OSHW#3 메모리 관리

2016104142 이광원

1. 자료구조, 설정 과정

```python
#메인 프로그램
M_size = int(input("Memory Size(K): ")) #메모리의 크기 입력
Memory = [0]*M_size #메모리를 리스트로 구현
Requests, Locations, Sizes = [], [], [] #Request의 id, 메모리 위치, 크기 저장 리스트

while 1:
    R_id = int(input("Request ID(0 to exit): "))    #조작할 Request의 ID 입력
    if R_id == 0: break                             #프로그램 종료
    R_size = int(input("Request Memory Size(K): ")) #Request가 요청할 메모리 크기 입력
    #새로운 Request 발생 시
    if R_id not in Requests:
        Best_Fit(R_id, R_size)  #Best Fit Case로 메모리 할당
    else:
        #Request가 가진 메모리 초기화
        if R_size == 0: Free(R_id)
        #중복된 Request 요청 발생 시
        else:
            print("\nThere is already a Request allocated with the same id\n")
```

```
Memory Size(K): 256
Request ID(0 to exit): 1
Request Memory Size(K): 64

REQUEST 1: 64K
```

- ID, Size를 한 쌍씩 입력

```
Request ID(0 to exit): 1
Request Memory Size(K): 64

There is already a Request allocated with the same id

Request ID(0 to exit):
```

- 중복 ID 생성 불가(삭제 시 재생성 가능)

2. Request 메모리 관리 - 할당

```python
#메모리 할당 위치, 잔여 메모리, Block(Hole) 정보 출력
def print_Alloc(r_id, loc, size):
    _, holes = Check_Hole() #빈 메모리 크기 확인
    free = 0    #빈 메모리 크기 초기화

    for i in range(len(holes)): free += holes[i]    #전체 빈 메모리 크기 계산

    print("\nREQUEST %d: %dK" % (r_id, size))
    print("-Request %d allocated at address %dK" % (r_id, loc))
    print("-%dK free, %d block(s), average size = %fK\n" % (free, len(holes), free/ len(holes)))
```

```
Request ID(0 to exit): 1
Request Memory Size(K): 64

REQUEST 1: 64K
-Request 1 allocated at address 0K
-192K free, 1 block(s), average size = 192.000000K

Request ID(0 to exit): 2
Request Memory Size(K): 64

REQUEST 2: 64K
-Request 2 allocated at address 64K
-128K free, 1 block(s), average size = 128.000000K

Request ID(0 to exit):
```

- Free Size, Block 수, 평균 Size, 개별 출력

```python
#메모리를 압축하여 빈 메모리를 Block(Hole)으로 변환
def Compaction(free):
    for _ in range(free):   #빈 메모리를 뒤로 이동
        Memory.remove(0)
        Memory.append(0)
    print("\n-Compaction complete, memory relocated")   #압축 완료

    move = 0    #이동 변수
    for i in range(len(Locations)): #압축 결과에 따라 메모리 위치 정보 수정
        Locations[i] = move #위치 정보 수정
        move += Sizes[i]
```

```
Request ID(0 to exit): 2
Request Memory Size(K): 64

REQUEST 2: 64K
-Request 2 allocated at address 128K
-64K free, 1 block(s), average size = 64.000000K

Request ID(0 to exit): 3
Request Memory Size(K): 72

Unable to allocate due to lack of Memory

Request ID(0 to exit):
```

- 메모리를 초과하는 요청 처리

2. Request 메모리 관리 - 초기화

```python
#할당 중인 메모리를 초기화
def Free(r_id):
    #종료 시킬 Request 내용 검색, 삭제
    for i in range(len(Requests)):
        if Requests[i] == r_id:
            loc = Locations.pop(i)
            size = Sizes.pop(i)

    Requests.remove(r_id)   #Request 삭제

    start, sizes = Check_Hole() #빈 메모리 정보 확인
    s_coal = -1 #할당 되었던 메모리 앞 Hole의 시작 주소 초기화
```

```python
print("\nFREE REQUEST", r_id, "(%dK)" % size)
print("-Request %d freed at address %dK" % (r_id, loc))
print("-%dK free, %d block(s), average size = %fK\n" % (free, len(holes), free/len(holes)))
```

```
Request ID(0 to exit): 1
Request Memory Size(K): 0

FREE REQUEST 1 (64K)
-Request 1 freed at address 0K
-144K free, 2 block(s), average size = 72.000000K

Request ID(0 to exit): 3
Request Memory Size(K): 0

FREE REQUEST 3 (32K)
-Request 3 freed at address 128K
-176K free, 3 block(s), average size = 58.666667K

Request ID(0 to exit):
```

- 가지고 있는 메모리 초기화

3. 메모리 압축(Compaction)

```python
#메모리를 압축하여 빈 메모리를 Block(Hole)으로 변환
def Compaction(free):
    for _ in range(free):   #빈 메모리를 뒤로 이동
        Memory.remove(0)
        Memory.append(0)
    print("\n-Compaction complete, memory relocated")   #압축 완료

    move = 0    #이동 변수
    for i in range(len(Locations)): #압축 결과에 따라 메모리 위치 정보 수정
        Locations[i] = move #위치 정보 수정
        move += Sizes[i]
```

```
Request ID(0 to exit): 4
Request Memory Size(K): 100

-Compaction complete, memory relocated

REQUEST 4: 100K
-Request 4 allocated at address 128K
-28K free, 1 block(s), average size = 28.000000K

Request ID(0 to exit):
```

- 모든 Hole을 하나의 Block으로 압축

4. 메모리 병합(Coalescing)

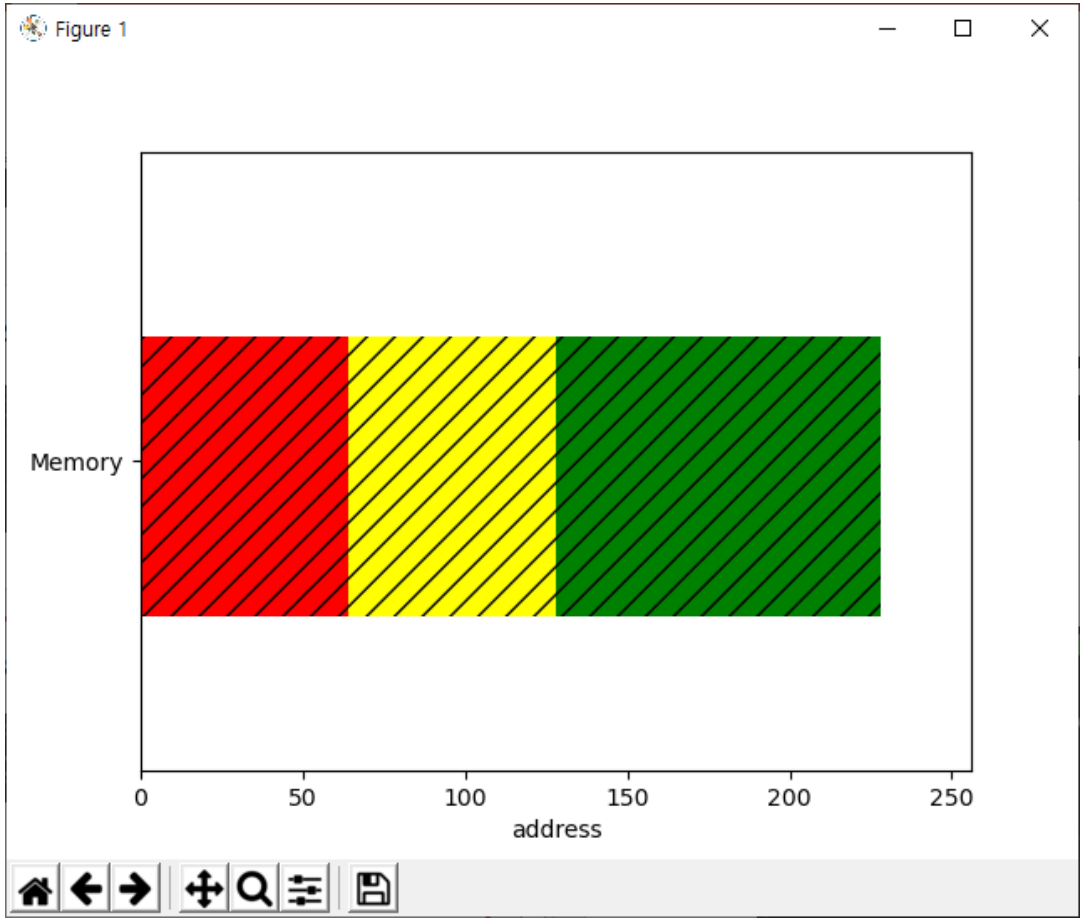
```
#할당 되었던 메모리 바로 뒤의 Hole과 병합
if loc+size != len(Memory)-1 and Memory[loc+size] == 0:
    print("-Coalescing blocks at addresses %dK and %dK" % (loc, loc+size))

#할당 되었던 메모리 바로 앞의 Hole과 병합
if s_coal != -1:
    print("-Coalescing blocks at addresses %dK and %dK" % (s_coal, loc))

print("-%dK free, %d block(s), average size = %fK\n" % (free, len(holes), free/len(holes)))
```

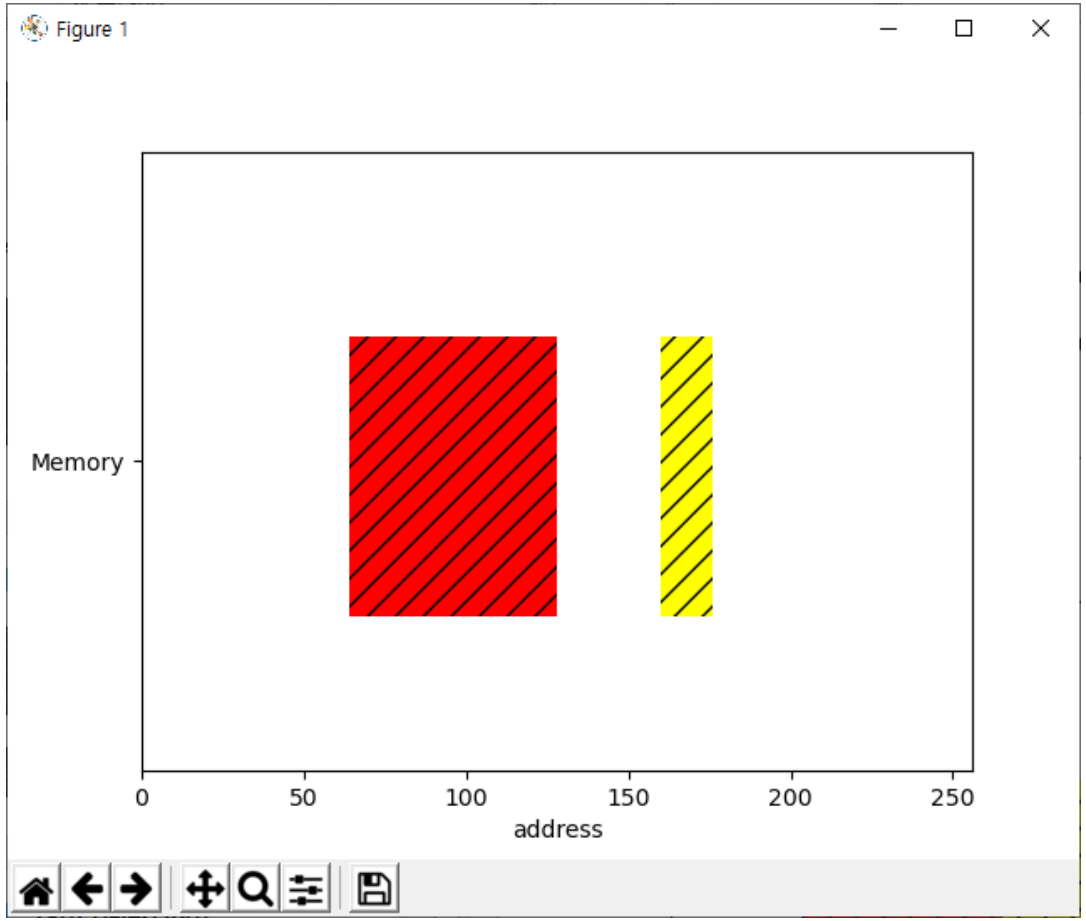
```
Request ID(0 to exit): 2
Request Memory Size(K): 0

FREE REQUEST 2 (64K)
-Request 2 freed at address 64K
-Coalescing blocks at addresses 0K and 64K
-208K free, 2 block(s), average size = 104.000000K
```

```
Request ID(0 to exit): 5
Request Memory Size(K): 0

FREE REQUEST 5 (32K)
-Request 5 freed at address 128K
-Coalescing blocks at addresses 128K and 160K
-Coalescing blocks at addresses 0K and 128K
-256K free, 1 block(s), average size = 256.000000K

Request ID(0 to exit):
```

- 새 빈 메모리를 앞,뒤 Hole과 병합

5. 프로그램 종료





- 0 입력 시 이미지 출력 및 프로그램 종료