# Laboratory 7

Title of the Laboratory Exercise: Programs for memory management algorithms

1. Introduction and Purpose of Experiment

In a multiprogramming system, the user part of memory must be further subdivided to accommodate multiple processes. This task of subdivision is carried out dynamically done by the operating system known as memory management. By solving these problems students will become familiar with the implementations of memory management algorithms in dynamic memory partitioning scheme of operating system.

1. Aim and Objectives

Aim

* To develop a simulator for memory management algorithms

Objectives

At the end of this lab, the student will be able to

* Apply memory management algorithms wherever required
* Develop simulators for the algorithms

1. Experimental Procedure
   * + Analyse the problem statement
     + Design an algorithm for the given problem statement and develop a flowchart/pseudo-code
     + Implement the algorithm in C language
     + Compile the C program
     + Test the implemented program
     + Document the Results
     + Analyse and discuss the outcomes of your experiment
2. Questions

Implement a simulator for the memory management algorithms with the provision of compaction and garbage collection

1. First fit
2. Best fit
3. Worst fit
4. Calculations/Computations/Algorithms
5. Presentation of Results

#include <algorithm>

#include <iostream>

#include <iterator>

#include <list>

using block\_list\_t = **std**::**list**<**std**::**pair**<char, int>>;

using proc\_list\_t = **std**::**list**<**std**::**pair**<char, int>>;

**std**::**ostream**& operator<<(**std**::**ostream**& out, **std**::**list**<**std**::**pair**<char, int>> list) {

    out << "START -> ";

    for (auto [mem\_name, mem\_size] : list) {

        out << mem\_name << " " << mem\_size << " -> ";

    }

    out << "END" << **std**::endl;

    return out;

}

void **worst\_fit**(block\_list\_t blist, proc\_list\_t plist) {

**std**::cout << "Worst Fit" << **std**::endl;

    for (auto [proc\_name, proc\_mem] : plist) {

        block\_list\_t::iterator worst = blist.**end**();

        for (auto it = blist.**begin**(); it != blist.**end**(); it++) {

            auto [block\_name, block\_size] = \*it;

            if (block\_name != 'H')

                continue;

            if (block\_size >= proc\_mem) {

                if (worst == blist.**end**())

                    worst = it;

                else if (block\_size > worst->second)

                    worst = it;

            }

        }

*// found the worst fit*

        if (worst != blist.**end**()) {

            auto [block\_name, block\_size] = \*worst;

            int rem\_mem = block\_size - proc\_mem;

            if (rem\_mem > 0) {

                (\*worst) = {proc\_name, proc\_mem};

                blist.**insert**(**std**::**next**(worst), {'H', rem\_mem});

            } else if (rem\_mem == 0) {

                (\*worst) = {proc\_name, proc\_mem};

            }

        }

    }

**std**::cout << "Memory Allocation List" << **std**::endl;

**std**::cout << blist;

}

void **best\_fit**(block\_list\_t blist, proc\_list\_t plist) {

**std**::cout << "Best Fit" << **std**::endl;

    for (auto [proc\_name, proc\_mem] : plist) {

        block\_list\_t::iterator best = blist.**end**();

        for (auto it = blist.**begin**(); it != blist.**end**(); it++) {

            auto [block\_name, block\_size] = \*it;

            if (block\_name != 'H')

                continue;

            if (block\_size >= proc\_mem) {

                if (best == blist.**end**())

                    best = it;

                else if (block\_size < best->second)

                    best = it;

            }

        }

*// found the best fit*

        if (best != blist.**end**()) {

            auto [block\_name, block\_size] = \*best;

            int rem\_mem = block\_size - proc\_mem;

            if (rem\_mem > 0) {

                (\*best) = {proc\_name, proc\_mem};

                blist.**insert**(**std**::**next**(best), {'H', rem\_mem});

            } else if (rem\_mem == 0) {

                (\*best) = {proc\_name, proc\_mem};

            }

        }

    }

**std**::cout << "Memory Allocation List" << **std**::endl;

**std**::cout << blist;

}

void **first\_fit**(block\_list\_t blist, proc\_list\_t plist) {

**std**::cout << "First Fit" << **std**::endl;

    for (auto [proc\_name, proc\_mem] : plist) {

        for (auto it = blist.**begin**(); it != blist.**end**(); it++) {

            auto [block\_name, block\_size] = \*it;

            if (block\_name != 'H')

                continue;

            int rem\_mem = block\_size - proc\_mem;

            if (rem\_mem > 0) {

                (\*it) = {proc\_name, proc\_mem};

                blist.**insert**(**std**::**next**(it), {'H', rem\_mem});

                break;

            } else if (rem\_mem == 0) {

                (\*it) = {proc\_name, proc\_mem};

                break;

            }

        }

    }

**std**::cout << "Memory Allocation List" << **std**::endl;

**std**::cout << blist;

}

int **main**(int argc, char\* argv[]) {

    block\_list\_t block\_list{{'H', 100}, {'H', 500}, {'H', 200}, {'H', 300}, {'H', 600}};

    proc\_list\_t proc\_list{{'A', 212}, {'B', 417}, {'C', 112}, {'D', 426}};

**std**::cout << "Memory Before Allocation" << **std**::endl;

**std**::cout << block\_list;

**std**::cout << "Process List" << **std**::endl;

**std**::cout << proc\_list;

**std**::cout << **std**::endl;

**first\_fit**(block\_list, proc\_list);

**std**::cout << **std**::endl;

**best\_fit**(block\_list, proc\_list);

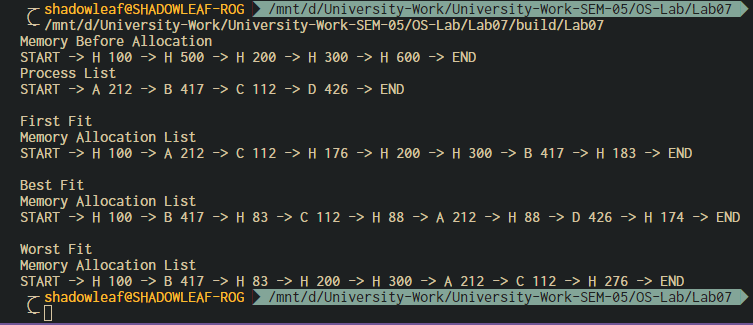
**std**::cout << **std**::endl;

**worst\_fit**(block\_list, proc\_list);

    return EXIT\_SUCCESS;

}

1. Analysis and Discussions



1. Conclusions
2. Comments

1. Limitations of Experiments

2. Limitations of Results

3. Learning happened

4. Recommendations