운영체제 Lab03

2015140124
전자공학과
진우빈

1. 프로그램의 구조

```
#pragma once

struct Process {
	int Id; // Process ID
	int Size; // Process Size
	int Pos; // Process Store Position
	Process* next;
};

class Memory {
public:
	Memory(int size);
	Memory(const Memory& mem);
	~Memory();

	void Alloc(int id, int size, bool flag);
	void Coalescing(bool flag);
	void Compaction(bool flag);
	void ProcessMove(int pos[2]);
	int GetHoleSize();
	void Print();
	int GetNum();

	int maxNum;
	int maxSize;
	Process* init;
};
```

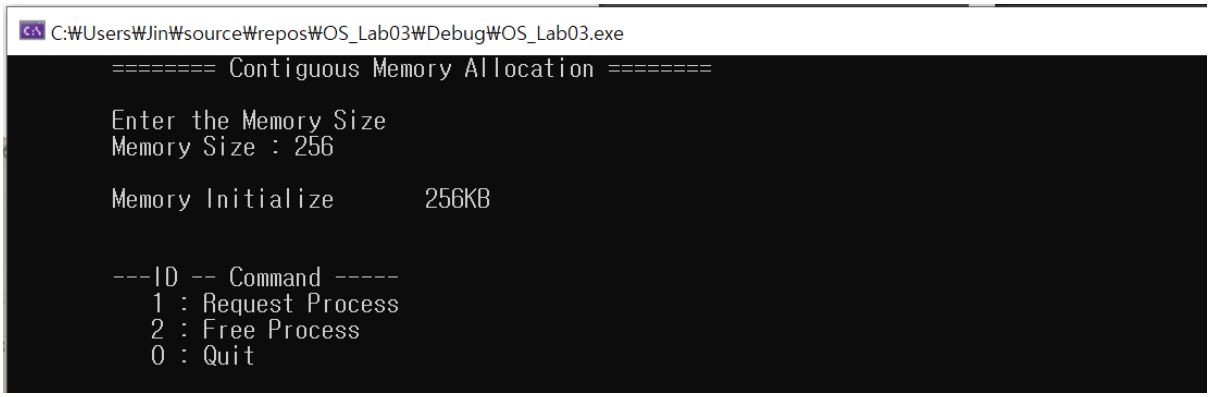
Process Structrure를 만들어 Process 넘버, Process의 크기, Process의 저장 위치를 입력으로 갖습니다. Process* next는 구조체의 다음 노드를 가리킵니다.
Print() 함수는 메모리의 정보를 출력합니다. Process를 Request 하거나 Free 할 때마다 메모리의 구조를 대략적으로 출력하여 어떻게 사용되고 있는지를 확인할 수 있게 하였습니다.
이후에 Main 문에서는 처음에 메모리의 크기를 전체 설정하고 While문을 돌면서 사용자의 입력을 받습니다. 1번은 프로세스 Request, 2번은 프로세스 Free, 0번은 프로그램 종료를 의미합니다.

2. 사용 함수

```
void Alloc(int id, int size, bool flag); // 프로세스 할당 함수
void Coalescing(bool flag); // Alloc 할 때 Coalescing이 필요하다면 수행
void Compaction(bool flag); // Alloc할 때 Compaction이 필요하다면 수행
void ProcessMove(int pos[2]); // 프로세스를 옮기는 함수
int GetHoleSize(); // 전체 Hole의 사이즈를 계산하고 return
void Print(); // 메모리가 어떻게 사용되고 있는 지, Block, Average Size 출력
int GetNum(); // 프로세스의 개수 return
```

3. 실행 화면

Pdf 2페이지에 있는 사진을 예시로 하여 진행 하였습니다.



<Memory Size 입력 받고 이후의 Application 실행>

<1번 째 입력: Number 1 & Size 64 Request>

```
Process Number : 2
Process Size : 64


REQUEST 2: 64KB
Best Fit : Allocated at  64KB

Memory  256KB

        0KB ~ 64KB              Process1(64KB)
        64KB ~ 128KB            Process2(64KB)
        128KB ~ 256KB           Hole(128KB)


128KB Free      1 Block(s)      Average Size = 128KB
```

<2번 째 입력: Number 2 & Size 64 Request>

```
Process Number : 3
Process Size : 32


REQUEST 3: 32KB
Best Fit : Allocated at  128KB

Memory  256KB

        0KB ~ 64KB              Process1(64KB)
        64KB ~ 128KB            Process2(64KB)
        128KB ~ 160KB           Process3(32KB)
        160KB ~ 256KB           Hole(96KB)


96KB Free       1 Block(s)      Average Size = 96KB
```

<3번 째 입력: Number 3 & Size 32 Request>

```
Process Number : 4
Process Size : 16


REQUEST 4: 16KB
Best Fit : Allocated at  160KB

Memory  256KB

        0KB ~ 64KB              Process1(64KB)
        64KB ~ 128KB            Process2(64KB)
        128KB ~ 160KB           Process3(32KB)
        160KB ~ 176KB           Process4(16KB)
        176KB ~ 256KB           Hole(80KB)


80KB Free       1 Block(s)      Average Size = 80KB
```

<4번 째 입력: Number 4 & Size 16 Request>

```
Enter the Process Number

Process Number : 1


Free Process 1: 64KB
Memory  256KB

        0KB ~ 64KB              Hole(64KB)
        64KB ~ 128KB            Process2(64KB)
        128KB ~ 160KB           Process3(32KB)
        160KB ~ 176KB           Process4(16KB)
        176KB ~ 256KB           Hole(80KB)


144KB Free      2 Block(s)      Average Size = 72KB
```

<5번 째 입력: Process 1 Free>

```
Process Number : 3


Free Process 3: 32KB
Memory  256KB

        0KB ~ 64KB              Hole(64KB)
        64KB ~ 128KB            Process2(64KB)
        128KB ~ 160KB           Hole(32KB)
        160KB ~ 176KB           Process4(16KB)
        176KB ~ 256KB           Hole(80KB)


176KB Free      3 Block(s)      Average Size = 58.6667KB
```

<6번 째 입력: Process 3 Free>

```
Process Number : 5
Process Size : 32


REQUEST 5: 32KB
Best Fit : Allocated at  128KB

Memory  256KB

        0KB ~ 64KB              Hole(64KB)
        64KB ~ 128KB            Process2(64KB)
        128KB ~ 160KB           Process5(32KB)
        160KB ~ 176KB           Process4(16KB)
        176KB ~ 256KB           Hole(80KB)


144KB Free      2 Block(s)      Average Size = 72KB
```

<7번 째 입력: Number 5 & Size 32 Request>

```
Process Number : 2


Free Process 2: 64KB
Coalescing : 0KB & 64KB
Memory  256KB

        0KB ~ 128KB             Hole(128KB)
        128KB ~ 160KB           Process5(32KB)
        160KB ~ 176KB           Process4(16KB)
        176KB ~ 256KB           Hole(80KB)


208KB Free      2 Block(s)      Average Size = 104KB
```

<6번 째 입력: Process 2 Free 이 때 Coalescing이 자동으로 수행 되어짐 이후 함수 종료>

아래는 Pdf 예시가 아닌 임의의 환경에서 Compaction을 수행한 환경입니다.

```
REQUEST 6: 80KB
Compaction ...
        Process2 64KB Moves to 0KB
        Process4 16KB Moves to 240KB

Total Move! :   80KB

Memory  256KB

        0KB ~ 64KB              Process2(64KB)
        64KB ~ 176KB            Hole(112KB)
        176KB ~ 240KB           Process5(64KB)
        240KB ~ 256KB           Process4(16KB)


112KB Free      1 Block(s)      Average Size = 112KB
Best Fit : Allocated at  64KB

Memory  256KB

        0KB ~ 64KB              Process2(64KB)
        64KB ~ 144KB            Process6(80KB)
        144KB ~ 176KB           Hole(32KB)
        176KB ~ 240KB           Process5(64KB)
        240KB ~ 256KB           Process4(16KB)
```

1. **외부단편화로 인해 프로세스를 메모리 위에 올릴 수 없다고 판단될 시 Compaction**
2. **총 80KB의 메모리를 옮겨주어 Compaction 수행**
3. **위의 메모리 Print는 Compaction 직후 메모리 환경**
4. **아래 메모리 Print는 Compaction 이후 Process 할당 후 메모리 환경**
5.

Exe 실행 파일은 첨부가 되지 않아 소스코드를 첨부합니다. (Visual Studio2019에서 실행함)