
2016 컴퓨터 소프트웨어 설계

최종보고서

2조 - 버거킹 강남점			
직책	학과	학번	성명
팀장	컴퓨터공학과	20124829	이선희
부팀장	컴퓨터공학과	20124973	한우탁
팀원	컴퓨터공학과	20124932	박용희
팀원	컴퓨터공학과	20114815	노광준
팀원	컴퓨터공학과	20114787	장성일



조선대학교
CHOSUN UNIVERSITY

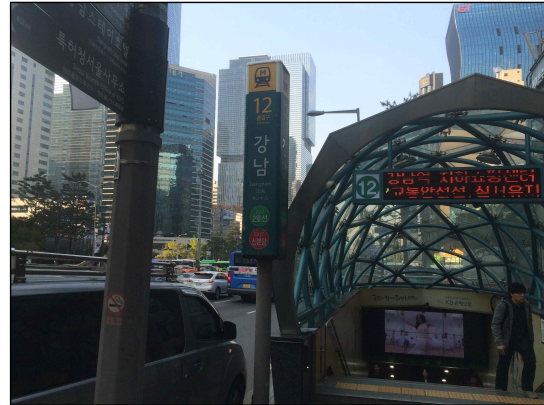
목 차

제 1 장 프로젝트 필요성	1
제 2 장 프로젝트 목적	2
제 3 장 프로젝트 설계	3
3.1 기능	3
3.2 상세설계	4
3.2.1 Windows 화면 설계(GUI)	4
3.2.2 각 쓰레드별 pseudo code	6
3.2.3 메모리 DB를 위한 트리 구현 전략	9
3.2.4 DB의 저장 및 복구 전략	11
3.2.5 네트워크를 이용한 서버와 클라이언트의 연결	12
제 4 장 프로젝트 구현	17
4.1 주요 구현 내용	18
4.1.1 Thread, Semaphore & GUI	18
4.1.2 DB, Network, File I/O	25
제 5 장 팀원별 역할분담 및 후기	32
제 6 장 개발일정	35
제 7 장 기대효과	36
7.1 전체 프로그램 활용 가능성	36
7.2 기술적 기대효과	36
7.3 산업적 기대효과	37
부록. 소스코드	38

1. 프로젝트 필요성



[그림 1] 강남역 12번 출구 맥도날드



[그림 2] 강남역 12번 출구

강남역은 국토교통부가 조사한 바와 같이 이용객이 가장 많은 전철역이자, 출퇴근 시간대에 혼잡도가 가장 높은 곳으로 나타났다. 강남역 12번 출구 맥도날드에서는 이러한 조사결과를 바탕으로 많은 손님들이 입장해 이용할 것을 대비하고 만족도를 높이하고자 시뮬레이션을 통해 매장 내 혼잡 요소 및 개선 필요 요소를 파악하려고 한다. 또한 향후 드라이브 스루 서비스를 제공하게 될 때의 상황, 손님들이 가장 많이 이용하는 메뉴 등을 종합적으로 분석하려고 한다.

이런 상황을 종합적으로 분석할 수 있는 시뮬레이션 프로그램을 이용한다면, 효율적으로 매장 구조를 변경하고, 매출을 높일 수 있는 좋은 도구로써 사용할 수 있을 것이다. 또한 다른 맥도날드 지점에서도 이용할 수 있을 것이다.

추가적으로 맥도날드 지점을 열고자 하는 예비 업주는 자신이 매장을 운영하고, 발생할 수 있는 문제점, 인테리어 등을 미리 계획할 수 있도록 도움을 줄 수 있는 프로그램으로 이용할 수 있을 것이다.

2. 프로젝트 목적

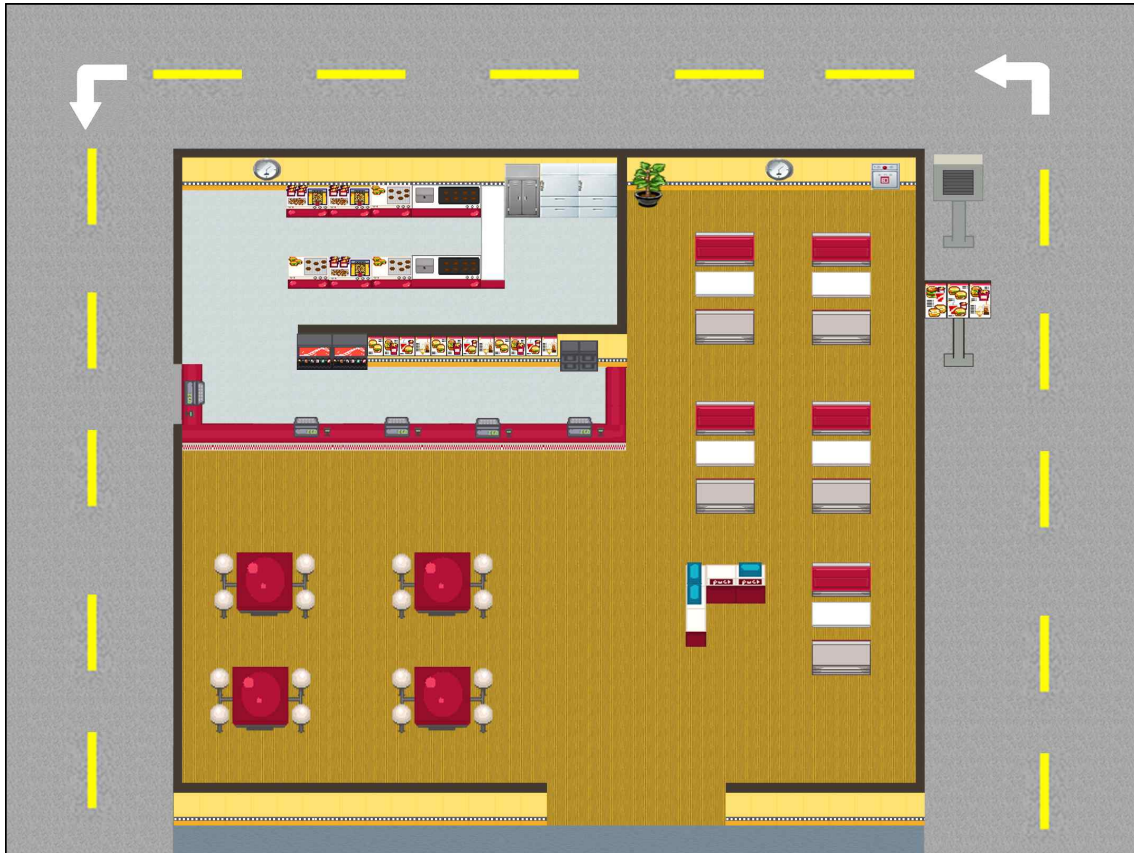
주 목적 : 맥도날드 매장/드라이브 스루 이용 시뮬레이션 제작



시뮬레이션을 제작하는데 있어서 발생하는 상황(손님 입장, 주문, 식사 등)을 여러 개의 **쓰레드와 세마포어**, **큐** 등을 이용해서 구현하려고 한다. 특히, 이러한 상황을 **GUI**로 구현하고 시뮬레이션 동작 중 발생하는 데이터에 대해서는 **AVL tree 데이터베이스**를 통해서 처리한다. 또한 그 데이터를 **TCP/IP 소켓 프로그래밍**을 통해서 서버와 클라이언트 간에 통신하는 기능을 수행하는 프로그램을 제작하고자 한다.

3. 프로젝트 설계

3.1 기능



본 프로젝트에서 제작할 시뮬레이션 프로그램은 위와 같은 화면에서 손님들과 드라이브 스루를 이용하기 위한 자동차가 입장하고 각각 서비스를 이용한다. 손님들은 매장에 20명 정도가 입장하며, 주문을 받는 직원은 3명, 요리사는 2명, 결제를 도우는 직원은 1명으로 구성이 된다. 드라이브 스루를 이용하는 손님은 자동차를 타고 드라이브 스루로 입장하고, 주문과 결제를 진행한다. 전체 프로그램 기능은 시뮬레이션을 실행하는데 필요한 각각의 기능으로 구성이 되어있다.

손님들과 자동차, 매장 내/외 직원들은 각각 쓰레드가 동작함으로써 구현이 된다. 각각 동작하는데 있어서 입장하는 인원, 주문을 한 후 요리에 들어감, 서비스를 제공하는데 순서를 지키는 점 등을 세마포어의 카운팅, 동기화, 상호 배제 등을 이용해서 구현한다. 주문하는 내용과 손님 정보 등은 AVL tree 데이터베이스의 노드로써 관리가 되며, 이러한 내용들을 서버와 클라이언트가 통신할 수 있도록 TCP/IP 소켓 프로그래밍을 이용한다.

3.2 상세 설계

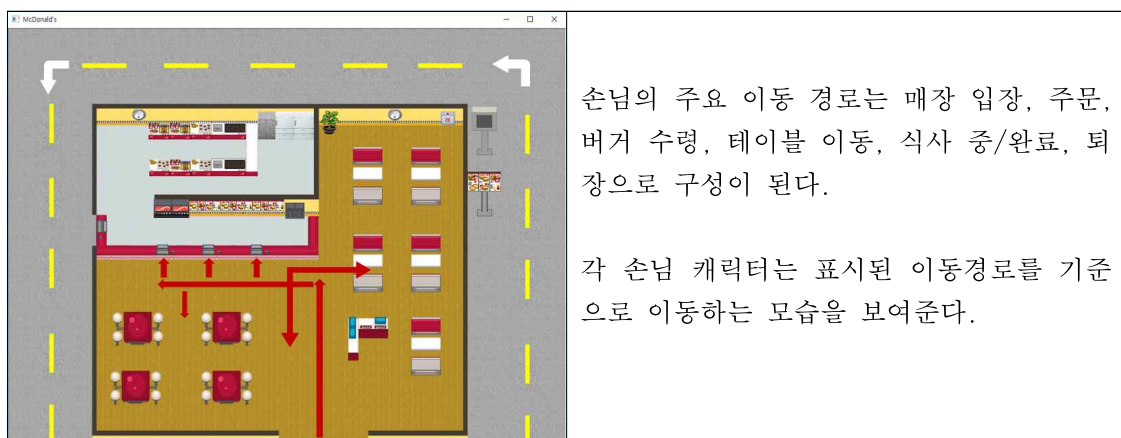
3.2.1 Windows 화면 설계(GUI)



그림 4 GUI 배경 화면

프로그램의 GUI Window 화면은 위 [그림 4]와 같이 실제 매장의 이미지와 같이 구성하였다. 이 배경 화면을 기준으로 각각 손님, 직원, 자동차 등이 동작을 수행한다.

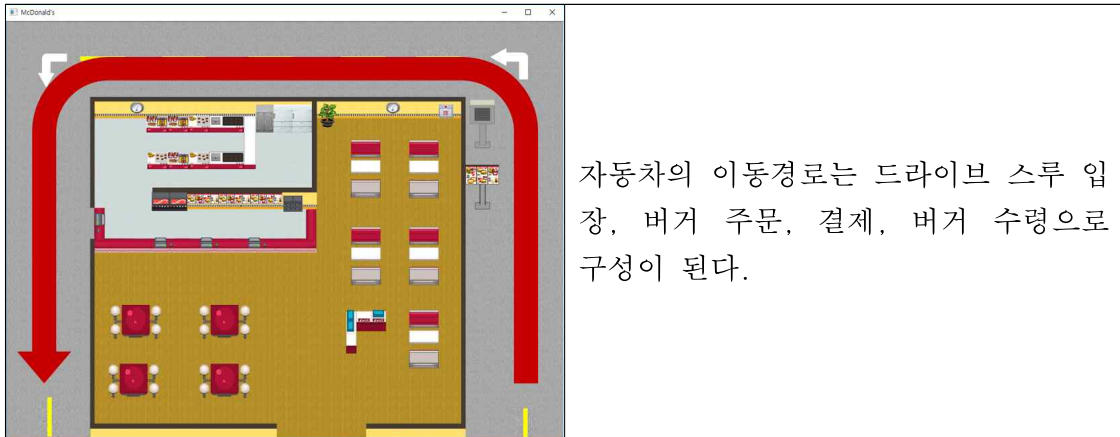
(1) 손님 이동 경로



손님의 주요 이동 경로는 매장 입장, 주문, 버거 수령, 테이블 이동, 식사 중/완료, 퇴장으로 구성이 된다.

각 손님 캐릭터는 표시된 이동경로를 기준으로 이동하는 모습을 보여준다.

(2) 자동차 이동 경로

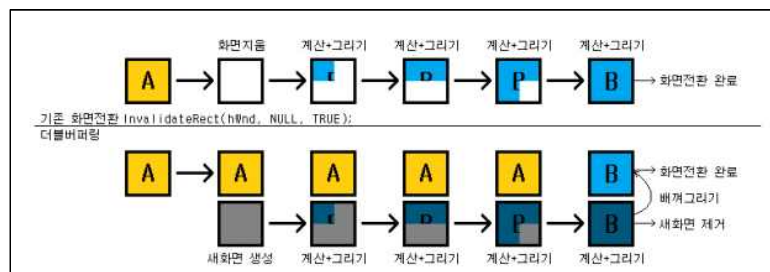


(3) 캐릭터/자동차 이미지



(4) 더블버퍼링

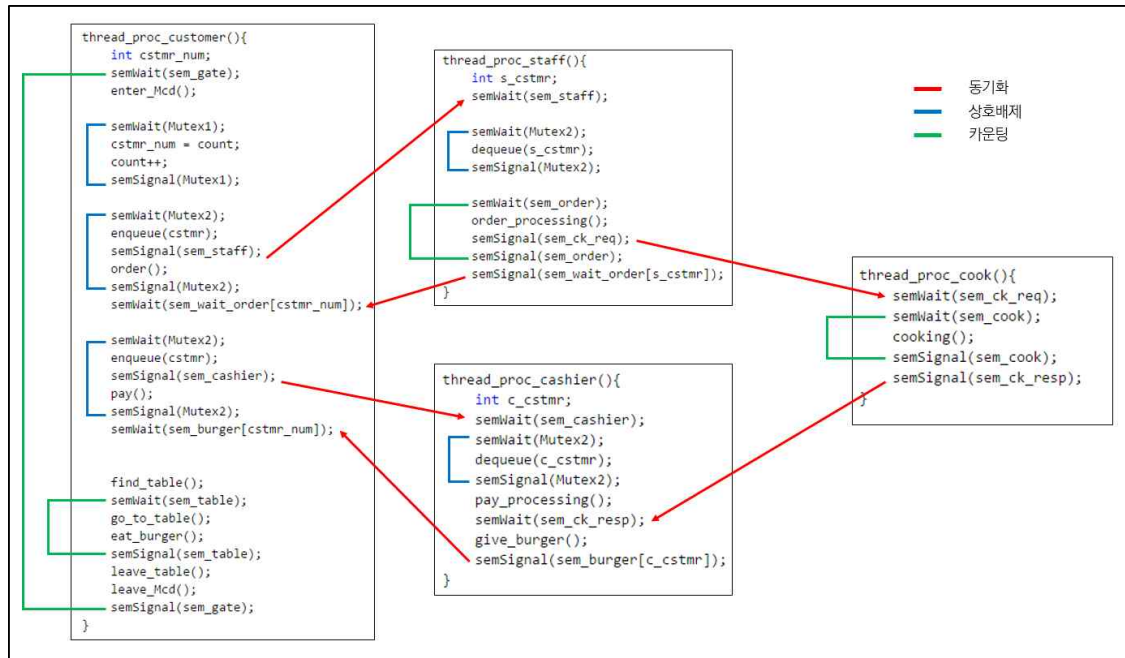
GUI를 통해서 표현하고자 하는 이미지가 많을수록 화면의 깜빡임이 심해져 더블버퍼링 기술을 필요로 한다. 더블버퍼링은 여러 개의 이미지를 따로 출력하는 것이 아닌, 메모리에 미리 그린 후 한 번에 출력함으로써 깜빡임이 없도록 한다.



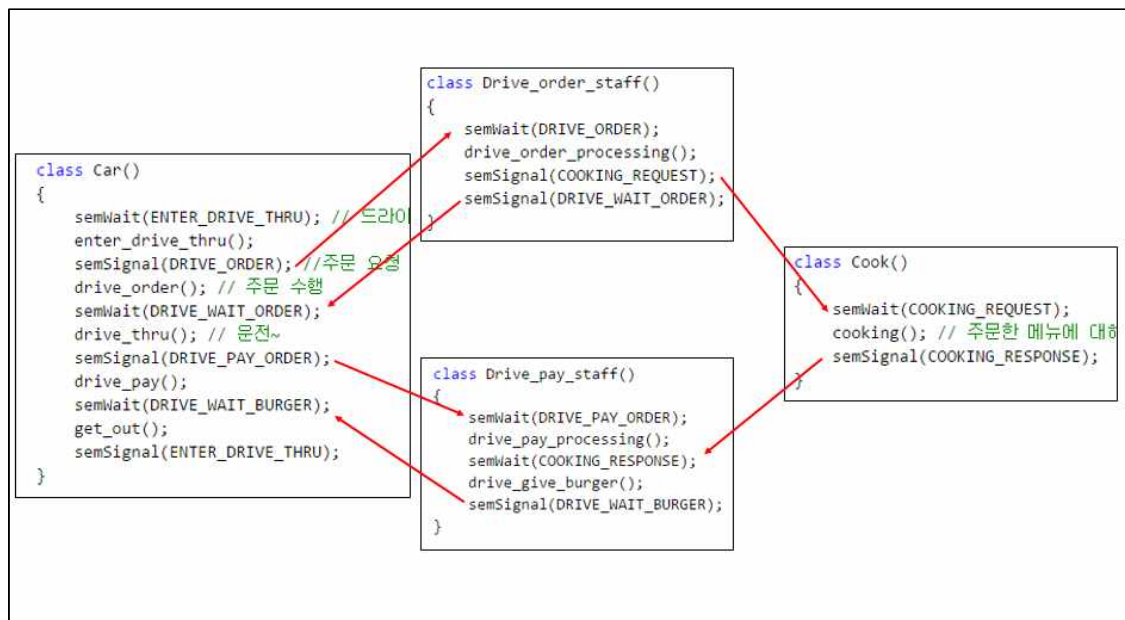
3.2.2 각 쓰레드별 pseudo code

각 쓰레드별 pseudo code는 다음과 같다.

(1) 손님, 직원, 캐셔, 요리사 쓰레드



(2) 자동차, 드라이브 직원, 드라이브 캐셔, 요리사 쓰레드



(3) 카운팅, 동기화, 상호배제 세마포어

a. 카운팅 세마포어

smp_gate : 매장 내 입장하는 손님의 수를 20명으로 제한한다.

smp_table : 매장 내에서 식사할 수 있는 테이블은 제한되어 있다. 이 수를 카운팅 한다.

smp_order : 직원의 수는 3명으로 동시에 주문을 할 수 있는 최대 손님 수도 3명이다.

smp_cook : 동시에 요리할 수 있는 요리사의 수는 2명이다.

smp_enter_drive_thru : 드라이브 스루에 입장할 수 있는 자동차의 수를 제한한다.

b. 동기화 세마포어

smp_staff[THREAD_CNT] : 손님이 주문을 받는 직원 앞에 도착했을 때 직원이 주문을 수행하는 것으로 동기화를 한다.

smp_wait_order[THREAD_CNT] : 주문이 완료되었다는 신호를 직원이 손님에게 주기 전까지 기다림으로써 동기화를 수행한다.

smp_cashier[THREAD_CNT] : 주문이 완료된 후 손님이 캐셔에게 계산을 해줄 것을 요청할 때 캐셔가 계산을 수행한다.

smp_burger[THREAD_CNT] : 직원이 만들어진 버거를 손님에게 주는 동기화를 수행한다.

smp_ck_req[THREAD_CNT] : 직원이 요리사에게 요리를 요청하는 과정에서 동기화를 수행한다. 직원이 요리사에게 버거를 만들어 달라고 요청할 때 요리사는 요리를 해야 한다.

smp_ck_resp[THREAD_CNT] : 요리사가 버거를 다 만들고 직원에게 다 만들어졌다고 알리는 과정의 동기화를 수행한다. 버거가 다 만들어지면 직원은 손님에

게 버거를 전해준다.

smp_drive_order[THREAD_CNT] : 자동차 내 손님이 직원에게 주문을 요청하는 과정의 동기화를 수행한다.

smp_drive_wait_order[THREAD_CNT] : 주문이 완료된 후에 계산을 하러 이동을 한다.

smp_drive_pay_order[THREAD_CNT] : 계산하는 곳에 가서 계산하는 직원에게 계산을 요청하면 계산하는 직원이 그때 계산을 수행한다.

smp_drive_wait_burger[THREAD_CNT] : 자동차에서 버거를 직원이 전해주기까지 기다린다.

c. 상호배제 세마포어

smp_mutex1 : 입장한 손님에 대한 번호를 부여해주기 위해서 전역변수를 사용하였다. 이 전역변수에 대해서 손님 쓰레드들이 접근을 하게 되는데 이때 상호배제를 수행하여 전역변수를 보호한다.

smp_qmutex1 : 입장한 손님의 주문, 계산을 수행하는데 공정하게 순서대로 한명씩 이루어질 수 있도록 큐를 이용하는데, 이 큐는 여러 쓰레드가 동시에 접근하면 안되는 임계영역에 해당하므로 접근할 때 상호배제를 수행한다.

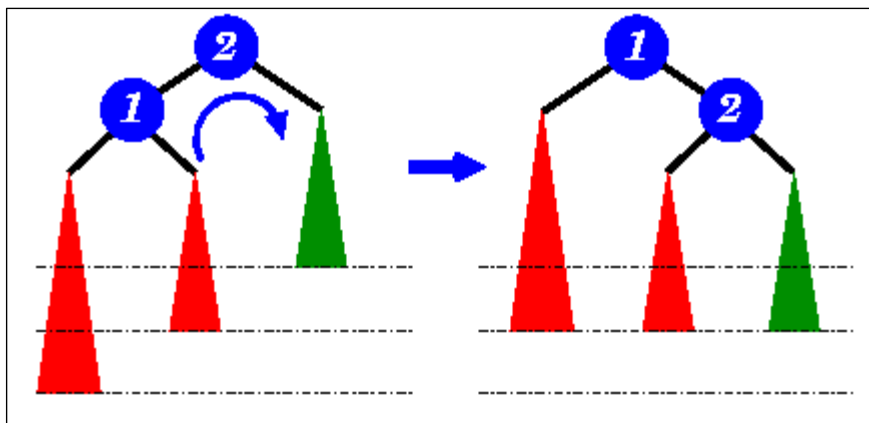
smp_qmutex2 : 손님에게 공정한 순서로 서비스를 제공하기 위해서 직원뿐만 아니라 요리사 또한 서비스를 받아야 하는 순서를 알고 있어야 한다. 역시 큐를 이용하며 해당 큐 또한 상호배제가 필요하다.

smp_qmutex3 : 계산 순서 또한 공정하게 진행하기 위해서 계산원도 큐를 통해서 손님의 순서를 가져온다. 해당 큐에 대한 상호배제를 위한 세마포어이다.

3.2.3 메모리 DB를 위한 트리 구현 전략

(1) DB구현에 사용한 Tree에 대해 기술

DB구현에 사용할 Tree로 초기에 AVL tree와 B tree에 대해서 자료조사를 수행했다. 조사 결과에 따르면, AVL tree는 탐색속도가 빠르고 트리 전체를 재배열 시키지 않아도 트리의 균형이 유지가 되고 B-tree는 대용량일수록 효율적이며, 짝수 홀수에 따라 알고리즘이 달라지는 문제가 있어서 트리로는 AVL tree를 사용하기로 하였다.



AVL tree의 시간복잡도를 살펴보았을 때 검색, 삽입, 삭제를 수행할 때 평균의 경우와 최악의 경우 둘 다 $O(\log n)$ 의 시간복잡도를 갖는 것을 확인할 수 있다.

삽입과 삭제는 균형인수를 기준으로 회전을 통해 노드들을 배치하고, 탐색은 해당 key를 통해 노드가 있는지 알려준다. 균형인수는 서브트리의 높이 차로 이 균형인수를 바탕으로 상황에 맞는 회전을 수행해야 한다. 이때 각 상황에 맞게 LL, RR, LR, RL 회전을 수행해준다. 이 회전을 함으로써 트리는 균형을 갖추게 되고 결과적으로 탐색 시간을 줄이게 된다.

삽입은 노드의 키 값을 기준으로 균형에 맞추어 왼쪽이나 오른쪽으로 삽입이 되고, 탐색은 키 값을 기준으로 작으면 왼쪽 서브트리, 크면 오른쪽 서브트리를 탐색하는 형태로 구성이 된다. 수정은 기존의 있는 노드를 삭제하고 새로 삽입하거나, 기존 노드를 수정하고 균형을 맞추는 회전을 수행함으로써 구현할 수 있다. 삭제 또한 키 값으로 해당하는 노드를 찾고 삭제를 한 후에 회전을 다시 수행해줌으로써 트리의 균형을 유지시켜준다.

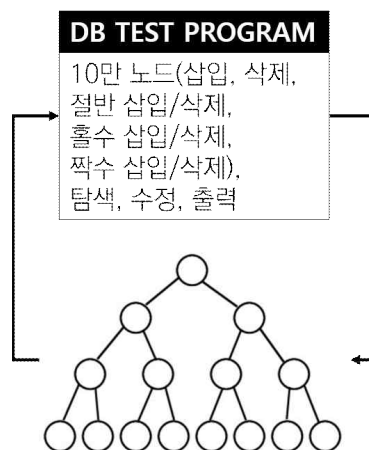
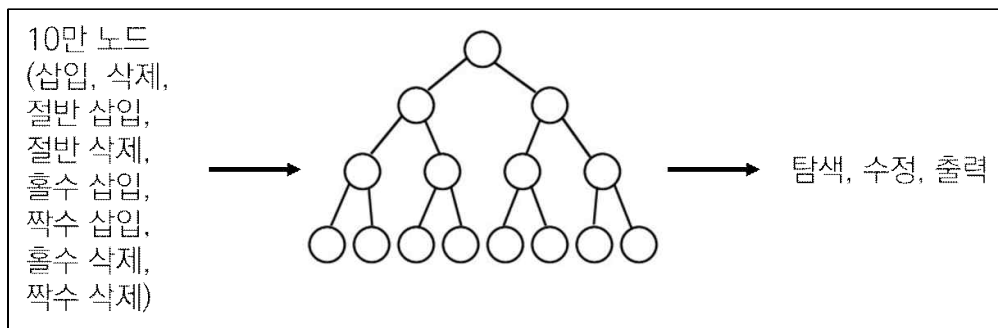
(2) Tree Node 구조체(DB 레코드) 설계

DB 레코드는 다음과 같이 정의하였다.

필드명	데이터 타입	내용
key	int	레코드 키(Primary Key) 값
point_card	bool	포인트 카드 유무
menu	string(char*)	메뉴 주문, 개수, 가격 정보
cal	string(char*)	계산 방식(현금/카드)
nowtime	string(char*)	주문 시각

위의 레코드를 기준으로 AVL Node를 설계한다. 노드는 삽입, 삭제, 탐색하는데 사용하는 키 값과 주문한 메뉴 등에 대한 정보를 담고 있으며 이 노드는 하나의 정보 단위로써 클래스로 설계한다. 메뉴 정보에 대해서는 메뉴 이름과 가격을 동시에 가져오기 위한 자료구조로써 map을 사용한다.

(3) DB 테스트 전략



AVL Tree로 생성한 DB를 테스트하기 위해서 삽입, 삭제 등의 경우에 10만개의 노드를 삽입하고 삭제하는 테스트를 수행하여 Tree에 문제가 있는지 테스트를 진행하도록 설계했다. 테스트는 랜덤한 값을 가지는 노드를 삽입하고 삭제하는 방식으로 테스트 결과 문제없이 Tree에 저장하고 삭제가 되는 것으로 테스트가 완료가 된다.

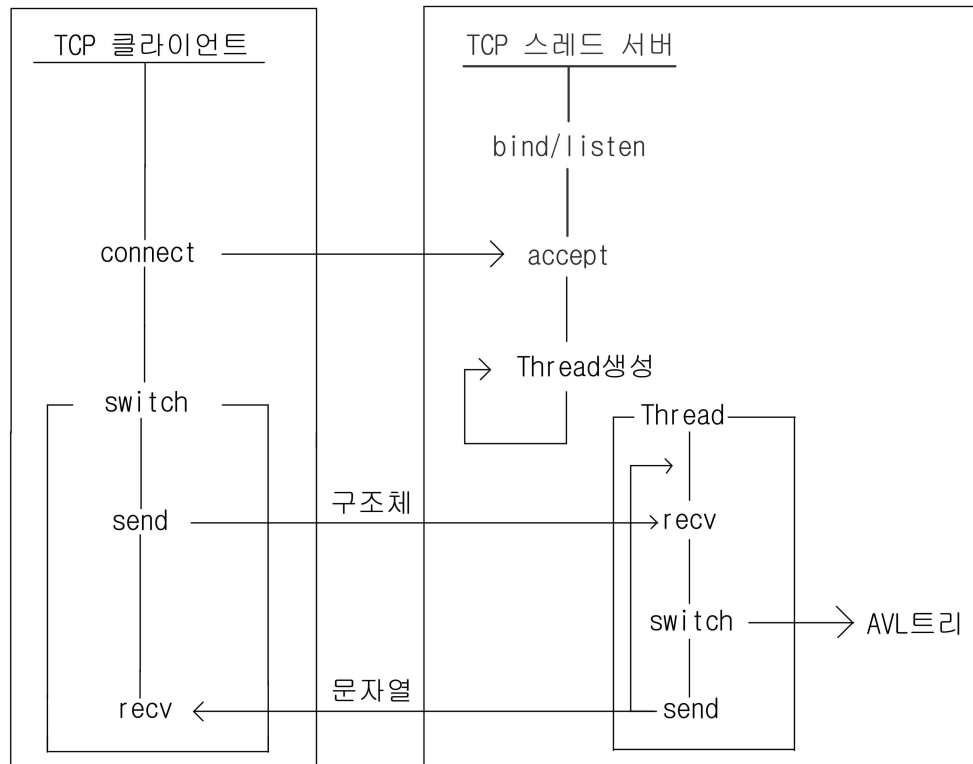
3.2.4 DB의 저장 및 복구전략

시뮬레이션 프로그램을 시작하고 종료할 때 결과의 내용이나 DB의 데이터를 파일로 저장하고 복구하기 위해서 File I/O를 수행한다.

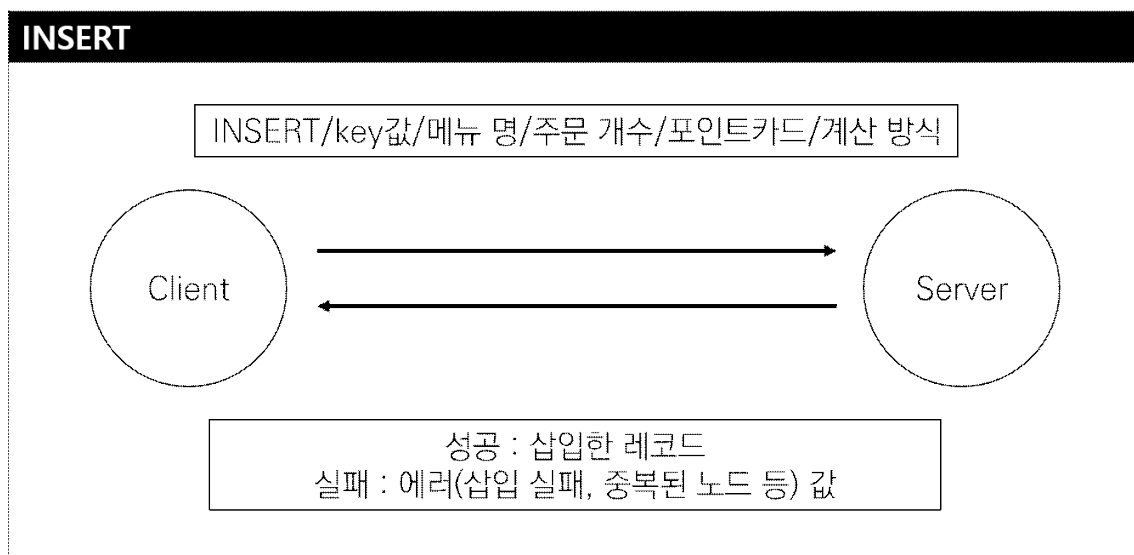
File I/O를 수행할 때 노드의 데이터에 대해서는 텍스트 형태가 아닌 바이너리 형태로 저장을 한다. 바이너리 형태를 이용할 경우 실제 40000원이라는 데이터에서 40000이라는 값을 저장할 때 텍스트로 저장한다면 5바이트가 필요하지만 바이너리로 저장할 경우에는 2바이트로 충분히 저장이 가능하다. DB에 저장하고 복구하기 위한 전략으로는 트리의 데이터를 어떻게 탐색하느냐가 중요한데, 탐색 방식의 후보로 너비 우선 탐색이나 깊이 우선 탐색 등의 방법을 후보로 두고 어떤 방식이 유리할지 조사를 한 후에 적용할 계획이다.

3.2.5 네트워크를 이용한 서버와 클라이언트의 연결

TCP/IP 소켓 프로그래밍을 통해서 서버와 클라이언트를 제작해서 레코드에 대한 정보를 주고받는 네트워크 통신 구조는 다음과 같이 설계하였다.



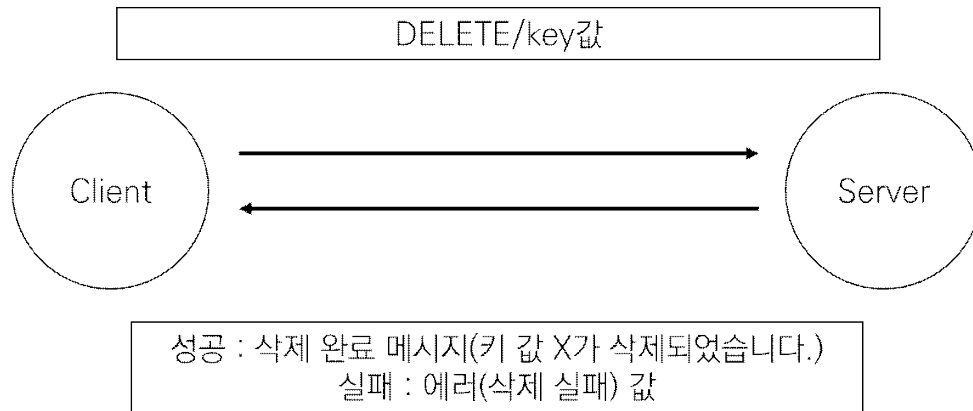
데이터 전송 단위는 레코드 한 개 단위로 전송을 주고받는 형태로 설계하였고, 실제 데이터를 주고받는 프로토콜은 다음과 같이 설계하였다.



수신한 레코드를 이용해 새로운 노드를 생성, AVL트리에 삽입한다. 입력 받은 키

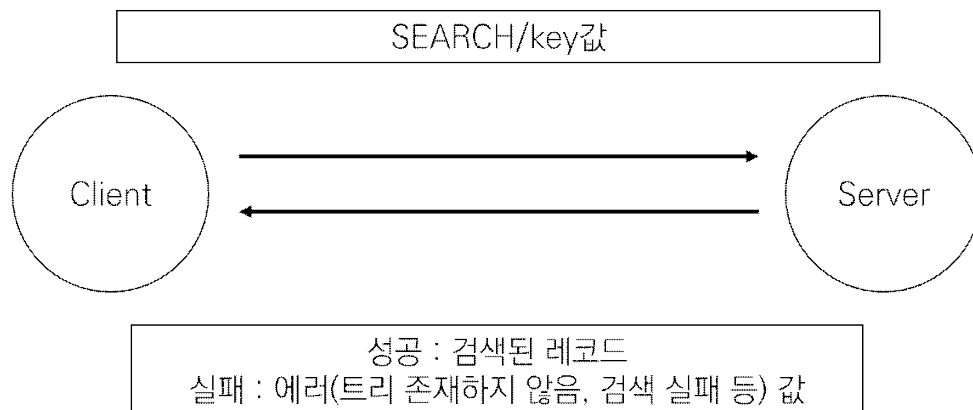
값이 이미 AVL트리에 존재하면 중복된 키 값 에러 발생(-6), 삽입 성공 시 삽입한 레코드 값을 송신

DELETE



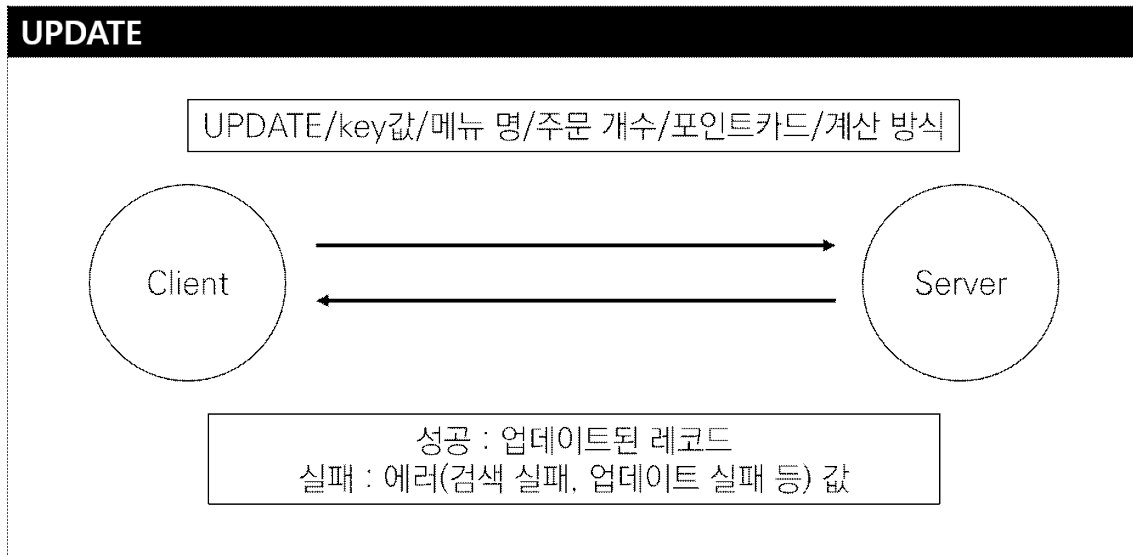
수신한 키값에 해당하는 노드를 삭제한다. 삭제 전 Search를 수행, 찾을 수 없을 경우 삭제 실패 에러 발생(-7), 삭제 성공 시 삭제 성공 메시지 송신

SEARCH



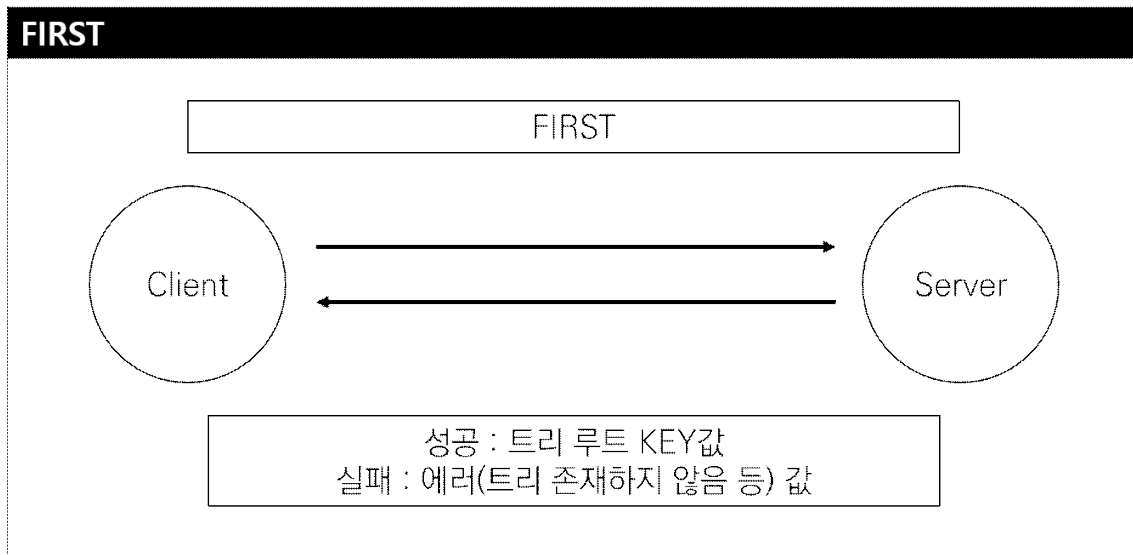
수신한 키값에 해당하는 노드를 검색한다. 검색 실패 시 검색 실패 에러 발생(-8), 검색 성공 시 검색한 레코드 값 송신

UPDATE



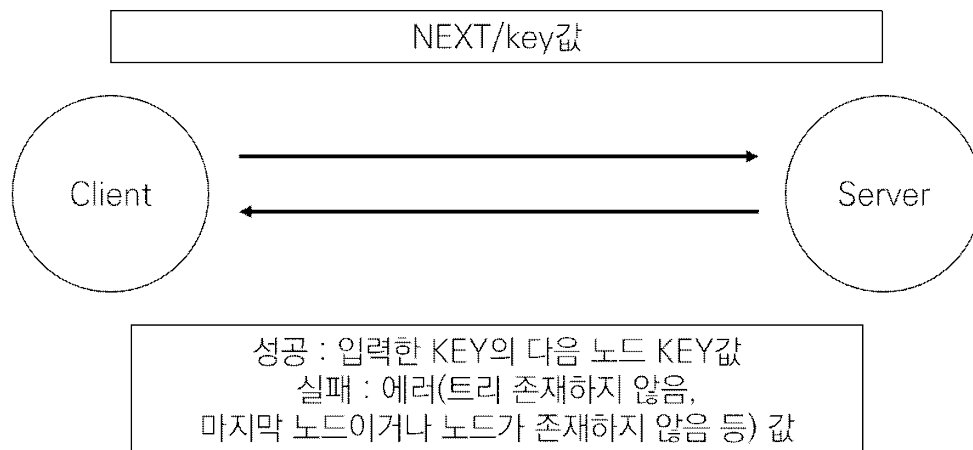
수신한 키 값의 레코드를 수신 받은 레코드로 변경한다. 수정 전 Search를 수행, 찾을 수 없을 경우 수정 실패 에러 발생(-9), 수정 성공 시 수정된 레코드를 송신

FIRST



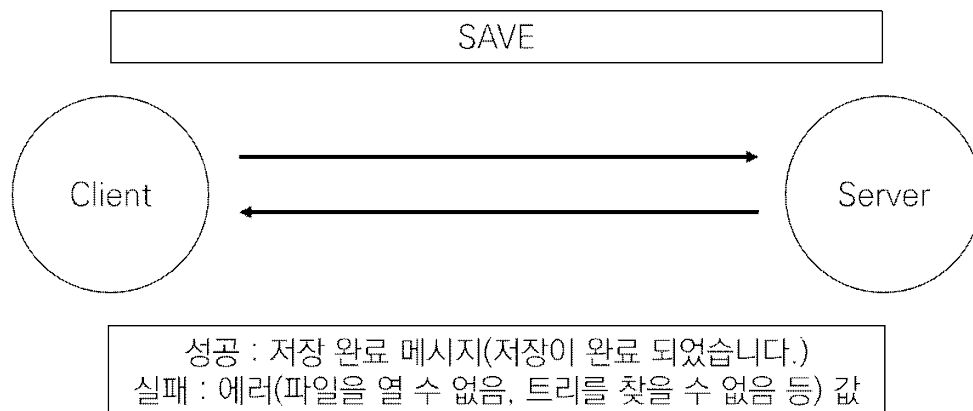
트리의 첫 번째에 해당하는 노드의 키값을 송신한다. 트리가 없을 경우 트리 없음 에러 발생(-3)

NEXT



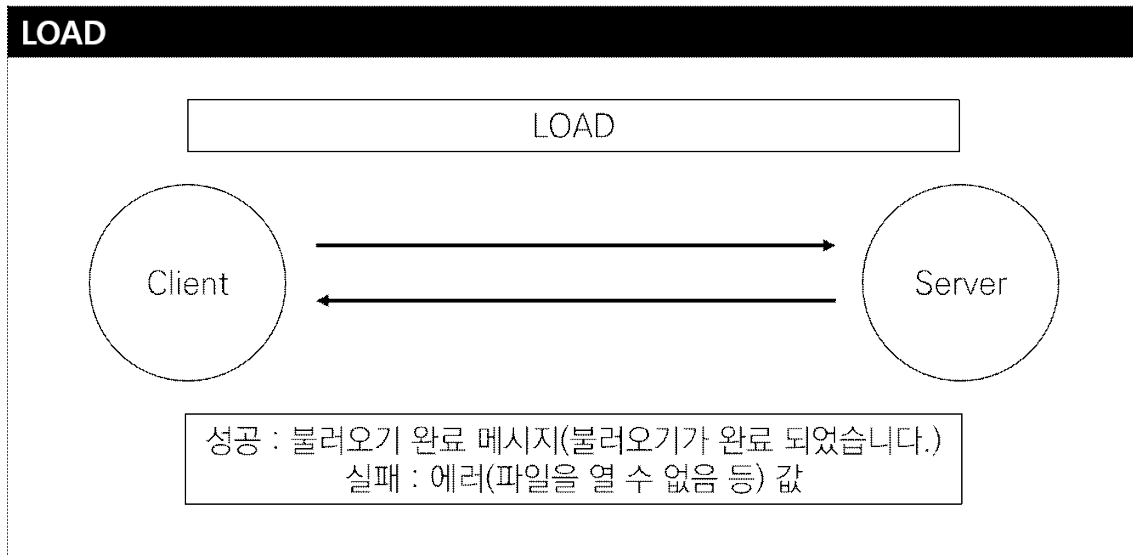
수신한 키 값의 다음 노드의 키값을 송신한다. 수신한 키값이 마지막 노드이거나 존재하지 않을 경우 마지막 노드 에러 발생(-10)

SAVE



AVL 트리를 data.bin 저장한다. 저장 실패 시 파일 열기 실패 에러 발생(-4)

LOAD



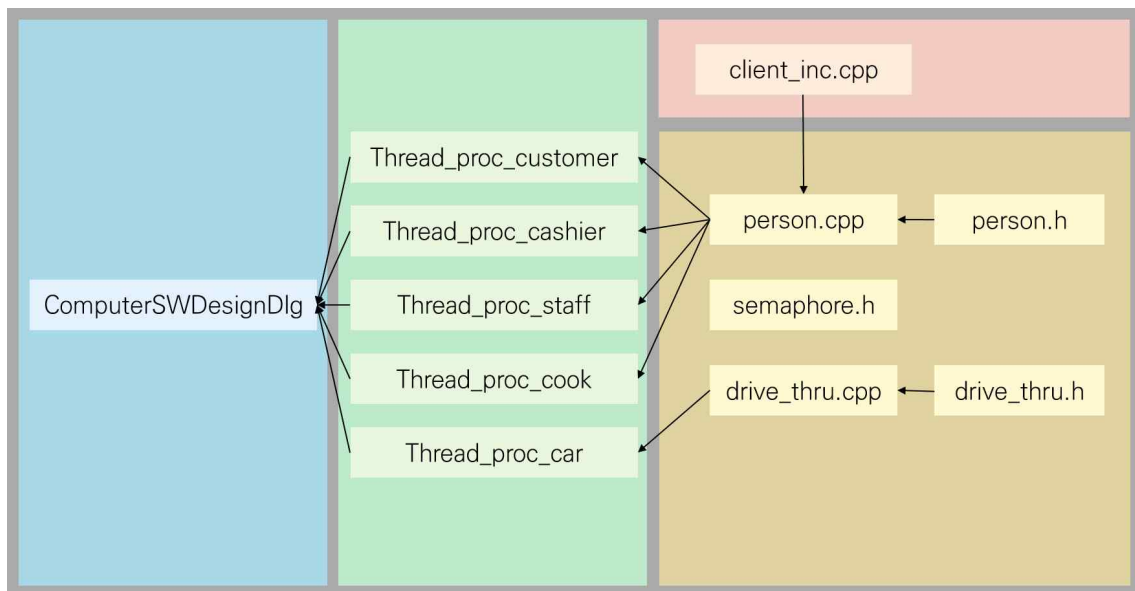
data.bin에서 레코드를 읽어와 AVL트리에 삽입한다. 불러오기 실패 시 파일 열기 실패 에러 발생(-4)

서버의 경우 다수의 클라이언트로부터 들어오는 접속/서비스 요청을 처리하기 위해 스레드를 사용하였으며 DB에 동시에 접근하는 것을 막기 위해서 상호배제를 수행하였다.

4. 프로젝트 구현

본 프로젝트는 앞서 진행한 설계를 바탕으로 구현되었으며, 구현 환경은 다음과 같이 구성되었다.

프로젝트 구현 환경	
개발언어	C++
개발도구	Visual Studio 2015
개발환경	Windows



전체 프로그램의 구현 파일 구성은 위와 같이 이루어져 있다. GUI 부분은 ComputerSWDesignDlg에서 담당하면서 동시에 쓰레드 시작함수들을 실행시켜준다. 각각의 쓰레드 시작함수에서는 손님, 캐셔, 직원, 요리사 등의 객체에 대한 주소 값을 받아서 캐릭터들의 동작을 진행시켜준다. 각 캐릭터들은 클래스로 정의되어 있으며 person.cpp와 person.h에 구현되어 있다. 마찬가지로 자동차에 관한 부분은 drive_thru.cpp와 drive_thru.h에 구현되어 있다. 각 쓰레드가 실행되면서 필요한 세마포어에 대한 정보는 semaphore.h에 선언되어 있어 별도의 파일마다 따로 선언할 필요 없이 구현되어 있다. 네트워크는 client_inc.cpp 파일을 포함시키면 별도의 다른 파일에 대한 참조 없이 관련된 함수에 접근이 가능하도록 구현되었다.

4.1 주요 구현 내용

4.1.1 Thread, Semaphore & GUI

(1) Thread 생성

```
ComputerSWDesignDlg.cpp
/* Create Staff threads */
for (i = 0; i < 3; i++) {
    thread_staff[i] = CreateThread(NULL, 0,
(LPTHREAD_START_ROUTINE)thread_proc_staff, &obj_stf[i], 0, &staff_id);
    if (thread_staff[i] == NULL) {
        _tprintf(_T("CreateThread _staff_ error : %d\n"),
GetLastError());
        return 1;
    }
}

/* Create cashier thread */
thread_cashier = CreateThread(NULL, 0,
(LPTHREAD_START_ROUTINE)thread_proc_cashier, NULL, 0, &cashier_id);
if (thread_cashier == NULL) {
    _tprintf(_T("CreateThread _cashier_ error : %d\n"), GetLastError());
    return 1;
}

/* Create Customer threads */
for (i = 0; i < THREAD_CNT; i++) {
    thread_customer[i] = CreateThread(NULL, 0,
(LPTHREAD_START_ROUTINE)thread_proc_customer, &obj_cstmr[i], 0, &customer_id);
    if (thread_customer[i] == NULL) {
        _tprintf(_T("CreateThread _customer_ error : %d\n"),
GetLastError());
        return 1;
    }
}
}
```

필요한 스레드를 생성하는 것을 확인할 수 있다. 여러 개의 스레드가 필요한 만큼 스레드 배열을 생성하고 적절히 CreateThread 함수를 이용하여 스레드를 생성하고 실행시켜준다.

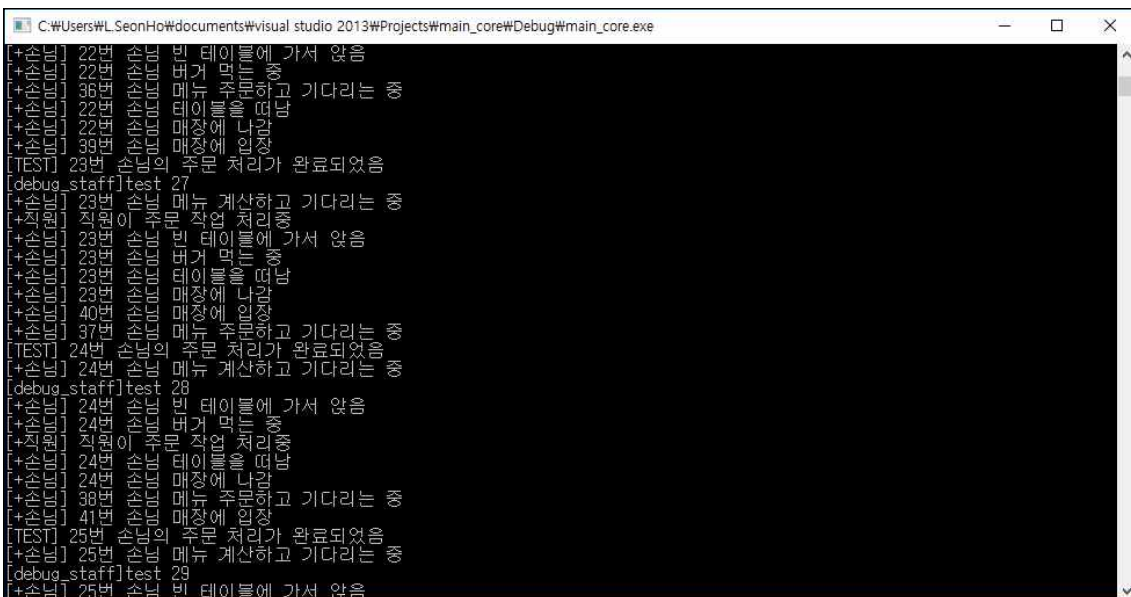
(2) 손님 Thread 실행 함수

```
thread_proc_customer.cpp
wait_enter = WaitForSingleObject(smp_gate, INFINITE);
obj_cstmr->enter_Mcd(); // 매장에 입장

// 전역변수 customer_number가 임계영역이므로 상호배제 수행
mutex1 = WaitForSingleObject(smp_mutex1, INFINITE);
cstmr_num = customer_number;
customer_number++;
if (!ReleaseSemaphore(smp_mutex1, 1, NULL))
    printf("ReleaseSemaphore _mutex1_ error : %d\n", GetLastError());

// 전역변수 Queue에 접근할 때 상호배제 수행
mutex2 = WaitForSingleObject(smp_qmutex1, INFINITE);
Queue1.push(cstmr_num);
// Queue 상호배제 해제
if (!ReleaseSemaphore(smp_qmutex1, 1, NULL))
    printf("ReleaseSemaphore _mutex2_ error : %d\n", GetLastError());
```

손님 Thread의 실행 함수의 일부이다. 각 손님 객체의 메소드를 실행하면서 시뮬레이션을 진행하고 있으며, 각 쓰레드에 대한 번호를 얻어온다. 이때 전역변수에 대한 접근에 대해서 상호배제를 수행하고 있고, 이후에 손님의 번호를 일종의 번호표처럼 주문 큐에 집어넣을 때, 큐에 대한 상호배제를 세마포어를 통해서 수행하고 있다.



```
C:\Users\W.L.Seon\Documents\visual studio 2013\Projects\main_core\Debug\main_core.exe
[+손님] 22번 손님 빈 테이블에 가서 앉음
[+손님] 22번 손님 버거 먹는 중
[+손님] 36번 손님 메뉴 주문하고 기다리는 중
[+손님] 22번 손님 테이블을 떠남
[+손님] 22번 손님 매장에 나갈
[+손님] 36번 손님 매장에 입장
[TEST] 23번 손님의 주문 처리가 완료되었음
[debug_staff] test 27
[+손님] 23번 손님 메뉴 계산하고 기다리는 중
[+직원] 직원 이 주문 작업 처리중
[+손님] 23번 손님 빈 테이블에 가서 앉음
[+손님] 23번 손님 버거 먹는 중
[+손님] 23번 손님 테이블을 떠남
[+손님] 23번 손님 매장에 나갈
[+손님] 40번 손님 매장에 입장
[+손님] 37번 손님 메뉴 주문하고 기다리는 중
[TEST] 24번 손님의 주문 처리가 완료되었음
[+손님] 24번 손님 메뉴 계산하고 기다리는 중
[debug_staff] test 28
[+손님] 24번 손님 빈 테이블에 가서 앉음
[+손님] 24번 손님 버거 먹는 중
[+직원] 직원 이 주문 작업 처리중
[+손님] 24번 손님 테이블을 떠남
[+손님] 24번 손님 매장에 나갈
[+손님] 38번 손님 메뉴 주문하고 기다리는 중
[+손님] 41번 손님 매장에 입장
[TEST] 25번 손님의 주문 처리가 완료되었음
[+손님] 25번 손님 메뉴 계산하고 기다리는 중
[debug_staff] test 29
[+손님] 25번 손님 빈 테이블에 가서 앉음
```

실제 해당하는 내용을 CLI 환경에서 실행시켰을 때 정상적으로 손님들의 번호에 맞춰서 공정하게 진행이 되고 있다는 것을 확인할 수 있다. 또한 각 메소드들이 실행

행될 때 GUI 상으로 캐릭터들의 좌표를 바꿔줌으로써 시뮬레이션을 진행한다.

(3) 손님 클래스

```
person.h
/* Customer Class */
class Customer
{
    int direction; // 진행 방향을 나타내는 변수
    int avata; // 캐릭터 종류를 나타내는 변수
    int motion; // 캐릭터 진행 동작을 나타내는 변수
    PT cstmr_point;
    char char_buf[1000];
public:
    Customer();
    ~Customer();
    /* Customer function */
    int get_direction();
    int get_avata();
    int get_motion();
    int get_point_x();
    int get_point_y();
    char get_char_buf();
    void Test();
    void enter_Mcd();
    void order(int);
    void pay();
    void find_table();
    void go_to_table(int);
    void eat_burger();
    void leave_table();
    void leave_Mcd();
    void Random();
    int order_num,sit,sit_R, sit_L;
    int n = 0;
    int order_p;
};
```

손님 클래스에는 손님이 가져야 될 정보들을 멤버 변수로 손님이 해야되는 행동들을 메소드로 정의해두었다.

```

person.cpp
void Customer::enter_Mcd(){
    for (int i = 0; i < 30; i++) {
        cstmr_point.y -= 10;
        motion += 1;
        if (motion == 3)
            motion = 1;
        Sleep(150);
    }
}

```

손님 클래스의 메소드들에 대한 내용을 보면 위의 메소드와 같이 손님이 고유하게 가지는 좌표 값들을 수정해주는 역할만 수행한다는 것을 확인할 수 있다.

(4) Semaphore 생성

```

ComputerSWDesignDlg.cpp
// [카운팅] 매장안으로 손님이 20명만 입장하도록 제한한다.
smp_gate = CreateSemaphore(NULL, MAX_CUSTOMER_CNT, MAX_CUSTOMER_CNT, NULL);
if (smp_gate == NULL) {
    _tprintf(_T("CreateSemaphore _gate_ error : %d\n"), GetLastError());
    return 1;
}

for (i = 0; i < THREAD_CNT; i++) {
    // [동기화] 손님이 직원에게 갔을 때 직원이 주문을 받는다.
    smp_staff[i] = CreateSemaphore(NULL, 0, 1, NULL);
    if (smp_staff == NULL) {
        _tprintf(_T("CreateSemaphore _staff_ error : %d\n"),
        GetLastError());
        return 1;
    }
}
}

```

카운팅 용도로 사용할 세마포어의 경우는 초기 값을 카운팅할 수 만큼으로 설정해 주고 생성해주면 된다. 동기화 용도로 사용하는 세마포어는 세마포어 배열을 생성하여 각각 생성해주어야 한다. 세마포어 생성에는 CreateSemaphore 함수를 이용한다. 생성된 세마포어를 각 쓰레드 실행 함수에서 사용해야 하는데, 프로그램 구조상 실행 함수들을 별도의 cpp 파일로 분리를 해놓은 상황이므로 세마포어에 대한 선언은 따로 헤더파일로 구성하여 사용하였다.

semaphore.h

```
#ifndef SEMAPHORE_H
#define SEMAPHORE_H

#ifdef PERSON
#include <Windows.h>
#endif

extern HANDLE smp_gate;
extern HANDLE smp_staff[50];
extern HANDLE smp_wait_order[50];
extern HANDLE smp_cashier[50];
extern HANDLE smp_burger[50];
extern HANDLE smp_table;
extern HANDLE smp_ck_req[50];
extern HANDLE smp_ck_resp[50];
extern HANDLE smp_mutex1;
extern HANDLE smp_wait_mutex;
extern HANDLE smp_qmutex1;
extern HANDLE smp_qmutex2;
extern HANDLE smp_qmutex3;
extern HANDLE smp_dc_eMcd;
extern HANDLE smp_dc_order;

/* drive thru */
extern HANDLE smp_wait_dt;

#endif SEMAPHORE_H
```

(5) GUI 구현

GUI 구현은 기본적으로 MFC를 이용하였다. MFC의 다이얼로그를 사용함으로써 배경 그래픽과 캐릭터들에 대한 표현을 하였다. MFC에서는 GUI에 필요한 요소들을 각각 클래스로 정의한다. 따라서 MFC 사용자는 해당 클래스들에 메시지 처리 함수나 필요한 경우 사용자 정의 함수를 구현해서 이용한다.

```
ComputerSWDesignDlg.cpp
BOOL CComputerSWDesignDlg::OnInitDialog()
{
    CDialogEx::OnInitDialog();

    // 메인 다이얼로그 외부 참조 포인터 초기화
    pDlg = this;

    obj_cstmr = new Customer[50];
    obj_stf = new Staff[3];
    obj_stf[0].set_point(500, 365);
    obj_stf[1].set_point(410, 365);
    obj_stf[2].set_point(320, 365);
    // 이후 쓰레드, 세마포어, 윈도우 창 생성과 관련된 루틴 진행
}
```

다이얼로그 클래스에서는 OnInitDialog() 메소드를 이용할 수 있는데 이 메소드에서는 초기 다이얼로그에 관련된 부분을 다룬다. 필요한 초기 작업이 있으면 여기서 구현한다. 본 프로젝트에서는 다이얼로그가 생성되면서 필요한 쓰레드, 세마포어 등이 생성되게 했다.

```
ComputerSWDesignDlg.cpp
void CComputerSWDesignDlg::OnPaint()
{
    if (IsIconic()){
    else
    {
        CPaintDC dc(this);

        CDC memDC;
        CDC mdcOffScreen;
        CBitmap bmpOffScreen;

        CBitmap *oldbitmap;
```

OnPaint() 함수에서는 실질적으로 배경과 캐릭터들을 찍어내는 작업을 수행했다. DC에 이미지를 그리기 위해서 현재 다이얼로그의 DC를 얻어오고, 더블버퍼링 작

업을 위해서 메모리 DC를 생성했다.

```
ComputerSWDesignDlg.cpp
A1_F1.LoadBitmapW(IDB_A1_F1);
A1_F1.GetObject(sizeof(BITMAP), (LPVOID)&A1_bmpinfo);
```

이 함수 안에서 필요한 비트맵 이미지도 로드하여 준다. 비트맵 이미지는 미리 리소스로 등록을 시켜둔다.

```
ComputerSWDesignDlg.cpp
memDC.CreateCompatibleDC(&dc);
mdcOffScreen.CreateCompatibleDC(&dc);
bmpOffScreen.CreateCompatibleBitmap(&dc, m_Bitmap.bmWidth, m_Bitmap.bmHeight);
oldbitmap = mdcOffScreen.SelectObject(&bmpOffScreen);
```

memDC와 mdcOffScreen은 DC에 그리기 위한 영역이므로 관련된 처리를 해주고 DC에 대한 정보를 넘겨주고 실질적으로 그리기 위한 일종의 그리기 도구를 선택해주는 SelectObject를 호출하여준다.

```
ComputerSWDesignDlg.cpp
memDC.SelectObject(&A1_H1);
mdcOffScreen.TransparentBlt(obj_cstmr[i].get_point_x(),
obj_cstmr[i].get_point_y(), A1_bmpinfo.bmWidth / 4, A1_bmpinfo.bmHeight / 4,
&memDC, 0, 0, A1_bmpinfo.bmWidth, A1_bmpinfo.bmHeight, RGB(0, 255, 0));
```

이후 객체의 상태에 따른 비트맵을 적절히 memDC와 mdcOffScreen에 그려준다. 이렇게 그려진 메모리 DC들은 아래와 같이 최종적으로 한번에 출력이 된다.

```
ComputerSWDesignDlg.cpp
dc.BitBlt(0, 0, m_Bitmap.bmWidth, m_Bitmap.bmHeight, &mdcOffScreen, 0, 0,
SRCCOPY);
```

이러한 일련의 작업을 통해서 GUI 구현이 완성된다.

4.1.2 DB, Network, File I/O

(1) AVL Tree 구현

```
SERVER_AVL.h
class AvlNode
{
public:
    int key;
    char menu[MAX_MENU_LENGTH];
    AvlNode *left, *right;
    int height;
    bool point_card;           // 포인트카드 유:1 , 무: 0
    char cal[20];              // 지불방식 현금,카드
    char nowtime[sizeof "2016-11-22T06:08:09Z"];
public:
    AvlNode(int key, char menu[MAX_MENU_LENGTH], int height, bool point_card,
char cal[20])
    { ... }

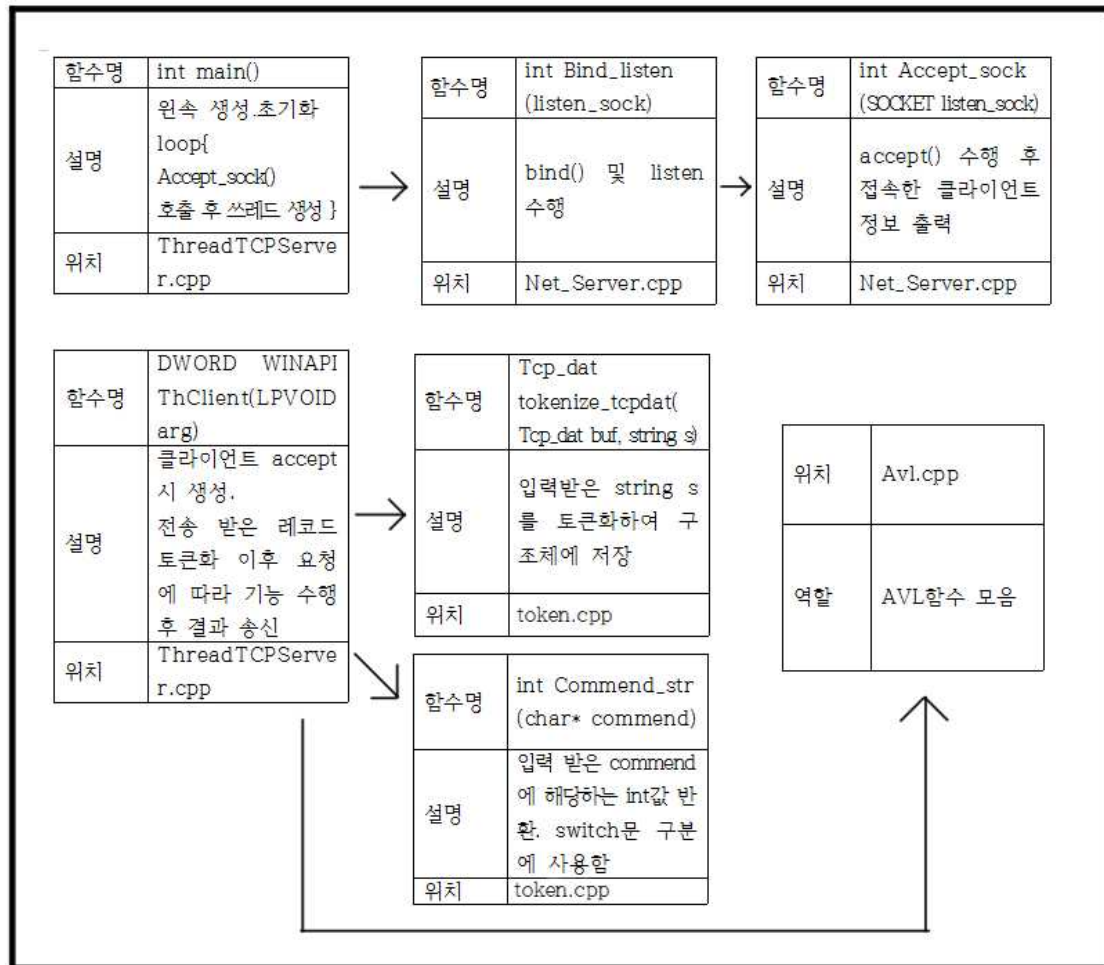
    AvlNode(int key, char menu[MAX_MENU_LENGTH], int height, bool point_card,
char cal[20], char loadtime[sizeof "9999-12-31T99:99:99Z"])
    { ... }

};

AvlNode *right_Rotate(AvlNode *y);/* 오른쪽으로 회전시키는 함수 */
AvlNode *left_Rotate(AvlNode *x);/* 왼쪽으로 회전시키는 함수 */
int height(AvlNode *N);/* 트리의 높이를 얻는 함수 */
int getBalance(AvlNode *N);/* 노드N의 균형인수 반환하는 함수 */
AvlNode* insert(AvlNode* rootNode, AvlNode* newNode);/* Avl 트리의 삽입 함수 */
AvlNode* deleteNode(AvlNode* root, int key);/* Avl 트리의 삭제 함수 */
void updateNode(int x, AvlNode* node, Tcp_dat buf);/* Avl 트리의 수정 함수 */
AvlNode * searchNode(int x, AvlNode* &node);/* Avl 트리의 탐색 함수 */
void preOrder(AvlNode *root);/* Avl 트리의 전위순회(pre-order) 및 출력 함수 */
AvlNode * minValueNode(AvlNode* node);/* 노드 삭제할 시 가장 작은 key값을 찾아주는 함수 */
```

AVL Tree에 필요한 노드에 대한 정보와 메소드들은 하나의 클래스로 구현함으로써 이후 사용하는데 편리하도록 구현하였다.

(2) Network Server 구현



네트워크 서버는 위와 같은 순서 도표로 진행이 된다. 서버는 기본적으로 클라이언트가 요청한 DB 트랜잭션을 처리해주는 역할을 하는데 트랜잭션의 경우 설계단계에서 프로토콜 형태로 각 수행 절차 등을 정의해두었다. 그 내용을 기반으로 구현한 각 기능별 서버의 수행절차는 다음과 같다.

함수	클라이언트	서버
order_send	레코드를 전송 후 수신 대기	레코드 수신, 레코드를 토큰화한 후 key값 체크, key가 음수일 경우 에러.
insert	레코드 생성 후 전송 INSERT/KEY/메뉴/주문개수/포인트 카드/계산방식 수신 대기	key값이 0일 경우 key값 자동 생성 후 insert key값이 0이 아닐 경우 search 수행, key에 해당하는 노드 존재 시 에러 없을 시 삽입. 삽입된 노드 정보 송신
search	레코드 생성 후 전송	key값을 검색.

	SEARCH/KEY/0/0/0/0 수신 대기	검색 실패 시 에러 성공 시 검색한 노드 정보 송신
delete	레코드 생성 후 전송 DELETE/KEY/0/0/0/0 수신 대기	key값을 검색 검색 실패 시 에러 성공 시 노드 삭제. 삭제 성공 메시지 송신
update	레코드 생성 후 전송 UPDATE/KEY/메뉴/주문개수/포인트 카드/ 계산방식 수신대기	key값을 검색 검색 실패 시 에러 성공 시 노드 수정 수정된 노드 정보 송신
next	레코드 생성 후 전송 NEXT/KEY/0/0/0/0 수신대기	key값이 마지막 노드이거나 없는 노드이면 에러 성공 시 key의 다음 노드의 키를 송신
first	레코드 생성 후 전송 FIRST/0/0/0/0/0 수신대기	트리가 없을 경우 에러 성공 시 트리의 첫 번째 노드 의 키를 송신
random	레코드 생성 후 전송 RANDOM/0/0/0/0/0 수신대기	vector에 저장돼있는 key값 들 중 하나를 무작위로 가져 와 송신
preorder	최초 first 실행 후 반복문 do{ Search() }while(Next == 0)	client 요청 수행
allDelete	First() do{ Com_Search() }while(First == 0)	client 요청 수행
save	레코드 생성 후 전송 save/0/0/0/0/0 수신대기	트리가 없을 시 에러 파일 오픈 실패 시 에러 save 수행 후 성공 메시지 송신
load	allDelete 수행 후 레코드 생성 전송 load/0/0/0/0/0 수신대기	allDelete 수행 후 파일 오픈 실패 시 에러 load 수행 후 성공 메시지 송신

실제 서버 프로그램에서는 클라이언트의 접속 요청만을 기다리고 있으며 접속 요청이 들어올 경우 서비스를 처리해줄 쓰레드를 하나 생성해준다. 다음은 실제 해당 내용이 구현된 코드의 내용이다.

ThreadTcpServer.cpp

```

int main(int argc, char* argv[])
{
    //CriticalSection 초기화
    InitializeCriticalSection(&cs);

    int retval;
    //원속 초기화
    WSADATA wsa;
    if (WSAStartup(MAKEWORD(2, 2), &wsa) != 0)
        return -1;

    SOCKET listen_sock = socket(AF_INET, SOCK_STREAM, 0);
    if (listen_sock == INVALID_SOCKET) err_quit("socket (");

    retval = Bind_listen(listen_sock);

    SOCKET client_sock;
    HANDLE hThread;
    DWORD ThreadId;

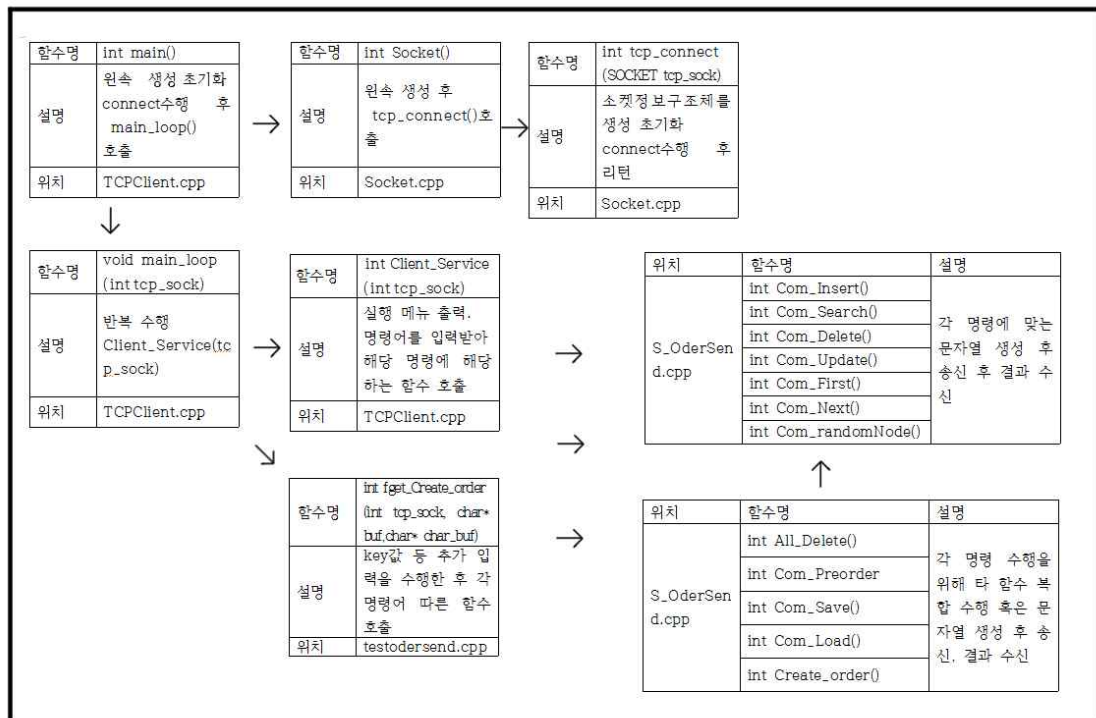
    insert_map();

    while (1)
    {
        cout << "클라이언트 Connect 대기 중..." << endl;
        client_sock = Accept_sock(listen_sock);
        hThread = CreateThread(NULL, 0, ThClient, (LPVOID)client_sock, 0,
&ThreadId);

        if (hThread == NULL)
            printf("[오류] 쓰레드 생성 실패\n");
        else
            CloseHandle(hThread);
    }
    closesocket(listen_sock);
    //CriticalSection 제거
    DeleteCriticalSection(&cs);
    //원속 종료
    WSACleanup();
    return 0;
}

```

(3) Network Client 구현



네트워크 클라이언트는 위와 같은 순서 도표로 진행이 된다. 클라이언트는 DB 트랜잭션 명령을 날리는 역할을 하는데, RPC 기법을 이용하여 클라이언트는 로컬에서 DB 트랜잭션을 수행하는 것처럼 느끼게 된다. 마찬가지로 트랜잭션에 관해서는 설계단계에서 정의해둔 프로토콜을 따른다. 예시로 Insert 명령의 경우는 다음과 같이 구현이 되어 있다.

```

S_OderSend.cpp
int Com_Insert(int tcp_sock, char* char_buf, int key) {
    int err=0;
    cout << "WtWt:::insert 실행합니다." << endl;
    sprintf(char_buf, "insert/%d/%s/%d/%d/%s",key, menu[random(20)],
    random(10) + 1, random(2), cal[random(2)]);

    if ((err = order_send(tcp_sock, char_buf)) < 0) {
        return err;
    }
    if ((err = atoi(char_buf)) < 0) {
        return err;
    }
    cout << "WtWt:::수신 데이터 : " << char_buf << endl;
    return 0;
}

```

(4) File I/O 구현

File I/O의 경우 서버측에서 수행하는 작업으로 다음과 같이 구현하였다.

```
ThreadTcpServer.cpp
//save
if (root == NULL){
    cout << "트리가 비었습니다." << endl;
    sprintf(insert_buf, "-3");
}
else{
    fout.open("data.bin", ios::binary | ios::out);
    if (fout.is_open()) {
        cout << "저장합니다." << endl;
        next_nodekey = root->key;
        do {
            newNode = searchNode(next_nodekey, root);
            if (judge) {
                buf = tokenize_tcpdat_menu(buf, newNode->menu);
                file_buf.menu = map_at(buf.menu, 0);
                file_buf.num = atoi(buf.num);
                file_buf.cal = map_at(newNode, 0);
                strcpy(file_buf.time, newNode->nowtime);
                file_buf.key = newNode->key;
                file_buf.point_card = newNode->point_card;
                fout.write((char*)&file_buf, sizeof(file_buf));
                Next_Node(root, newNode->key);
            }
        } while (!(next_nodekey < 0));
        sprintf(insert_buf, "저장 완료");
        cout << "저장 완료" << endl;
    }
    else{
        cout << "파일 오픈 실패" << endl;
        sprintf(insert_buf, "-4");
    }
    fout.close();
}
retval = send(client_sock, (const char*)insert_buf, strlen(insert_buf), 0);
if (retval == SOCKET_ERROR) {
    err_display("send()");
    break;
}
break;
```

ThreadTcpServer .cpp

```

//load
fin.open("data.bin", ios::in | ios::binary);
if (fin.is_open()) {
    cout << "파일을 불러오는 중 입니다." << endl;
    while ((fin.read((char*)&file_buf, sizeof(file_buf)))) {
        price = map_at(return_menu[file_buf.menu], 1)*file_buf.num;
        //메뉴 레코드 생성
        sprintf(insert_buf, " ||주문 내역 :Wt%SWt갯수 :Wt%SWt가격 :%d||", return_menu[file_buf.menu], file_buf.num, price);

        //복호화 및 노드 생성
        newNode = new AvlNode(file_buf.key, insert_buf, 0,
file_buf.point_card, return_menu[file_buf.cal],file_buf.time);
        //트리에 삽입
        root = insert(root, newNode);
        //random key를 위한 nodekey 삽입
        nodekey.push_back(newNode->key);
        if (key <= file_buf.key){
            key = file_buf.key + 1;
        }
    }
    cout << "불러오기 완료" << endl;
    sprintf(insert_buf, "불러오기 완료");
}
else {
    cout << "파일 오픈 실패" << endl;
    sprintf(insert_buf, "-4");
}
fin.close();
retval = send(client_sock, (const char*)insert_buf, strlen(insert_buf), 0);
if (retval == SOCKET_ERROR) {
    err_display("send()");
    break;
}
break;

```

5. 팀원별 역할분담 및 후기

이름	내용
이선호	<ul style="list-style-type: none"> • 손님, 직원, 드라이브 스루의 자동차 동작 등 시뮬레이션 프로그램 전반적인 동작에 관한 멀티 쓰레드 및 세마포어 구현 • 전체 프로그램 구조(서버/클라이언트 역할, File I/O, 멀티 쓰레드/세마포어, GUI) 설계 및 구현 검토 • 멀티 쓰레드 & GUI 통합 작업 및 더블버퍼링 수정 • GUI 이미지출력 및 이동함수 구현 • 각 역할별 구현 내용 검토 및 조정 • 주간 보고서 내용 검토 및 조정
한우탁	<ul style="list-style-type: none"> • AVL Tree / B Tree 자료 조사 • AVL Tree 노드 구현 • AVL Tree 삽입, 삭제, 수정, 탐색 등 기능 구현 • AVL Tree 테스트 • AVL Tree 내용 File I/O 자료 조사 • File I/O 구현
박용희	<ul style="list-style-type: none"> • 배경/캐릭터 제작 관련 자료 조사 • 맵칩을 이용한 배경 제작 • 캐릭터 제작 및 합성 • 더블 버퍼링 구현을 통한 화면 깜빡임 해소 • GUI 이미지출력 및 이동함수 구현
노광준	<ul style="list-style-type: none"> • TCP/IP 소켓 프로그래밍 자료 조사 • 멀티 쓰레드를 통한 데이터 처리를 수행하는 서버 프로그램 제작 • 데이터 처리 요청을 전송하는 클라이언트 프로그램 제작 • 네트워크 통신 프로토콜 정의 및 구현 • 서버/클라이언트 프로그램 테스트 • AVL Tree 프로그램과 서버/클라이언트 프로그램 통합 • File I/O 구현
장성일	<ul style="list-style-type: none"> • 주간 보고서 작성 및 출력 • 프로젝트 일정 점검 • 업무 분담 검토 및 결과물 취합 • 진행 과정 캡처 파일 및 이미지 파일 관리 • 프로젝트 소스 파일 버전 관리 및 백업

이름	후기
이선호	다시는 MFC를 사용하지 않아야겠다고 생각했으며 소프트웨어 공학, 스택오버플로우, 구글이 왜 존재하는지 알게 되었다. 아직도 C++ 활용 능력이 부족하다는 것을 깨달았다. 새벽 3시~4시에 작업을 시켜도 열심히 해준 팀원들에게 매우 감사하다. 다들 각자 맡은 바 역할을 잘 해주어서 성공적으로 프로젝트가 끝난 것 같다.
한우탁	Avl Tree로 선택하기 전에 Tree 종류들을 살펴보았다. 여러 종류의 Tree의 장단점을 비교결과 Avl Tree로 선택하게 됐는데 막상 구현하려니 힘들었다. 처음 짜보는 알고리즘이었기 때문이다. 그래도 차근차근 책과 인터넷을 통해 틀을 만들어놓고 진행을 하였고 Avl Tree를 우리 조의 프로젝트에 맞게 재구성하는 작업을 계속하였다. 많은 기능을 넣기 위해 테스트를 수십 번 진행한 결과 완벽하게 구현이 되었다. 또 File I/O를 구현하는데 좀 까다로운 면이 있었다. 몇 번이나 코드를 다시 짰는지 모르겠다. 다행히도 팀원 노광준과 협력하여 도움을 많이 얻었고 같이 구현에 성공하였다. 이번 프로젝트를 통해 협업의 중요성과 전체적인 맥락에서 살펴볼 수 있는 안목이 중요하다는 사실을 다시 한번 느끼게 되었다. 많은 것을 배우고 경험할 수 있는 시간이었다.
박용희	그림배경과 GUI를 담당하게 되었는데 포토샵도 만져 본적이 없고 프로그램 짜는 능력도 부족하여 부담이 굉장히 컸다. 캐릭터이미지와 배경작업을 하기 위해 많은 이미지를 보고 조립해보고 이리저리 만져가며 포토샵에 대해서 많이 알게 되었고 앞으로의 프로그램 개발에 있어서 많은 도움이 될 것 같다. 그림 작업을 마치고 GUI코딩을 하기 시작했는데 MFC를 학교에서 수강 한 적이 있어서 채택했지만 수업 때는 보지 못했던 기능들이 정말 많았고 처음에 화면에 배경과 캐릭터를 출력하는데 문제가 없었지만 캐릭터 모션과 이동하는 것은 잠자기 직전까지 생각할 정도로 고민이 많았었다. 여차저차 잘 진행 되는지 알았는데 쓰레드와 결합하는 과정에서 GUI가 잘 안돼서 썩 같아있었고 이 때 조장의 도움을 많이 받았다. 이번 프로그램 제작을 통해 지금까지 열심히 안했던 것이 많이 후회가 되었고 많은 것을 배울 수 있었고 조원들에게도 많은 도움을 받을 수 있었고 많은 것을 경험 할 수 있었던 좋은 시간이 되었다.
노광준	네트워크를 맨 처음 담당하게 됐을 때는 많이 걱정이 됐다. 네트워크 관련 지식은 이론으로만 알고 있고 직접 코딩해본적은 없었기 때문이다. 하지만 자료조사를 하고 공부해서 처음 통신에 성공했을 때 매우 기뻐했다. 하지만 통신이 성공했다고 내 역할은 끝이 아니었다. 통신을 통해 무언가를 조작하는 것은 생각처럼 쉽지 않았다. 통신이 아니라면 간단하게 됐을 전체 출력 기능 같은 몇몇 기능은 통신을 통해선 수많은 반복을 통해야했고 새로운 함수를 만들기를 요구하기도 했다. 이런 기능을 하나하나 테스트 프로그램에 구현하면서 많은 자료를 보고 새로운 것을 많이 배웠다. 많은 것을 배운 프로젝트였다.

장성일	<p>처음 설계를 하게 되었을 때는 프로그램을 잘 짜는 것도 아니여서 어설프게 이걸 하겠다고 했다가 프로젝트를 망칠까봐 걱정이 들었다. 하지만 뛰어난 팀원들을 만난 덕분에 직접 프로그램을 개발하지 않고 팀원들은 보조하는 역할을 맡게 되었다. 솔직히 처음에는 좋았다. 보조하는 역할이라 그리 힘든 일도 없었고 많은 것을 할 필요 없었다. 그러나 시간이 점점 지날수록 내 자신이 한심하게 느껴졌다. 매주 모여서 열심히 프로그램을 개발하기 위해 토론하는 팀원들을 보며 아무 말도 못하고 뒤에서 무기력한 내 모습이 정말 싫어졌다. 그럴수록 팀원들에게 더 도움이 되려 했지만 할 수 있는 일은 한계적이었다. 차라리 어설프더라도 프로그램을 맡아 해봤으면 했다. 하지만 수확은 있었다. 보고서를 쓰기 위해서는 필연적으로 모든 코드들을 볼 수가 있었다. 하는 일이 없어 시간이 많았던 나는 이 코드들은 보고 팀원들이 매주 무엇이 바뀌었는지 알려 주어 공부를 할 수 있었다. 비록 이번 프로젝트에서는 보조적인 업무만 했지만 다음 프로젝트에서는 한 파트를 맡아 개발할 수 있는 발판을 마련해준 좋은 기회였다.</p>
-----	--

6. 개발일정

분류	상세 분류	9월			10월				11월				12월			
		2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
계획 수립	시나리오 선정, 일정 계획 수립															
요구 분석	요구사항 분석(통신, DB, GUI 등)															
Thread & Semaphore	Thread & Semaphore 설계															
	각 객체별 Thread 구현															
	Semaphore(카운팅, 상호배제) 구현															
	Semaphore(동기화) 구현															
	Thread & Semaphore 테스트															
GUI	MFC/GDI+ 자료 조사															
	Window 구조 설계															
	전체 GUI 설계															
	Background GUI 출력 구현															
	각 객체별 이미지 출력 구현															
	GUI 테스트															
Database	AVL Tree, B Tree 자료 조사															
	AVL Tree 구조 설계															
	AVL Tree 구현															
	AVL Tree 테스트															
TCP/IP	Socket 통신 프로그래밍 자료 조사															
	Server & Client 설계															
	Server & Client 구현															
	Server & Client 테스트															
File I/O	File I/O 설계															
	File I/O 구현															
	File I/O 테스트															
병합	각 모듈 병합															
테스트	종합 테스트															
보고서 작성	주간 보고서 작성															

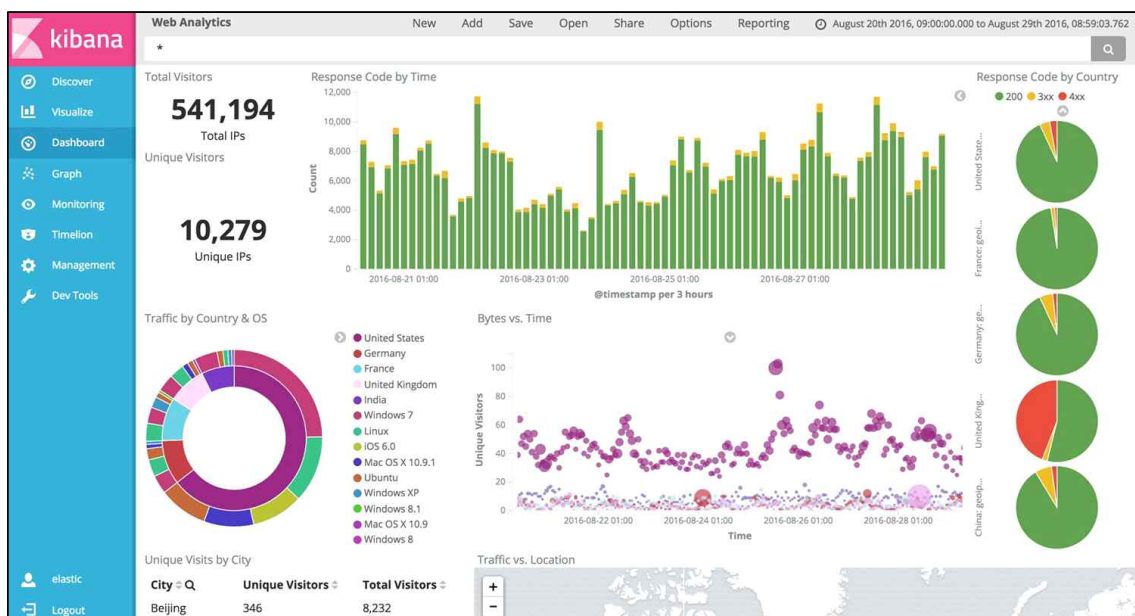
7. 기대효과

7.1 전체 프로그램 활용 가능성

이 시뮬레이션 프로그램을 통해 강남역 12번 출구의 맥도날드 매장의 손님들에 대한 시뮬레이션을 해볼 수 있으며, 이를 통해서 영업과 관련해서 필요한 내용들을 예측할 수 있으며, 필요에 따라서는 관련 데이터를 축적할 수도 있다.



시뮬레이션에서 발생한 데이터는 CSV나 json 형태로 가공하여 Elasticsearch나 Kibana등을 이용하여 데이터 분석을 수행할 수 있다.



7.2 기술적 기대효과

해당 프로그램을 통해서 얻을 수 있는 기술적 기대효과로는 멀티 쓰레드 프로그램을 수행하는데 적합한 환경을 갖추고 있는지 테스트 하는 용도로써도 사용할 수 있을 것이다. 그 외에도 tree 및 서버/클라이언트 통신을 수행하는데 적절한지 테스트할 수 있을 것이다.

7.3 산업적 기대효과

해당 시뮬레이션 프로그램의 GUI나 레코드 항목 등을 수정한다면 다른 업체나 업계에서도 충분히 활용 가능한 시뮬레이션 프로그램으로 사용할 수 있다. 또한 데이터 처리 능력이나 분석 능력이나 리포팅 기능이 추가된다면 더욱 산업적으로 활용 가치가 높은 프로그램으로써 평가받으며 사용될 수 있을 것으로 기대가 된다.

- 부록 -

컴퓨터 소프트웨어 설계
구현 코드