

U18CO018
Shubham Shekhaliya
OS – Assignment 6

1. To implement first fit, best fit and worst fit storage allocation algorithms for memory management.

Description

A set of holes, of various sizes is scattered through the memory at any given time. When a process arrives and needs the memory, the system searches for a hole that is large enough for this process. The first-fit, best-fit and worst-fit are strategies used to select a free hole from the set of available holes.

Implementation details

Free space is maintained as a linked list of nodes with each node having the starting byte address and the ending byte address of a free block. Each memory request consists of the process-id and the amount of storage space required in bytes. Allocated memory space is again maintained as a linked list of nodes with each node having the process-id, starting byte address and the ending byte address of the allocated space. When a process finishes (taken as input) the appropriate node from the allocated list should be deleted and this free disk space should be added to the free space list. [Care should be taken to merge contiguous free blocks into one single block. This result in deleting more than one node from the free space list and changing the start and end address in the appropriate node]. For allocation use first fit, worst fit and best fit.

First-Fit

Allocate the first hole that is big enough. Searching can start either at the beginning of the set of holes or where the previous first-fit search ended. You can stop searching as soon as you find a free hole that is large enough.

Best-Fit

Allocate the smallest hole that is big enough. You must search the entire list unless the list is kept ordered by size. The strategy produces the smallest leftover hole.

Worst fit

Allocate the biggest hole.

Code:-

```
#include <bits/stdc++.h>
using namespace std;

vector< pair< int,int> > freeSpace;
vector < vector<int> > allocatedSpace;

void FirstFit(int id, int size) {
    bool found = false;
    int i = 0;
    for (pair<int, int> block: freeSpace) {
        if (block.second - block.first + 1 >= size) {
            // found
            found = true;
            int startAdd = block.first;
            int EndAdd = block.first + size - 1;
            allocatedSpace.push_back({startAdd, EndAdd, id});
            sort(allocatedSpace.begin(), allocatedSpace.end());
            if (EndAdd == block.second) {
                // remove the entry
                freeSpace.erase(freeSpace.begin() + i);
            } else {
                // Modify Entry
                freeSpace[i].first = freeSpace[i].first + size;
            }
            break;
        }
        i++;
    }
    if (!found) {
        cout << " Sorry No Space Available....." << endl;
    }
}
```

```

void BestFit(int id, int size) {
    bool found = false;
    int mnSize = INT_MAX;
    int index = -1;
    int startAdd, EndAdd;
    int i = 0;
    for (pair<int, int> block:freeSpace) {
        if (block.second - block.first + 1 >= size) {
            // found
            found = true;
            if (block.second - block.first + 1 < mnSize) {
                index = i;
                startAdd = block.first;
                EndAdd = block.first + size - 1;
                mnSize = block.second - block.first + 1;
            }
        }
        i++;
    }
    if (!found) {
        cout << " Sorry No Space Available....." << endl;
        return;
    }
    pair<int, int> block = freeSpace[index];
    allocatedSpace.push_back({startAdd, EndAdd, id});
    sort(allocatedSpace.begin(), allocatedSpace.end());
    if (EndAdd == block.second) {
        // remove the entry
        freeSpace.erase(freeSpace.begin() + index);
    } else {
        // Modify Entry
        freeSpace[index].first = freeSpace[index].first + size;
    }
}

```

```

void WorstFit(int id,int size) {
    bool found = false;
    int mxSize = INT_MIN;
    int index = -1;
    int startAdd, EndAdd;
    int i = 0;
    for (pair<int, int> block:freeSpace) {
        if (block.second - block.first + 1 >= size) {
            // found
            found = true;

```

```

        if (block.second - block.first + 1 > mxSize) {
            index = i;
            startAdd = block.first;
            EndAdd = block.first + size - 1;
            mxSize = block.second - block.first + 1;
        }
    }
    i++;
}
if (!found) {
    cout << " Sorry No Space Available....." << endl;
    return;
}
pair<int, int> block = freeSpace[index];
allocatedSpace.push_back({startAdd, EndAdd, id});
sort(allocatedSpace.begin(), allocatedSpace.end());
if (EndAdd == block.second) {
    // remove the entry
    freeSpace.erase(freeSpace.begin() + index);
} else {
    // Modify Entry
    freeSpace[index].first = freeSpace[index].first + size;
}
}

void DeleteAllocatedSpace(int id) {
    int start = -1, end = -1;
    for (int i = 0; i < allocatedSpace.size(); i++) {
        if (allocatedSpace[i][2] == id) {
            start = allocatedSpace[i][0];
            end = allocatedSpace[i][1];
            allocatedSpace.erase(allocatedSpace.begin() + i);
            break;
        }
    }
    if (start != -1) {
        // Check for merging
        for (auto & i : freeSpace) {
            if (end + 1 == i.first) {
                i.first = start;
                return;
            }
            if (i.second + 1 == start) {
                i.second = end;
            }
        }
    }
}

```

```

        return;
    }
}

// add Free Block
freeSpace.emplace_back(start, end);
sort(freeSpace.begin(), freeSpace.end());
}
}

void display() {
    cout<<"Free Space list \n";
    for (pair<int, int> p:freeSpace)
        cout << "{" << p.first << "," << p.second << "}\n";

    cout<<" ----- \n";
    cout<<" allocated Space list \n";
    for(int i=0;i<allocatedSpace.size();i++)
        cout << "{" << allocatedSpace[i][0] << "," << allocatedSpace[i][1] << ","
" << allocatedSpace[i][2] << " }\n";
}

int main() {
    freeSpace.push_back({0, 100});
    int id;
    while (true) {
        cout<< "Enter 1 to add new proces \n"
            "Enter 2 to remove one process \n"
            "Enter 3 to See memory allocation\n"
            "Enter 4 to exit\n";

        int type;
        cin >> type;
        if (type == 1) {
            cout << "Enter Process ID " << endl;
            cin >> id;
            bool flag = false;
            for (int i = 0; i < allocatedSpace.size(); i++) {
                if (allocatedSpace[i][2] == id) {
                    cout << "ID already Exists .. \n";
                    flag = true;
                }
            }
            if (flag)
                continue;
            cout << "Enter Size of required space \n";
            int size;

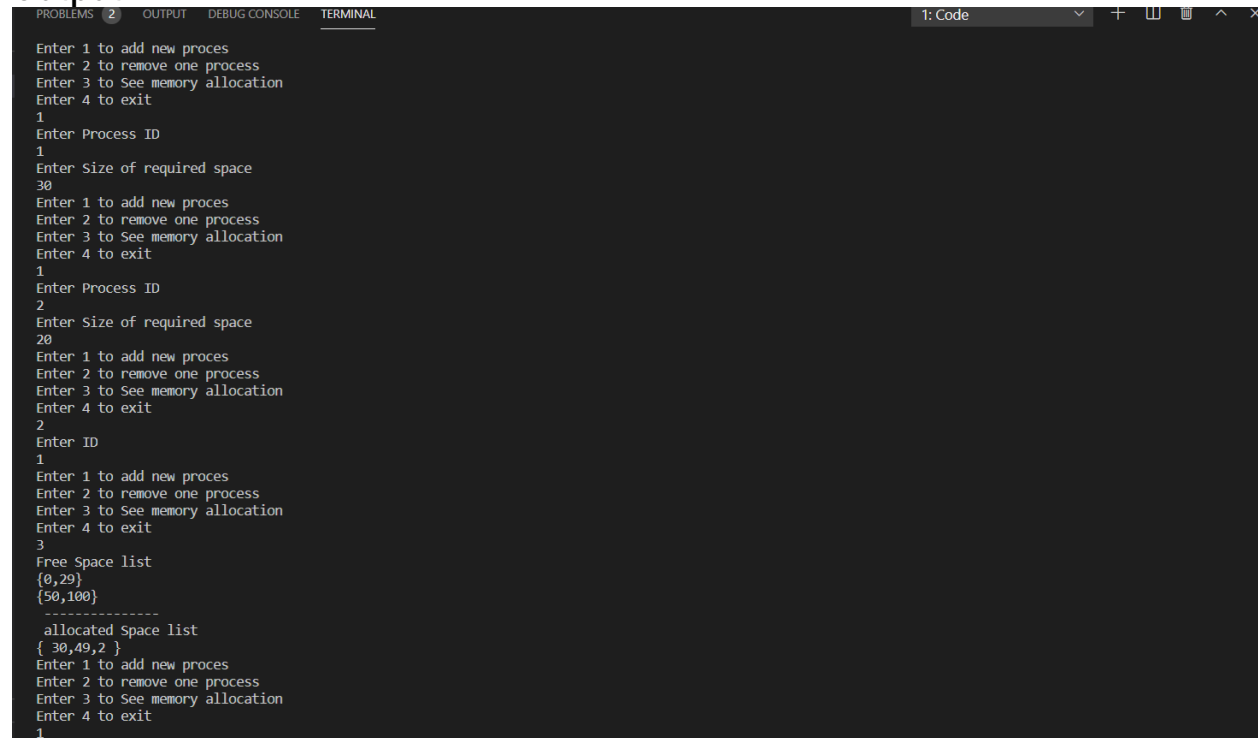
```

```

        cin >> size;
        WorstFit(id, size);
    } else if (type == 2) {
        cout << "Enter ID " << endl;
        cin >> id;
        DeleteAllocatedSpace(id);
    } else if (type == 3) {
        display();
    } else if (type == 4) {
        break;
    } else {
        cout << "Enter valid choice !! ";
    }
}
display();
return 0;
}

```

Output:-



```

PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL 1: Code
Enter 1 to add new proces
Enter 2 to remove one process
Enter 3 to See memory allocation
Enter 4 to exit
1
Enter Process ID
1
Enter Size of required space
30
Enter 1 to add new proces
Enter 2 to remove one process
Enter 3 to See memory allocation
Enter 4 to exit
1
Enter Process ID
2
Enter Size of required space
20
Enter 1 to add new proces
Enter 2 to remove one process
Enter 3 to See memory allocation
Enter 4 to exit
2
Enter ID
1
Enter 1 to add new proces
Enter 2 to remove one process
Enter 3 to See memory allocation
Enter 4 to exit
3
Free Space list
{0,29}
{50,100}
-----
allocated Space list
{ 30,49,2 }
Enter 1 to add new proces
Enter 2 to remove one process
Enter 3 to See memory allocation
Enter 4 to exit
1

```

```
PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL 1: Code
Enter 4 to exit
3
Free Space list
{0,29}
{50,100}
-----
allocated Space list
{ 30,49,2 }
Enter 1 to add new proces
Enter 2 to remove one process
Enter 3 to See memory allocation
Enter 4 to exit
1
Enter Process ID
3
Enter Size of required space
20
Enter 1 to add new proces
Enter 2 to remove one process
Enter 3 to See memory allocation
Enter 4 to exit
3
Free Space list
{0,29}
{70,100}
-----
allocated Space list
{ 30,49,2 }
{ 50,69,3 }
Enter 1 to add new proces
Enter 2 to remove one process
Enter 3 to See memory allocation
Enter 4 to exit
4
Free Space list
{0,29}
{70,100}
-----
allocated Space list
{ 30,49,2 }
{ 50,69,3 }
```

2. Write a program that implements the following Page replacement algorithm.

i) LRU (Least Recently Used)

```
#include<bits/stdc++.h>
using namespace std;
int main() {
    int n;
    cout << "Enter number of page\n";
    cin >> n;
    int pages[n];
    cout << "Enter the reference string\n";
    for (int i = 0; i < n; i++) {
        cin >> pages[i];
    }
    int capacity;
    cout << "Enter number of frame\n";
    cin >> capacity;

    set<int> frame;
    map<int, int> indexes;
    int page_faults = 0;
    for (int i = 0; i < n; i++) {
        bool flag=false;
        if (frame.size() < capacity) {
```

```

        if (frame.find(pages[i]) == frame.end()) {
            frame.insert(pages[i]);
            page_faults++;
            flag= true;
        }
        indexes[pages[i]] = i;
    } else {
        if (frame.find(pages[i]) == frame.end()) {
            int lru = INT_MAX, val;
            for (auto it:frame) {
                if (indexes[it] < lru) {
                    lru = indexes[it];
                    val = it;
                }
            }
            frame.erase(val);
            indexes.erase(val);
            frame.insert(pages[i]);
            page_faults++;
            flag = true;
        }
        indexes[pages[i]] = i;
    }
    cout << "id = " << i << " page[i] = " << pages[i] << " --> ";
    for (auto it:frame)
        cout << it << " ";
    if(flag) cout<<" Miss\n";
    else cout<<" Hit\n";
}
cout << "no of page faults = " << page_faults << endl;
return 0;
}

```

Output:-

```

D:\6th SEM\OS>cd "d:\6th SEM\OS\Assignment6\" && g++ second1.cpp -o second1 && "d:\6th SEM\OS\Assignment6\"second1
Enter number of page
12
Enter the reference string
1 2 3 4 1 2 5 1 2 3 4 5
Enter number of frame
4
id = 0 page[i] = 1 --> 1 Miss
id = 1 page[i] = 2 --> 1 2 Miss
id = 2 page[i] = 3 --> 1 2 3 Miss
id = 3 page[i] = 4 --> 1 2 3 4 Miss
id = 4 page[i] = 1 --> 1 2 3 4 Hit
id = 5 page[i] = 2 --> 1 2 3 4 Hit
id = 6 page[i] = 5 --> 1 2 4 5 Miss
id = 7 page[i] = 1 --> 1 2 4 5 Hit
id = 8 page[i] = 2 --> 1 2 4 5 Hit
id = 9 page[i] = 3 --> 1 2 3 5 Miss
id = 10 page[i] = 4 --> 1 2 3 4 Miss
id = 11 page[i] = 5 --> 2 3 4 5 Miss
no of page faults = 8

```


ii) Optimal Page Replacement algorithm

```
#include <bits/stdc++.h>
using namespace std;
int main() {
    int n;
    cout << "Enter number of page\n";
    cin >> n;
    int pages[n];
    cout << "Enter the reference string\n";
    for (int i = 0; i < n; i++) {
        cin >> pages[i];
    }
    int capacity;
    cout << "Enter number of frame\n";
    cin >> capacity;

    int page_faults = 0;
    set<int> frame;
    int hit = 0;
    for (int i = 0; i < n; i++) {
        bool flag=false;
        if (frame.find(pages[i])==frame.end()) {
            page_faults++;
            flag=true;
            if (frame.size() < capacity)
                frame.insert(pages[i]);
            else {
                int value = -1, farthest = i + 1;
                for (auto it:frame) {
                    int j;
                    for (j = i + 1; j < n; j++) {
                        if (it == pages[j]) {
                            if (j > farthest) {
                                farthest = j;
                                value = it;
                            }
                        }
                    }
                    break;
                }
                if (j == n) {
                    value = it;
                    break;
                }
            }
        }
        if (value == -1) {
```

```

        frame.erase(frame.begin());
    } else {
        frame.erase(value);
    }
    frame.insert(pages[i]);
}

}

cout << "id = " << i << " page[i] = " << pages[i] << " --> ";
for (auto it:frame)
    cout << it << " ";
if(flag) cout<<" Miss\n";
else cout<<" Hit\n";
}

cout << "no of page faults = " << page_faults << endl;
return 0;
}

```

Output:-

```

d:\6th SEM\OS\Assignment6>cd "d:\6th SEM\OS\Assignment6\" && g++ second2.cpp -o second2 && "d:\6th SEM\OS\Assignment6\"second2
Enter number of page
12
Enter the reference string
1 2 3 4 1 2 5 1 2 3 4 5
Enter number of frame
4
id = 0 page[i] = 1 --> 1      Miss
id = 1 page[i] = 2 --> 1 2      Miss
id = 2 page[i] = 3 --> 1 2 3      Miss
id = 3 page[i] = 4 --> 1 2 3 4      Miss
id = 4 page[i] = 1 --> 1 2 3 4      Hit
id = 5 page[i] = 2 --> 1 2 3 4      Hit
id = 6 page[i] = 5 --> 1 2 3 5      Miss
id = 7 page[i] = 1 --> 1 2 3 5      Hit
id = 8 page[i] = 2 --> 1 2 3 5      Hit
id = 9 page[i] = 3 --> 1 2 3 5      Hit
id = 10 page[i] = 4 --> 2 3 4 5      Miss
id = 11 page[i] = 5 --> 2 3 4 5      Hit
no of page faults = 6

```