# ЦЕЛЬ РАБОТЫ

Исследовать стратегии и методы управления виртуальной памятью в современных ОС.

# ПОСТАНОВКА ЗАДАЧИ

1. Разработать программу, моделирующую работу алгоритмов замещения страниц FIFO и BIFO. В алгоритме BIFO последовательность обращений к страницам следует определить для каждого задания J1, ..., JM.
2. Исходные данные к программе:

* объём оперативной памяти в страницах (Vоп);
* последовательность обращений к страницам REF(1), REF(2), ..., REF(N);

1. Результаты:

* план размещения страниц в оперативной (реальной) памяти в моменты времени t1, ..., tN;
* число страничных прерываний из-за отсутствия необходимой страницы (Nстр).

1. Сравнить результаты моделирования для заданных алгоритмов замещения страниц.

# ОПИСАНИЕ ПРОГРАММЫ

При запуске программы реализующей алгоритм замещения страниц FIFO, предлагается ввести объем оперативной памяти в страницах(RAM volume), максимальный номер страницы для генерации (maxRefValue) и количество операций. Последовательность подаваемых страниц выбирается с помощью генератора случайных чисел. Далее на экран выводится номер момента времени, план размещения страниц, а так же флаг – F если произошло замещение. По завершению работы программы выводится количество страничных прерываний.

При запуске программы реализующей алгоритм замещения страниц BIFO, так же предлагается ввести объем оперативной памяти в страницах(RAM volume), максимальный номер страницы для генерации (maxRefValue), количество процессов использующих операционную память и количество операций. Далее предлагается ввести количество страниц оперативной памяти выделяемых для каждого из процессов. При некорректном вводе предлагается повторный ввод. Последовательность подаваемых страниц выбирается с помощью генератора случайных чисел. Как и в реализации алгоритма FIFO на экран выводится номер момента времени, план размещения страниц, а так же флаг – F если произошло замещение. По завершению работы программы выводится количество страничных прерываний.

# ТЕКСТ ПРОГРАММ

1. Текст программы реализации алгоритма замещения страниц FIFO

#include <iostream>

#include <iomanip>

#include <vector>

#include <conio.h>

typedef std::vector<size\_t> vectValue;

void refRandomFill(vectValue &REF, size\_t &maxValue) {

for (size\_t i = 0; i < REF.size(); ++i) {

REF[i] = 1 + (rand() % (maxValue));

}

}

bool isZero(size\_t &val) {

if (val == 0) {

return true;

}

else {

return false;

}

}

bool isExist(vectValue &vect, size\_t &val, size\_t firstIndex, size\_t lastIndex) {

for (size\_t i = firstIndex; i < lastIndex; ++i) {

if (vect[i] == val) return true;

}

return false;

}

void printVector(vectValue &vect, size\_t firstIndex, size\_t lastIndex) {

std::cout << "-------" << std::endl;

for (size\_t i = firstIndex; i < lastIndex; ++i) {

if (isZero(vect[i])) {

std::cout << "| " << std::setw(6) << " | " << std::endl;

}

else {

std::cout << "| " << std::setw(3) << vect[i] << " | " << std::endl;

}

std::cout << "-------" << std::endl;

}

}

size\_t tryFIFO(vectValue &vect, vectValue &REF, size\_t indexOfFirst, size\_t indexOfLast) {

size\_t Fcounter = 0;

size\_t tempIndex = 0;

bool fullestFlag = false;

bool fFlag = false;

for (size\_t k = 0; k < REF.size(); ++k) {

if (!isExist(vect, REF[k], 0, vect.size())) {

vect[tempIndex++] = REF[k];

fFlag = true;

if (!fullestFlag) {

fFlag = false;

}

}

else {

fFlag = false;

}

std::cout << std::setw(5) << "T = " << (k + 1) << std::endl;

std::cout << REF[k] << std::endl;

printVector(vect, 0, vect.size());

if (fFlag) {

++Fcounter;

std::cout << std::setw(3) << "F" << std::endl;

}

if (tempIndex == vect.size()) {

tempIndex = 0;

fullestFlag = true;

}

std::cout << std::endl;

\_getch();

}

return Fcounter;

}

int main() {

size\_t operAmount = 0, V = 0, maxRefValue = 0;

//\*

std::cout << "Enter RAM volume: ";

std::cin >> V;

std::cout << "Enter maxRefValue: ";

std::cin >> maxRefValue;

std::cout << "Enter operation amount: ";

std::cin >> operAmount;

vectValue REF(operAmount);

refRandomFill(REF, maxRefValue);

vectValue FIFO\_ARR(V);

size\_t F1 = tryFIFO(FIFO\_ARR, REF, 0, FIFO\_ARR.size());

std::cout << "Amount of breaks FIFO: " << F1 << std::endl;

//\*/

\_getch();

return 0;

}

2. Текст программы реализации алгоритма замещения страниц BIFO

#include <iostream>

#include <iomanip>

#include <vector>

#include <conio.h>

typedef std::vector<size\_t> vectValue;

typedef std::vector<std::vector<size\_t> > matrix;

bool isExistOnColumn(matrix &matr, size\_t &val, size\_t column) {

for (size\_t i = 0; i < matr.size(); ++i) {

if (matr[i][column] == val) {

return true;

}

}

return false;

}

void matrixRandomFill(matrix &matrix, size\_t maxValue) {

for (size\_t i = 0; i < matrix.size(); ++i) {

for (size\_t j = 0; j < matrix[i].size(); ++j) {

size\_t tempRandomNum = 0;

do {

tempRandomNum = 1 + (rand() % (maxValue));

matrix[i][j] = tempRandomNum;

} while (!isExistOnColumn(matrix, tempRandomNum, j));

}

}

}

bool isZero(size\_t &val) {

if (val == 0) {

return true;

}

else {

return false;

}

}

bool isExist(vectValue &vect, size\_t &val, size\_t firstIndex, size\_t lastIndex) {

for (size\_t i = firstIndex; i < lastIndex; ++i) {

if (vect[i] == val) return true;

}

return false;

}

void printVector(vectValue &vect, size\_t firstIndex, size\_t lastIndex) {

std::cout << "-------" << std::endl;

for (size\_t i = firstIndex; i < lastIndex; ++i) {

if (isZero(vect[i])) {

std::cout << "| " << std::setw(6) << " | " << std::endl;

}

else {

std::cout << "| " << std::setw(3) << vect[i] << " | " << std::endl;

}

std::cout << "-------" << std::endl;

}

}

bool FIFO(vectValue &vect, size\_t processNUm, matrix &REF, size\_t colNum, size\_t indexOfFirst, size\_t indexOfLast) {

static std::vector<size\_t> temp(REF.size());

static std::vector<bool> tempResult(REF.size());

std::cout << "task" << processNUm + 1 << std::endl;

bool Fresult = false;

if (!isExist(vect, REF[processNUm][colNum], indexOfFirst, indexOfLast)) {

size\_t tempIndex = (indexOfFirst + temp[processNUm]++);

size\_t tempValue = vect[tempIndex];

vect[tempIndex] = REF[processNUm][colNum];

if (temp[processNUm] == (indexOfLast - indexOfFirst)) {

tempResult[processNUm] = true;

temp[processNUm] = 0;

}

if ((tempResult[processNUm]) && (tempValue != 0)) {

Fresult = true;

}

}

std::cout << REF[processNUm][colNum] << std::endl;

printVector(vect, indexOfFirst, indexOfLast);

if (Fresult)std::cout << std::setw(4) << "F";

std::cout << std::endl;

return Fresult;

}

size\_t tryBIFO(vectValue &vect, matrix &REF, size\_t taskAmount, vectValue taskDiv) {

size\_t Fcounter = 0;

for (size\_t k = 0; k < REF[0].size(); ++k) {

std::cout << std::setw(5) << "T = " << (k + 1) << std::endl;

for (size\_t i = 1; i <= taskAmount; ++i) {

Fcounter += FIFO(vect, i - 1, REF, k, taskDiv[i - 1], taskDiv[i]);

}

\_getch();

std::cout << std::setw(5) << "-------------------------------- "<< std::endl;

}

return Fcounter;

}

size\_t vectSum(vectValue &vect) {

size\_t sum = 0;

for (size\_t i = 0; i < vect.size(); ++i) {

sum += vect[i];

}

return sum;

}

int main() {

size\_t operAmount = 0, V = 0, maxRefValue = 0;

size\_t processesAmount = 0;

std::cout << "Enter RAM volume: ";

std::cin >> V;

std::cout << "Enter maxRefValue: ";

std::cin >> maxRefValue;

std::cout << "Enter processes amount: ";

std::cin >> processesAmount;

std::cout << "Enter operation amount: ";

std::cin >> operAmount;

std::cout << std::endl;

vectValue taskDiv(processesAmount + 1);

bool result = false;

size\_t temp = 0;

do {

if (result) {

temp = 0;

std::cout << "input error, enter correctly pages amount for processes!" << std::endl;

}

for (size\_t i = 1; i < (processesAmount); ++i) {

std::cout << "Enter pages amount for " << (i) << " process:";

size\_t task = 0;

std::cin >> task;

temp += task;

taskDiv[i] = taskDiv[i - 1] + task;

}

taskDiv[processesAmount] = V;

result = true;

} while ((temp) > (V-1));

vectValue BIFO\_ARR(V);

matrix BIFO\_REF(processesAmount, std::vector<size\_t>(operAmount));

matrixRandomFill(BIFO\_REF, maxRefValue);

size\_t F2 = tryBIFO(BIFO\_ARR, BIFO\_REF, processesAmount, taskDiv);

std::cout << "Amount of breaks BIFO: " << F2 << std::endl;

\_getch();

return 0;

}



# ТЕСТИРОВАНИЕ ПРОГРАММЫ

Для проверки работоспособности программы моделирующей алгоритм замещения страниц по методике FIFO, было разработано 2 тестовых примера, с разными наборами исходных данных:

При первом запуске на вход подавалась заведомо меньшее количество входящих страниц, чем возможно разместить в оперативной памяти. Ожидаемый результат: все страницы размещены в оперативной памяти без вызова прерываний. Результат работы программы изображен на рисунке 1

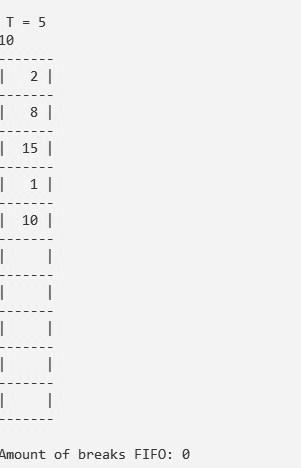


Рисунок 1 – Результат выполнения программы алгоритма FIFO с количеством страниц меньшим объема оперативной памяти

В качестве второго теста, на вход был подан набор данных содержащий определенные номера страниц, при этом максимальный номер которых не превышает размерности оперативной памяти, в результате ожидается полное размещение входящих страниц и отсутствие прерываний. Результат виден на рисунке 2.

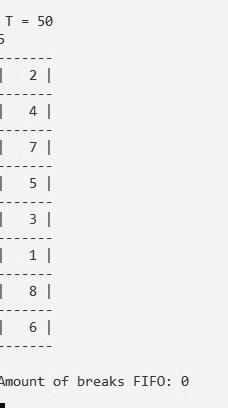


Рисунок 2 – Результат выполнения программы алгоритма FIFO с номерами страниц не превышающими размерность оперативной памяти

Для проверки работоспособности программы моделирующей алгоритм замещения страниц по методике BIFO, так же было разработано 2 тестовых примера:

В первом случае программе был передан параметр указывающий, что оперативной памятью пользуется один процесс, вследствие чего ожидается работа программы как обычного алгоритма FIFO, что видно на рисунке 3.

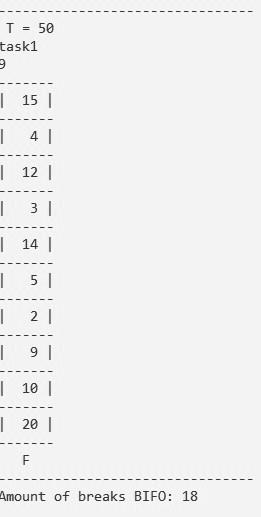


Рисунок 3 – Результат выполнения программы алгоритма BIFO для одного процесса

Во втором случае программа была запущенна с передачей ей 4 процессов, при чем первому процессу выделялось 3 страницы оперативной памяти, второму процессу – 2 страницы, 3 процессу – 1 страница и 4 – 4 страницы. Результат виден на рисунке 4.

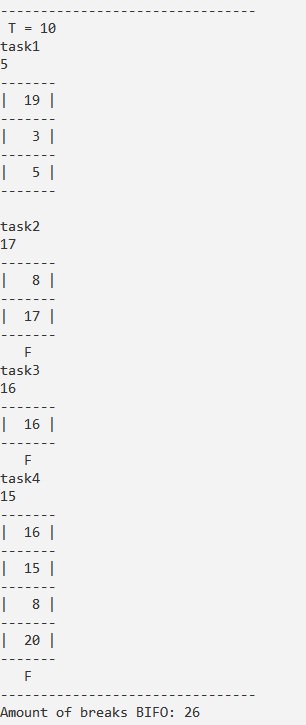


Рисунок 4 – Результат выполнения программы алгоритма BIFO для четырех процессов с пользовательским указанием выделения страниц

ВЫВОД

При выполнении лабораторной работы была написаны и протестированы программы на языке C++, моделирующие алгоритмы замещения страниц оперативной памяти FIFO и BIFO. При выполнении программ в качестве исходных данных использовались: объем оперативной памяти (в страницах), последовательность страниц подаваемых в оперативную память и их номера.

По результатам тестирования, можно сделать вывод, что программа работоспособны и поставленная цель достигнута.