio - 1;
321     pthread_attr_setschedparam(&rt_sched_attr[1], &rt_param[1]);
322     rc = pthread_create(&threads[1],       // pointer to thread descriptor
323                         &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)
```

```
377         perror("pthread_create for service 5");
378     else{
379 
380         printf("pthread_create successful for service 5\n");
381         syslog(LOG_CRIT, "pthread_create successful for service 5\n");
382     }
383 
384     // Service_6 = RT_MAX-2 @ 1 Hz
385     //
386     rt_param[6].sched_priority = rt_max_prio - 2;
387     pthread_attr_setschedparam(&rt_sched_attr[6], &rt_param[6]);
388     rc = pthread_create(&threads[6], &rt_sched_attr[6], Service_6, (void *)&
    (threadParams[6]));
389     if (rc < 0)
390         perror("pthread_create for service 6");
391     else{
392 
393         syslog(LOG_CRIT, "pthread_create successful for service 6\n");
394     }
395 
396     // Service_7 = RT_MIN   0.1 Hz
397     //
398     rt_param[7].sched_priority = rt_min_prio;
399     pthread_attr_setschedparam(&rt_sched_attr[7], &rt_param[7]);
400     rc = pthread_create(&threads[7], &rt_sched_attr[7], Service_7, (void *)&
    (threadParams[7]));
401     if (rc < 0)
402         perror("pthread_create for service 7");
403     else{
404 
405         printf("pthread_create successful for service 7\n");
406         syslog(LOG_CRIT, "pthread_create successful for service 7\n");
407     }
408 
409     // Wait for service threads to initialize and await release by sequencer.
410     //
411     // Note that the sleep is not necessary of RT service threads are created wtih
412     // correct POSIX SCHED_FIFO priorities compared to non-RT priority of this main
413     // program.
414     //
415     // usleep(1000000);
416 
417     // Create Sequencer thread, which like a cyclic executive, is highest prio
418     printf("Start sequencer\n");
419     syslog(LOG_CRIT, "Start sequencer\n");
420     threadParams[0].sequencePeriods=900;
421 
422     // Sequencer = RT_MAX   @ 30 Hz
423     //
424     rt_param[0].sched_priority = rt_max_prio;
425     pthread_attr_setschedparam(&rt_sched_attr[0], &rt_param[0]);
426     rc = pthread_create(&threads[0], &rt_sched_attr[0], Sequencer, (void *)&
    (threadParams[0]));
427     if (rc < 0)
428         perror("pthread_create for sequencer service 0");
429     else{
430 
```

```
431         printf("pthread_create successful for sequeencer service 0\n");
432         syslog(LOG_CRIT, "pthread_create successful for sequeencer service 0\n");
433     }
434 
435     for (i = 0; i < NUM_THREADS; i++)
436         pthread_join(threads[i], NULL);
437 
438     printf("\nTEST COMPLETE\n");
439     syslog(LOG_CRIT, "\nTEST COMPLETE\n");
440 }
441 
442 void *Sequencer(void *threadp)
443 {
444     struct timeval current_time_val;
445     struct timespec delay_time = {0,16666666}; // delay for 16.67 msec, 60 Hz
446     struct timespec remaining_time;
447     double current_time;
448     double residual;
449     int rc, delay_cnt=0;
450     unsigned long long seqCnt=0;
451     threadParams_t *threadParams = (threadParams_t *)threadp;
452 
453     gettimeofday(&current_time_val, (struct timezone *)0);
454     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);
455     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);
456 
457     do
458     {
459         delay_cnt = 0;
460         residual = 0.0;
461 
462         gettimeofday(&current_time_val, (struct timezone *)0);
463         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);
464 
465         do
466         {
467             rc = nanosleep(&delay_time, &remaining_time);
468 
469             if (rc == EINTR)
470             {
471                 residual = remaining_time.tv_sec + ((double)remaining_time.tv_nsec /
(double)NANOSEC_PER_SEC);
472 
473                 if (residual > 0.0)
474                     printf("residual=%lf, sec=%d, nsec=%d\n", residual, (int)
remaining_time.tv_sec, (int)remaining_time.tv_nsec);
475 
476                 delay_cnt++;
477             }
478             else if (rc < 0)
479             {
480                 perror("Sequencer nanosleep");
481                 exit(-1);
```

```
482             }
483
484         } while ((residual > 0.0) && (delay_cnt < 100));
485
486         seqCnt++;
487         gettimeofday(&current_time_val, (struct timezone *)0);
488         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);
489
490         if (delay_cnt > 1)
491             printf("Sequencer looping delay %d\n", delay_cnt);
492
493         // Release each service at a sub-rate of the generic sequencer rate
494
495         // Servcie_1 = RT_MAX-1 @ 30 Hz
496         if((seqCnt % 2) == 0)
497         {
498             syslog(LOG_CRIT, "Task 1 (Frame Sampler thread) Released \n");
499             sem_post(&semS1); // Frame Sampler thread
500         }
501
502         // Service_2 = RT_MAX-2 @ 10 Hz
503         if((seqCnt % 6) == 0)
504         {
505             syslog(LOG_CRIT, "Task 2 (Time-stamp with Image Analysis thread) Released
    \n");
506             sem_post(&semS2); // Time-stamp with Image Analysis thread
507         }
508
509         // Service_3 = RT_MAX-3 @ 5 Hz
510         if((seqCnt % 12) == 0)
511         {
512             syslog(LOG_CRIT, "Task 3 ( Difference Image Proc thread) Released \n");
513             sem_post(&semS3); // Difference Image Proc thread
514         }
515
516         // Service_4 = RT_MAX-2 @ 10 Hz
517         if((seqCnt % 6) == 0)
518         {
519             syslog(LOG_CRIT, "Task 4 (Time-stamp Image Save to File thread) Released
    \n");
520             sem_post(&semS4); // Time-stamp Image Save to File thread
521         }
522
523         // Service_5 = RT_MAX-3 @ 5 Hz
524         if((seqCnt % 12) == 0)
525         {
526             syslog(LOG_CRIT, "Task 5 (Processed Image Save to File thread) Released
    \n");
527             sem_post(&semS5); // Processed Image Save to File thread
528         }
529
530         // Service_6 = RT_MAX-2 @ 10 Hz
531         if((seqCnt % 6) == 0)
532         {
533             syslog(LOG_CRIT, "Task 6 (Send Time-stamped Image to Remote thread)
```

```
Released \n");
534             sem_post(&semS6); // Send Time-stamped Image to Remote thread
535         }
536
537         // Service_7 = RT_MIN   1 Hz
538         if((seqCnt % 60) == 0)
539         {
540             syslog(LOG_CRIT, "Task 7 (10 sec Tick Debug thread) Released \n");
541             sem_post(&semS7); // 10 sec Tick Debug thread
542         }
543
544         gettimeofday(&current_time_val, NULL);
545         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);
546
547     } while (!abortTest && (seqCnt < threadParams->sequencePeriods));
548
549     sem_post(&semS1);
550     sem_post(&semS2);
551     sem_post(&semS3);
552     sem_post(&semS4);
553     sem_post(&semS5);
554     sem_post(&semS6);
555     sem_post(&semS7);
556     abortS1 = TRUE;
557     abortS2 = TRUE;
558     abortS3 = TRUE;
559     abortS4 = TRUE;
560     abortS5 = TRUE;
561     abortS6 = TRUE;
562     abortS7 = TRUE;
563     print_data();
564
565     pthread_exit((void *)0);
566 }
567
568 void *Service_1(void *threadp)
569 {
570     double start, end, total;
571     threadParams_t *threadParams = (threadParams_t *)threadp;
572
573     read_time(&start);
574     syslog(LOG_CRIT, "Task 1, Frame Sampler thread @ msec=%f \n", start);
575     printf("Task 1, Frame Sampler thread @ msec=%f \n", start);
576
577     while (!abortS1)
578     {
579         sem_wait(&semS1);
580
581         execution_cycle[0]++;
582         read_time(&start);
583         syslog(LOG_CRIT, "Task 1, Frame Sampler start %d @ msec=%f",
execution_cycle[0], start);
584         fibTest(ITERATION_COUNT);
585         read_time(&end);
586         total = end - start;
```

```
587         if(total > wcet[0]) wcet[0] = total;
588         execution_time[0] += total;
589         syslog(LOG_CRIT, "Task 1, Frame Sampler Execution complete @ msec=%f,
    execution time : %f ms\n", end, total);
590     }
591
592     pthread_exit((void *)0);
593 }
594
595 void *Service_2(void *threadp)
596 {
597
598     double start, end, total;
599     threadParams_t *threadParams = (threadParams_t *)threadp;
600
601     read_time(&start);
602     syslog(LOG_CRIT, "Task 2, Time-stamp with Image Analysis thread @ msec=%f \n",
    start);
603     printf("Task 2, Time-stamp with Image Analysis thread @ msec=%f \n", start);
604
605     while (!abortS2)
606     {
607         sem_wait(&semS2);
608
609         execution_cycle[1]++;
610         read_time(&start);
611         syslog(LOG_CRIT, "Task 2, Time-stamp with Image Analysis thread start %d @
    msec=%f", execution_cycle[1], start);
612         fibTest(ITERATION_COUNT);
613         read_time(&end);
614         total = end - start;
615         if(total > wcet[1]) wcet[1] = total;
616         execution_time[1] += total;
617         syslog(LOG_CRIT, "Task 2, Time-stamp with Image Analysis thread Execution
    complete @ msec=%f, execution time : %f ms\n", end, total);
618     }
619
620     pthread_exit((void *)0);
621
622 }
623
624 void *Service_3(void *threadp)
625 {
626
627     double start, end, total;
628     threadParams_t *threadParams = (threadParams_t *)threadp;
629
630     read_time(&start);
631     syslog(LOG_CRIT, "Task 3, Difference Image Proc thread @ msec=%f \n", start);
632     printf("Task 3, Difference Image Proc thread @ msec=%f \n", start);
633
634     while (!abortS3)
635     {
636         sem_wait(&semS3);
637
638         execution_cycle[2]++;
639         read_time(&start);
```

```
640         syslog(LOG_CRIT, "Task 3, Difference Image Proc  start %d @ msec=%f",
    execution_cycle[2], start);
641         fibTest(ITERATION_COUNT);
642         read_time(&end);
643         total = end - start;
644         if(total > wcet[2]) wcet[2] = total;
645         execution_time[2] += total;
646         syslog(LOG_CRIT, "Task 3, Difference Image Proc Execution complete @ msec=%f,
    execution time : %f ms\n", end, total);
647     }
648
649     pthread_exit((void *)0);
650 }
651
652 void *Service_4(void *threadp)
653 {
654
655     double start, end, total;
656     threadParams_t *threadParams = (threadParams_t *)threadp;
657
658     read_time(&start);
659     syslog(LOG_CRIT, "Task 4, Time-stamp Image Save to File thread @ msec=%f \n",
    start);
660     printf("Task 4, Time-stamp Image Save to File thread @ msec=%f \n", start);
661
662     while (!abortS4)
663     {
664         sem_wait(&semS4);
665
666         execution_cycle[3]++;
667         read_time(&start);
668         syslog(LOG_CRIT, "Task 4, Time-stamp Image Save to File start %d @ msec=%f",
    execution_cycle[3], start);
669         fibTest(ITERATION_COUNT);
670         read_time(&end);
671         total = end - start;
672         if(total > wcet[3]) wcet[3] = total;
673         execution_time[3] += total;
674         syslog(LOG_CRIT, "Task 4, Time-stamp Image Save to File Execution complete @
    msec=%f, execution time : %f ms\n", end, total);
675     }
676
677     pthread_exit((void *)0);
678
679 }
680
681 void *Service_5(void *threadp)
682 {
683
684     double start, end, total;
685     threadParams_t *threadParams = (threadParams_t *)threadp;
686
687     read_time(&start);
688     syslog(LOG_CRIT, "Task 5, Processed Image Save to File thread @ msec=%f \n",
    start);
689     printf("Task 5, Processed Image Save to File thread @ msec=%f \n", start);
690
```

```
691     while (!abortS5)
692     {
693         sem_wait(&semS5);
694 
695         execution_cycle[4]++;
696         read_time(&start);
697         syslog(LOG_CRIT, "Task 5, Processed Image Save to File start %d @ msec=%f",
    execution_cycle[4], start);
698         fibTest(ITERATION_COUNT);
699         read_time(&end);
700         total = end - start;
701         if(total > wcet[4]) wcet[4] = total;
702         execution_time[4] += total;
703         syslog(LOG_CRIT, "Task 5, Processed Image Save to File Execution complete @
    msec=%f, execution time : %f ms\n", end, total);
704     }
705 
706     pthread_exit((void *)0);
707 
708 
709 }
710 
711 void *Service_6(void *threadp)
712 {
713 
714     double start, end, total;
715     threadParams_t *threadParams = (threadParams_t *)threadp;
716 
717     read_time(&start);
718     syslog(LOG_CRIT, "Task 6, Send Time-stamped Image to Remote thread @ msec=%f \n",
    start);
719     printf("Task 6, Send Time-stamped Image to Remote thread @ msec=%f \n", start);
720 
721     while (!abortS6)
722     {
723         sem_wait(&semS6);
724 
725         execution_cycle[5]++;
726         read_time(&start);
727         syslog(LOG_CRIT, "Task 6, Send Time-stamped Image to Remote start %d @ msec=
    %f", execution_cycle[5], start);
728         fibTest(ITERATION_COUNT);
729         read_time(&end);
730         total = end - start;
731         if(total > wcet[5]) wcet[5] = total;
732         execution_time[5] += total;
733         syslog(LOG_CRIT, "Task 6, Send Time-stamped Image to Remote Execution
    complete @ msec=%f, execution time : %f ms\n", end, total);
734     }
735 
736     pthread_exit((void *)0);
737 
738 }
739 
740 void *Service_7(void *threadp)
741 {
742     double start, end, total;
```

```
743     threadParams_t *threadParams = (threadParams_t *)threadp;
744
745     read_time(&start);
746     syslog(LOG_CRIT, "Task 7, 10 sec Tick Debug thread @ msec=%f \n", start);
747     printf("Task 7, 10 sec Tick Debug Thread @ msec=%f \n", start);
748
749     while (!abortS7)
750     {
751         sem_wait(&semS7);
752
753         execution_cycle[6]++;
754         read_time(&start);
755         syslog(LOG_CRIT, "Task 7, 10 sec Tick Debug start %d @ msec=%f",
    execution_cycle[6], start);
756         fibTest(ITERATION_COUNT);
757         read_time(&end);
758         total = end - start;
759         if(total > wcet[6]) wcet[6] = total;
760         execution_time[6] += total;
761         syslog(LOG_CRIT, "Task 7, 10 sec Tick Debug Execution complete @ msec=%f,
    execution time : %f ms\n", end, total);
762     }
763
764     pthread_exit((void *)0);
765 }
766
767 double getTimeMsec(void)
768 {
769     struct timespec event_ts = {0, 0};
770
771     clock_gettime(CLOCK_MONOTONIC, &event_ts);
772     return ((event_ts.tv_sec) * 1000.0) + ((event_ts.tv_nsec) / 1000000.0);
773 }
774
775 void print_scheduler(void)
776 {
777     int schedType;
778
779     schedType = sched_getscheduler(getpid());
780
781     switch (schedType)
782     {
783     case SCHED_FIFO:
784         printf("Pthread Policy is SCHED_FIFO\n");
785         break;
786     case SCHED_OTHER:
787         printf("Pthread Policy is SCHED_OTHER\n");
788         exit(-1);
789         break;
790     case SCHED_RR:
791         printf("Pthread Policy is SCHED_RR\n");
792         exit(-1);
793         break;
794     default:
795         printf("Pthread Policy is UNKNOWN\n");
796         exit(-1);
797     }
```

```
798 }
799
```