ECEN 5623
Chutao Wei

Problem 1:

Case 1: Toyota ABS Software

The Toyota hybrid vehicle Prius 2010 has software glitch in their ABS system, which lead to a small delay between the driver stepping the break and the system hold the wheels. Tens of accidents are related to this. Toyota also had to recall a lot of cars and do the software update.

The ABS software here do not have predictable response due to the unpredictable software glitch that happened. Since it is a small delay that happened in an unpredictable manner.

Case 2: Therac-25

Therac-25 is a computer-controlled radiation therapy machine. Couple concurrent programming errors are found in the sotfware, which led to radiation overdose on 6 patients. The issues are:

1. Dangerous mode is allowed to be switched when it shouldn't be.
2. When blocking operation should be in place and halt system from more commands, the software decides to let user keep push buttons.

However, this is not just software issues. The engineering team did not follows safety design principles during engineering process.

1. No independent code review
2. Poor documentation and user manual
3. When using software from old model, the engineering team did not take extra precaution
4. The project depends too much on software. All control logic is software based. No way for hardware to provide information back to software, which lead to open-loop control logic(for application like this you definitely want a close-loop). I wonder if they ever hired a hardware engineer or they just assign work to other firms to do the hardware. This should be a pretty obvious thing.
5. They never test it!!!!

Case 3: ATT 4ESS Upgrade

ATT update the 4ESS(a central piece of device that interact with many other switches on the network) software lead to crashing of the whole long distance network in 1990.

To simplify the problem describe in the ariticle. Think about 3 switches A, B, and C. Switch A and Switch B and Switch C is installed with the new software. The new software forgot to include the logic to tell other switches that it finishes updating. The software will just do its normal routing work right after new software is installed. The scenario is described below

1. C told B, "I need to reboot, error occured"
2. B says, "OK, I will mark Switch C is rebotting."
3. After a while
4. C says, "Hi C, I need to route this balabalabala"
5. B says, "What the hell? I thought you are still rebooting. I think there might be something wrong with me. I better reboot, now"
6. B told A and B, "Hi A and B, I need to reboot, error occurred"
7. A and C says , "OK, I will mark Switch B is rebooting."
8. After a while
9. B says "Hi A and C, I need to route these balabalabala"
10. A and C say, "what the hell? I thought you are still rebooting. I think there might be something wrong with me. I better reboot, now"

ECEN 5623
Chutao Wei

Then the story is obvious, the reboot spreads all over the network because the new software forgot to tell other it finished rebooting.

This is a mistake in software requirement that is presented here. They means that the engineers did not test their “backup options”(which is reboot in this case). Or you can say the test engineers do not test all its code path. The error the happened on the first switch might be a normal error, but becomes the trigger of other error which is more server.

ECEN 5623
Chutao Wei

Problem 2:

Case: Aegis Missile cruiser USS System Failure

a) All systems on cruiser use Windows NT. This is not a big deal probably back then. As today we would use a more reliable and deterministic system.
   The cause of the very first failure is dividing by 0. This should be very obvious to EE or CS people. This is an exception that you would need to write a handler for, or you can have input check to guard against dividing by zero.
   The system administrator of the cruiser entered 0 into the data field for the Remote Data Base Manager. This little action led to a divided by zero system fault and crashes on all LAN consoles.
   Gregory blames the system administrator as the root cause. However, this obviously looks like an engineering failure to me. The software is bad since it does not have input check.

b) Kevin Deaton seems to have the same view as I do. A good system should have input check. He also mentioned that system should be loosely coupled, which means system should be relative independent and should not crash just because other system feed bad data.

c) The system administrator is not the root cause. Using windows NT is not the root cause.
   The root cause is that the engineers(either software or firmware) did not have input check.

d) Again I dont think system matters in this special case. Dividing by zero led to system exception is something every embedded system engineer should know. No matter it is on Linux, freeRTOS or just simply Cyclic Executive, dividing by zero exception would always happen. This is probably just common sense that software engineers forgot to take into account.

ECEN 5623
Chutao Wei

Problem 3:

I decided to do my individual project on extended lab.

Platform and OS:
Raspberry Pi 4, Raspbian (Debian based system).

Time-Lapse Image Acquisition MINIMUM REQUIREMENTS:

3. 640x480
4. 1 Hz rate for image or 24 Hz for encoding continuous frames
5. Format in PPM P3
6. no error should happen in 2000 sec
7. Add timestamp and system name to all images

Time-Lapse Image Acquisition VERIFICATION MINIMUM REQUIREMENTS:

1. exactly 1800 frame in 30 mins. Tolerance is 1800 sec +/- 1 sec
2. Melting ice :) (is probably not very safe to put it on my laptop)
3. First Video: Melting ice and external clock use ffmpeg
4. Second video: Interesting thing either indoor or outdoor

Time-Lapse Image Acquisition TARGET GOALS:

11. Continuous download of frames over Ethernet(I might use WiFi) so that you can run indefinitely and never run out of space on your flash file system which should maintain only the last 2000 frames.

Time-Lapse Image Acquisition STRETCH-GOALS:

e) Run at a higher frame rate of 10 Hz with continuous download, this time for 9 minutes, which will produce up to 6000 frames (about 6GB uncompressed for 640x480) and repeat the jitter and accumulated latency verification at this higher rate.

ECEN 5623
Chutao Wei

Problem 4:
Provide all major functional capability requirements

Capability 1:
Take image every 1 sec and save in 640x480 .ppm format.

Capability 2:
Stream image every 1 sec to my Ubuntu laptop.

Capability 3:
Meet timing requirements within 30 minutes. Tolerance is 1800 sec +/- 1 sec.

Capability 4 (stretch goal):
Now change frame rate to 10 Hz and meet timing requirement for 9 minutes. Tolerance is 540 sec +/- 0.1 sec.

ECEN 5623
Chutao Wei

Problem 5:
Complete a high-level real-time software system functional description

Project Overview:

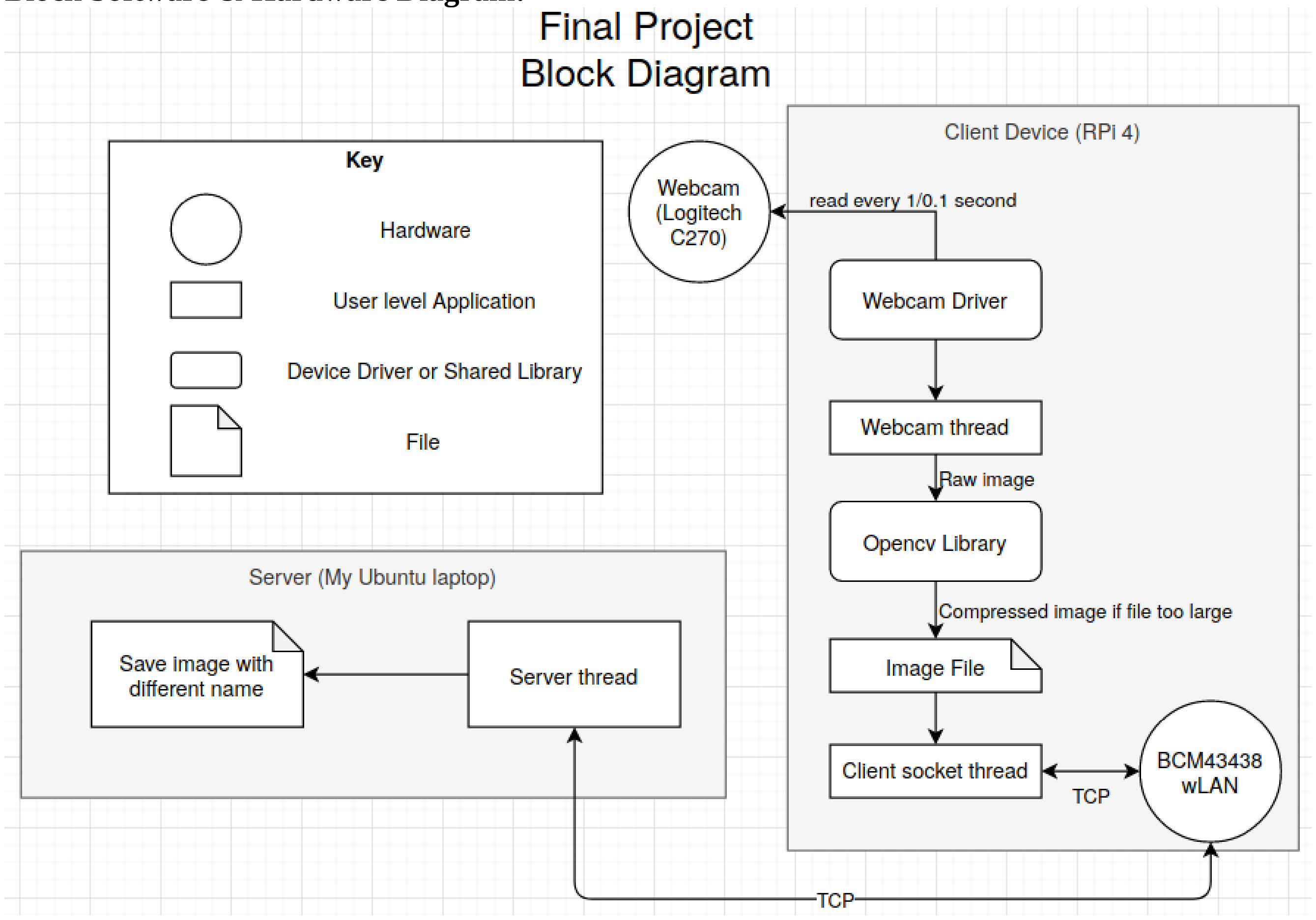
The Raspberry Pi 4(RPi 4) project should be able to take images and send the image over to my host computer every 1/0.1 sec.

Services are:

- S_1: Take image and save image
- S_2: Send image to my host using TCP/IP protocol

My host computer should receive it and store all images sent

Block Software & Hardware Diagram:



ECEN 5623
Chutao Wei

Problem 6:

Provide all major real-time service requirements with a description of each S_i including C_i, T_i, D_i

Services are:
- S_1: Take image and save image
- S_2: Send image to my host using TCP protocal

Computation Times are:
- C_1: TBD
- C_2: TBD

Periods are:
- T_1: 1/0.1 sec
- T_2: 1/0.1 sec

Deadkube are:
- D_2: 1/0.1 sec
- D_2: 1/0.1 sec

RMLUB:

$$\frac{C1}{T1}+\frac{C2}{T2}<2\times\left(2^{0.5}-1\right)=0.8284$$

In case $T_1 = T_2 = 1$:

$$C1+C2<0.8284\,sec$$

In case $T_1 = T_2 = 0.1$:

$$C1+C2<0.08284\,sec$$