**facedetect.cpp**

```cpp
#include "opencv2/objdetect.hpp"
#include "opencv2/videoio.hpp"
#include "opencv2/highgui.hpp"
#include "opencv2/imgproc.hpp"

#include <iostream>
#include <stdio.h>

#include <pthread.h>
#include <sched.h>
#include <unistd.h>
#include <stddef.h>
#include <stdlib.h>
#include <sys/sysinfo.h>
#include <stdbool.h>
#include <semaphore.h>
#include <sys/time.h>
#include <mqueue.h>
#include <math.h>
#include <signal.h>
#include <atomic>

/*
* Example 9
* C: 35 4 3
* T: 50 100 100
* D: 50 50 50
*
* Task 0, WCET=35, Period=50, Utility Sum = 0.700000
* Task 1, WCET=4, Period=100, Utility Sum = 0.740000
* Task 2, WCET=3, Period=100, Utility Sum = 0.770000
*
* Total Utility Sum = 0.770000
* LUB = 0.779763
* RM LUB: Feasible
* Completion time feasibility: Feasible
* Scheduling point feasibility: Feasible
* Deadline monotonic: Feasible
*
* (Period)
* Total utility in EDF: 0.770000 Which is less than 1.0
* EDF on Period: Feasible
* Total utility in LLF: 0.770000 Which is less than 1.0
* LLF on Period: Feasible

* (Deadline)
* Total utility in EDF: 0.840000 Which is less than 1.0
* EDF on Deadline: Feasible
* Total utility in LLF: 0.840000 Which is less than 1.0
* LLF on Deadline: Feasible
*
*/

```

```
54 #define FRAME_HEIGHT 320
55 #define FRAME_WIDTH 240
56
57 #define NANOSEC_PER_SEC 1000000000
58
59 #define OVERALL_DEADLINE 150
60 #define FACE_DETECTION_DEADLINE 50
61 #define SERVO_ACTUATION_DEADLINE 50
62 #define SERVO_SHOOT_DEADLINE 50
63
64 #define NUMBER_OF_TASKS 3
65
66 #define CUSTOM_MQ_NAME "/send_receive_mq"
67
68 struct mq_attr mq_attr;
69 mqd_t message_queue_instance;
70
71 /** Global variables */
72 cv::String faceCascadePath;
73 cv::CascadeClassifier faceCascade;
74 double overall_start_time, overall_stop_time;
75 double face_recognition_start_ms;
76 double face_recognition_end_ms;
77
78 double wcet_servo_actuation;
79 double wcet_servo_shoot;
80 double wcet_face_recognition;
81 double wcet_overall;
82
83 int overall_deadline_miss;
84 int face_detection_deadline_miss;
85 int servo_actuation_deadline_miss;
86 int servo_shoot_deadline_miss;
87
88 int starting_count = 0;
89
90 volatile bool exit_flag = false;
91 std::atomic<bool> stop_timer = false;
92
93 #ifdef IS_RPI
94
95 #define NUM_GPIO 32
96
97 #define SERVO1_PIN 4
98 #define SERVO2_PIN 23
99 #define LASER_PIN 18
100
101 #define MIN_WIDTH 1000
102 #define MAX_WIDTH 2000
103 #define SERVO_RANGE 180
104
105 #include <pigpio.h>
106
107 void change_servo_degree(int output_pin, uint8_t degree)
108 {
109     if (degree < 0)
```

```
110     {
111         degree = 0;
112     }
113     else if (degree > SERVO_RANGE)
114     {
115         degree = SERVO_RANGE;
116     }
117 
118     int pwmWidth = MIN_WIDTH + (degree * (MAX_WIDTH - MIN_WIDTH) / SERVO_RANGE);
119 
120     gpioServo(output_pin, pwmWidth);
121 }
122 
123 #endif
124 
125 typedef struct
126 {
127     int threadId;
128 } ThreadArgs_t;
129 
130 typedef struct
131 {
132     int period;
133     int burst_time;
134     struct sched_param priority_param;
135     void *(*thread_handle)(void *);
136     pthread_t thread;
137     ThreadArgs_t thread_args;
138     void *return_Value;
139     pthread_attr_t attribute;
140     int target_cpu;
141 } RmTask_t;
142 
143 typedef struct
144 {
145     int x1;
146     int y1;
147     int x2;
148     int y2;
149 } Points_t;
150 
151 sem_t semaphore_face_detect, semaphore_servo_actuator, semaphore_servo_shoot;
152 
153 double read_time(double *var)
154 {
155     struct timeval tv;
156     if (gettimeofday(&tv, NULL) != 0)
157     {
158         perror("readTOD");
159         return 0.0;
160     }
161     else
162     {
163         *var = ((double)(((double)tv.tv_sec * 1000) + (((double)tv.tv_usec) / 1000.0)
    ));
164     }
```

```
165     return (*var);
166 }
167 
168 void delay_ns(int ns)
169 {
170 
171     double residual;
172     struct timeval current_time_val;
173     struct timespec delay_time = {0, ns}; // delay for 33.33 msec, 30 Hz
174     struct timespec remaining_time;
175     int rc;
176 
177     gettimeofday(&current_time_val, (struct timezone *)0);
178 
179     rc = nanosleep(&delay_time, &remaining_time);
180 
181     if (rc == EINTR)
182     {
183         residual = remaining_time.tv_sec + ((double)remaining_time.tv_nsec / (double)
    NANOSEC_PER_SEC);
184 
185         if (residual > 0.0)
186             printf("residual=%lf, sec=%d, nsec=%d\n", residual, (int)
    remaining_time.tv_sec, (int)remaining_time.tv_nsec);
187     }
188     else if (rc < 0)
189     {
190         perror("delay_ns nanosleep");
191         exit(-1);
192     }
193 }
194 
195 
196 
197 void *Sequencer(void *threadp)
198 {
199     struct timeval current_time_val;
200     struct timespec delay_time = {0, 50000000}; // delay for 33.33 msec, 30 Hz
201     struct timespec remaining_time;
202     double current_time;
203     double residual;
204     int rc, delay_cnt = 0;
205     unsigned long long seqCnt = 0;
206 
207     gettimeofday(&current_time_val, (struct timezone *)0);
208     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);
209     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);
210 
211     do
212     {
213         delay_cnt = 0;
214         residual = 0.0;
215 
216         gettimeofday(&current_time_val, (struct timezone *)0);
```

```
217         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);
218
219         delay_ns(50000000);
220
221         seqCnt++;
222         gettimeofday(&current_time_val, (struct timezone *)0);
223         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);
224
225
226         syslog(LOG_CRIT, "Task 1 (Frame Sampler thread) Released \n");
227         sem_post(&semaphore_face_detect); // Frame Sampler thread
228
229         syslog(LOG_CRIT, "Task 2 (Servo Actuation) Released \n");
230         sem_post(&semaphore_servo_actuator); // Time-stamp with Image Analysis thread
231
232         syslog(LOG_CRIT, "Task 3 (Servo Shoot) Released \n");
233         sem_post(&semaphore_servo_shoot); // Difference Image Proc thread
234
235         gettimeofday(&current_time_val, NULL);
236         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);
237
238     } while (!!exit_flag);
239
240     sem_post(&semaphore_face_detect);
241     sem_post(&semaphore_servo_actuator);
242     sem_post(&semaphore_servo_shoot);
243
244
245     pthread_exit((void *)0);
246 }
247
248
249 Points_t detectFaceOpenCVLBP(cv::CascadeClassifier faceCascade, cv::Mat &frameGray, int inHeight = 300, int inWidth = 0)
250 {
251     std::vector<cv::Rect> faces;
252     faceCascade.detectMultiScale(frameGray, faces);
253
254     if (!faces.empty())
255     {
256         int x1 = faces[0].x;
257         int y1 = faces[0].y;
258         int x2 = faces[0].x + faces[0].width;
259         int y2 = faces[0].y + faces[0].height;
260         cv::rectangle(frameGray, cv::Point(x1, y1), cv::Point(x2, y2), cv::Scalar(0, 255, 0), 2);
261         return {x1, y1, x2, y2};
262     }
263     return {0, 0, 0, 0};
264 }
265
266 void *FaceDetectService(void *args)
```

```
267 {
268
269     RmTask_t *task_parameters = (RmTask_t *)args;
270
271     struct sched_param schedule_param;
272     int policy, cpucore;
273     pthread_t thread;
274     cpu_set_t cpuset;
275
276     thread = pthread_self();
277     cpucore = sched_getcpu();
278
279     pthread_getschedparam(pthread_self(), &policy, &schedule_param);
280     CPU_ZERO(&cpuset);
281     pthread_getaffinity_np(thread, sizeof(cpu_set_t), &cpuset);
282
283     std::string faceCascadePath = "./lbpcascade_frontalface.xml";
284     cv::CascadeClassifier faceCascade;
285
286     if (!faceCascade.load(faceCascadePath))
287     {
288         printf("--(!)Error loading face cascade\n");
289         return NULL;
290     }
291
292     cv::VideoCapture source;
293     source.open(0, cv::CAP_V4L);
294
295     cv::Mat frame, frameGray;
296     cv::Size frameSize(FRAME_HEIGHT, FRAME_WIDTH);
297
298     double fps, execute_ms;
299
300     while (!exit_flag)
301     {
302         sem_wait(&semaphore_face_detect);
303         read_time(&face_recognition_start_ms);
304
305         source >> frame;
306         if (frame.empty())
307             break;
308         cv::imshow("Original Frame", frame);
309         cv::resize(frame, frame, frameSize);
310         cv::cvtColor(frame, frameGray, cv::COLOR_BGR2GRAY);
311
312         Points_t face_points = detectFaceOpenCVLBP(faceCascade, frameGray);
313
314         cv::imshow("OpenCV - LBP Face Detection", frameGray);
315
316         if (face_points.x1 != 0 && face_points.x2 != 0)
317         {
318             // Allocate memory for the Points_t structure
319             Points_t *points_buffer_ptr = (Points_t *)malloc(sizeof(Points_t));
320
321             // Copy the face_points data into the allocated memory
322             memcpy(points_buffer_ptr, &face_points, sizeof(Points_t));
```

```
323
324             // Send the message containing the Points_t structure
325             if (mq_send(message_queue_instance, (const char *)points_buffer_ptr,
    sizeof(Points_t), 0) == -1)
326             {
327                 perror("mq_send");
328                 free(points_buffer_ptr);
329             }
330         }
331         else
332         {
333 #ifdef IS_RPI
334             gpioWrite(LASER_PIN, 0);
335 #endif
336         }
337
338         read_time(&face_recognition_end_ms);
339         execute_ms = (face_recognition_end_ms - face_recognition_start_ms);
340         fps = 1000 / execute_ms;
341
342         if (wcet_face_recognition < execute_ms && starting_count > 5)
343         {
344             wcet_face_recognition = execute_ms;
345         }
346         if (execute_ms > FACE_DETECTION_DEADLINE && starting_count > 5)
347         {
348             face_detection_deadline_miss++;
349         }
350
351         // printf("| FPS                                | %.2f       |\n", fps);
352         // printf("| Execution Time                     | %.2f ms    |\n\n", execute_ms)
    ;
353
354         int k = cv::waitKey(5);
355         if (k == 27)
356         {
357             Points_t *points_buffer_ptr = (Points_t *)malloc(sizeof(Points_t));
358             memcpy(points_buffer_ptr, &face_points, sizeof(Points_t));
359
360             exit_flag = true;
361
362             sem_post(&semaphore_face_detect);
363             mq_send(message_queue_instance, (const char *)points_buffer_ptr,
    sizeof(Points_t), 0);
364             sem_post(&semaphore_servo_shoot);
365             // set flag
366             break;
367         }
368
369         if (starting_count < 9)
370         {
371             starting_count++;
372         }
373     }
374     printf("Face Detection service ended !\n\r");
375
376     cv::destroyAllWindows();
```

```
377     return NULL;
378 }
379 
380 void *ServoActuatorService(void *args)
381 {
382     RmTask_t *task_parameters = (RmTask_t *)args;
383 
384     struct sched_param schedule_param;
385     int policy, cpucore;
386     pthread_t thread;
387     cpu_set_t cpuset;
388     double execution_complete_time_for_a_loop;
389     double execution_start_time_for_a_loop;
390 
391     thread = pthread_self();
392     cpucore = sched_getcpu();
393 
394     pthread_getschedparam(pthread_self(), &policy, &schedule_param);
395     CPU_ZERO(&cpuset);
396     pthread_getaffinity_np(thread, sizeof(cpu_set_t), &cpuset);
397 
398     int center_x, center_y;
399     double angle_pan = 0;
400     double angle_tilt = 0;
401     int angle_pan_int;
402     int angle_tilt_int;
403 
404 #ifdef IS_RPI
405 
406     std::cout << "Starting Thread 1 now! Press CTRL+C to exit" << std::endl;
407 
408     do
409     {
410 
411         Points_t received_points;
412         ssize_t received_size = mq_receive(message_queue_instance, (char *)&
    received_points, sizeof(Points_t), NULL);
413 
414         if (received_size == -1)
415         {
416             if (errno == EINTR)
417             {
418                 // Interrupted by signal, check exit_flag and continue if not set
419                 continue;
420             }
421             perror("mq_receive");
422         }
423 
424         else
425         {
426             read_time(&execution_start_time_for_a_loop);
427             // printf("receiver - Received message | X1 = %d X2 = %d Y1 = %d Y2 = %d
    | Received message of size = %ld\n",
428             //        received_points.x1, received_points.x2, received_points.y1,
    received_points.y2, received_size);
429 
430             center_x = (received_points.x1 + received_points.x2) / 2;
```

```
431             center_y = (received_points.y1 + received_points.y2) / 2;
432 
433             if (center_x > 160)
434             {
435                 angle_pan = atan(((320.0 - center_x) / 160.0)) * (180.0 / M_PI);
436             }
437             else
438             {
439                 angle_pan = atan(((160.0 - center_x) / 160.0)) * (180.0 / M_PI);
440                 angle_pan = 50 + angle_pan;
441             }
442 
443             angle_tilt = atan(((240.0 - center_y) / 160.0)) * (180.0 / M_PI);
444 
445             angle_pan_int = (int)angle_pan;
446             angle_tilt_int = (int)angle_tilt;
447 
448             // printf("Angle pan %d Angle tilt %d\n\r", angle_pan_int,
    angle_tilt_int);
449 
450             change_servo_degree(SERVO1_PIN, angle_pan_int);
451             change_servo_degree(SERVO2_PIN, angle_tilt_int);
452 
453             read_time(&execution_complete_time_for_a_loop);
454 
455             double execution_time = execution_complete_time_for_a_loop -
    execution_start_time_for_a_loop;
456 
457             // printf("| Execution time for Servo Actuation      | %.2f ms    |\n\n",
    execution_time);
458 
459             if (wcet_servo_actuation < execution_time && starting_count > 5)
460             {
461                 wcet_servo_actuation = execution_time;
462             }
463             if (execution_time > SERVO_ACTUATION_DEADLINE && starting_count > 5)
464             {
465                 servo_actuation_deadline_miss++;
466             }
467 
468 
469         }
470     } while (!exit_flag);
471     printf("Servo Actuation service ended !\n\r");
472 
473 #endif
474 
475     return NULL;
476 }
477 
478 void *ServoShootService(void *args)
479 {
480     RmTask_t *task_parameters = (RmTask_t *)args;
481 
482     double execution_complete_time_for_a_servo_shoot;
483     double execution_start_time_for_a_servo_shoot;
484 
```

```
485     struct sched_param schedule_param;
486     int policy, cpucore;
487     pthread_t thread;
488     cpu_set_t cpuset;
489 
490     thread = pthread_self();
491     cpucore = sched_getcpu();
492 
493     pthread_getschedparam(pthread_self(), &policy, &schedule_param);
494     CPU_ZERO(&cpuset);
495     pthread_getaffinity_np(thread, sizeof(cpu_set_t), &cpuset);
496 
497 #ifdef IS_RPI
498 
499     std::cout << "Starting Thread 2 now! Press CTRL+C to exit" << std::endl;
500 
501     int degree = 0;
502     do
503     {
504 
505 
506         sem_wait(&semaphore_servo_shoot);
507 
508         read_time(&execution_start_time_for_a_servo_shoot);
509 
510         gpioWrite(LASER_PIN, 1);
511 
512         read_time(&execution_complete_time_for_a_servo_shoot);
513 
514         double execution_time = execution_complete_time_for_a_servo_shoot -
    execution_start_time_for_a_servo_shoot;
515 
516         double overall_response_time = execution_complete_time_for_a_servo_shoot -
    face_recognition_start_ms;
517 
518         // printf("| Execution time for Servo Shoot          | %.2f ms    |\n",
    execution_time);
519         // printf("| Overall response time                   | %.2f ms    |\n\n",
    overall_response_time);
520 
521         if (wcet_servo_shoot < execution_time && starting_count > 5)
522         {
523             wcet_servo_shoot = execution_time;
524         }
525         if (execution_time > SERVO_SHOOT_DEADLINE && starting_count > 5)
526         {
527             servo_shoot_deadline_miss++;
528         }
529         if (overall_response_time > OVERALL_DEADLINE && starting_count > 5)
530         {
531             overall_deadline_miss++;
532         }
533         if (overall_response_time > wcet_overall && starting_count > 5 && !exit_flag)
534         {
535             wcet_overall = overall_response_time;
536         }
537 
```

```
538         sem_post(&semaphore_face_detect);
539     } while (!exit_flag);
540     printf("Servo shoot service ended !\n\r");
541 
542     gpioWrite(LASER_PIN, 0);
543 
544 #endif
545 
546     return NULL;
547 }
548 
549 void printFinalTable()
550 {
551     printf("+-------------------------------------------------+------------+\n");
552     printf("| Metric                                          | Value      |\n");
553     printf("+-------------------------------------------------+------------+\n");
554     printf("| Overall Deadline Miss Count                     | %11d |\n", overall_deadline_miss);
555     printf("| Face Detection Deadline Miss Count              | %11d |\n", face_detection_deadline_miss);
556     printf("| Servo Actuation Deadline Miss Count             | %11d |\n", servo_actuation_deadline_miss);
557     printf("| Servo Shoot Deadline Miss Count                 | %11d |\n", servo_shoot_deadline_miss);
558     printf("| Face Recognition Worst-Case Execution Time      | %8.2f ms |\n", wcet_face_recognition);
559     printf("| Servo Actuation Worst-Case Execution Time       | %8.2f ms |\n", wcet_servo_actuation);
560     printf("| Servo Shoot Worst-Case Execution Time           | %8.2f ms |\n", wcet_servo_shoot);
561     printf("| OverAll Response Worst-Case Execution Time      | %8.2f ms |\n", wcet_overall);
562     printf("+-------------------------------------------------+------------+\n");
563 }
564 
565 void print_scheduler(void)
566 {
567     int schedType;
568     schedType = sched_getscheduler(getpid());
569     switch (schedType)
570     {
571     case SCHED_FIFO:
572         printf("Pthread Policy is SCHED_FIFO\n");
573         break;
574     case SCHED_OTHER:
575         printf("Pthread Policy is SCHED_OTHER\n");
576         break;
577     case SCHED_RR:
578         printf("Pthread Policy is SCHED_OTHER\n");
579         break;
580     default:
581         printf("Pthread Policy is UNKNOWN\n");
582     }
583 }
584 
585 int main(int argc, const char **argv)
586 {
587 
```

```
588 #ifdef IS_RPI
589
590     std::cout << "Raspberry PI " << std::endl;
591
592     if (gpioInitialise() < 0)
593         return -1;
594
595     gpioSetMode(LASER_PIN, PI_OUTPUT);
596
597 #else
598
599     std::cout << "Linux System " << std::endl;
600
601 #endif
602
603     pthread_t threads[NUMBER_OF_TASKS];
604     cpu_set_t threadcpu;
605
606     /* setup common message q attributes */
607     mq_attr.mq_maxmsg = 10;
608     mq_attr.mq_msgsize = sizeof(Points_t);
609     mq_attr.mq_flags = 0;
610
611     mq_unlink(CUSTOM_MQ_NAME); // Unlink if the previous message queue exists
612
613     message_queue_instance = mq_open(CUSTOM_MQ_NAME, O_CREAT | O_RDWR, S_IRWXU, &
mq_attr);
614     if (message_queue_instance == (mqd_t)(-1))
615     {
616         perror("mq_open");
617     }
618
619     int rt_max_prio = sched_get_priority_max(SCHED_FIFO);
620     int rt_min_prio = sched_get_priority_min(SCHED_FIFO);
621
622     RmTask_t tasks[NUMBER_OF_TASKS] = {
623         // {.period = 20,     // ms
624         //  .burst_time = 10, // ms
625         //  .priority_param = {rt_max_prio},
626         //  .thread_handle = &FaceDetectService,
627         //  .thread = threads[0],
628         //  .thread_args = {0},
629         //  .target_cpu = 2},
630         {20, 10, {rt_max_prio}, &FaceDetectService, threads[0], {0}, NULL, tasks[0]
.attribute, 1},
631         {50, 20, {rt_max_prio - 1}, &ServoActuatorService, threads[1], {1}, NULL,
tasks[1].attribute, 0},
632         {50, 20, {rt_max_prio - 2}, &ServoShootService, threads[2], {2}, NULL,
tasks[2].attribute, 0}
633
634     };
635     // Initialize Semaphore
636     sem_init(&semaphore_face_detect, false, 1);
637     sem_init(&semaphore_servo_actuator, false, 1);
638     sem_init(&semaphore_servo_shoot, false, 1);
639
640     pthread_attr_t attribute_flags_for_main; // for schedular type, priority
```

```
641     struct sched_param main_priority_param;
642 
643     printf("This system has %d processors configured and %d processors available.\n", get_nprocs_conf(), get_nprocs());
644 
645     printf("Before adjustments to scheduling policy:\n");
646     print_scheduler();
647 
648     CPU_ZERO(&threadcpu); // clear all the cpus in cpuset
649 
650     main_priority_param.sched_priority = rt_max_prio;
651     for (int i = 0; i < NUMBER_OF_TASKS; i++)
652     {
653         CPU_SET(tasks[i].target_cpu, &threadcpu);
654 
655         // initialize attributes
656         pthread_attr_init(&tasks[i].attribute);
657 
658         pthread_attr_setinheritsched(&tasks[i].attribute, PTHREAD_EXPLICIT_SCHED);
659         pthread_attr_setschedpolicy(&tasks[i].attribute, SCHED_FIFO);
660         pthread_attr_setschedparam(&tasks[i].attribute, &tasks[i].priority_param);
661         pthread_attr_setaffinity_np(&tasks[i].attribute, sizeof(cpu_set_t), &threadcpu);
662 
663         CPU_ZERO(&threadcpu);
664     }
665 
666     pthread_attr_init(&attribute_flags_for_main);
667 
668     pthread_attr_setinheritsched(&attribute_flags_for_main, PTHREAD_EXPLICIT_SCHED);
669     pthread_attr_setschedpolicy(&attribute_flags_for_main, SCHED_FIFO);
670     pthread_attr_setaffinity_np(&attribute_flags_for_main, sizeof(cpu_set_t), &threadcpu);
671 
672     // Main thread is already created we have to modify the priority and scheduling scheme
673     int status_setting_schedular = sched_setscheduler(getpid(), SCHED_FIFO, &main_priority_param);
674     if (status_setting_schedular)
675     {
676         printf("ERROR; sched_setscheduler rc is %d\n", status_setting_schedular);
677         perror(NULL);
678         exit(-1);
679     }
680 
681     printf("After adjustments to scheduling policy:\n");
682     print_scheduler();
683 
684     read_time(&overall_start_time);
685 
686     for (int i = 0; i < NUMBER_OF_TASKS; i++)
687     {
688         // Create a thread
689         // First paramter is thread which we want to create
690         // Second parameter is the flags that we want to give it to
691         // third parameter is the routine we want to give
692         // Fourth parameter is the value
```

```
693         printf("Setting thread %d to core %d\n", i, tasks[i].target_cpu);
694 
695         if (pthread_create(&tasks[i].thread, &tasks[i].attribute, tasks[i]
    .thread_handle, &tasks[i]) != 0)
696         {
697             perror("Create_Fail");
698         }
699     }
700 
701     printf("Test Conducted over %lf msec\n", (double)(overall_stop_time -
    overall_start_time));
702 
703     for (int i = 0; i < NUMBER_OF_TASKS; i++)
704     {
705         pthread_join(tasks[i].thread, &tasks[i].return_Value);
706     }
707 
708     // Cleanup actions
709     mq_close(message_queue_instance);
710     mq_unlink(CUSTOM_MQ_NAME);
711 
712 #ifdef IS_RPI
713     gpioTerminate();
714 #endif
715 
716     if (pthread_attr_destroy(&tasks[0].attribute) != 0)
717         perror("attr destroy");
718     if (pthread_attr_destroy(&tasks[1].attribute) != 0)
719         perror("attr destroy");
720     if (pthread_attr_destroy(&tasks[2].attribute) != 0)
721         perror("attr destroy");
722 
723     sem_destroy(&semaphore_face_detect);
724     sem_destroy(&semaphore_servo_actuator);
725     sem_destroy(&semaphore_servo_shoot);
726 
727     printFinalTable();
728 
729 #ifdef IS_RPI
730     gpioTerminate();
731 #endif
732 }
733 
```