**Project Report**

**Spring 2017**

**Advanced Operating System**

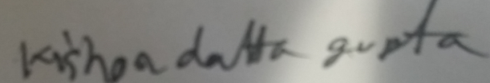
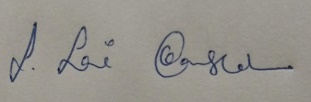
**Name: Kishor Datta Gupta ID: L20421951**

**Name: Sai Gavaskar Sakalam ID: L20419319**

**Date: 30/04/2017**

**List of Collaborators:**

1. **Distribute System Concepts and design**
2. **Stackoverflow**
3. **Wikipedia**
4. **Opengl Tutorials**
5. **C++ tutorials**
6. **http://www.cs.rpi.edu/~moorthy/Courses/os98/Pgms/socket.html**

**Signature  
Signature:**

# Introduction:

## Bayou :

Bayou is a replicated, weakly consistent storage system designed for a mobile computing environment that includes portable machines with less than ideal network connectivity. To maximize availability, users can read and write any accessible replica. Bayou’s design has focused on supporting application-specific mechanisms to detect and resolve the update conflicts that naturally arise in such a system, ensuring that replicas move towards eventual consistency, and defining a protocol by which the resolution of update conflicts stabilizes.

Designed to support:

* a variety of non-real-time collaborative applications
* applications that might be used by individuals at different hosts at different times

Examples:

* Meeting room scheduler
* Bibliographic database

Basic System Model:

* Each data collection is replicated in full at a number of servers
* API supports R / W
* Weakly consistent replication model
* Session guarantees
* Anti-entropy sessions

Conflict Detection and Resolution

* Storage systems must provide means for an application to specify its notion of a conflict along with its policy for resolving conflicts
* Resolution can vary according to the semantics of the application

Replica Consistency:

All servers move towards eventual consistency because

* Writes perform in the same, well- defined order
* conflict detection and merge procedures are deterministic (servers resolve the same conflicts in the same manner)

A user may be granted:

* R/W privileges to data collection
* Server privileges to maintain a replica

**Eventual consistency**:

This is a consistency model used in distributed computing to achieve high availability that informally guarantees that, if no new updates are made to a given data item, eventually all accesses to that item will return the last updated value. Eventual consistency is widely deployed in distributed systems, often under the moniker of optimistic replication, and has origins in early mobile computing projects. A system that has achieved eventual consistency is often said to have converged, or achieved replica convergence . Eventual consistency is a weak guarantee – most stronger models, like linearizability are trivially eventually consistent, but a system that is merely eventually consistent does not usually fulfill these stronger constraints.

Eventually consistent services are often classified as providing BASE (Basically Available, Soft state, Eventual consistency) semantics, in contrast to traditional ACID (Atomicity, Consistency, Isolation, Durability) guarantees Eventual consistency is sometimes criticized as increasing the complexity of distributed software applications. This is partly because eventual consistency is purely a livens guarantee (reads eventually return the same value) and does not make safety guarantees: an eventually consistent system can return any value before it converges.

Problem Statement:

In a building, there are eight meeting rooms, which are shared by 10 departments on an as-needed basis. Each department has a secretary who is responsible for reserving meeting rooms. Each reservation slot is of 1 hour duration, and rooms can only be reserved between 8 a.m. and 5 p.m. Your job is to implement a shared room reservation schedule that can be used by the secretaries.

Use the following conflict resolution rule: If multiple appointments overlap, the earliest entry for that slot prevails and others are cancelled. If there is a tie for the earliest entry, the secretaries' first names will be used to break the tie. Each secretary will work on a separate replica of the schedule and enter the reservation request stamped by the clock time. The clocks are synchronized. Dependency checks will be done during the anti-entropy sessions as in Bayou. The goal is eventual consistency.

# Project Requirement:

## User Requirement:

* Secretaries must be able to request for the reservation of meeting room at any time. Whether servers are available or not.
* The system must acknowledge the secretaries whether the reservation process is success or failure whenever server got live.
* The system must display information about the reserved meeting rooms.

## Reliability Requirement:

* A single server failure wont effect the system.
* Eventually all 10 secretary will have same meeting room schedule.
* It will be 24 hours available to use the system from any secretary whether its online or not.

## Performance Requirement:

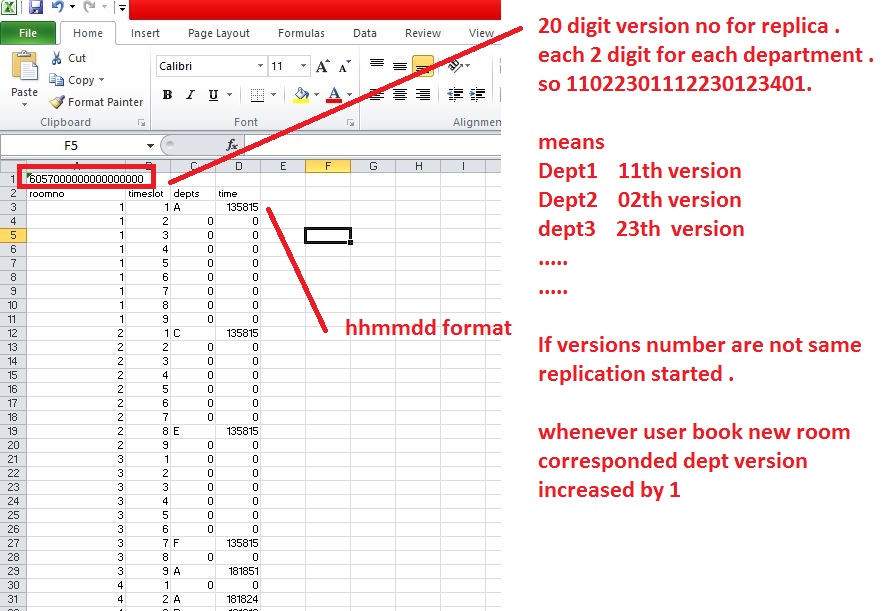
* System should support up to 10 concurrent request for reservation of meeting rooms.
* System must allow 8 different rooms to be reserved for each 1 hour reservation slot.

## Software Requirement:

* Operating system: Linux (Ubuntu/centos)
* IDE: Nontepad GCC Compiler
* OpenGL

# Implementation & Desgins:

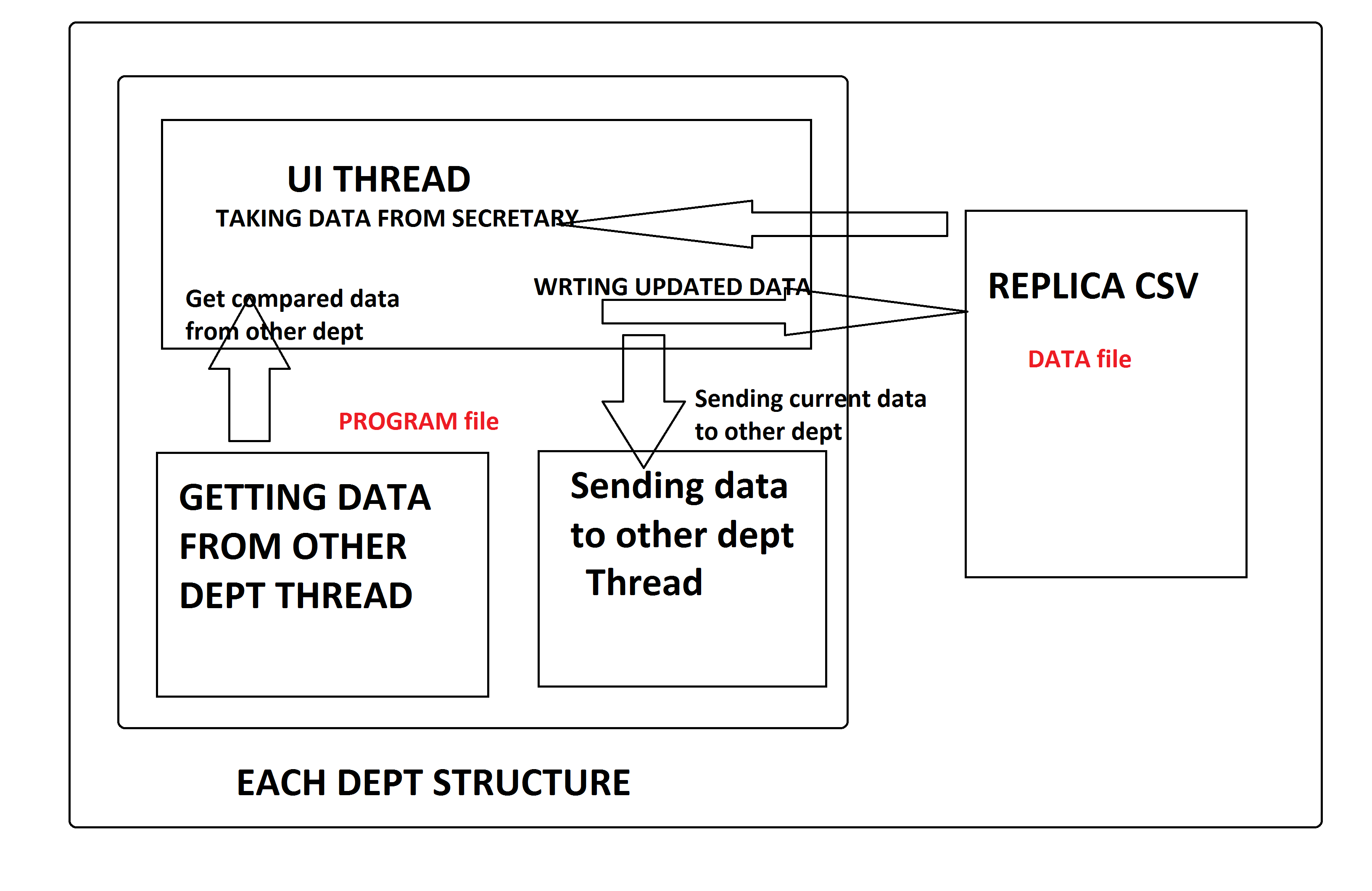
DATA MANAGEMENT



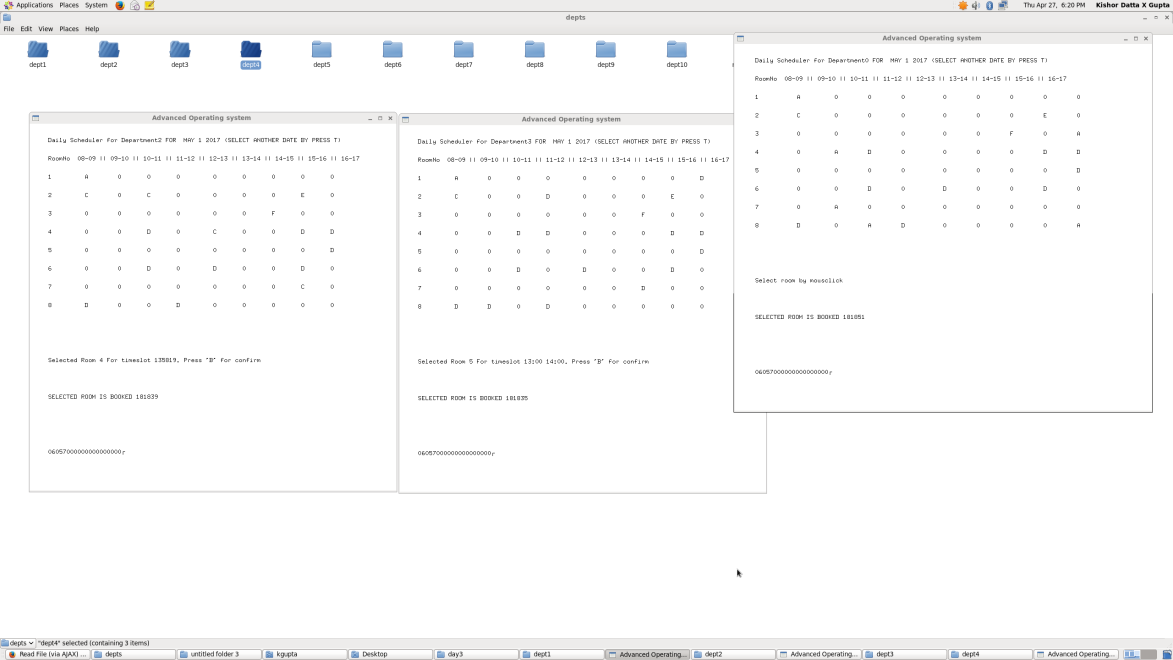
Note:

* From 8 am to 5 pm divided into 9 time slot and each assigned a Integer sequence 1 to 9.
* Eventually if servers are running they all reach a same csv file with same version number and same data.
* We need to use TCP as we need to be sure messages was sent before increment versions(later it not implemented)

**PROGRAM Management:**

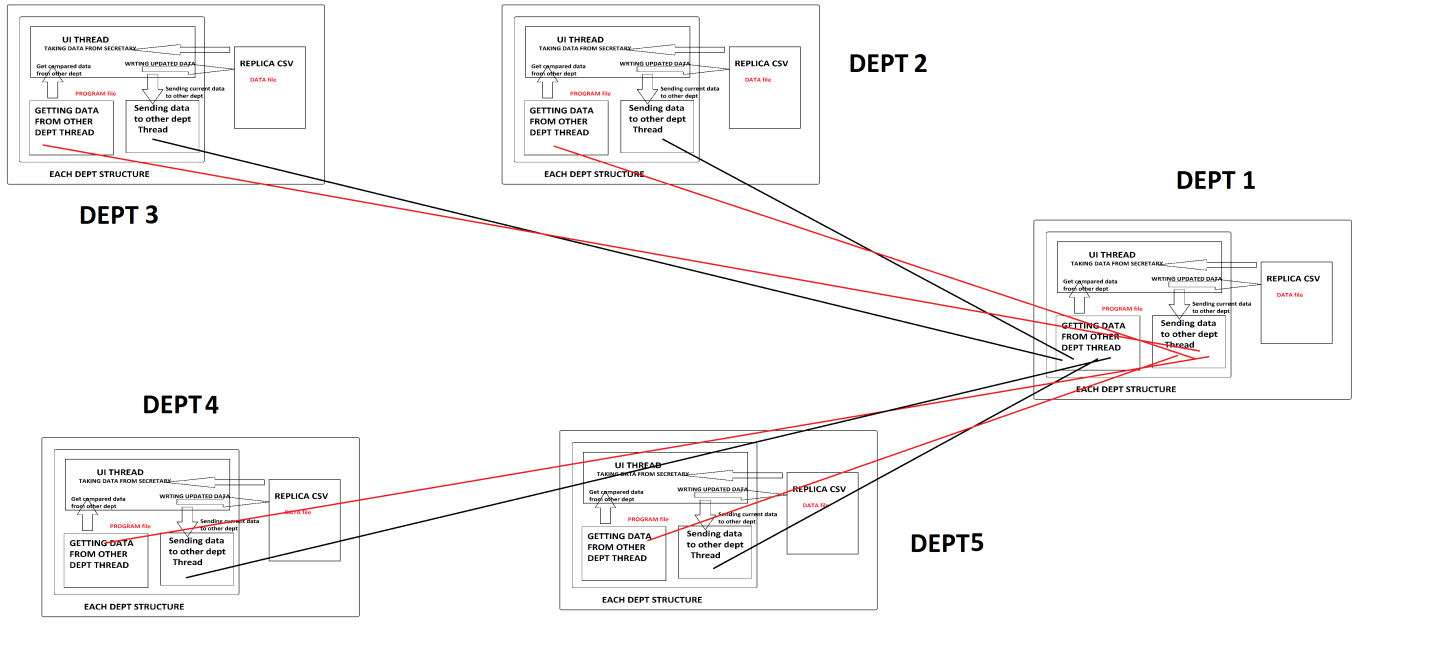
****  
**UI thread:**

Developed in openGL for one day schedule. Click on any field and press B will book that room.

in OPENGL sample screenshot

**NETWORK MANAGEMENT:**

Network flow for **dept1** sending and receiving response. Other depts. Has similar with all others



APP STEPS  
1. App starts

2. Load data and version from replica

3. Start UI thread, Listening thread, Receiving thread

UI THREAD:

* 1. USER SELECT A ROOM AND PRESS B TO BOOKED
  2. IF ROOM NOT OCCUPIED. OCCUPIED THE ROOM UPDATE THE DATA
  3. UPDATE THE VERSION NUMBER
  4. IF USER PRESS EXIT SAVE THE DATA IN REPLICA FILE AND CLOSE ALL THREE THREAD

ELSE GO TO 1 AND WAIT FOR USER INPUT

LISTENING THREAD:

1. CREATE A SERVER SOCKET
2. CLIENT CONNECT AND SEND VERSION NUMBER
3. IF VERSION NUMBER NOT MATCHED SERVER SEND REPLICA DATA OF ROOMS WHICH ARE BOOKED

ELSE SEND 0

1. CLOSE THE CLIENT SOCKET AND GO TO 1

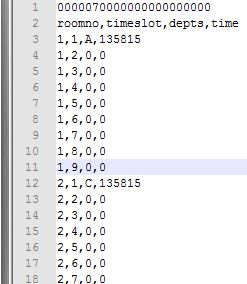
RECEIVING THREAD:

1. GET ALL OTHER DEPARTMENT SERVER IP AND PORTS ith to N
2. GET ith SERVER IP AND CONNECT WITH SERVER
3. SEND CURRENT VERSION NUMBER TO SERVER
4. IF SERVER SENDS 0, CLOSE THIS SOCKET AND INCREASE ith value AND GO TO 2

ELSE

1. SERVER SENDS DATA MATCH WITH EXISTING DATA AND UPDATE IF DATA HAVE NEW ENTRIES. AND INCREASE ith value AND GO TO 2

**Replica Data Sample**



**USE CASE:**

**Dept A server Dept B client**

Dept B sent “0000070000000000000000” IF Dept A find Version code is same with data he sent 0, other wise sent   
10,c,135815\n17,e,135815\n25,f,135815…(only booked room info)  
Dept B check his row 10,17,25…. With correspondind data, older data remained or alphabetical order preserved.

# Build Instructions:

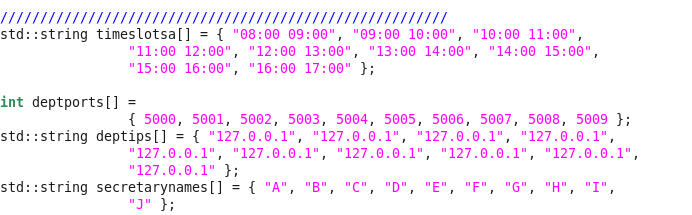
BUILD COMMAND: **gcc -o kda kd1.cpp -lglut -lGLU -lpthread**

RUN: BY CLICK THE KDA executable

NOTE: Replica.csv file need to be with samefolder at executables.

For Hosting in multi computer:

Need to add IP and port in Code deptips and deptsport array for respective department as in the image



To assign department in define select the number of the dept. 0 to 9 in define section

**Use Instructions:**

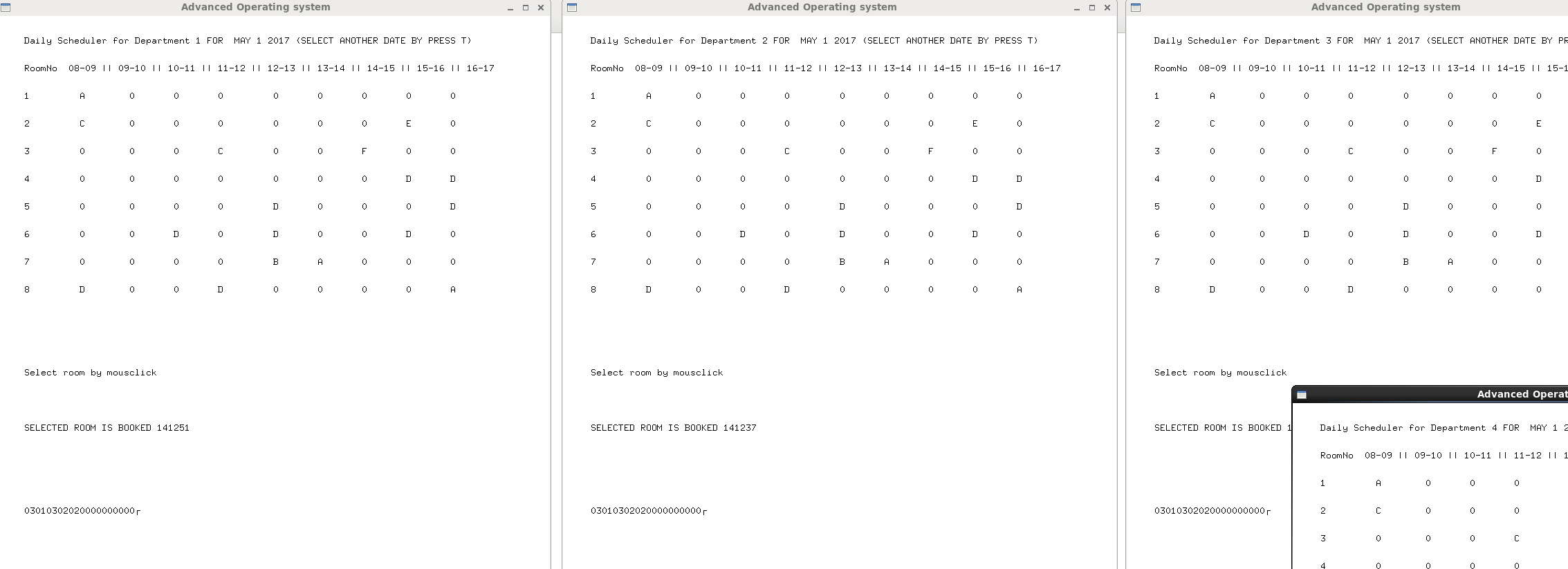
Click in a roombooked data in UI and press B will booked that room .

For refresing data need a mousclick on UI.

Need to close by ESC to proper save of data

Special Notes: Its only provide data for one day

# RESULTS:



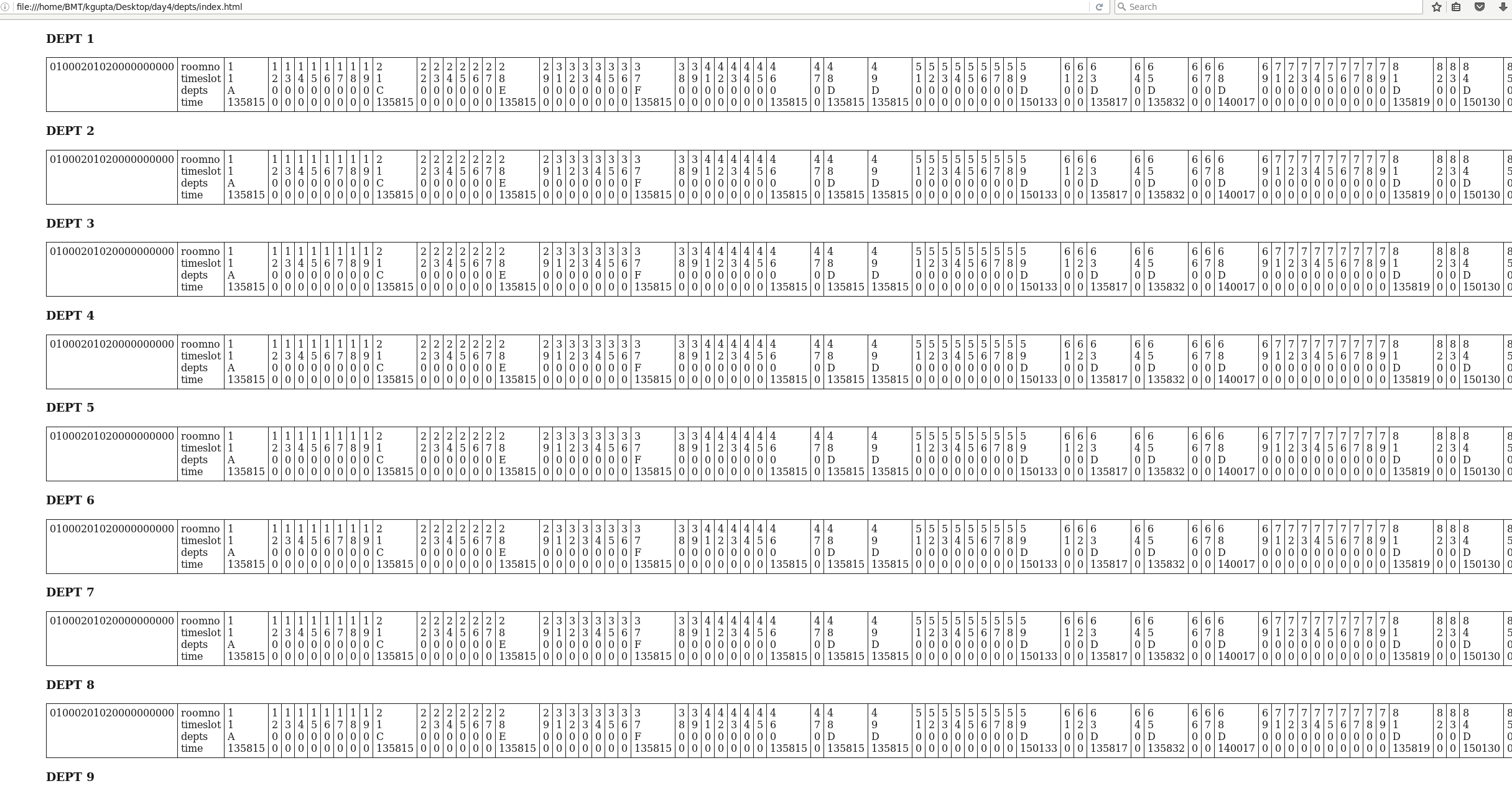
# Eventual Consistency:

All the departments are storing there data wherever server is online or not, when server online they just sync the data and updated. Eventually all 10 file reach an same data. Whenver they were online full time or not.

We followed **Active replication .** Each server receives client requests, and broadcasts them to the other servers. They collectively implement a fault-tolerant state machine. In presence of crash, all the correct processes reach the same next state. Agreement. Every correct replica receives all

the requests. Order. Every correct replica receives the requests in the same order. Agreement part is solved by atomic multicast. Order part is solved by partially order multicast. The order part solves the consensus problem where servers will agree about the next update. It requires a synchronous model.

**Here all the data from all replica showed in a html file in mid running. Eventually all data reach to eventual consistency using anti entropy**

****

**Client centric:**

Client communicates with different replica at different times. So this is a client centric model .we used Sequential consistency. In a result as its synchronous clocked we able to reach Eventual Consistency.  
  
**Conclusion:**

Biggest challenge is to reduce data between server to merge replicas data. We used TCP , and limit the size 2048byte. Even if all the room booked data would not crossed 2048 limit. Will be max 1800 byte. Using version in first transection reduce load, If one server is unavailable it just go to next server.

Project made for one day schedule to avoid complexity. As its demo project and main target was to understand and implement eventual consistency and gain anti entropy. We are successful to implemenet that.

Saving in local . data saved in local after each booked or click closed . so unexpected close will create problem and lost of data. But as it can get data again from other server it will not ost data if it already passed is data to other server. So data failure was save that way

UI as a problem to view latest data need to refresh window. We tried to make auto refresh but in opengl it seems problematic to call UI thread refresh from other thread. Its limitations of opengl system