Umbrella Rental System

By

Shuodong Zhang

Xinyi Wu

Yiheng Xu

Final Report for ECE 445, Senior Design, Spring 2017

TA: Kexin Hui

3 May 2017

Group No. 72

Abstract

Sudden rains frequently ambush students and faculties without an umbrella in our campus area. In such event, buying a new umbrella is neither convenient nor economical. We present a low cost umbrella rental system to help those people without an umbrella in time of need. In this paper, we focus on design, implementation and verification of our umbrella rental system. A discussion on cost, ethical considerations and future work is also included.

Contents

1. Introduction

1.1 Motivation and Purpose

Sudden rains cause discomfort to any people without an umbrella. Buying a new umbrella is both inconvenient and costly: there may not be a shop that sells umbrella nearby, and buying a new umbrella each time a sudden rain falls is a waste. Our umbrella rental system aims to provide an economic and convenient alternative to buying new umbrellas. For the convenience of our users, we should allow users to return an umbrella to a different location; this implicitly requires our design to allow multiple umbrella racks to work together. Noting that umbrellas are prone to damage, we must keep track of the damaged umbrellas so that they will not be rented again until repaired.

1.2 Features

- Automatically choose an undamaged umbrella for rent, and an empty slot for return
- Authenticate users and umbrellas using RFID tags
- Allow a user to report damages to the umbrella
- Prevent a user that has damaged an umbrella from renting temporarily
- Multiple umbrella racks connected to one server
- High reliability which allows 24/7 operation, even with unstable network

2 Design

2.1 Design Overview

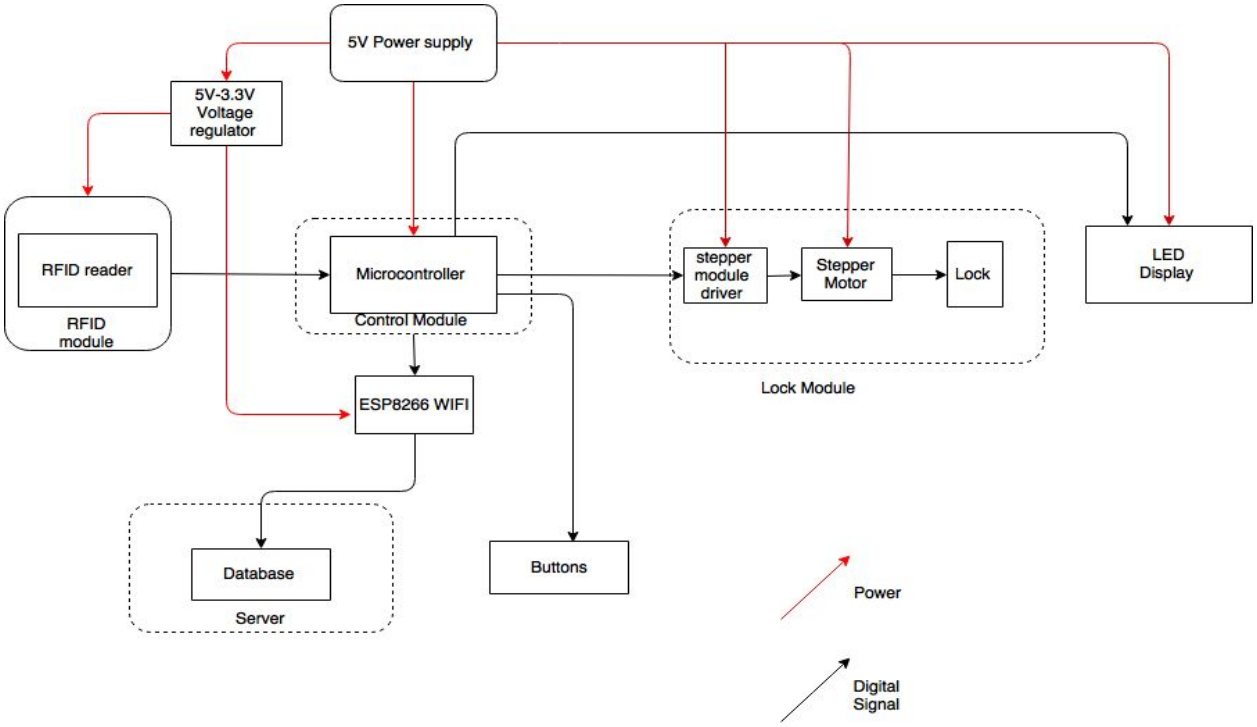

Figure 1: System Block Diagram

Our project consists of 7 modules: Power module, LCD display, Wi-Fi module, RFID module, lock module, control module, and server software.

2.2 Module Details

2.2.1 LCD Display

The LCD display unit contains 1602 LCD Display Screen with backlight adjust. The screen takes input from microcontroller and display the required content on it. In our case, we use it to display the current status of the whole system, user id, wifi communication status and damage report selection.It only reads the ASCii code data, so we need to transfer the raw data to ASCii code in order to make it display correctly.

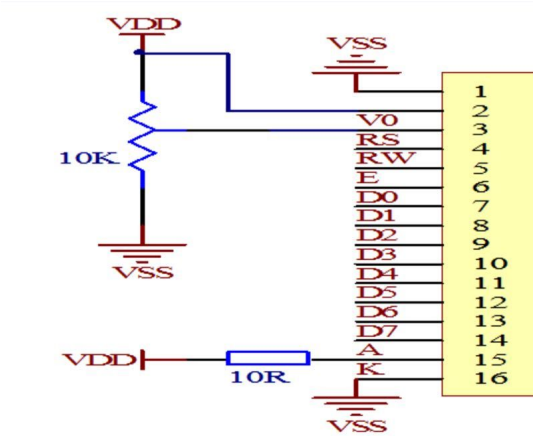

Figure 2: LCD pin assignment

1602 LCD display is powered by 5v DC voltage from the power pin on the PCB. It can display 16 characters per line and 2 lines maximum. Also, we could adjust the backlight of display by turning the 10 kΩ variable resistor.

Figure 3: 1602 LCD display

2.2.2 Power Module

Our Power module uses a 5V 2.1A wall power adaptor to connect between plug and our pcb. It provides stable 5V to support our circuit. Our Microcontroller, Motor driver, stepper motor and Lcd display all requires 5V supply, so we can directly wire their power inputs to the power pin on our pcb. Due to specifications of our RFID sensor and Wifi chip, we have to use a 5V-3.3V voltage regulator to step down the supply in order to power them safely.

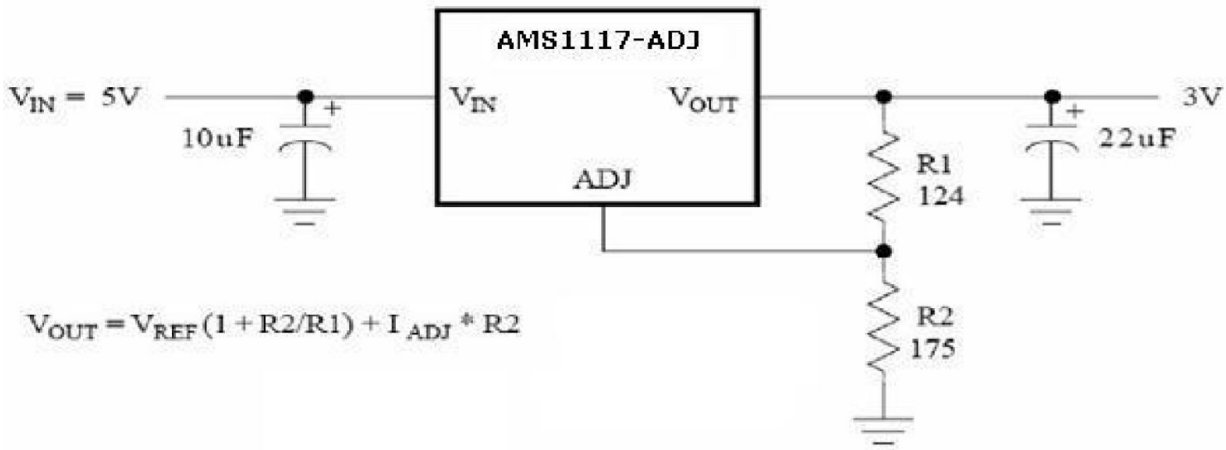

Figure 4: 5V to 3.3V Schematics [2]

2.2.3 Wi-Fi Module

The Wi-Fi Module handles the communication between the microcontroller and the server. It consists of a ESP8266 wireless adapter. The ESP8266 chip communicates to the microcontroller using serial interface. It is connected to the internet by joining in a local wireless network. Through this internet connection it is able to connect to our server located at a static public IP.

The ESP8266 chip requires 3.3V DC input as its power. Its RX/TX pins operates at 3.3V under normal operations.[1] Our microcontroller operates under 5V power supply, which gives a measured 4.2V at its TX pin. After repetitive testing and debugging over a one-week period, we decide to directly connect TX pin of the microcontroller to RX pin of ESP8266 chip, and use a pull-up resistor to set the signal from TX pin of ESP8266 to RX pin of microcontroller to 5V, as shown in Figure 5.

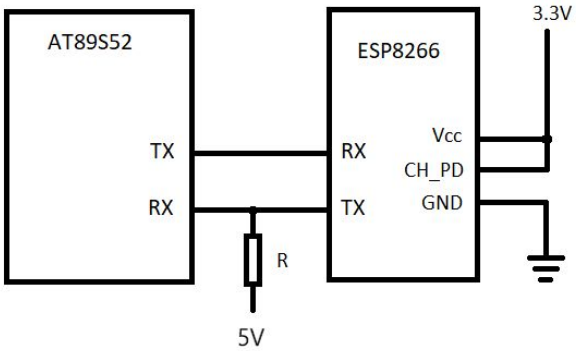

Figure 5: Interfacing ESP8266 with AT89S52

2.2.4 Lock Module

The lock module consists of three main components: BYJ-48 Stepper Motor with ULN2003 Darlington Transistor, mechanical motor-to-lock setup, and DEMUX IC for minimal microcontroller pin usage.

The BYJ-48 motor used to drive the lock is a 4 phase 5V DC stepper motor. It takes four input signals from microcontroller, first passed through ULN2003 Darlington Transistor to amplify currents, as seen in

the figure below.

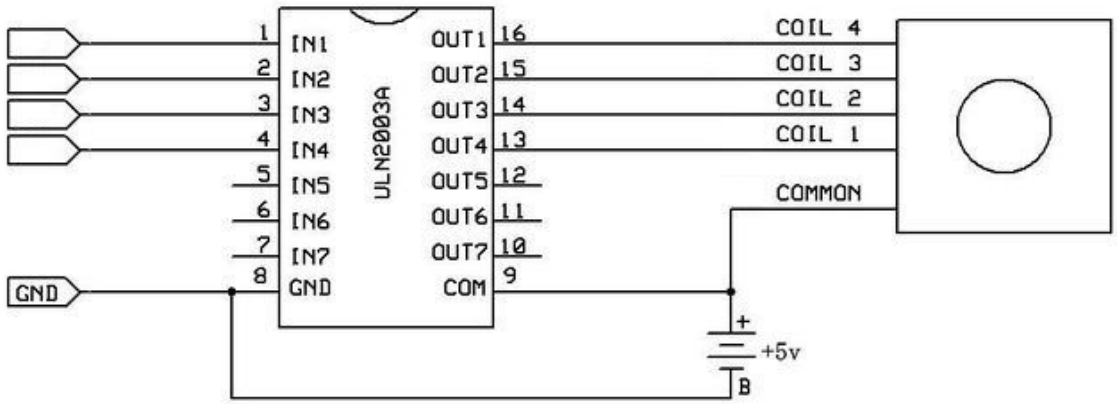

Figure 6 : BYJ-48 Stepper Motor with ULN2003 Setup [3]

The stepper motor is controlled by waves of input pulses from the microcontroller. Each set of pulse includes 4 steps, and one pulse will rotate the motor by 5.625°. The input pulse is shown in Table 1:

Step	Port Data	Pin 3	Pin 2	Pin 1	Pin 0
1	0x03	0	0	1	1
2	0x06	0	1	1	0
3	0x0C	1	1	0	0
4	0x09	1	0	0	1

Table 1: Stepper Motor Input Pulse

Because of this the stepper motor can have precise control of rotation. And for reverse direction simply send the 4 signal steps in reverse order would make the motor turn in reverse. Speed can be controlled by adjusting frequency (delay between each pulse signal) of the input pulse.

Next is the mechanical setup between lock and the motor. Our lock is modified from existing umbrella lock driven by manual key rotations. The final mechanical design lets the motor rotate a tension wire connected to a secondary spring loaded trigger that connects to the actual lock arm axle (also spring loaded), when tension wire is pulled back by the motor, the secondary trigger pulls the lock arm axle triggering it to eject outward releasing the umbrella. Then, motor turns in reverse direction and the secondary trigger resets back to “lock” position by its spring, allowing lock arm axle to be locked in place again.

Last component of the Lock Module is the Demultiplexer group. Normally each motor requires 4 signal

pins from Microcontroller, and there would normally be 10 to 20 umbrella locks on each umbrella rack. Solution to relieve pin usage is to dd select pins and demultiplexer. Since only one motor will be operating at a time, only one set of 4 motor signals is needed. Through implementation of demultiplexer IC and select pin, the exact motor would be selected to receive the signal pulse and all other motors would receive null signals thus halted. As seen in table below, usage of DEMUX can drastically reduce signal pin usage.

	4 Signal Pin per Motor V.S. 4 Signal Pin with DEMUX Select	
Pins Needed	4*n	4 + x
Controllable Motor	n	2^x

Table 2: Motor Pin Usage

The multiplexer used is Texas Instrument 74AC139 DUAL 2 to 4 DEMUX. Each of the 4 motor signals will go through one DEMUX and the select bits are shared. The following is the truth table for 74AC139:

FUNCTION TABLE
(each decoder/demultiplexer)

INPUTS			OUTPUTS			
$\overline{G}$	SELECT					
	B	A	Y0	Y1	Y2	Y3
H	X	X	H	H	H	H
L	L	L	L	H	H	H
L	L	H	H	L	H	H
L	H	L	H	H	L	H
L	H	H	H	H	H	L

Figure 7: 74AC139 Truth Table [4]

However as seen from table above, the 74AC139 will set the selected output to LOW and others to HIGH, thus requiring all signals to go through inverter gates before feeding them to the motors. Therefor final schematics of DEMUX and inverter group is shown below:

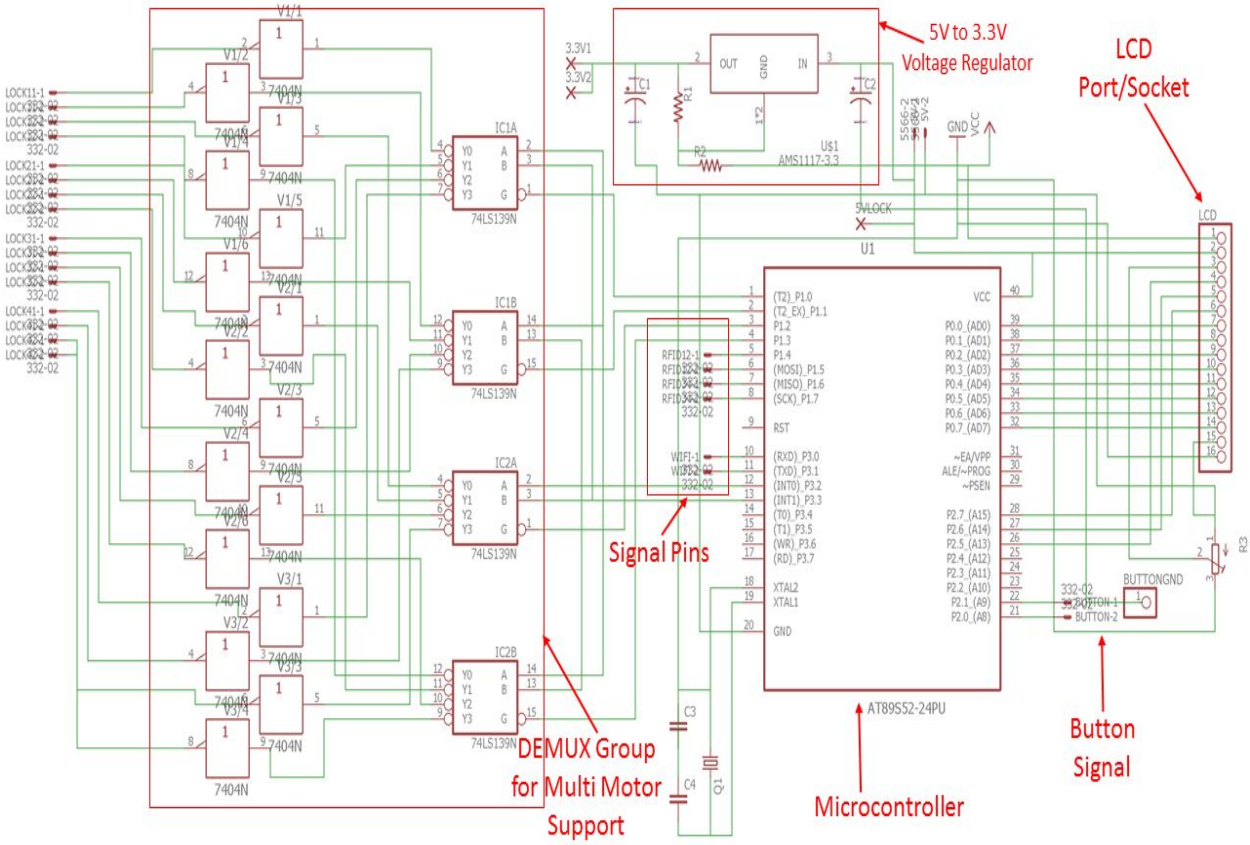

Figure 8 : PCB Schematics for DEMUX Group

2.2.5 RFID Module

Our RFID module is a RC522 RFID read sensor. It reads RFID tags/cards and output their datas(user ID for user card, umbrella ID for umbrella RFID tag) as digital signal to Control Module over SPI interface. The Id information will also show on display. Our RFID sensor works at 13.56MHz and works at 3.3V DC. We have to use a voltage regulator to step down the voltage. After actual test, the sensor read range is about 7cm which means it will not accidentally read RFiD when people walk pass by. There is no delay between user scans rfid and the id shows on the display.

Figure 9 : RC522 Sensor figure

Blow is the working flowchart of our RFID module:

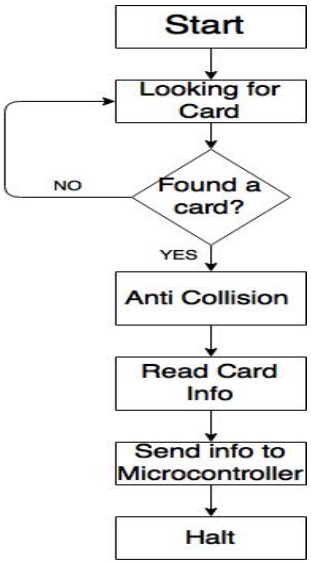

Figure 10 : RC522 working flowchart

2.2.6 Control Module

Our control module consists of an AT89S52 microcontroller. It is powered by a supply of 5V. It communicates with the LCD screen through GPIO pins, with RFID module through SPI interface, and with Wi-Fi module through serial interface. The pin assignment is shown in Figure 11.

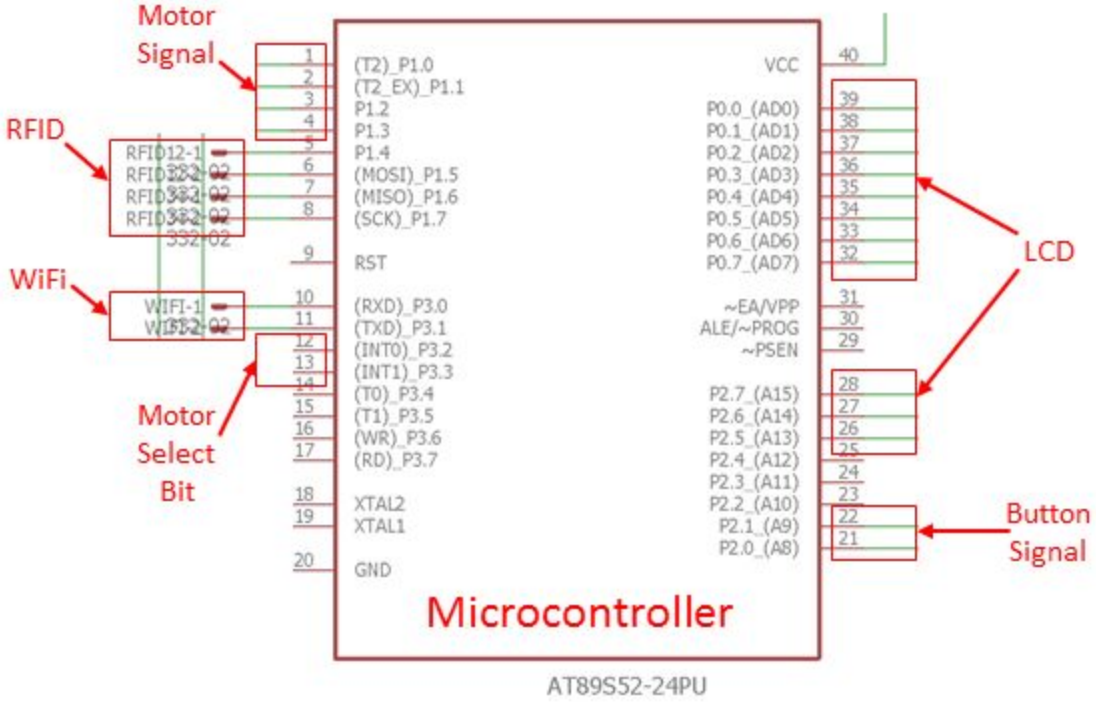

Figure 11: Microcontroller Pin Assignment

The operational flowchart of the microcontroller is shown in Figure 12.

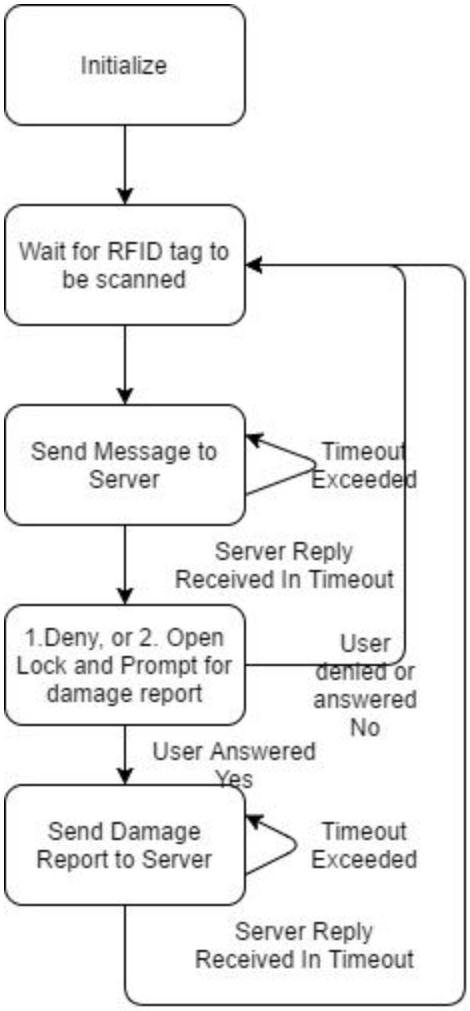

Figure 12: Control Module Flowchart

The damage report described in the figure above has two types. The first one is sent if a user reports damage immediately after renting an umbrella. In this case, we consider the damage to be caused by the previous user. The second one is sent if a user reports damage after returning an umbrella. In this case, we consider the damage to be caused by the current user. To report damage, the user has to press one of the two buttons connected to the microcontroller, which stand for “yes, report” and “no damage”, respectively.

2.2.7 Server Software

Our server software is a socket server program written in Python. It collects user inputs from the racks in the form of 16-Byte messages, records the status of the system in a MySQL database and gives instructions to each rack in 3 Bytes. It consists of a multithreaded socket program written in Python and a MySQL database. Figure 13 is an overview of the our server program.

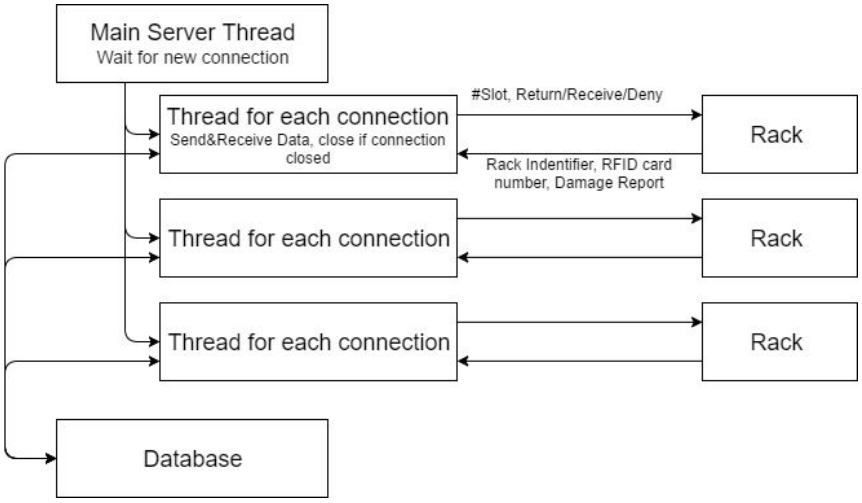

Figure 13: Server Overview

We choose to create a thread for each connection instead of using polling. This design choice gives our system extra robustness as each exception thrown in each thread will only terminate the thread, not the entire server.

The operational flowchart of each thread is shown in Figure 14.

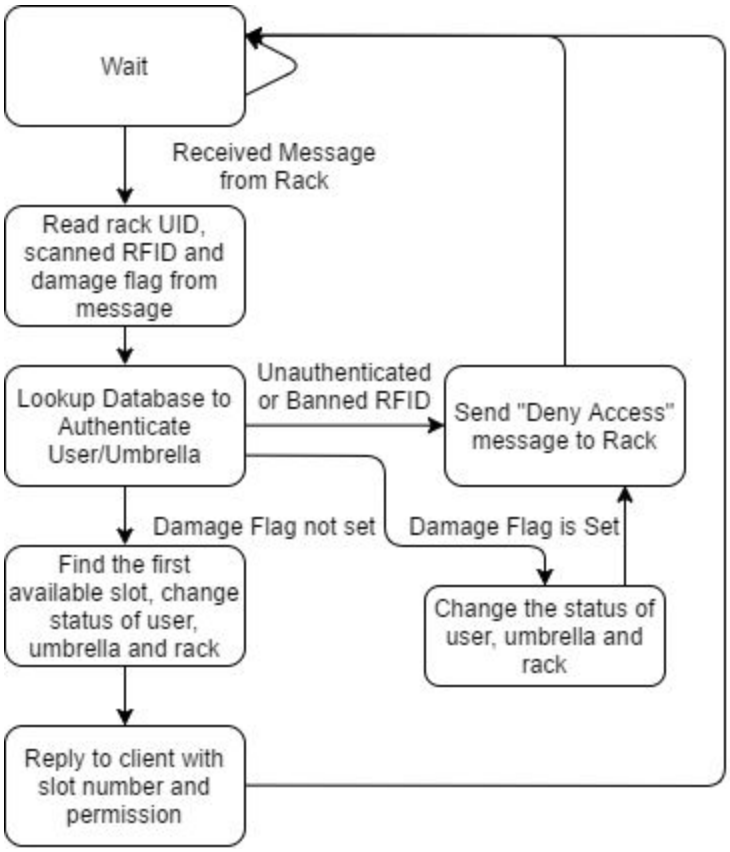

Figure 14: Server Flowchart

The source code of our server is included in Appendix B for reference.

2.3 Design Alternatives

As both the RFID module and the Wi-Fi module requires 3.3V power supply, an alternative is to power the microcontroller using 3.3V. However, our LCD screen must be powered by 5V, and its 11 data pins will need to be pulled up if microcontroller is powered by 3.3V. To simplify our circuit, we choose to power the microcontroller by 5V.

In our final design, the user that reports damage caused by a previous user is temporarily banned as well. This design decision may sound troublesome, as a user is banned without doing damage himself. Our reason to adopt this design is the security of the system. Assume that we do not ban such users temporarily. In this case, a malicious user can try to rent umbrella from each of the available slots on a rack with multiple attempts, reporting damage each time. By doing this, the malicious user can prevent any umbrellas on the rack from being rented, and the innocent previous users will be banned. Foreseeing this risk, we decide to ban all users who report damage temporarily.

3. Design Verification

All the requirements listed in our Requirements and Verification Table has been met. All the verification listed in the Requirements and Verification Table has passed, as shown in the following sections.

3.1 Power Module

We verify our power module by plugging adaptor to wall power supply and probe our power pin on the PCB. Multimeter shows 4.95V which is inside ±5% requirement. Next we connect 5-3.3V voltage regulator to the power pin. We probe the Vout pin of the regulator and get 3.275V on the multimeter.

Adding up the maximum working current for all components:

$$I_{total} = I_{RFID} + I_{microcontroller} + I_{LCD} + I_{motor} + I_{wifi} = 150mA + 125mA + 120mA + 200mA + 140mA = 735\ mA$$

We are providing 2A current, and it satisfies the current requirement.

3.2 LCD Display

We connect our 1602 LCD display to 5V power supply and probe its power pin. 4.95V shows on the multimeter and it satisfy our requirement for LCD. Next we verify LCD's backlight adjust feature. After turning the variable resistor, the backlight changes accordingly. In order to verify that 1602 could display all 10 numbers, 26 characters including upper and lower cases, we input them one by one from microcontroller. All the inputs shows up correctly.

3.3 Wi-Fi Module

We set the ESP8266 wireless adapter to a wireless network created by a phone running a mobile hotspot. This setting is persistent across chip reboots. We run a test server on public internet which

source code is included in Appendix C.

The message sent by the rack is shown in command-line correctly after we turned on the microcontroller. The mock reply is shown on the LCD correctly with occasional transmission loss (message not received) due to unstable phone signal.

Later, we verified the auto-reconnect feature after it has been implemented. We starts the microcontroller program after shutting down the mobile hotspot. Within 10 seconds after we turned on the mobile hotspot again, the server command-line window displays a message received from the device.

Therefore all requirements for Wi-Fi module and the last requirement in "Security and Reliability" section has been met.

3.4 Lock Module

Each lock went through series of testings prior to final assembly onto the rack. After motor and tension wire had been installed and locked in place they are powered on and tested with data pins from microcontroller to open and close at least 5 times, all within 5 seconds. If everything is in working order this lock will be assembled onto the rack and tested again. After all 3 locks had been assembled on rack, wirings for all three motor and ULN2003 driver are connected to the PCB and tested again. Each lock is set to open and close in sequence, and time of each open is measured at around 2 seconds, and close for another 2 seconds, achieving all requirements for Lock Module.

3.5 RFID Module

We verify our RFID module by first connect it to the 3.3V power supply. After probing it, the multimeter shows 3.28V which is an acceptable working voltage for the RFID. Next, we test its working current. Multimeter shows 80mA, which is below the maximum current allowed(100mA)[5]

After connecting RFID and LCD together to our microcontroller, we scan one of our RFID cards and the card ID immediately shows on the display. We try several scans later and LCD shows consistent card id, which means our RFID sensor can correctly read RFID card information.

3.6 Control Module

During our testing of the completed rack, our microcontroller program's execution matches the operational flowchart in Figure 12 after all other modules are attached; therefore the control module's requirements have been met.

3.7 Server Software

We tested the server program using a mock client program that simulates the operation of a real rack and displays messages it receives. The source code of this program is included in Appendix D. To verify the server's capability to respond to multiple racks, we have done additional tests with the server and: 1) the real physical rack and a mock client, and 2) 5 mock clients. We use mysql console to check the data correctness in the database during those tests. Our server program passes all the tests and thus

satisfies all the requirements.

In addition, we tested using our rack after restarting the server. As we have examined, records in the database is not lost. Therefore, the requirements in “security and reliability” section has been met as well.

4. Costs

4.1 Parts

Part	Manufacturer	Retail Cost ($)	Bulk Purchase Cost ($)
AT89S52	Atmel	$3 each	$1.16 each
PCB	PCB Way	$3.80 each	$0.559 each
Motor & Driver	Generic	$20 per 10 pair	$8.50 per 10 pair
RC522 RFID	Generic	$25	$4.6 per Reader + 10 Card
ESP8266 WiFi	Generic	$4 each	$2.20 each
1602 LCD	Generic	$3 each	$1.31 each
Rack	Generic	$15 each	$10 each
Power Supply	Generic	$6	$3
DEMUX IC	Texas Instrument	$8 per 4 pair	$1.152 per 4 pair
Other IC (Inverter, etc)	Texas Instrument	$4	$1.768
Total		**$91.80**	**$34.24**

Table 3: Parts & Costs

4.2 Labor

Name	Hours Invested	Hourly rate	Total
Shuodong Zhang	220	$30	$6600
Yiheng Xu	220	$30	$6600
Xinyi Wu	220	$30	$6600
Total Labor Cost	$19800		

Table 4: Labor Costs

5. Conclusion

5.1 Accomplishments

Final product of project before demonstration is fully functional meeting all initial design specifications. All Requirement had been met and Verified. We were able to incorporate all parts and modules into the product, complete writing driver, firmware and software codes, and achieve full function status with PCB/parts soldering and system assembly.

The Umbrella Rental System prototype is easy to use in general with fluent user experience reflected through ease of use, system latency, and LCD information prompt convenience. All high level requirements set in initial proposal had been met with our prototype: 24x7 service, power loss data preservation, easy to use system, damage reports, as well as low operation costs.

Furthermore, we exceeded our initial high level requirements for low manufacture costs, with requirement of $150, actual functional prototype cost of $91.80, and estimated bulk cost of only $34.24 (See section 4.1 Parts Costs).

5.2 Uncertainties

As a prototype the product still have many uncertainties that may lead to issues. First is its exposed wiring and circuitry. Due to time limitation and easier debugging capability, current prototype has many exposed wirings, and PCB is stored in an open to cardboard box. Not only is this not safe, it is also prone to water damage from wet umbrellas with the project being an umbrella rack.

Second, current rental database algorithms may have unconsidered corner cases for malicious activity prevention. This is also an uncertainty for system security and maintenance.

5.3 Ethical considerations

The IEEE and ACM codes of ethics both states that we should avoid harm to others, especially our users. By design, our product is only a verification and logging system for access of shared resource (umbrellas), and is not likely to make direct contact with our users. Therefore, throughout operations and usage of our product it is quite impossible to physically harm people.

The ACM Code of Ethics also states that we should respect privacy [6]. We log user activities by recording their IDs, leasing and return time (and possibly video recordings of them if we decide to utilize a security camera). We plan to limit our data collection by automatically deleting archived files when a certain time period has passed.

Our project takes responsibility of IEEE code of ethics #1 “To accept responsibility in making decisions consistent with the safety, health, and welfare of the public, and to disclose promptly factors that might endanger the public or the environment.” [7] This system will benefit the community by providing great

convenience and will not cause harm to public safety or the environment.

We will also try to do our best to follow IEEE code of ethics #3 “to be honest and realistic in stating claims or estimates based on available data.” [7] We will make sure that our function works based on the data we collect. To follow IEEE code of ethics #7, we will also be open to accept any critic or comment on our design and the technical implementation. All suggestions are helpful towards a successful project.

5.4 Future work

First the system could be improved with better wiring and circuitry design, better protection for PCB and completely redesigned liquid prevention for entire circuitry.

Second, the rental database algorithm could be improved for more complete protection against corner cases and malicious activities, a better balance between user experience and system security. For example current implementation bans current user account temporarily when damage is reported, even if current user is not responsible, which can cause negative user experience.

Third, a better user registration and account management interface is needed, this can be designed as websites or mobile applications.

Furthermore, payment system could be setup on the server database for commercial applications of the project.

Lastly, more security options could be added, for example encryption for RFID chips and authorization process to prevent fake/cloned RFIDs; additional sensor for umbrella return status, etc.

References

[1] ESP8266Datasheet Available at: https://espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf Accessed Mar 2017.

[2] AMS1117 datasheet, AMS-Semitech.com. Available at: http://ams-semitech.com/attachments/File/AMS1117_20120314.pdf Accessed Mar 2017.

[3] BYJ-48 Stepper Motor, Instructables.com. Available at: http://www.instructables.com/id/BYJ48-Stepper-Motor/ Accessed Mar 2017.

[4] CD74AC139 Dual 2 to 4 Decoder, Texas Instrument. Available at: http://www.ti.com/lit/ds/symlink/cd74ac139.pdf Accessed Mar 2017.

[5] RC522 Datasheet Available at: https://www.elecrow.com/download/MFRC522%20Datasheet.pdf Accessed Mar 2017.

[6] Acm.org "ACM ACM Code of Ethics and Professional Concuct", 2017, [online]. available: http://www.acm.org/about-acm/acm-code-of-ethics-and-professional-conduct. Accessed Mar 2017.

[7] Ieee.org "IEEE IEEE Code of Ethics", 2017 [online]. available : http://www.ieee.org/about/corporate/governance/p7-8.html.

Appendix A Requirement and Verification Table

Requirement	Verification	Score
Power Module 1.Provide stable voltage at 5V +/- 5%. 2.Provide at least 1A of sustained current at 5V voltage. 3. The voltage regulator circuit runs correctly, giving 3.3V +/- 0.1V of voltage with 5V input.	1. a.) Connect multimeter to power module Vcc and GND to measure voltage. b.) Verify this voltage is within 4.75V to 5.25V range. 2. a.) Connect a 5 Ohm resistor that can withstand 1A current between power module's Vcc and GND b.) Use a multimeter to verify that stable current of 1 A can be reached. 3. Use a multimeter to probe the output of voltage regulator circuit. The voltage output between Vout and GND should be between 3.2V and 3.4V.	0+0+5
RFID Module 1. Must support interface other than UART because microcontroller has only one set of UART I/O 2. Implement driver for RFID module, to allow scanning and sending correct 13.56Mhz RFID card data or instructions to/from microcontroller.	1. Our choice of RC522 as our RFID reader satisfy this requirement because it has SPI interface support 2. a.) First verify our AT89S52 microcontroller and LCD screen. b.) Program the microcontroller to read card number from the RFID module and display it on LCD screen. c.) Connect RFID module to microcontroller, let microcontroller to run and scan an RFID tag. See if the correct card number is displayed in 9 character format.	5 (0,5)

Control Module 1. Spec requirements: Microcontroller should operate at around 5V DC with adequate Vin tolerance for reliability in case of unstable power supply. Microcontroller of choice (AT89S52) should have support for interfacing with RFID, wifi, screen and lock modules. 2. Microcontroller software realizes all the high-level requirements and the designed functions described in the microcontroller operation flow chart, without hangs.	1. Check the microprocessor spec sheet[2] for verification. 2. Test the complete system as a user after all other verifications has been done. Go through all the states and transitions and see if the system releases umbrella in error or hangs in undefined states. The tests and expected results are: 1) Power on the rack. Expected result: system displays anything meaningful on the LCD screen in the initialization phase. 2) Wait for initialization to end. Expected Result: The LCD screen prompt user to scan a card after initialization ends. 3) Scan a RFID card/tag. Expected result: After the rack displays the card number, the server command line interface should show the correct 16-byte message received containing UID of the rack, the RFID card number and damage flag as 'N'. 4) Wait until the 3-character server-to-client message is displayed on the LCD. Expected Result: If the third digit of server-to-client message is "L" or "R", motor specified by the second character in the message should rotate. Otherwise the LCD screen should notified the user as being denied access. 5) Press a button when prompted by LCD screen. Expected Result: if Yes button is chosen, the server command line window should receive a message with damage flag 'D' or 'P' depending on the previous message from server contains 'R' or 'L'. 6) See if the rack returns to prompt for scan RFID tag again.	5 (0,5)

Lock Module 1. The drive for lock module is able to rotate each different lock's motor. 2. The lock must successfully open or close in a maximum of 5 seconds.	1&2. a.) Connect the stepper motor to the mechanical lock. b.) Program the stepper motor to open/close the lock. c.) Make sure the lock opens and closes correctly each within 5 seconds. 3. After testing LCD screen, RFID module and Wi-Fi module, Use our server and different RFID tags to make the rack call the motor unlock function with different inputs. At least 2 different locks should be able to rotate after this function is called.	5 (0,0,5)
Wifi Module Implement driver for wifi module so that: 1. Wifi Module can connect to preprogrammed wireless network and send data to master PC's listening port. Microcontroller can properly read data sent from the master PC through serial interface.	1. a. Check server command-line interface on master PC to see if our rack is connected to the server. b. Code the server on master PC to display data received to see if the data sent from the rack is correct. c. Write testing program to run on the microcontroller to display data read over serial interface. Check if data can be received correctly. Alternative c) : after verifying a) and b), code the server to reply and code the microcontroller to display received data from ESP8266.	8 (1+5+2)
Screen Module 1. Display shall have at least 2 lines, each with 16 characters; takes digital input from microcontroller, full ASCII support. 2. Implement driver for the screen so that the microcontroller can make screen display a message with length at most 16 characters.	1. Our choice of LCD 1602 module satisfy the first two requirements. It has 16x2 characters and full ASCII support. 2. After implementing our LCD screen driver on the microprocessor, write any program to display 16 ASCII characters on the screen.	5 (0,5)

Master PC Software 1. Master PC software correctly reads data from listening port. 2. Master PC database correctly perform the following operations: a. query for user and umbrella ID, b. add entry for user rent/return, c. refresh rack status, d. make decision correctly based on the data above 3. Software on Master PC links correctly the network interface and the database. It can pass the data read from network to inversely.	1. Write testing program to display data read from the connection. 2. During runtime, read the database for the mentioned data using manual MySQL database operations. Check if they are accessed correctly. 3. Display the server's reply in command-line interface. Determine if it is correct based on the running state of master PC and rack.	11 (0,6,5)
Security and Reliability 1. We aim for adequate reliability for 24/7 database system online usecase. The software on PC, if crashed, will not cause persistent data in the database to be lost. 2. In the case of microprocessor power loss and reboot, states of umbrella availability should not be lost. 3. In the case of connection lost, the rack will automatically reconnect to the Wi-Fi network and server's listening port and re-send the message.	1. a.) Force terminate the database software on PC. b.) Examine the data in the database. Data that was already stored in the database prior to the termination should not be lost. 2. a.) Unplug the rack from power. b.) Examine the rack information on master PC and ensure the data is not lost. 3. a) Before scanning RFID tag, power off the wireless router. b) Scan RFID tag and power on the router. c) See if the client-to-server is sent within 20 seconds after the Wi-Fi network becomes available by looking at server command-line interface.	6 (3,0,3)
	Total	50

Appendix B Server Program Source Code

B.1 Main Program: Server.py

```
#!/usr/bin/env python

import socket
import sys
import datetime
from thread import *
import database as db


if len(sys.argv) != 3:
	print 'python server.py IP_address PORT_number'
	sys.exit()

TCP_IP = sys.argv[1]
TCP_PORT = int(sys.argv[2])
BUFFER_SIZE = 20  # Normally 1024, but we want fast response

count = 0
list_conn = []
list_addr = []
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
print 'Socket Created'
try:
	s.bind((TCP_IP, TCP_PORT))
except socket.error as msg:
	print 'Bind failed. Error:' + str(msg[0]) + ' Message '+msg[1]
	sys.exit()
s.listen(10)

def clientthread(conn):
	##conn.send('Server ACK placeholder\n')
	while True:
		try:
			data=conn.recv(BUFFER_SIZE)
		except socket.error as msg:
			print 'Receive failed. Error:' + str(msg[0]) + ' Message '+msg[1]
			break
		if not data:
			break
		print 'Data received from: ',addr,' Data: ',data
		##process data, craft reply msg
		"""if data == "UcantBserious":
			inputrack = str(raw_input("Select rack number for address"+str(addr)))
			reply = "0"*(3-len(inputrack))+inputrack+"0"*3
			print "Reply is: ",reply"""
```

```
rackUID = str(data)[:-10]
ID = str(data)[6:-1]
Damage = str(data)[15:]
time = str(datetime.datetime.now())
print rackUID
print ID
print Damage
print time                    ##TODO 3/15/2017:ADD HISTORY

if db.validateRack(rackUID)==False: ##rack does not exist
        reply = "00D"         ##deny
elif db.validateUmbrella(ID)==True: ##umbrella is being returned by a user
        slot = db.findSlot(rackUID,"Empty")
        if slot == -1 and Damage == "N":  ##no empty slot for return
                reply = "00D"         ##deny
        else:
                link_id = db.trackUmbrella(ID) ##link_id must be of a user when umbrella is leased
                if Damage == "D":
                        ##if db.fetchUserStatus(link_id)!="damagereported": ##user haven't reported
damage when he leased this umbrella
                        reply = "00D"
                        prevSlot = db.getLastSlot(ID)
                        prevUser = db.findLastUser(rackUID,prevSlot)
                        db.updateUserStatusTime(prevUser,"returndamaged",time) ##mark user
damaged this umbrella
                        db.updateSlot(rackUID,prevSlot,"Damaged",link_id) ##mark slot as having a
damaged umbrella, last user is link_id
                        db.createHistory(prevUser,rackUID,prevSlot,time,"returnDamaged")
                elif len(link_id)<8:
                        reply = "00D"
                else:
                        reply = "0"*(2-len(str(slot)))+str(slot)+"R" ##Accept return
                        ##Assume no damage...
                        db.assignLastSlot(ID,slot)
                        db.updateUserStatusTime(link_id,"returned",time) ##update user status
                        db.updateSlot(rackUID,slot,"Available",link_id) ##update rack status, slot
available again, last user is link_id
                        db.createHistory(link_id,rackUID,slot,time,"return")
                        link_slot_ID=rackUID+str(slot) ##link umbrella with current slot
                        db.markUmbrella(ID,link_slot_ID)
elif db.validateUser(ID)==True:##Valid User ID
        if db.fetchUserStatus(ID)=="returned": ##new lease or return damaged
                slot = db.findSlot(rackUID,"Available")
                if slot == -1:  ##no available umbrella
                        reply = "00D"         ##deny
                elif Damage == 'N': ##available
                        reply = "0"*(2-len(str(slot)))+str(slot)+"L" ##Accept Lease
                        link_slot_ID = rackUID+str(slot) ##???
                        umbrella_id = db.findUmbrellaByLink(link_slot_ID) ##find the umbrella to be
leased
                        db.markUmbrella(umbrella_id,ID) ##link user ID with this umbrella
```

```
                db.updateSlot(rackUID,slot,"Empty",ID) ##update rack status
                db.updateUserStatusTime(ID,"leasing",time) ##update user status
                db.createHistory(ID,rackUID,slot,time,"lease")
            else:
                ##returndamaged
                ###############TODO:find last slot used by user, mark it damaged; mark
user as returndamaged;
                reply = "00D"
                prevSlot = db.getLastSlot(ID)
                db.updateSlot(rackUID,prevSlot,"Damaged",ID)
                db.updateUserStatusTime(ID,"returndamaged",time)
                db.createHistory(ID,rackUID,prevSlot,time,"returnDamaged")
        elif db.fetchUserStatus(ID)=="leasing": ##return or pre-lease report
            slot = db.findSlot(rackUID,"Empty")
            umbrella_id = db.findUmbrellaByLink(str(ID))
            if slot == -1:  ##no empty slot for return
                reply = "00D"          ##deny
            else: ##return by scanning user ID
                reply = "0"*(2-len(str(slot)))+str(slot)+"R"  ##Accept Return ##Bug: Opens slot
twice when no damage reported
                if Damage == "P": ##user reported pre-lease damage
                    db.updateUserStatusTime(ID,"damagereported",time)
                    """
                    last_user_ID = db.findLastUser(rackUID,slot)
                    if db.fetchUserStatus(last_user_ID)!="damagereported":

db.updateUserStatusTime(last_user_ID,"returndamaged",time)
                    """
                    db.updateSlot(rackUID,slot,"Damaged",ID)
                    db.createHistory(ID,rackUID,slot,time,"leaseDamaged")
                    db.createHistory(ID,rackUID,slot,time,"returnDamaged")
                elif Damage == "D": ##Return damaged but should consider the case in which
user reported damage before
                    ###if db.fetchUserStatus(ID)!="damagereported": ##user haven't
reported damage when he leased this umbrella
                    db.updateUserStatusTime(ID,"returndamaged",time) ##mark user
damaged this umbrella
                    db.updateSlot(rackUID,slot,"Damaged",ID) ##mark slot as having a
damaged umbrella, last user is ID
                    db.createHistory(ID,rackUID,slot,time,"returnDamaged")
                else: ##Assume No damage safe return, waiting for damage report
                    db.updateSlot(rackUID,slot,"Available",ID) ##mark slot available for
rent again, last user is link_id
                    db.assignLastSlot(ID,slot)
                    db.updateUserStatusTime(ID,"returned",time) ##update user as
returned
                    db.createHistory(ID,rackUID,slot,time,"return")
                link_slot_ID=rackUID+str(slot) ##link umbrella with current slot
                db.markUmbrella(umbrella_id,link_slot_ID)

        else:
            reply = "00D" ##deny because user damaged umbrella previously
```

```
		else:
			reply = "00D" ##deny because no ID match
		print "Reply is: "+reply

		##reply='Reply Undefined'
		try:
			conn.sendall(reply)
		except socket.error as msg:
			print 'Send failed. Error:' + str(msg[0]) + ' Message '+msg[1]
			break
	conn.close()


##conn, addr = s.accept()
##print 'Connection address:', addr
while 1:
	conn, addr = s.accept()
	print 'Connection address:', addr
	count = count + 1
	start_new_thread(clientthread,(conn,))
	print count
s.close()
```

B.2 Helper: database.py

```
import MySQLdb as mdb

def connect():

	return mdb.connect(host="localhost", user="ece445", passwd="12345678", db="foo2");
##Users
def createUser(userid,status,lasttime):
	db_rw=connect()
	cur=db_rw.cursor()
	cur.execute("INSERT INTO users(id, userid, status, lasttime) VALUES (NULL,%s,%s,%s);", (userid,status,lasttime))
	db_rw.commit()
def validateUser(userid):
	db_rw=connect()
	cur=db_rw.cursor()
	cur.execute("SELECT userid FROM users WHERE userid = %s;", (userid,))
	if cur.rowcount < 1:
		return False
	return True
def fetchUserStatus(userid):
	db_rw=connect()
	cur=db_rw.cursor()
	cur.execute("SELECT status FROM users WHERE userid = %s;", (userid,))
	return cur.fetchone()[0]
def updateUserStatusTime(userid,status,lasttime):
	db_rw=connect()
```

```
	cur=db_rw.cursor()
	cur.execute("UPDATE users SET status=%s, lasttime=%s WHERE userid = %s;",(status,lasttime,userid))
	db_rw.commit()
def fetchUserById(userid):
	db_rw=connect()
	cur=db_rw.cursor()
	cur.execute("SELECT * FROM users WHERE userid = %s;", (userid,))
	if cur.rowcount < 1:
		return None
	return cur.fetchone()
def listUsers(garbage):
	db_rw=connect()
	cur=db_rw.cursor()
	cur.execute("SELECT * FROM users;")
	return cur
##History
def createHistory(userid, rackUID, slot, time, operation):
	db_rw=connect()
	cur=db_rw.cursor()
	cur.execute("INSERT INTO history(id, userid, rackUID, slot, time, operation) VALUES (NULL, %s,%s,%s,%s,%s);",
(userid, rackUID,slot,time,operation))
	db_rw.commit()
def fetchHistory(garbage):
	db_rw=connect()
	cur=db_rw.cursor()
	cur.execute("SELECT * from history;")
	return cur
##Umbrella
def createUmbrella(umbrella_id,link_id):
	db_rw=connect()
	cur=db_rw.cursor()
	cur.execute("INSERT INTO umbrellas(id, umbrella_id, link_id) VALUES (NULL, %s, %s);", (umbrella_id,link_id))
	db_rw.commit()
def validateUmbrella(umbrella_id):
	db_rw=connect()
	cur=db_rw.cursor()
	cur.execute("SELECT umbrella_id from umbrellas WHERE umbrella_id = %s;", (umbrella_id,))
	if cur.rowcount < 1:
		return False
	return True
def trackUmbrella(umbrella_id):
	db_rw=connect()
	cur=db_rw.cursor()
	cur.execute("SELECT link_id from umbrellas WHERE umbrella_id = %s;",(umbrella_id,))
	link_id = cur.fetchone()[0]
	return link_id
def findUmbrellaByLink(link_id):
	db_rw=connect()
	cur=db_rw.cursor()
	cur.execute("SELECT umbrella_id from umbrellas WHERE link_id = %s;",(link_id,))
```

```
	umbrella_id=cur.fetchone()[0]
	return umbrella_id
def markUmbrella(umbrella_id,link_id):
	db_rw=connect()
	cur=db_rw.cursor()
	cur.execute("UPDATE umbrellas SET link_id = %s WHERE umbrella_id = %s",(link_id,umbrella_id))
	db_rw.commit()
def listUmbrellas(garbage):
	db_rw=connect()
	cur=db_rw.cursor()
	cur.execute("SELECT * FROM umbrellas;")
	return cur
#locations
def createRack(UID,slot,state,lastuser):
	if len(UID)!=6:
		return False
	db_rw=connect()
	cur=db_rw.cursor()
	cur.execute("DELETE FROM locations WHERE UID = %s;",(UID,))
	k = int(slot)
	for i in range (1,k+1):
		cur.execute("INSERT INTO locations(id,UID,slot,state,lastuser) VALUES
(NULL,%s,%s,%s,%s);",(UID,str(i),state,lastuser))
	db_rw.commit()
	return True
def validateRack(UID):
	db_rw=connect()
	cur=db_rw.cursor()
	cur.execute("SELECT * from locations WHERE UID = %s;",(UID,))
	if cur.rowcount < 1:
		return False
	return True
def findSlot(UID,state):
	db_rw=connect()
	cur=db_rw.cursor()
	cur.execute("SELECT slot from locations where UID = %s AND state = %s;",(UID,state))
	if cur.rowcount<1:
		return -1;
	else:
		slot=int(cur.fetchone()[0])
		return slot;
def updateSlot(UID,slot,state,lastuser):
	db_rw=connect()
	cur=db_rw.cursor()
	cur.execute("UPDATE locations SET state = %s, lastuser = %s WHERE UID=%s AND slot = %s;",(state,lastuser,UID,slot))
	db_rw.commit()
def findLastUser(UID,slot):
	db_rw=connect()
	cur=db_rw.cursor()
	cur.execute("SELECT lastuser from locations WHERE UID = %s AND slot = %s;",(UID,slot))
```

```
        userid=cur.fetchone()[0]
        return userid
def listLocation(garbage):
        db_rw=connect()
        cur=db_rw.cursor()
        cur.execute("SELECT * FROM locations;")
        return cur

#lastSlot
def assignLastSlot(itemid,slot):
        db_rw=connect()
        cur=db_rw.cursor()
        cur.execute("DELETE FROM lastSlot WHERE itemid = %s;",(itemid,))
        db_rw.commit()
        db_rw=connect()
        cur=db_rw.cursor()
        cur.execute("INSERT INTO lastSlot(itemid,slot) VALUES (%s,%s);",(itemid,slot))
        db_rw.commit()
def getLastSlot(itemid):
        db_rw=connect()
        cur=db_rw.cursor()
        cur.execute("SELECT slot FROM lastSlot WHERE itemid = %s;",(itemid,))
        slot = cur.fetchone()[0]
        return slot
```

Appendix C Test Server Source Code

```
#!/usr/bin/env python

import socket
import sys
import datetime
from thread import *
#import database as db


if len(sys.argv) != 3:
        print 'python server.py IP_address PORT_number'
        sys.exit()

TCP_IP = sys.argv[1]
TCP_PORT = int(sys.argv[2])
BUFFER_SIZE = 20  # Normally 1024, but we want fast response

count = 0
list_conn = []
list_addr = []
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
print 'Socket Created'
try:
```

```
        s.bind((TCP_IP, TCP_PORT))
except socket.error as msg:
        print 'Bind failed. Error:' + str(msg[0]) + ' Message '+msg[1]
        sys.exit()
s.listen(10)

def clientthread(conn):
        ##conn.send('Server ACK placeholder\n')
        while True:
                try:
                        data=conn.recv(BUFFER_SIZE)
                except socket.error as msg:
                        print 'Receive failed. Error:' + str(msg[0]) + ' Message '+msg[1]
                        break
                if not data:
                        break
                print 'Data received from: ',addr,' Data: ',data
                ##process data, craft reply msg
                """if data == "UcantBserious":
                        inputrack = str(raw_input("Select rack number for address"+str(addr)))
                        reply = "0"*(3-len(inputrack))+inputrack+"0"*3
                        print "Reply is: ",reply"""
                reply = "04R"
                print "Reply is: "+reply

                ##reply='Reply Undefined'
                try:
                        conn.sendall(reply)
                except socket.error as msg:
                        print 'Send failed. Error:' + str(msg[0]) + ' Message '+msg[1]
                        break
        conn.close()


##conn, addr = s.accept()
##print 'Connection address:', addr
while 1:
        conn, addr = s.accept()
        print 'Connection address:', addr
        count = count + 1
        start_new_thread(clientthread,(conn,))
        print count
s.close()
```

Appendix D Mock Client Source Code

```
#!/usr/bin/env python
```

```
import sys
import socket
import time
import random
##This program is written under python 2.7.3
##For python 3.0, replace: raw_input
if len(sys.argv) != 3:
        print 'python client.py IP_address PORT_number'

"""
input1 = str(raw_input("Enter rack number:"))
x = len(input1)
input2 = str(raw_input("Enter ID:"))
y = len(input2)
input3 = str(raw_input("Report Damage?"))
message = "0"*(3-x)+input1+"0"*(9-y)+input2+input3
print message
"""


def openslot(slot):
        print 'slot',slot,'opened.'

TCP_IP = sys.argv[1]
TCP_PORT = int(sys.argv[2])
BUFFER_SIZE = 100
temp = str(raw_input("Enter rack UID:"))
UID = "0"*(3-len(temp))+temp
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((TCP_IP, TCP_PORT))
#s.send("UcantBserious")
#data = s.recv(6)
#print "received data:", data
#rack = str(data)[:-3]
#x = len(rack)
while 1:
        ID = str(raw_input("Enter ID:"))
        y = len(ID)
        damage = str(raw_input("Report Damage?"))
        message = UID+"0"*(9-y)+ID+damage
        s.send(message)
        data = s.recv(3)
        print "received data:", data
        slot = int(str(data)[:-1])
        permission = str(data)[2:]
        if permission == "L":
                openslot(slot)
                print "Lease permitted."
        elif permission == "R":
```

```
        openslot(slot)
        print "Return accepted."
elif permission == "D":
        print "Denied."
else:
        print "Boom! You have entered undefined state"
##s.send(message)
##data = s.recv(BUFFER_SIZE)
##s.close()
garbage = raw_input("Enter something when you are ready for next iteration.")

##time.sleep(10*random.random())
```