seqgen.c

```
/* ========================================================================== */
/*                                                                            */
// Sam Siewert, December 2017
//
// Sequencer Generic
//
// 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 - 30 Hz
//                   [gives semaphores to all other services]
// Service_1 - 3 Hz  , every 10th Sequencer loop
//                   [buffers 3 images per second]
// Service_2 - 1 Hz  , every 30th Sequencer loop
//                   [time-stamp middle sample image with cvPutText or header]
// Service_3 - 0.5 Hz, every 60th Sequencer loop
//                   [difference current and previous time stamped images]
// Service_4 - 1 Hz, every 30th Sequencer loop
//                   [save time stamped image with cvSaveImage or write()]
// Service_5 - 0.5 Hz, every 60th Sequencer loop
//                   [save difference image with cvSaveImage or write()]
// Service_6 - 1 Hz, every 30th Sequencer loop
//                   [write current time-stamped image to TCP socket server]
// Service_7 - 0.1 Hz, every 300th Sequencer loop
//                   [syslog the time for debug]
//
// With the above, priorities by RM policy would be:
//
// Sequencer = RT_MAX   @ 30 Hz
// Servcie_1 = RT_MAX-1 @ 3 Hz
// Service_2 = RT_MAX-2 @ 1 Hz
// Service_3 = RT_MAX-3 @ 0.5 Hz
// Service_4 = RT_MAX-2 @ 1 Hz
// Service_5 = RT_MAX-3 @ 0.5 Hz
// Service_6 = RT_MAX-2 @ 1 Hz
```

```
// Service_7 = RT_MIN   0.1 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 466500 // 100 ms load

#define NUM_THREADS (7 + 1)

int abortTest = FALSE;
```

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

```
162     }
163
164 }
165
166 double read_time(double *var)
167 {
168     struct timeval tv;
169     if (gettimeofday(&tv, NULL) != 0)
170     {
171         perror("readTOD");
172         return 0.0;
173     }
174     else
175     {
176         *var = ((double)(((double)tv.tv_sec * 1000) + (((double)tv.tv_usec) / 1000.0)
    ));
177     }
178     return (*var);
179 }
180
181
182 void main(void)
183 {
184     struct timeval current_time_val;
185     int i, rc, scope;
186     cpu_set_t threadcpu;
187     pthread_t threads[NUM_THREADS];
188     threadParams_t threadParams[NUM_THREADS];
189     pthread_attr_t rt_sched_attr[NUM_THREADS];
190     int rt_max_prio, rt_min_prio;
191     struct sched_param rt_param[NUM_THREADS];
192     struct sched_param main_param;
193     pthread_attr_t main_attr;
194     pid_t mainpid;
195     cpu_set_t allcpuset;
196
197     printf("Starting Sequencer Demo\n");
198     syslog(LOG_CRIT, "Starting Sequencer Demo\n");
199
200     printf("testing Fib load with iterations :%d\n", ITERATION_COUNT);
201     double avg_time = 0;
202     for(int i=0;i<10;i++){
203         double start, end;
204         read_time(&start);
205         fibTest(ITERATION_COUNT);
206         read_time(&end);
207         double total_ex = end - start;
208         avg_time += total_ex;
209         printf("iteration %d) Start time: %f ms , end time: %f ms , execution time:
    %f ms\n\n",i, start, end, total_ex);
210         syslog(LOG_CRIT, "iteration %d) Start time: %f ms , end time: %f ms ,
    execution time: %f ms\n\n",i, start, end, total_ex);
211
212     }
213     printf("***** Average time %f *****\n", avg_time / 10);
214     syslog(LOG_CRIT, "***** Average time %f *****\n", avg_time / 10);
215
```

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

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

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

```
431         syslog(LOG_CRIT, "pthread_create successful for sequencer service 0\n");
432     }
433 
434     for (i = 0; i < NUM_THREADS; i++)
435         pthread_join(threads[i], NULL);
436 
437     printf("\nTEST COMPLETE\n");
438     syslog(LOG_CRIT, "\nTEST COMPLETE\n");
439 }
440 
441 void *Sequencer(void *threadp)
442 {
443     struct timeval current_time_val;
444     struct timespec delay_time = {0, 33333333}; // delay for 33.33 msec, 30 Hz
445     struct timespec remaining_time;
446     double current_time;
447     double residual;
448     int rc, delay_cnt = 0;
449     unsigned long long seqCnt = 0;
450     threadParams_t *threadParams = (threadParams_t *)threadp;
451 
452     gettimeofday(&current_time_val, (struct timezone *)0);
453     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);
454     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);
455 
456     do
457     {
458         delay_cnt = 0;
459         residual = 0.0;
460 
461         gettimeofday(&current_time_val, (struct timezone *)0);
462         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);
463 
464         do
465         {
466             rc = nanosleep(&delay_time, &remaining_time);
467 
468             if (rc == EINTR)
469             {
470                 residual = remaining_time.tv_sec + ((double)remaining_time.tv_nsec /
    (double)NANOSEC_PER_SEC);
471 
472                 if (residual > 0.0)
473                     printf("residual=%lf, sec=%d, nsec=%d\n", residual, (int)
    remaining_time.tv_sec, (int)remaining_time.tv_nsec);
474 
475                 delay_cnt++;
476             }
477             else if (rc < 0)
478             {
479                 perror("Sequencer nanosleep");
480                 exit(-1);
481             }
```

```
482
483         } while ((residual > 0.0) && (delay_cnt < 100));
484
485         seqCnt++;
486         gettimeofday(&current_time_val, (struct timezone *)0);
487         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);
488
489         if (delay_cnt > 1)
490             printf("Sequencer looping delay %d\n", delay_cnt);
491
492         // Release each service at a sub-rate of the generic sequencer rate
493
494         // Servcie_1 = RT_MAX-1 @ 3 Hz
495         if ((seqCnt % 10) == 0)
496         {
497             syslog(LOG_CRIT, "Task 1 (Frame Sampler thread) Released \n");
498             sem_post(&semS1); // Frame Sampler thread
499         }
500
501         // Service_2 = RT_MAX-2 @ 1 Hz
502         if ((seqCnt % 30) == 0)
503         {
504             syslog(LOG_CRIT, "Task 2 (Time-stamp with Image Analysis thread) Released
    \n");
505             sem_post(&semS2); // Time-stamp with Image Analysis thread
506         }
507
508         // Service_3 = RT_MAX-3 @ 0.5 Hz
509         if ((seqCnt % 60) == 0)
510         {
511             syslog(LOG_CRIT, "Task 3 ( Difference Image Proc thread) Released \n");
512             sem_post(&semS3); // Difference Image Proc thread
513         }
514
515         // Service_4 = RT_MAX-2 @ 1 Hz
516         if ((seqCnt % 30) == 0)
517         {
518             syslog(LOG_CRIT, "Task 4 (Time-stamp Image Save to File thread) Released
    \n");
519             sem_post(&semS4); // Time-stamp Image Save to File thread
520         }
521
522         // Service_5 = RT_MAX-3 @ 0.5 Hz
523         if ((seqCnt % 60) == 0)
524         {
525             syslog(LOG_CRIT, "Task 5 (Processed Image Save to File thread) Released
    \n");
526             sem_post(&semS5); // Processed Image Save to File thread
527         }
528
529         // Service_6 = RT_MAX-2 @ 1 Hz
530         if ((seqCnt % 30) == 0)
531         {
532             syslog(LOG_CRIT, "Task 6 (Send Time-stamped Image to Remote thread)
    Released \n");
```

```
533             sem_post(&semS6); // Send Time-stamped Image to Remote thread
534         }
535
536         // Service_7 = RT_MIN   0.1 Hz
537         if ((seqCnt % 300) == 0)
538         {
539             syslog(LOG_CRIT, "Task 7 (10 sec Tick Debug thread) Released \n");
540             sem_post(&semS7); // 10 sec Tick Debug thread
541         }
542
543         gettimeofday(&current_time_val, NULL);
544         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);
545
546     } while (!abortTest && (seqCnt < threadParams->sequencePeriods));
547
548     sem_post(&semS1);
549     sem_post(&semS2);
550     sem_post(&semS3);
551     sem_post(&semS4);
552     sem_post(&semS5);
553     sem_post(&semS6);
554     sem_post(&semS7);
555     abortS1 = TRUE;
556     abortS2 = TRUE;
557     abortS3 = TRUE;
558     abortS4 = TRUE;
559     abortS5 = TRUE;
560     abortS6 = TRUE;
561     abortS7 = TRUE;
562     print_data();
563
564     pthread_exit((void *)0);
565 }
566
567 void *Service_1(void *threadp)
568 {
569     double start, end, total;
570     threadParams_t *threadParams = (threadParams_t *)threadp;
571
572     read_time(&start);
573     syslog(LOG_CRIT, "Task 1, Frame Sampler thread @ msec=%f \n", start);
574     printf("Task 1, Frame Sampler thread @ msec=%f \n", start);
575
576     while (!abortS1)
577     {
578         sem_wait(&semS1);
579
580         execution_cycle[0]++;
581         read_time(&start);
582         syslog(LOG_CRIT, "Task 1, Frame Sampler start %d @ msec=%f",
    execution_cycle[0], start);
583         fibTest(ITERATION_COUNT);
584         read_time(&end);
585         total = end - start;
586         if(total > wcet[0]) wcet[0] = total;
```

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

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

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

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

```
797 }
798
```