

Fault Tolerant Computer Systems

FTCS 422

LAB Assignment

Lab Assignment 1

Individual Assignment

Due: week 3

Connecting two computer systems via a serial line is one of the most simple connections possible. Consider an embedded system (“EyeBox”) connected to a PC via a serial line (115,200 Baud):

Write a program on the EyeBot to:

- Read two potentiometers, connected to analog inputs
- Continuously transmit these values to the PC via the serial line.
- Data format is “%4d %4d\n”, so always one line of data with 10 characters is sent. Data values are 0 to 1023 . The data rate is about 10 packets per second.

Write a program on the PC to:

- Continuously read data packets from the serial line
- Each data packet represents a position value in 2D of a turtle graphics device (“etch-a-sketch”)
- Print the received data pair (or display graphically for an bonus point using fltk)
- Flag incorrect data (e.g. out-of-bounds, too big a change to previous value, etc.)

First, make sure your program is running correctly with the embedded system running without a fault. After that, make your PC program fault-tolerant in the sense that it will detect faults and will continue to run even if the embedded system will exhibit different faults. These faults could be in the line connection (e.g. serial line being disconnected/reconnected) or in the data being transmitted (incorrect data).

With your source code, submit a list of possible faults that could occur in the embedded system (or the serial line connection) and describe the approaches you have taken to overcome each individual fault.

Bonus point: Display the current turtle position graphically (using fltk) in a window, together with a message counter, time elapsed between two packets, and a “pause” and “exit” button.

Fault Tolerant Computer Systems

FTCS 422

LAB Assignment

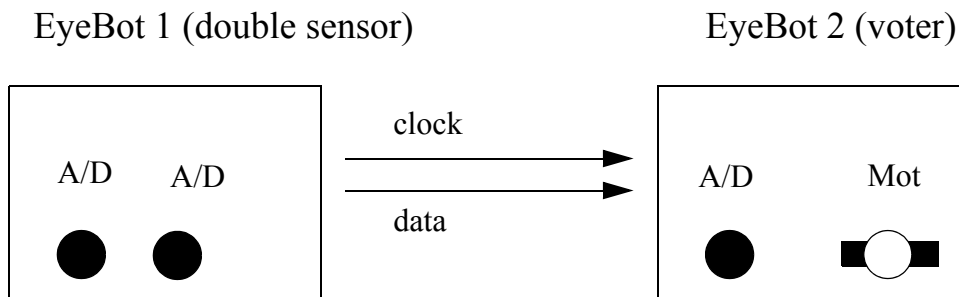
Lab Assignment 2

Group Assignment

Due: week 4

Implement a fault-tolerant temperature control system. This system will comprise:

- Three EyeBots with one **analog input** each (use potentiometer on experiment board).
*[Because of limited EyeBot numbers, we will use **two EyeBots**, one using 2 potentiom.]*
- One of the EyeBots doubles as **voter**. *[The one using only a single potentiometer.]*
- Temperature values should be digitised to 9 Bits range [0..511] before sending.
- The Eyebots are linked via 2 **digital lines** for **clock** and **data**.
Over these lines, implement a fault-detecting serial code with **even parity bit**.
- Use **Mid-Point Value** for the voter to display the correct temperature.
- For debugging purposes, each EyeBot should print its local temperature values - plus all the received ones for the voter.
- The voter EyeBot is connected to a servo (motor) for the valve of a cooling system. Adjust this motor according to the correct temperature (open valve for high temperature and vice versa).



Fault Tolerant Computer Systems

FTCS 422

LAB Assignment

Lab Assignment 3

Individual Assignment

Due: week 5

Connect an EyeBot to a PC and send text data from the PC to the EyeBot via the serial connection. The data needs to be encoded/decoded according to the following schema:

- On the PC: encode data using **7-4 Hamming** and **7-of-14** code (selectable by user)
- Use an fltk-Window with buttons to select between the two encodings, provide a text box for entering the string to be sent.
- On the EyeBot: read and decode data in either **7-4 Hamming** or **7-of-14** code
Use a KEY-button to select between the two codes

Note:

- Get your program working correctly without fltk first, then add it later. However, not using fltk at all will result in 20% points subtracted.
- A testing program (binary) for the PC will be provided, which generates correct data as well as data with single bit errors.

Fault Tolerant Computer Systems

FTCS 422

LAB Assignment

Lab Assignment 4

Individual Assignment

Due: week 6

Design a fault tolerant system that saves/loads one data file distributed on three disks, using RAID-4 single-bit error correction scheme (XOR two input words).

- Use a single workstations to store the files.
- Design program “**store**” to store a file.
This involves taking a file from the Unix file system, encoding it, splitting it into three segments, which are then stored as individual sub-files.
- Design program “**load**” to load a file.
This involves loading segments from 3 sub-files, decoding data (including error detection and correction), and writing a single data file back.

Notes

- Filenames: use “/tmp/ftcs.N” as a temporary file, where “N” is your student id number.
- You should be able to recover from a single disk failure.
- Your loading program should be able to report any errors single bit errors.

Marking

What we will do to assess your program:

- Write and read a file, comparing the two files using “diff”.
In this case no faults should be reported.
This may be tested on binary executable files.
- Corrupt one or two of the sub-files and re-test the program.

Bonus Point

Use sockets to run this application across multiple workstations.

Fault Tolerant Computer Systems

FTCS 422

LAB Assignment

Lab Assignment 5

Group Assignment

Due: week 7

Design a concurrent system that consists of a total number of 6 tasks as follows. Note that it is essential to use semaphore synchronization for exchanging data between tasks.

- Task 1: Left PSD
Read the value of the left PSD sensor and store the result in the output queue.
- Task 2: Right PSD
Read the value of the right PSD sensor and store the result in the output queue.
- Task 3: Camera
Read camera image and calculate the average gray scale value along the main diagonal and store the result in the output queue.
- Task 4: LCD output
Continuously read the output queue and print the text for each entry to the screen.
The interface between tasks 1..3 (“writers”) to task 4 (“reader”) is via a global queue, to make sure output requests are executed in the order they are submitted.
Each queue entry consists of a text position (x,y) and the actual text string (max. 16 characters, terminated by a NULL character).
- Task 5: Watchdog
Implement a watchdog task that checks whether tasks 1..4 are still running. Print an error message if one of the tasks have terminated, indicating which task has failed.
- Task 6: Testing
Pressing KEY1 .. KEY4 should terminate task 1 .. task 4, resp., for testing purposes.
A task can be terminated by calling `OSkill` with the corresponding task number. These task numbers have to be kept in a global array when starting the tasks with `OSspawn`, so the testing task has access to them.

Bonus point: Automatically re-start a terminated task.

Fault Tolerant Computer Systems

FTCS 422

LAB Assignment

Lab Assignment 6

Group Assignment

Due: week 8

Implement a fuzzy logic controller for driving a LabBot parallel to a curved wall at a desired velocity, independent of its load (e.g. going up or down a slope).

Part 1

Start with a fuzzy controller for a **single wheel**. Feed back the difference between desired and actual velocity (error) and the wheel acceleration (rate of change, error_dot) to the fuzzy controller.

Adjust the fuzzy parameters until system performance is satisfactory for a range of different speeds and various changes between different speeds.

Part 2

Extend the fuzzy controller from part 1 to a fuzzy controller for driving a vehicle with two driven wheels in differential drive mode. The goal is to drive the vehicle at a constant speed at a **set distance from a wall** to the left of the robot.

This requires to run e.g. the left wheel at a constant speed and to change the speed of the right wheel depending on the changes in distance to the left wall.

Smooth wall curvatures can be expected; there will be no sharp corners.

Note that because of manufacturing tolerances, different loads, and opposing driving directions of the two motors, the robot's two motors may run at slightly different speeds when set to the same speed value.

Fault Tolerant Computer Systems

FTCS 422

LAB Assignment

Lab Assignment 7

Individual Assignment

Due: week 9

Consider the programming language “*Drive*”, defined in EBNF:

program	=	{ statement “;” } .
statement	=	TIMES constant DO program END IF psd “<” constant DO program END STRAIGHT BACK LEFT RIGHT .
psd	=	PSD_FRONT PSD_LEFT PSD_RIGHT .
constant	=	digit {digit} . (* max. 5 *)
digit	=	‘0’ .. ‘9’ .

Sample “Drive” program:

```
TIMES 4 DO STRAIGHT; RIGHT; END;  
LEFT; LEFT;  
IF PSD_FRONT < 40 DO STRAIGHT; END;
```

Part 1: Implement a scanner for the language symbols and constants in Drive

Print the input program as a sequence of tokens.

Part 2: Implement a recursive descent parser for Drive

The parser calls the scanner and checks the syntactical correctness of the program. Error are to be reported – parsing stops at the first error encountered.

Note: White space (“ ”, “\n”, “\r”, “\t”) may be contained between successive tokens.

Bonus point: Extend the language to handle variables.

Hints

1. Implement a function to return the current input symbol `symboltype CurSymbol();`
2. Use scanner from previous lab to advance one symbol `symboltype NextSymbol();`

3. Use function “error”:

```
void error(char *s)
{ printf("ERROR: %s\n", s); /* print error message */
  myexit();                /* terminate program */
}
```

4. Use function “check” to see if a specific keyword is present:

```
bool check(symboltype s)
{ if (s == CurSymbol())
  { NextSymbol(); return TRUE; } /* symbol is present */
  else return FALSE;           /* symbol is not there */
}
```

5. Use function “accept” if a specific keyword has to follow:

```
void accept(symboltype s)
{ if (!check(s)) error("wrong symbol"); /* must accept */
}
```

Fault Tolerant Computer Systems

FTCS 422

LAB Assignment

Lab Assignment 8

Individual Assignment

Due: week 10

Implement a remote control system that drives the robot from a PC. Use fltk to design a graphical user interface.

- Implement buttons for: forward, left, right, backwards.
The robot should drive a short distance (10cm) for forward backward and turn a small angle (30 deg.) for left/right.
- Implement a text **box** for entering a robot program in language “**Drive**” (see previous lab).
- Implement a “**compile**” button to check the program for correctness and generate the proper command sequence
- Implement a “**run**” button to send the compiled sequence of driving commands to the robot
- Implement a “**stop**” button to stop command sending immediately and stop the robot.

On the Robot:

- For every driving command, use the PSD sensors. Do not execute a driving command if a minimum safety distance is breached.

Fault Tolerant Computer Systems

FTCS 422

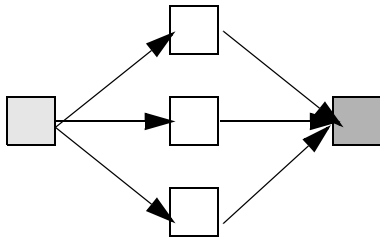
LAB Assignment

Lab Assignment 9

Group Assignment

Due: week 11

Implement a distributed system comprising five PCs, communicating via sockets. ASCII data from one source is being relayed via three PCs to one destination. (*Primitives for socket communication will be provided*).



- For every message, create a unique tag (integer) at the source PC.
- Each message from the source PC is to be sent (with its tag) to each of the three intermediate PCs.
- The destination PC has to print each incoming message, but has to filter out duplicates which it identifies by the message tags.
- Tags need to be created in ascending order. That way, the receiver PC does not have to store any tags. It only prints messages with a higher tag than the previous one.

Notes

- Each group will be assigned two unique socket numbers.
- Each intermediate PC needs to print out received messages for debugging and marking purposes

Bonus point:

- Create a nice fltk user interface

Fault Tolerant Computer Systems

FTCS 422

LAB Assignment

Lab Assignment 10

Group Assignment

Due: week 12

Mastermind

The game of mastermind is played with two players, the mastermind and the codebreaker. Initially, the mastermind sets a code which consists of a sequence of four numbers ranging from 1 to 8, without repetition.

Examples of legal codes are:

- 2, 3, 8, 7
- 5, 1, 2, 8

Once the mastermind has set the code, the codebreaker's task is to deduce the code within six attempts. If the codebreaker can achieve this, then the codebreaker wins, otherwise the mastermind wins.

Each time the codebreaker makes a guess, the mastermind is obliged to respond with feedback. The mastermind starts at the first position of the code and checks the corresponding position within the guess. If the code number matches exactly with the guess number, the mastermind adds a black to the return. If not, the mastermind checks the remainder of the guess for the number and if it exists, adds a white to the return. This procedure is repeated until all four positions have been checked. The mastermind then returns the feedback.

Examples of feedback:

Code	Guess	Feedback
1, 4, 5, 8	1, 5, 4, 7	1 Black, 2 White
1, 4, 5, 8	2, 4, 7, 5	1 Black, 1 White
1, 4, 5, 8	2, 3, 6, 7	Nothing
1, 4, 5, 8	1, 5, 4, 8	2 Black, 2 White
1, 4, 5, 8	1, 4, 5, 8	4 Black (breaker wins)

Lab Assignment

Implement a client and server program which allows a user to play mastermind, using all the techniques and skills you have learnt in this course. Use sockets to communicate across the internet like in the previous lab. All communication between the client and server should be encoded using a 7/4/3 hamming code. Please include a testing facility to allow the demonstrator to change bits.

After building the client and server, you should be able to interface with anyone everyone else's program because a uniform protocol is employed (see below). Non-compliance to protocol should generate an error message.

The client

The client should interface with the player. In particular, it has the following requirements:

- Nice-looking fault-tolerant GUI which handles ignorant player input is essential.
- Connect to any mastermind server. The user should be able to change IP address and port.
- Allows the user to enter a nickname (4 chars exactly) and displays the game time.
- Enter guesses and receive feedback. The guess should take the format "Gnnnnddd" where the n represents a char (nickname) and d represents a decimal digit (guess).

The server

The server should handle up to 3 games simultaneously. That is, 3 people should be allowed to play with the same server at any given time. If more people want to join in, an error message can be returned to the client by sending "F" to indicate that it is full. In particular, the server has the following requirements:

- Set a code for each game. This should be generated randomly.
- Process guesses and return feedback. Feedback should take the form "Add" where the first digit represents the number of black and the second digit represents the number of white.

Bonus Point

Incorporate a "hall of fame" feature to the client and server:

For the client: Request and display the hall-of-fame. This should message the server with "H". When the server replies, it should display it in a nice format to the player.

For the server: A hall-of-fame should be maintained recording nicknames of the top 3 games sorted by guesses taken. Each time the hall-of-fame is updated, it is to be saved to a file on the server's hard-drive. Return hall-of-fame data to a requesting client. The format should be "Wnnnnddd", where n is the nickname of the top player and "ddd" are the scores of the three top entries.

ASIDE – read the following if you are interested in this topic

[James Ng]

How is mastermind similar to web-browsing? Consider a typical transaction in IE browser and mastermind client:

- Mastermind: button click to send message to server. IE: click on link to request HTML file from HTTP server.
- Mastermind: client prepares protocol compliant message to send to server specified by IP. IE: prepares HTTP protocol compliant request and translates www name into IP using a domain name server (DNS) lookup.
- Mastermind: client sends message to server via sockets. IE: sends message to server via sockets.
- Mastermind: Server receives and processes request. IE: HTTP server finds requested file.
- Mastermind: Server sends back protocol compliant request via sockets. IE: HTTP server sends requested file back in HTTP via sockets.
- Mastermind: Client receives message and displays it in a nice manner to client. IE: Client receives file. If file is HTML, then IE displays it in a nice manner to client. Can view the html file by “view source” in IE.

Note that the HTTP server does not have to be finding a file which already exists, although that was the traditional way when the internet was first born. It can do anything it likes, but if it wants to display something on the IE browser, it must return a HTML file. Doing anything it likes gives rise to things such as CGI, which can dynamically create new HTML files depending on the request it receives.

Trapdoors

For those of you who are interested in computer security, here’s a thought. What if, against my lab specifications, you decided to program the server to respond to the message “Xnnnd”. When a client sends this message to the server, the server updates the hall-of-fame with *nnnn* as nickname, *d* as number of attempts.

No harm is done in this instance, but imagine if you are a bank which bought a program to run their banking systems. Would you like it if the developer could enter a message and change the amount of money in their account circumventing all protection mechanisms? This sort of attack is called a trapdoor and it can also be performed through software patches and upgrades.

How can you defend yourself against this if you are a buyer of off-the-shelf software? You can’t use a firewall because it will see that communication by that process through its authorized port as legitimate. You can’t use anti-virus because its not smart enough to know that that code is “bad”.

If the developer is unwilling to supply you their source code, then it’s very difficult. The only way you could possibly discover it is if you disassembled the executable file and found that small piece of code, but that’s like looking for a needle in a haystack. You’re essentially defenceless and simply trusting the developer not to implement a trapdoor.

If you have the source code, it’s very much easier to check but still time consuming, especially if they have coded very sloppily. But developers do not like giving up their source code. If you want it, you’d probably have to pay a decent amount more, plus sign non-disclosure agreements. How many pieces of software have you bought which offer the source code along with it?

For those of you who study commerce, this is agency theory at its finest and the principal will price-protect itself to compensate for this problem, just as shareholders price-protect themselves against managers who act in their own interests.