Operating System

경영학부

2012203079

정상원

Contiguous Memory Allocation

먼저 연속 메모리 할당을 간략하게 설명하겠습니다. 연속 메모리 할당 방식은 각 프로세스가 필요로하는 메모리를 Batch로 불러와서 메모리에 연속적으로 할당하는 방법입니다. 연속적으로 할당한다는 것은 하나의 프로세스의 메모리가 할당될 시에 물리적인 주소가 끊임이 없이 연결되어 있다는 것을 의미합니다. 실제로는 더이상 사용되지 않는 방법이라고는 하지만, 이러한 메모리 할당 방식은 세가지로 나눌 수 있습니다. 메모리 전체를 훑다가 처음으로 프로세스의 필요에 맞는 메모리 블록이 나오면 메모리를 할당하는 First fit, 모든 비어있는 메모리를 확인 한 다음 가장 비슷하거나 딱 맞는 메모리 블록을 찾아 할당하는 Best fit, 그리고 마지막으로 가장 큰 메모리 블록부터 프로세스를 할당하는 Worst fit 방식이 있는데, 저의 코드에서 메모리 객체는 비어있는 메모리의 사이즈를 확인해서 저장하는 배열을 따로 두지 않았기 때문에 가장 직관적인 First fit 방식을 채택하였습니다. 저는 먼저 이러한 메모리 할당 방식을 시뮬레이션 하기 위해 Process 클래스를 만들었습니다. Process 클래스 각 객체는 해당 프로세의 Id, 필요로하는 메모리의 크기, 변수는 bottom과 top으로 선언되어 있지만 각각 Base와 limit으로 메모리의 시작점과 끝점을 저장할 수 있는 변수를 두었습니다.

Process 객체의 id는 0부터 시작하여 순차적으로 할당하였고, size는 무작위 숫자를 발생시켜 1부터 30까지의 사이즈를 할당하였습니다.

다음으로 Memory 객체는 사이즈를 100으로 잡았습니다. Memory 내 가장 중요하게 작업 해야했던 append method는 말 그대로 비어있는 메모리에 프로세스를 할당하는 함수입니다. Append method가 불러지면 먼저 boolean으로 선언된 checker 변수가 false로 선언됩니다. 이 변수는 나중에 메모리에 프로세스가 할당될 수 있는 자리가 났을 때 true로 바뀌고 메모리를 iterate하는 loop를 빠져나오기 위한 변수입니다. 다음은 메모리에 가용 공간이 있는지를 확인하기 위한 변수 tmpsize를 append method가 메모리 내 주소를 하나씩 iterate할 때마다 0으로 초기화 시킵니다. Tmpsize는 후에 메모리에 연속된 할당 가능한 자리가 있는지를 확인하기 위해 사용됩니다. 메모리 위에서 주소를 하나씩 증가시키며 사용중이라면 1, 사용중이지 않다면 0으로 쓰여있는 숫자를 확인하여 이 숫자가 0 이라면 tmpsize를 하나씩 증가시킵니다. 이 작업을 수행하는 도중 1을 만나게 된다면 내부 loop을 빠져나와 다시 tmpsize를 0으로 초기화 시키고 시작하는 메모리를 하나 증가시킨 뒤 작업을 진행합니다.

만약 tmpsize를 증가시키다가 1을 만나지 않아서 tmpsize와 process\_size의 값이 같아진다면 그것은 본 프로세스가 연속적으로 할당 될 수 있는 공간이 존재한다는 것을 의미하므로 checker을 true로 바꾸고 iteration loop을 빠져나옵니다. 다음으로 본 프로세스가 할당되는 공간을 모두 1로 선언하고 method가 종료됩니다.

Main 함수에서는 위 작업을 순서에 맞게 진행하게끔 코드를 작성하였습니다. 먼저 메모리를 생성하고, 메모리를 0으로 초기화 한 다음 프로세스를 세개 생성하고 다시 랜덤하게 두개를 제거하는 방식으로 진행됩니다. 프로세스에게 메모리를 할당하려고 하는 도중 더이상 그 프로세스를 할당할 공간이 존재하지 않는다면 오류가 발생하며 종료됩니다.

아래는 코드입니다.

#include <iostream>

#include <math.h>

#include <time.h>

using namespace std;

class Process {

public:

int pro\_id;

int pro\_size;

int pro\_bottom;

int pro\_top;

Process() {};

Process(int size, int id) {

this->pro\_size = size;

this->pro\_id = id;

//cout << "Process Created " << pro\_id << endl;

}

~Process() {

}

void memloc(int bottom, int top) {

this->pro\_bottom = bottom;

this->pro\_top = top;

}

};

class Memory {

public:

int m\_size = 100;

int M[100]; // memory size 100 for now.

bool checker = false;

Memory() {

cout << "Memory Created" << endl;

}

void append(int process\_size, Process& p) {

checker = false;

for (int i = 0; i < 100; i++) {

int tmpsize = 0; // To check memory sector availability

if (checker == false) {

for (int k = i; k < i + process\_size; k++) {

if (M[k] == 1) {

checker = false;

break;

}

else if(M[k] == 0){

tmpsize++;

}

if (tmpsize == process\_size) {

p.pro\_bottom = i;

p.pro\_top = i + p.pro\_size - 1;

checker = true;

continue;

}

else if (100 - i < p.pro\_size) {

cout << "================================CANNOT WRITE " << p.pro\_id << " ===============================" << endl;

return;

}

}

}

if (checker == true && i == p.pro\_top + 1) {

cout << "Writing complete" << endl; break;

}

else if (checker == true && i <= p.pro\_top) {

M[i] = 1;

}

}

}

void pop(int process\_size, int top, int bottom) {

for (int i = bottom; i <= top; i++) {

M[i] = 0;

}

}

void print\_memory\_status() {

for (int i = 0; i < 10; i++) {

for (int k = i \* 10; k < (i \* 10) + 10; k++) {

cout << M[k] << " ";

}

cout << endl;

}

}

};

void main() {

//Memory declaration

Memory M;

Memory\* ptr\_M;

ptr\_M = &M;

//Initialize the memory. Size 100. 0 == empty 1 == true.

for (int i = 0; i < 100; i++) {

ptr\_M->M[i] = 0;

}

//Printing the initial memory status

ptr\_M->print\_memory\_status();

//Process Array and itration number

Process p[1000];

int process\_itr = 0;

//Random number variables declaration

int randsize\_1;

int randsize\_2;

int randsize\_3;

int round = -1; // To count how many rounds undertaken

bool control = false;

do {

round++;

srand(time(NULL));

randsize\_1 = rand() % 30 + 1;

srand(time(NULL) + 1);

randsize\_2 = rand() % 30 + 1;

srand(time(NULL) + 2);

randsize\_3 = rand() % 30 + 1;

//Creating Processes

p[process\_itr] = Process(randsize\_1, process\_itr);

ptr\_M->append(randsize\_1, p[process\_itr]);

process\_itr++;

p[process\_itr] = Process(randsize\_2, process\_itr);

ptr\_M->append(randsize\_2, p[process\_itr]);

process\_itr++;

p[process\_itr] = Process(randsize\_3, process\_itr);

ptr\_M->append(randsize\_3, p[process\_itr]);

process\_itr++;

//Random number for removing random processes

srand(time(NULL) + 5);

int rand\_rmv = rand() % 3 + 1;

srand(time(NULL) + 10);

int rand\_rmv\_2 = rand() % 3 + 1;

cout << "Before removing process " << p[process\_itr - rand\_rmv].pro\_id << " and " << p[process\_itr - rand\_rmv\_2].pro\_id << " " << endl;

ptr\_M->print\_memory\_status();

//Checking Memory Usage

cout << "Process ID : " << p[process\_itr - 3].pro\_id << " Process Size : " << p[process\_itr - 3].pro\_size << " Bottom : " << p[process\_itr - 3].pro\_bottom <<

" Top : " << p[process\_itr - 3].pro\_top << endl;

cout << "Process ID : " << p[process\_itr - 2].pro\_id << " Process Size : " << p[process\_itr - 2].pro\_size << " Bottom : " << p[process\_itr - 2].pro\_bottom <<

" Top : " << p[process\_itr - 2].pro\_top << endl;

cout << "Process ID : " << p[process\_itr - 1].pro\_id << " Process Size : " << p[process\_itr - 1].pro\_size << " Bottom : " << p[process\_itr - 1].pro\_bottom <<

" Top : " << p[process\_itr - 1].pro\_top << endl;

//Appending Processes to a memory

ptr\_M->pop(p[process\_itr - rand\_rmv].pro\_size, p[process\_itr - rand\_rmv].pro\_top, p[process\_itr - rand\_rmv].pro\_bottom);

ptr\_M->pop(p[process\_itr - rand\_rmv\_2].pro\_size, p[process\_itr - rand\_rmv\_2].pro\_top, p[process\_itr - rand\_rmv\_2].pro\_bottom);

cout << "After removing processes" << endl;

ptr\_M->print\_memory\_status();

cout << "Continue? "; cin >> control; // 0 -> continue 1 -> quit

} while (control == false);

}

결과에 대하여 먼저 설명하자면 메모리 사이즈를 100으로 잡고 프로세스의 사이즈를 1 부터 30까지 랜덤으로 한 결과 3, 4, 5번째 프로세스를 제거하고 나서부터 메모리가 조금씩 더러워 지기 시작하더니 아홉번째 프로세스까지 할당한 뒤 10번째 프로세스에게는 더 이상 할당 될 공간이 돌아가지 않았습니다. 9, 10, 11번째 프로세스의 크기는 각각 19, 22, 25였는데, 사실 메모리 내 할애가능 총 공간의 크기는 60이나 되었습니다. 위 세개의 프로세스를 다 더했을 때 사이즈가 66이긴 하지만 프로세스를 연속적으로 할당하지 않았다면 셋중 두개는 들어갈 수 있엇지만, 연속 메모리 할당 방식 하에서는 9번째 프로세스 하나를 위한 자리밖에 없었습니다. 위 시뮬레이션에서 연속 프로세스 할당의 단점을 발견 할 수 있었습니다.

빨간색으로 표시된 메모리가 오류가 발생하기 직전의 메모리 상태입니다. 메모리 사이즈연속 100에 프로세스 사이즈 1~30으로는 메모리 할당 방식의 단점을 극대화하여 표현하기 힘들다고 판단하여 메모리 사이즈를 1000으로 잡고 프로세스 사이즈는 1~80으로 하여 한번 더 시뮬레이션 하고 그 마지막 결과도 마지막 페이지에 같이 첨부하였습니다. 그 결과 중간중간 빨간색으로 하이라이트 해 놓은 아까운 메모리 자원의 낭비를 확인 할 수 있었습니다.

==============================Result==============================

Memory Created

0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0

Writing complete

Before removing process 2 and 0

1 1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 0 0 0

0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0

Process ID : 0 Process Size : 6 Bottom : 0 Top : 5

Process ID : 1 Process Size : 9 Bottom : 6 Top : 14

Process ID : 2 Process Size : 12 Bottom : 15 Top : 26

After removing processes

0 0 0 0 0 0 1 1 1 1

1 1 1 1 1 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0

Continue? 0

Writing complete

Writing complete

Writing complete

Before removing process 5 and 3

0 0 0 0 0 0 1 1 1 1

1 1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1 1

1 1 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0

Process ID : 3 Process Size : 9 Bottom : 15 Top : 23

Process ID : 4 Process Size : 12 Bottom : 24 Top : 35

Process ID : 5 Process Size : 16 Bottom : 36 Top : 51

After removing processes

0 0 0 0 0 0 1 1 1 1

1 1 1 1 1 0 0 0 0 0

0 0 0 0 1 1 1 1 1 1

1 1 1 1 1 1 0 0 0 0

0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0

Continue? 0

Writing complete

Writing complete

Writing complete

Before removing process 7 and 6

0 0 0 0 0 0 1 1 1 1

1 1 1 1 1 0 0 0 0 0

0 0 0 0 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1 1

1 1 1 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0

Process ID : 6 Process Size : 12 Bottom : 36 Top : 47

Process ID : 7 Process Size : 16 Bottom : 48 Top : 63

Process ID : 8 Process Size : 19 Bottom : 64 Top : 82

After removing processes

0 0 0 0 0 0 1 1 1 1

1 1 1 1 1 0 0 0 0 0

0 0 0 0 1 1 1 1 1 1

1 1 1 1 1 1 0 0 0 0

0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0

0 0 0 0 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1 1

1 1 1 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0

Continue? 0

Writing complete

================================CANNOT WRITE 10 ===============================

================================CANNOT WRITE 11 ===============================

Before removing process 10 and 11

0 0 0 0 0 0 1 1 1 1

1 1 1 1 1 0 0 0 0 0

0 0 0 0 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1 1

1 1 1 1 1 0 0 0 0 0

0 0 0 0 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1 1

1 1 1 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0

Process ID : 9 Process Size : 19 Bottom : 36 Top : 54

Process ID : 10 Process Size : 22 Bottom : -858993460 Top : -858993460

Process ID : 11 Process Size : 25 Bottom : -858993460 Top : -858993460

===========================추가 결과==============================

================================CANNOT WRITE 67 ===============================

Writing complete

Before removing process 66 and 67

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0

Process ID : 66 Process Size : 75 Bottom : 922 Top : 996

Process ID : 67 Process Size : 78 Bottom : -858993460 Top : -858993460

Process ID : 68 Process Size : 2 Bottom : 20 Top : 21