알고리즘
팀 프로젝트

프로젝트 명	Load Balancing
팀 명	GGG
문서 제목	결과보고서

Version	FINAL
Date	2016-11-28

팀원	조 성룡
	최 종호
	홍 은표
지도교수	최 준수 교수

문서 정보 / 수정 내역

Filename	결과보고서 - LoadBalancing.doc
원안작성자	조성룡, 최종호, 홍은표
수정작업자	조성룡, 최종호, 홍은표

수정날짜	대표수정자	Revision	추가/수정 항목	내 용
2016-11-27	조성룡	초안	최초 작성	틀 구성
2016-11-28	조성룡	FINAL	내용 수정	양식을 토대로 작성

본 양식은 컴퓨터공학부 캡스톤 디자인 I 과목의 프로젝트 결과보고서 작성을 위한 기본 양식입니다. 문서의 필수 항목을 제시하는 것이니 폰트, 문단 구조 등의 디자인 부분은 자유롭게 설정하기 바랍니다. 양식 내에 붉은 색으로 기술한 부분은 지우고 작성하기 바랍니다.

목 차

1 프로젝트 목표

1.1 상황가정

목적은 효율적인 주문의 분배에 있다. Load Balancing 구현에 앞서 구체적인 상황이 필요했다. 이에 현 카페의 실태에 맞는 상황을 경험과 여러 자료조사를 통해 가정했다. 상황은 다음과 같다.

1명의 오더
2명의 바리스타
2개의 커피머신
커피는 아메리카노 , 카페라떼 , 마끼야또 3개로 한정

이와 같은 환경 속에서 손님에 의한 주문은 순서대로 input.txt적혀 있고 이 내용을 오더가 전달하며 Load Balancing 알고리즘을 통해 바리스타1과 바리스타2에게 적절히 분배가 된다.

1.2 주문

input.txt내용은 주문 순서만 정한다면 직접 수정 할 수 있으며 주문 순서는 다음과 같다.

(N개의 커피종류) (커피이름) (커피개수) .. N개 만큼 반복

커피의 종류는 헤더파일의 수정을 통해 수정 할 수 있다. 다음은 프로그램 구동에 쓰인 input.txt를 나타낸다.

```
1  3 ame 2 lat 1 mac 4
2  2 ame 4 lat 2
3  1 mac 5
```

위의 input을 보면 총 주문은 3개가 있으며 ame는 아메리카노, lat은 카페라떼, mac는 마끼야또를 의미한다. 1번 째 주문은 2개의 아메리카노, 1개의 카페라떼, 4개의 마끼야또를 의미한다. 2번 째 주문은 1번 째 주문이 끝나야 진행이 되며 3번 째 주문까지 완료가 되면 프로그램은 종료한다.

2 알고리즘 & 소스

2.1 알고리즘

오더는 input.txt에 따라 만들어야 할 커피를 적절히 바리스타1과 바리스타2 에게 분배한다. 분배의 기준은 바리스타가 쉬고 있을 경우이다. 단, 한 세트 내에서만 해당되며 한 세트의 주문이 끝나고 쉬고 있는 경우에는 다음 주문까지는 쉴 수 있다. 예를 들면 위에서 주어진 input.txt의 경우 다음과 같은 진행이 된다.

1번 째 주문의 진행
바리스타1이 ame1을 만든다. 바리스타2이 ame2를 만든다.
바리스타1이 lat1을 만든다. 바리스타2이 mac1을 만든다.
바리스타1이 mac2을 만든다. 바리스타2이 mac3을 만든다.
바리스타1이 mac4을 만든다. 바리스타2는 쉬게 된다.

커피마다 만드는 시간이 모두 다르기 때문에 이와 같은 알고리즘이 가능하며 아메리카노는 3초, 카페라떼는 4초, 마끼야또는 5초로 정했다. 각 커피에 대해 만드는 시간은 FuncVariable.h에서 #define으로 정의 되어진 부분에서 수정이 가능하다. 바리스타1과 바리스타2가 커피를 만들면 Complete구조체에 쌓이게 되고 OrderQue의 주문 내용과 일치가 되면 손님에게 내어주게 된다.

프로그램은 main.cpp, FuncVariable.h, FuncVariable.cpp로 구성된다.

2.2 FuncVariable.h

```
1   #pragma once
2
3   #ifndef __FUNCVARIABLE_H__
4   #define __FUNCVARIABLE_H__
5
6   #include<iostream>
7   #include<fstream>
8   #include<queue>
9   #include<cstdlib>
10
11  using namespace std;
12
13  #define ame 3
14  #define lat 4
15  #define mac 5
16  #define MAX 30
17
18  typedef struct OrderQue {
19      int Num_americano;
20      int Num_caffelatte;
21      int Num_macchiato;// 음료의 갯수 0:아메 , 1:라떼 , 2:마꺄또
22  }OrderQue;
23
24  typedef struct Completed {
25      int finish = 0;
26      int fin_americano = 0;
27      int fin_caffelatte = 0;
28      int fin_macchiato = 0;
29  }Completed;
```

#define으로 정의된 항목인 ame, lat, mac는 아메리카노, 카페라떼, 마끼야또를 의미하며 이에 할당된 정수는 각각의 커피를 만드는데 걸리는 시간을 의미한다.

OrderQue는 input.txt에서 불러오는 값을 저장하며 주문받는 커피의 개수를 각각 저장한다.

Completed는 바리스타1과 바리스타2가 만든 커피를 담아두는 구조체이다. 담아둔 커피가 OrderQue의 커피 개수와 일치하게 되면 출력을 한다.

```
30  typedef struct Barista {
31      int making_time = 0;
32      char Beverage_name[30];
33  }Barista;
34
35  extern queue<OrderQue> LeftBeverages;
36  extern Completed List[MAX];
37  extern Barista Barista1;
38  extern Barista Barista2;
39
40  // 각 음료가 주문 받은 것 만큼 채워지면 분배완료 ..
41  extern int americano, Atemp1, Btemp1;
42  extern int caffelatte, Atemp2, Btemp2;
43  extern int macchiato, Atemp3, Btemp3;
44  extern int index; // List 배열 접근을 위한 index
45  extern OrderQue temp;
46
47  void working();
48  void ReceiveOrder(char* Order, int n);
49  void tempInit();
50  void InsertBarista();
51  void display(Barista Barista1, Barista Barista2);
52
53  #endif
```

Barista 구조체는 OrderQue의 내용을 하나씩 분배 받는다. 이 때 만들어야 하는 커피의 종류와 그 커피를 만드는데 걸리는 시간을 할당 받는다.

이 페이지에서 함수에 대한 설명은 간략히 하겠다.
working() 이 함수는 바리스타1,2의 일을 구현한 함수다.
ReceiverOrder() input.txt의 내용을 읽어들여 OrderQue에 저장한다.
tempinit() 큐에 대한 초기화를 진행한다.
InsertBarista() 바리스타1과 2에게 분배를 해주는 함수이다.
display() 커피 제조과정을 출력해주는 함수이다.

2.3 FuncVariable.cpp

```
1   #include"FuncVariable.h"
2
3   // 주문 받는 내역을 큐에 넣어주는 함수 //
4   void ReceiveOrder(char* Order, int n) {
5       if (strcmp(Order, "ame") == 0)
6           temp.Num_americano = n;
7       if (strcmp(Order, "lat") == 0)
8           temp.Num_caffelatte = n;
9       if (strcmp(Order, "mac") == 0)
10          temp.Num_macchiato = n;
11  }

    void tempInit() {
        temp.Num_americano = 0;
        temp.Num_caffelatte = 0;
        temp.Num_macchiato = 0;
    }
```

ReceiveOrder함수는 input.txt의 내용을 OrderQue에 저장하는 역할을 한다. 이 때 OrderQue는 구조체이기 때문에 temp라는 변수를 활용하여 OrderQue에 push를 하게 된다. push를 하게 되면 버퍼를 지우기위해 다시 tempinit을 불러 큐를 초기화해주고 새로운 값을 input.txt에서 불러와 OrderQue에 저장한다. 이를 파일의 끝이 생길 때 까지 반복한다.

```
47  void working() {
48
49      if (Barista1.making_time != 0)
50          Barista1.making_time--;
51      if (Barista2.making_time != 0)
52          Barista2.making_time--;
53
54      if (Barista1.making_time == 0) {
55          if (strcmp(Barista1.Beverage_name, "americano") == 0) {
56              if (Atemp1 == List[index].fin_americano - Btemp1) {}
57              else Atemp1++;
58          }
59          else if (strcmp(Barista1.Beverage_name, "caffelatte") == 0) {
60              if (Atemp2 == List[index].fin_caffelatte - Btemp2) {}
61              else Atemp2++;
62          }
63          else if (strcmp(Barista1.Beverage_name, "macchiato") == 0) {
64              if (Atemp3 == List[index].fin_macchiato - Btemp3) {}
65              else Atemp3++;
66          }
67      }
```

```
68      if (Barista2.making_time == 0) {
69          if (strcmp(Barista2.Beverage_name, "americano") == 0) {
70              if (Btemp1 == List[index].fin_americano - Atemp1) {}
71              else Btemp1++;
72          }
73          else if (strcmp(Barista2.Beverage_name, "caffelatte") == 0) {
74              if (Btemp2 == List[index].fin_caffelatte - Atemp2) {}
75              else Btemp2++;
76          }
77          else if (strcmp(Barista2.Beverage_name, "macchiato") == 0) {
78              if (Btemp3 == List[index].fin_macchiato - Atemp3) {}
79              else Btemp3++;
80          }
81      }
82      _sleep(500);
83  }
```

working() 함수는 호출 될 때 마다 Barista의 making_time을 1을 감소시킨다. _sleep으로 초단위를 조정 할 수 있으며 making_time의 기준이 1초이기에 _sleep(1000)으로 설정하는 것이 좋다. 위의 사진에서는 디버깅을 위해 500으로 설정해주었다. 바리스타1 혹은 2의 making_time이 0이 된다면 새로운 주문을 OrderQue에서 받아 올 것이다. 이 때 한 세트 내에서의 분배만을 허용한다. 이에 대한 예시는 2.1 알고리즘에 자세히 기술되어 있다.

바리스타1 혹은 바리스타2가 만든 커피는 구조체 complete로 구성된 List에 저장이 되며 List내부에서 각 커피 개수가 OrderQue에서 주문을 받은 각 커피 개수와 일치하게 된다면 해당 주문은 완전히 완료가 된 상태이며 display 함수를 통해 완료가 되었음을 알리는 출력을 하게 된다.

위의 소스에서 Atemp - 1 , 2 , 3과 Btemp - 1 , 2 , 3은 바리스타1과 바리스타2가 만든 커피가 중복이 되면 안되므로 연동을 위해 만들어준 변수이다. 아메리카노의 경우를 보자면 다음과 같다.

바리스타1이 만든 아메리카노의 개수 -> Atemp1
바리스타2가 만든 아메리카노의 개수 -> Btemp1

두 변수를 이용하여 만들어야 할 커피의 개수를 구할 수 있다.

2.4 main.cpp

아래의 사진은 각 변수에 대한 선언을 의미한다.

```
1   #include "FuncVariable.h"
2
3   queue<OrderQue> LeftBeverages;
4   Completed List[MAX];
5   Barista Barista1;
6   Barista Barista2;
7
8   // 각 음료가 주문 받은 것 만큼 채워지면 분배완료 ..
9   int americano, Atemp1, Btemp1;
10  int caffelatte, Atemp2, Btemp2;
11  int macchiato, Atemp3, Btemp3;
12  int index; // List 배열 접근을 위한 index
13  OrderQue temp;
```

```
15  int main(void)
16  {
17      index = 0;
18      int n; // 음료 갯수
19      int OrderN; // 주문받은 음료 종류
20      char* Order = new char(sizeof(char) * 30); // 주문하는 사람
21      ifstream in;
22      in.open("input.txt");
23
24      while (!in.eof()) {
25          tempInit();
26          in >> OrderN;
27
28          // 주문을 받는다. //
29          for (int i = 0; i < OrderN; i++)
30          {
31              in >> Order >> n;
32              ReceiveOrder(Order, n);
33              cout << n << endl;
34          }
35          LeftBeverages.push(temp); // 주문 내용이 큐에 들어가 있음
36                                    // 바리스타에게 분배 (바리스타가 일 없으면 다음 주문 받을 음료 배분)
37                                    // 시간초는 슬립을 이용해서...
38                                    // A  , B------ , 완료 될 때 마다 음료별 카운트 증가
39                                    // 세트는 우리가 다 받아 놨으니 모이면 성공.
40                                    // 다 끝나면 다음 손님 주문 받기.
41          cout << LeftBeverages.front().Num_americano << endl;
42      }
```

파일의 끝에 도달 할 때 까지 input.txt에서 읽어와 OrderQue인 LeftBeverages에 저장한다.

```
43
44      while (!LeftBeverages.empty()) {
45          Atemp1 = Atemp2 = Atemp3 = 0; // 바리스타1로 부터 채워진 음료...
46          Btemp1 = Btemp2 = Btemp3 = 0; // 바리스타2로 부터 채워진 음료...
47          List[index].fin_americano = americano = LeftBeverages.front().Num_americano;
48          List[index].fin_caffelatte = caffelatte = LeftBeverages.front().Num_caffelatte;
49          List[index].fin_macchiato = macchiato = LeftBeverages.front().Num_macchiato;
50
51          LeftBeverages.pop();
52          int complete = 0;
53          while (true) {
54              if (Barista1.making_time == 0) { // 바리스타1이 쉬고 있으면 주문 부여
55                  if (americano != 0) {
56                      Barista1.making_time = ame; // 아메리카노 만드는 시간 ㄱㄱ
57                      strcpy(Barista1.Beverage_name, "americano");
58                      americano--;
59                  }
60                  else if (caffelatte != 0) {
61                      Barista1.making_time = lat;
62                      strcpy(Barista1.Beverage_name, "caffelatte");
63                      caffelatte--;
64                  }
65                  else if (macchiato != 0) {
66                      Barista1.making_time = mac;
67                      strcpy(Barista1.Beverage_name, "macchiato");
68                      macchiato--;
69                  }
```

OrderQue인 LeftBeverages가 빌 때 까지 working을 반복하게 되고 List와 비교해서 주문받은 커피의 개수가 일치가 되면 display함수 내부에서 일련의 동작을 통해 하나의 주문 완료를 알리게 된다. 주문 하나가 끝나게 되면 LeftBeverages 큐에서 다음 주문에 대한 목록을 pop하게 되고 바리스타1 혹은 바리스타2가 만들어야 할 커피를 분배받게 된다.

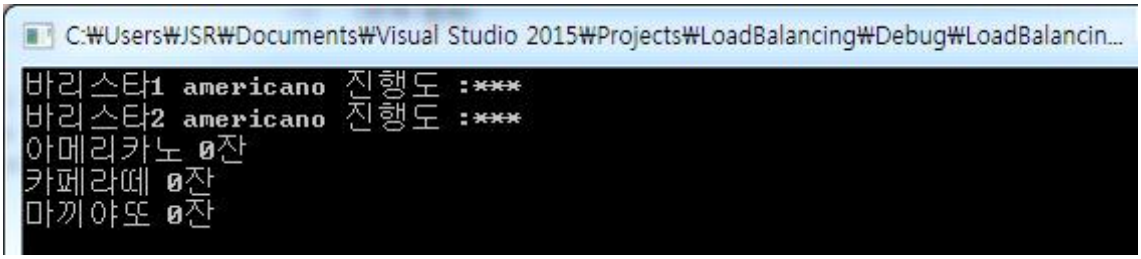
```
70              }
71
72              if (Barista2.making_time == 0) { // 바리스타2이 쉬고 있으면 주문 부여
73                  if (americano != 0) {
74                      Barista2.making_time = ame; // 아메리카노 만드는 시간 ㄱㄱ
75                      strcpy(Barista2.Beverage_name, "americano");
76                      americano--;
77                  }
78                  else if (caffelatte != 0) { // 라떼 만드는 시간
79                      Barista2.making_time = lat;
80                      strcpy(Barista2.Beverage_name, "caffelatte");
81                      caffelatte--;
82                  }
83                  else if (macchiato != 0) { // 마꺄또 만드는 시간
84                      Barista2.making_time = mac;
85                      strcpy(Barista2.Beverage_name, "macchiato");
86                      macchiato--;
87                  }
88
89              }
90
91              // 일하는 부분 //
92              display(Barista1, Barista2);
93              if (Barista1.making_time == 0 && Barista2.making_time == 0) { // 마지막 * 제거용...
94                  display(Barista1, Barista2);
95              }
```

```
91               // 일하는 부분 //
92               display(Barista1, Barista2);
93               if (Barista1.making_time == 0 && Barista2.making_time == 0) { // 마지막 * 제거용...
94                   display(Barista1, Barista2);
95               }
96
97               // 세트 맞춰졌으니 진동벨 울림//
98               if (americano == 0 && caffelatte == 0
99                   && macchiato == 0 && Barista1.making_time == 0
100                  && Barista2.making_time == 0) {
101                  List[index++].finish = 1;
102                  break;
103              }
104          }
105          //display(Barista1 , Barista2);
106          _sleep(200);
107      }
108
109      cout << "주문하신 " << index << "번 째 메뉴 " <<
110          "아메리카노 " << List[index - 1].fin_americano <<
111          "잔 카페라떼 " << List[index - 1].fin_caffelatte <<
112          "잔 마끼야또 " << List[index - 1].fin_macchiato <<
113          "잔 나왔습니다.^0^" << endl;
114  }
```

3 결과 화면

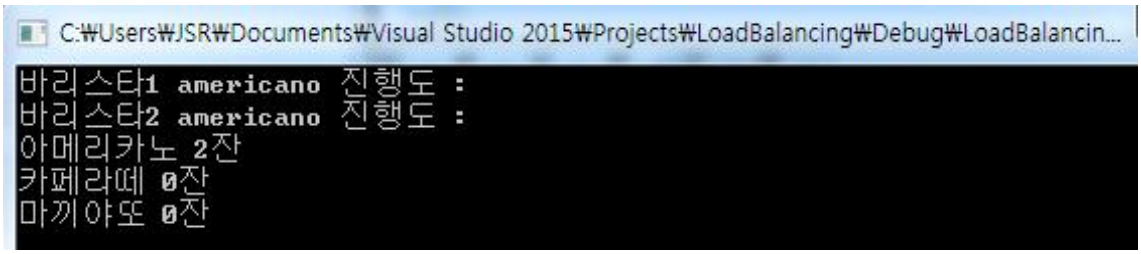
다음의 결과는 4p에 기술된 input.txt에 적힌 값을 토대로 한 값이다.

초기화면

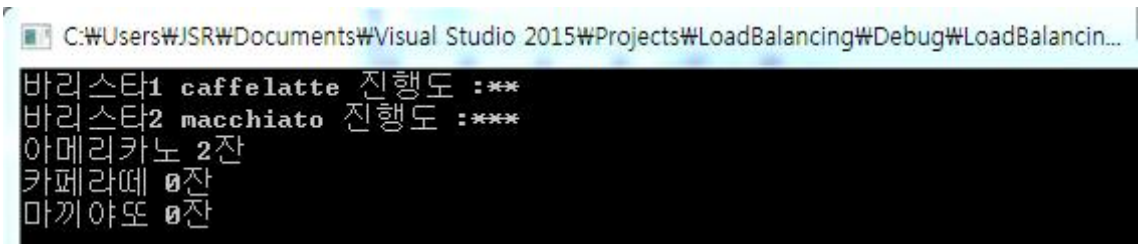


아메리카노 2잔을 만든 후의 화면

아메리카노가 0잔에서 2잔으로 변한 모습을 볼 수 있다.

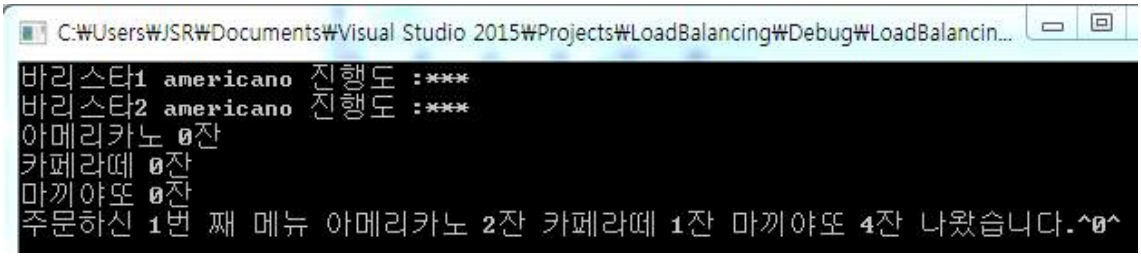


카페라떼 1잔과 마끼야또 1잔을 만드는 중



첫 번째 주문이 완료되고 난 후의 화면

결과창을 보게 되면 쌓여있던 커피들은 모두 0잔으로 초기화되고 첫 주문인 아메리카노 2잔, 카페라떼 1잔, 마끼야또 4잔을 정확히 출력하는 모습을 볼 수 있다.



받은 주문을 모두 완료한 후의 화면

화면을 보면 input.txt에서의 값과 일치하는 커피의 종류와 개수를 출력하는 모습을 볼 수 있다.