

Block.hpp 에 FileHashBuffer 추가

```
78     }
79
80     class FileHashBuffer
81     {
82     public:
83         vector<string> FileHashVector;
84         int HashCount;
85
86         FileHashBuffer()
87         {
88             this->HashCount = 0;
89         }
90
91         FileHashBuffer(string FileHash)
92         {
93             this->HashCount = 0;
94             InsertFileHash(FileHash);
95         }
96
97         bool IsFull()
98         {
99             if (HashCount >= 5) return true; else return false;
100         }
101
102         void InsertFileHash(string FileHash)
103         {
104             if (IsFull())
105             {
106                 cout << "File hash Count value exceeded 5" << endl;
107             }
108
109             HashCount++;
110             FileHashVector.push_back(FileHash);
111         }
112
113         string GetFileHash(int index)
114         {
115             assert(index < HashCount);
116             return FileHashVector.at(index);
117         }
118
119         void FileHashBufferDump()
120         {
121             cout << "FileHashBufferDump : " << endl;
```

BlockChain.hpp에 HashBufferList추가

```

98 class FileHashBufferList
99 {
100 public:
101     list<FileHashBuffer> List;
102
103     FileHashBufferList()
104     {
105         this->List.push_back(FileHashBuffer());
106     }
107
108     // 파일 해쉬를 넣었는데 5개가 다 찼으면 true리턴
109     bool InsertFileHash(string FileHash)
110     {
111         if (this->List.back().IsFull())
112         {
113             List.push_back(FileHashBuffer(FileHash));
114             return false;
115         }
116         else
117         {
118             List.back().InsertFileHash(FileHash);
119             if (List.back().IsFull())
120             {
121                 return true;
122             }
123             else false;
124         }
125     }
126
127     string GetFileHash(int BufferIndex, int HashIndex)
128     {
129         assert(BufferIndex < List.size());
130
131         auto it = List.begin();
132         for (int i = 0; i < BufferIndex; i++) it++;
133         return it->GetFileHash(HashIndex);
134     }
135
136     void ListDump()
137     {
138         int index = 0;
139         for (auto it = this->List.begin(); it != this->List.end(); it++)
140         {
141             cout << "[" + to_string(index) + "]" + ' ';
142             it->FileHashBufferDump();
143         }

```

사용법 (main함수)

```
27
28
29 FileHashBufferList HashBufferList;
30
31 int main()
32 {
33     for (int i = 1; i < 7; i++)
34     {
35         // 파일 해쉬값을 넣고 넣었는데 5개가 다 차면 true를 리턴
36         if (true == HashBufferList.InsertFileHash(to_string(i)))
37         {
38             cout << "Full Five Hash" << i << endl;
39         }
40     }
41
42     // 0번 버퍼의 i번째 해쉬값을 가져옴
43     for (int i = 0; i < 5; i++)
44         cout << HashBufferList.GetFileHash(0, i) << endl;
45
46     for (int i = 0; i < 1; i++)
47         cout << HashBufferList.GetFileHash(1, i) << endl;
48
49     //유효하지 않은 위치의 해쉬값을 가져오면 에러코드 발생
50     //cout << HashBufferList.GetFileHash(1, 1) << endl;
51
52     // 유효하지 않은 위치의 해쉬값을 가져오면 에러코드 발생
53     //cout << HashBufferList.GetFileHash(2, 1) << endl;
54
55     HashBufferList.ListDump();
56
57     return 0;
58 }
59
60
```

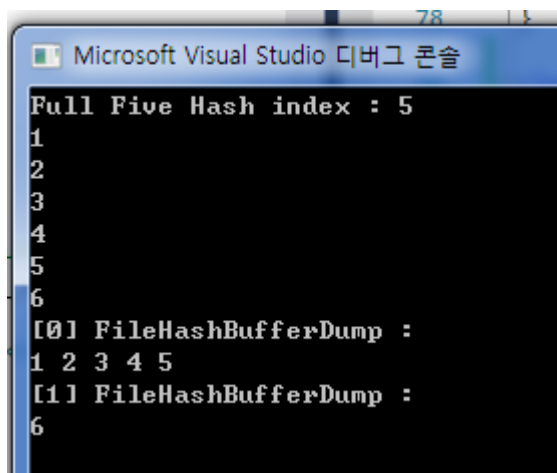
HashBufferList는 채굴 및 블록체인 저장에 앞서 외부로부터 들어오는 클라이언트 들의 파일 해쉬 값을 저장하기 위해 사용되는 자료구조이다. 36번 줄의 InsertFileHash 매서드는 파일 해쉬값을 버퍼에 넣는다. 이 때 만약 5개가 다 차면 true를 리턴해 알려준다. 이 기능을 이용해 채굴 타이밍을 알 수 있다.

44번줄의 GetFileHash는 버퍼 인덱스, 해쉬 인덱스 두 가지 정수값을 가지고 이에 해당하는 파일 해쉬값을 리턴해준다.

50번줄과 53번줄은 실행해보면 에러가 나는데 이에 대한 기본적인 대처를 보여준다. (어느 코드의 어느 라인에서 버그가 났는지 콘솔로 알려준다)

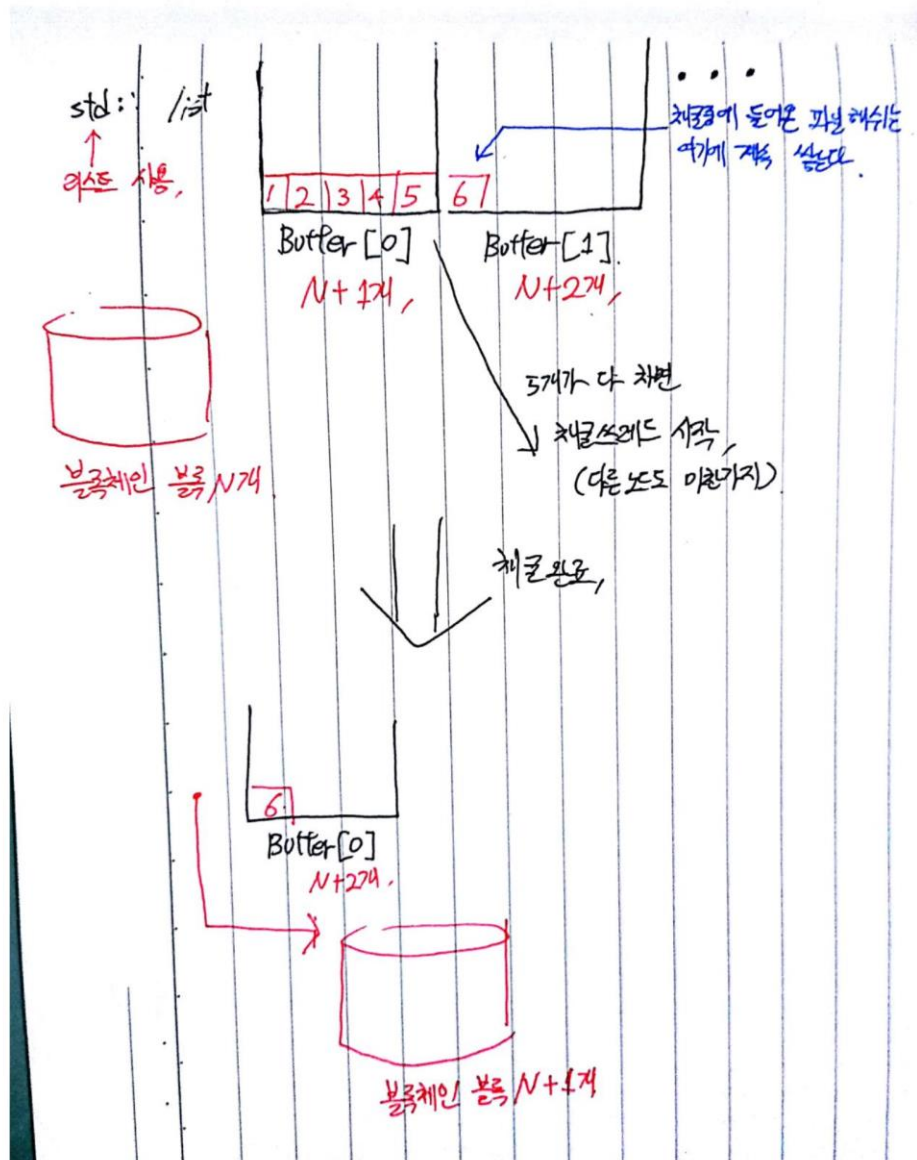
55번줄의 ListDump는 파일해쉬버퍼리스트의 모든 내용을 그대로 출력한다.

실행사진



```
Microsoft Visual Studio 디버그 콘솔
Full Five Hash index : 5
1
2
3
4
5
6
[0] FileHashBufferDump :
1 2 3 4 5
[1] FileHashBufferDump :
6
```

참고로 이 문서의 내용은 다음 그림을 구현한 것이다.



위 그림은 채굴 스레드에서 난스 계산에 성공했을 경우의 처리이다. (이때 HashBuffer 리스트는 블록체인의 블록의 연장선으로 취급되기 때문에 얼마든지 접근가능(사용가능)하다)

Buffer[0]의 파일 해쉬 5개가 모두 채워졌기 때문에 채굴 스레드가 돌아가기 시작한다. 다른 네트워크에 위치한 다른 노드들도 마찬가지이다. 여기서 만약 난스 계산에 성공했을 경우 다음과

