EE5903 REAL TIME SYSTEMS CA2

Reviewer: Ryan Christopher Louie, A0149643X

Original Designer: Lukas Philipp Brunke, A0149064A

1. Did the CA1 design document provide you a clear understanding of the intended design? Would you be able to implement the system based on the design document? (It may be worth attempting implementation to properly answer this question, although you are not required to submit any code that you attempt).

Lukas's design document provided sufficient details which made clear the intended design of real-time apple processing system. The main diagram did an excellent job separating out the three threads of processing. The flow of data was a particular strong point, showing the structure of data (any developer could clearly interpret the structure needed to support the data flowing through the messaging queues) and how the data would move (using domain-specific terminology like messaging queues makes the expectations on implementation clear as well). In addition, the accompanying paragraphs in the report were concise without loss of logical clarity; they explained the crucial logic related to timing for the three main processes.

I will say, however, that additional design diagrams would have made the process easier for a potential developer to implement this system. The crux of the system design was not detailed in diagrammatic form, which might have led improper interpretation of the design during implementation.

In Figure 1, I detail an addition the design diagram, which is an activity diagram showing data flows for the photo processing thread. In giving specific logic related to the timing deadline of the external processing unit, a programmer could have an easier time implementing the design intention of discarding apples for which the quality is unknown by the time the apple is under the actuator.

2. What are the advantages and disadvantages of the CA1 design adopted? How did it differ from your own CA1 design. What improvements would you suggest to the CA1 design, if any?

The design proposed by Lukas is quite similar to mine; this may be related to the fact that we collaborated on CA1. The use of three threads/processes, as well as message queues to communicate between them, was the core similarity. The way Lukas's design handles the logic of discarding apples after 5 seconds have passed is different. His design states that he discards apples that have not been processed after 5 seconds. In contrast, my design has no mechanism by which apples which are still being processed can be discarded. Subsequent apples which have been blocked by previous apple processing are moved to the quality message queue immediately, marking them as UNKNOWN, and the actuator discards these UNKNOWN apples.

3. Was the CA1 implementation in accordance with the design?

Lukas's CA1 implementation slightly differed to the proposed design. The detail in his design about discarding apples that have taken longer than 5 seconds to be processed does not get fully implemented! To his implementation's defense, the structure of three threads, two message queues in between, and most of the timing logic located in the actuator thread follows closely to the design diagram. The discrepancy may be due to the fact that the diagram, as is, does not satisfy the design intention of discarding apples that are still being processed by the `process_photo` function after 5 seconds have elapsed. The "Process Photo" process in the Data Flow Diagram can be expanded, as explained above; if this expansion includes a way for apples

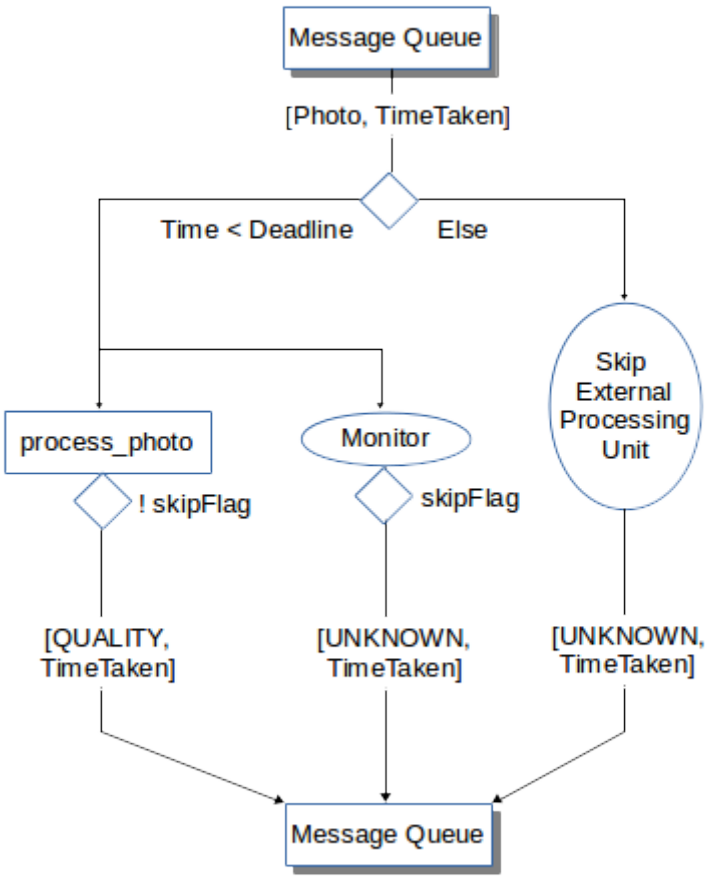

Figure 1: ADDITIONAL DIAGRAM SUGGESTION MADE BY ME

to be sent to the actuator prior to the External Processing Unit (`process_photo` method) returning, the original design intention, of throwing away all apples if their quality has not been determined in the deadline, could be satisfied.

4. Were there any notable features or shortcomings that you noticed in the CA1 implementation? What improvements would you suggest to the CA1 implementation, if any?

One issue I saw when running the program was in the following lines of the code (159-160):

```
double processTimeMS = actuatorTime.tv_usec - startTime.tv_usec;
printf("MILlISECONDS: %f\n", processTimeMS);
```

In the program output, `processTimeMS` printed negative values, which is non nonsensical. I would recommend that this be debugged, as the process time should always be a positive number. This may be influencing the timing of when the actuator discards apples.

5. Try running the CA1 code provided by your peer under different test conditions. Document your findings in the report:

No stress test on a 4 cpu system

```
-- TEST RESULTS --
Elapsed time (minutes): 5.09
Number of apples: 300
Number of bad apples discarded: 18
Number of good apples discarded: 22
Number of bad apples packed: 11
Number of apples spoilt due to bad apples: 105
Revenue from sale of good apples: $144
Revenue loss due to system shortcomings: $127 (47%)
```

Using the command `stress --cpu 1` on a 4 cpu system

```
-- TEST RESULTS --
Elapsed time (minutes): 5.09
Number of apples: 300
Number of bad apples discarded: 29
Number of good apples discarded: 3
Number of bad apples packed: 3
Number of apples spoilt due to bad apples: 45
Revenue from sale of good apples: $220
Revenue loss due to system shortcomings: $48 (18%)
```

Using the command `stress --cpu 4` on a 4 cpu system

```
-- TEST RESULTS --
Elapsed time (minutes): 5.10
Number of apples: 300
Number of bad apples discarded: 24
Number of good apples discarded: 15
Number of bad apples packed: 2
Number of apples spoilt due to bad apples: 22
Revenue from sale of good apples: $237
Revenue loss due to system shortcomings: $37 (14%)
```

6. Summarize the performance of the CA1 code under different conditions (document the relevant conditions). Can you explain your the observed variability in performance in terms of the design or implementation decisions? Do you think there is scope for improvement in performance? In case you test any of your recommended changes to the design/code, you may wish to include code snippets and performance improvement details in an appendix to your report (appendix does not count towards the page limit). This is optional, but encouraged, as it will allow you to test your ideas, and enhance your learning experience.

From the final test results, we can see that for Total Number Bad Apples,

```
No Stress, 1 CPU Stress, 4 CPU Stress,
26       , 32          , 26
```

and for Number of Bad Apples packed,

```
No Stress, 1 CPU Stress, 4 CPU Stress,
11       , 3           , 2
```

The results here are inconclusive from single tests, as there a large element of randomness in the types of apples sent to the external processing unit, and the time the external processing unit requires. I (the reviewer) did not have time to run these stress tests for multiple trials to obtain an average performance. As stated, the current implementation does not require obvious improvements; the implementation of threads, message queues, and the structures sent in the messages are all quite efficient.

In regards to the design, there is improvements to be made, as shown in the expanded diagram I have provided in the beginning of this review. Successful implementation of this design would result in the number of bad apples packed to be zero, and probably the number of bad and good apples to increase slightly (as we are discarding all apples if there quality could not be determined successfully or in time).

I made a quick edit to the implementation, reflecting my suggested redesign of the processing thread, which is `runProcess` in Lukas's code. In particular, a separate `monitor` thread is spun off, which manages the sending to the next message queue if the time deadline has arrived, without the `process_photo` function finishing.

Unfortunately, it seems like there are still bad apples going through the pipeline (not all of them are being discarded). There might be some bugs in the timing, but in general, more apples are being discarded, which meets the design intention.

```
-- TEST RESULTS --
Elapsed time (minutes): 5.11
Number of apples: 300
Number of bad apples discarded: 27
Number of good apples discarded: 17
Number of bad apples packed: 2
Number of apples spoilt due to bad apples: 46
Revenue from sale of good apples: $208
Revenue loss due to system shortcomings: $63 (23%)
```

My improved implementation reflecting this redesign is given below

```
void* runProcess(void* p) { // process thread implementation

    printf("Processing thread started\n");

    int counter = 0;

    do { // begin do-loop
        counter++;
```

```
int interruptFlag = 0;

// receive photo and time data
struct photo_msgbuf bufPempty;
msgrcv(msqidPhoto, &bufPempty, sizePhoto, 1, 0);

// get startTime info for everyone
struct timeval startTime = bufPempty.photoTime.startingTime;

// define monitor function
void * monitor(void* p) {
    while (interruptFlag == 0) {
        // get the current time
        struct timeval currentTime;
        gettimeofday(&currentTime, NULL);

        // calculate the time elapsed
        double processTimeS = currentTime.tv_sec - startTime.tv_sec;
        // assert( processTimeS >= 0 );
        double processTimeMS = currentTime.tv_usec - startTime.tv_usec;
        // assert( processTimeMS >= 0 );
        double processTime = processTimeS + processTimeMS/1000000;

        // overtime
        if (processTime >= 5.0) {
            printf("damn slow. Bypassing the external processing unit!");
            // we're sending an UNKNOWN to the next message queue
            // without letting the external processing unit finish
            interruptFlag = 1;

            // send message with UNKNOWN and time data
            struct qualTime_msgbuf bufQualTime = {2, {UNKNOWN, startTime}};
            msgsnd(msqidProcess, &bufQualTime, sizeProcess, 0);

        }

        // poll every .10 seconds
        usleep(0.10*1000000);
    }
}
// spin off monitor thread
pthread_t monitorThread;
pthread_create(&monitorThread, NULL, &monitor, NULL);

// process apple and get its quality
QUALITY photoQuality = process_photo(bufPempty.photoTime.photo);

// do the normal course of action if the external processing unit was not superseeded
if (interruptFlag == 0) {
    if (photoQuality == GOOD)
    {
        printf("Quality is GOOD for apple %d\n", counter);
    } else if (photoQuality == BAD) {
```

```
                printf("Quality is BAD for apple %d\n", counter);
            } else {
                printf("Quality is UNKNOWN for apple %d\n", counter);
            }

            // send message with apple quaility and time data
            struct qualTime_msgbuf bufQualTime = {2, {photoQuality, startTime}};
            msgsnd(msqidProcess, &bufQualTime, sizeProcess, 0);
        }

    //} while(counter <= numApples);
    } while(more_apples() == 1); // end do-looop

    printf("Processing thread ended\n");

    return NULL; // exit thread
}
```