# P20

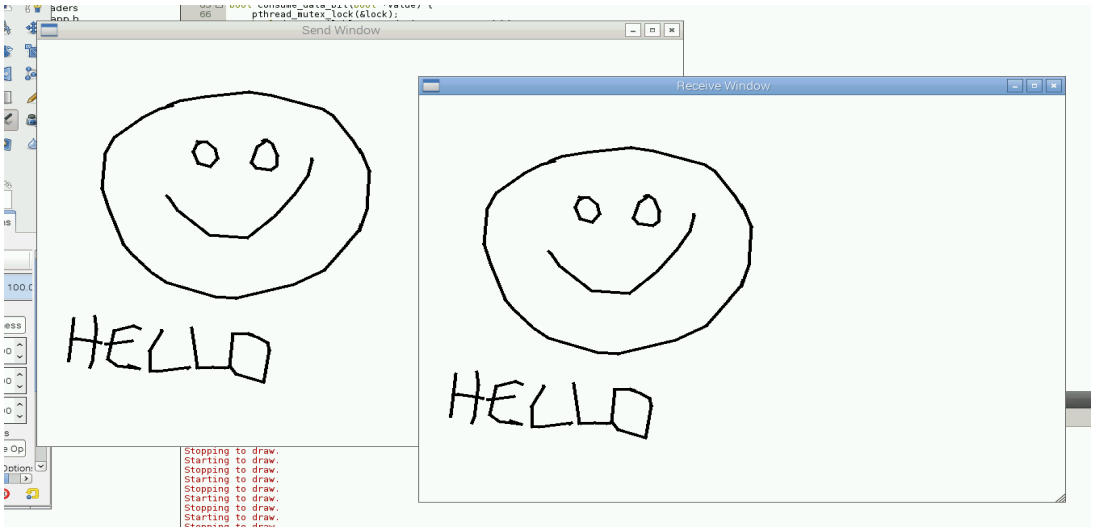
## Whiteboard chat for the Raspberry Pi

This project accounts for 30% of your final mark. You will work on this project with your Lab partner.

By the end of this project you should have developed an interactive whiteboard client/server that pairs of users can use to view pictures, shapes, etc., drawn by each other on their respective Raspberry Pi's. This will involve implementing a suitable GUI for the send and receive windows, devising a protocol to exchange drawings and physically communicating using the GPIO pins.

Students are expected/allowed to work on this project before and in between the lab sessions, where you can come for the lab sessions for further guidance.

**When you complete the project, please submit your code, scanned copies of your logbook and any results collected on a word document by 19th May 2022 here:** https://handin.ecs.soton.ac.uk/handin/2122/ELEC1204/1/ These will be used to moderate your marks awarded in the lab.




See a video of a potential solution

## Schedule

Preparation time : 39 hours

Lab time : 6 hours

## Items provided

Tools : None

Components : 7 jumper leads, Raspberry Pi 3 Model B+ and 16GB Micro-SD card (from P6 and P7)

Equipment : HDMI capable monitor

Software : Same Raspbian image file as P6 and P7

## Items to bring

- Essentials: A full list is available on the Laboratory website at https://secure.ecs.soton.ac.uk/notes/ellabs/databook/essentials/
- A USB stick for transferring files between Pi's.
- 2x Raspberry Pi
- 2x Raspberry Pi SD card (kept from P6)
- 2x Raspberry Pi power supply
- 2x Raspberry Pi HDMI cable
- 2x Raspberry Pi keyboard
- 2x Raspberry Pi mouse

i.e., Each pair needs 2 Raspberry Pi's for the final phase of this lab.

This project accounts for 30% of your final mark. You should spend up to 39 hours preparation and 6 hours in the lab. You will work on this project with your Lab partner. You are expected to work on this project before and in between the lab sessions, where you can come for the lab sessions for further guidance.

> **Academic Integrity** – *If you undertake the preparation jointly with other students, it is important that you acknowledge this fact in your logbook. Similarly, you may want to use sources from the internet or books to help answer some of the questions. Again, record any sources in your logbook.*

## Revision History

| | | |
|---|---|---|
| February 2015 | Tristan Aubrey-Jones and Mohammed El-Hajjar | First version created |
| February 2017 | Mohammed El-Hajjar | Update for Raspberry Pi 3 |
| February 2018 | Mohammed El-Hajjar | Update for further clarifications |
| February 2019 | Mohammed El-Hajjar | Update for Pi 3 Model B+ |
| March 2022 | Michail Pligouroudis | Update handin link |

# 1 Aims, Learning Outcomes and Outline

This project aims to:

- Develop confidence designing larger C++ applications which use multiple libraries.
- Develop an understanding of communication protocols and how the various layers of network applications (and the hardware stack) work.
- Get the opportunity to use advanced C++ features like abstract and template classes.

Having successfully completed the project, you will be able to:

- Design your own communication protocols/network-based applications.
- Use threads in C++ and use mutexes to make thread-safe collections.
- Use the GPIO interface on the Raspberry Pi.

The objective of this project is to design and develop an interactive whiteboard chat client/server application, where pairs of users can use their Raspberry Pi's to draw diagrams and exchange them remotely by connecting various pins of the Pi's GPIO interface. You will do this by designing, developing and testing the application a layer at a time.

The Qt5 tutorial at http://zetcode.com/gui/qt5/ and documentation at http://doc.qt.io/ will again be useful. Documentation for the STL classes and collections can be viewed at http://www.cplusplus.com/reference/. A good tutorial for the pthreads library (which you will use to create threads that send and receive over the GPIO interface) can be found at https://computing.llnl.gov/tutorials/pthreads/. Finally, you will be using the http://wiringpi.com/ library to read and write to the GPIO pins on the Pi and will need to know the pin wiringPi GPIO pin numbering scheme for the B+ Pi's at http://wiringpi.com/pins/.

# 2 Background and design

## 2.1 Read up on threads and mutexes

You will need to use multiple threads in your client/server to respond to window events (the main thread) as well as handling sending and receiving data over the GPIO interface. You should therefore:

- Read up on threads and the pthreads library.
- Read about race conditions and how to use locks/mutexes to make a collection "thread-safe".

## 2.2 Read up on using the GPIO interface

You will need to use the GPIO interface and wiringPi library to pass drawing-commands between Raspberry Pi's. You should read up on how to do this. (**You will be manually sending the data as a bit stream, so you don't need to know about SPI or I2C etc**.)

## 2.3 Design your application

Your application will have a "send" window that allows the user to draw on it (by catching mouse and/or keyboard events) and a "receive" window that will display drawings coming over the GPIO interface from the other user as they are being drawn. This drawing must at least involve drawing line diagrams and clearing the screen. To implement this functionality you must design several aspects (or layers) of your application, and therefore must answer the following questions:

- How will the send-window allow users to draw diagrams? How will it display diagrams as they are being drawn, and how will it retain these diagrams so that they don't disappear when the window is repainted?
- How will you represent the drawing commands so that they can be sent to the other user whilst they are being drawn?
- How will you serialize these commands into packets to be sent over the GPIO interface?
- How will you use threads to send and receive these packets, while the rest of the application keeps running? How will you use mutexes to make any relevant collections "thread-safe"?
- How will you convert binary packets into a stream of 1's and 0's? How will you control the GPIO pins to transmit this stream in a reliable way? (You can use multiple GPIO pins for this). For example, you may need to signal when a bit is ready to be read, and when the remote Pi has finished reading the current bit.
- How will you receive and buffer packets at the other end? How will you deserialize them? How will you draw them on the receive window? How will you retain the currently received diagram so that when the window is repainted the diagram isn't lost?

## 3 Implementation

To start this project, connect up your Raspberry Pi, start Qt creator and open the "lab20" project", which you can find on the Pi in '/home/pi/Documents/ELEC1204/P20'. Until the last step in your application you will simply display in the receive-window what is being drawn in the send-window. This will allow you to test your application one layer at a time, until when all the layers are working, where you can connect your two Raspberry Pi's together. It is recommended to start with the GUI/application layer and then move down to the physical layer, rather than the other way around. This can be done using the following steps.

### 3.1 The GUI send and receive windows – 5 marks

Make a Qt application with two windows (or two parts of the same window), one send/local window that the user draws on, and another receive/remote window that displays the diagram coming from the other user. Implement various "drawing command" classes for the different drawing actions you support (at least line drawing and screen clearing). Make your send window respond to mouse/keyboard events, create these commands, and display them locally so that they don't disappear when the window is repainted. Pass these commands to the receive window when they are created, and make the receive window display them, so that they don't disappear when it is repainted.

### 3.2 Serialize and deserialize drawing-commands – 5 marks

Write code that serializes (i.e., transforms into binary form like arrays of chars) these commands, and deserializes them. Change the send and receive windows so that commands are serialized and deserialized when passed from send to receive.

### 3.3 Implement send- and receive-threads – 5 marks

Implement send and receive threads, where the send thread takes serialized commands and will send these over the GPIO, and the receive thread will read data from the GPIO, and pass them to

the receive window. **Don't pass the data over the GPIO yet. For now, just test this by passing serialized commands using a queue. You may want to implement a thread-safe queue template class to do this.**

### 3.4 Implement your communication protocol using booleans – 8 marks

Before actually using the physical GPIO interface, implement and test your bit-stream communication protocol, by toggling shared Boolean variables instead of writing to GPIO pins. Remember you need to think about how to signal when a bit is ready to read, and when it has been read by the remote Pi when you do this.

Hint: you can achieve this with 3 jumper leads in each direction (i.e., 3 to connect a Pi's output to its own input, or 6 to connect two Pi's together). You may need to use mutexes to avoid race conditions.

### 3.5 Read and write to physical GPIO pins – 7 marks

Change your send/receive threads to use the wiringPi library to actually write to the GPIO pins. To start, connect your Pi to loopback (i.e. connect the send pins to the receive pins on the same Pi). Ideally this would be done via resistors to protect the Pi's if two pins are accidentally connected together that are both in output mode. Note that you will need to open a terminal and run your application using "sudo" to give it permission to use the GPIO pins. Once this is working, put your whiteboard application on your neighbour's Raspberry Pi (using a USB stick or by saving it on the SD card), and connect the Pi's together. You will need to connect the GPIO grounds together, but be careful not to connect the 5V or 3.3V pins to each other. Test whether your application works when connected to another Pi.

## *4* Optional Additional Work

*Marks will only be awarded for this section if you have already completed all of Section 3 to an excellent standard and with excellent understanding.*

- Implement more drawing features and commands, e.g., colours, text etc.
- Make your communication protocol more robust, so that you can disconnect and reconnect GPIO pins during transmission. This may involve:
  - Resetting communications after send/receives have got out of sync.
  - Using parity/check-sums.
  - Use proper handshaking so that whole commands/packets are not discarded until they have been acknowledged by the remote Pi, and so that resends are attempted when data is lost.