SYMULACJA PAMIĘCI WIRTUALNEJ

```
this->frames = frames;
	this->pages = (fileSize/allocUnitSize)+1;
	this->allocUnitSize = allocUnitSize;
	pageList = new record[pages];
	ramList = new record[frames];


	for(int i=0; i< pages; i++)
	{
		pageList[i].memory=0;
		pageList[i].page=0;
```

2012-01-19

Przy użyciu algorytmu FIFO

Imię i Nazwisko: Bartosz Adamiak

Indeks 101917

Grupa: I-3-2

Informatyka, Stacjonarne I stopnia, rok 2, semestr 3

Systemy Operacyjne

Symulacja pamięci wirtualnej

PRZY UŻYCIU ALGORYTMU FIFO

1. TREŚĆ ZADANIA

W systemie pamięci wirtualnej, w którym dostępne są 3 ramki, adres składa się z 8 bitów a rozmiar strony wynosi 32 bajty. W systemie tym realizowany jest ciąg odniesień do komórek pamięci o następujących adresach: 40, 190, 60, 100, 160, 90, 130, 120, 150,70, 50, 180. Jak będzie się zmieniać zawartość ramek w wyniku realizacji tego ciągu oraz ile będzie błędów strony, jeśli zastosujemy algorytm:
(a) FIFO (First In First Out)

2. OPIS ROZWIĄZANIA PROBLEMU

a. Podstawowa struktura danych

```
class Unit
{
public:
	char * datablock;
	Unit();
	Unit(int datablockSize);
	char show(int
datablockPosition);
	char show();
	~Unit(void);
};
```

datablock – tablica n-bajtowa gdzie n to rozmiar podstawowej jednostki alokacji

show – metoda wyświetlająca

z parametrem – pojedynczy bajt

bez parametru – cały wiersz pamięci

b. Klasy użyte w programie

I. **Ram –** odpowiedzialna za pojedyncze ramki pamięci
II. **PageFile –** odpowiedzialna za pojedyncze strony
III. **FIFO –** zapisywanie danych dotyczących realizacji metody FIFO
   a. elementToChange - indeks najdawniej użytej ramki
   b. lastElement – indeks ostatniej ramki
IV. **Manager –** klasa zarządzająca pamięcią ram i pamięcią wirtualną, spełnia zadanie zarządcy pamięci

```
class FIFO
	FIFO(int
lastElement);
	int elementToChange;
	int lastElement;
	void next();
```

```
class Manager
{
	struct record
	{
		int page, memory;
		bool pagefault;
	};

	record * pageList;
	record * ramList;
	Ram * r;
	PageFile * pf;

	FIFO * fifo;
	Manager(int fileSize, int
frames, int allocUnitSize);
	void call(int logicAddres);
	void load(FILE * f);
	void writeStatus();
	~Manager(void);
}
```

a. Struktura rekord – podstawowa jednostka organizacji tablicy stron
   i. page – numer strony (używane w tablicy stron)
   ii. memory – numer ramki (używane do ustawiania flagi PF w tablicy stron przy usuwaniu danych z ramki)
   iii. pagefault – (flaga PF)
      1. true – strona tylko w pamięci wirtalnej
      2. false – strona w pamięci wirtualnej i RAM
      3. pageList, ramList – odpowiednio talica stron i tablica pamięci ram
b. r – pamięć RAM o długości frames (zawiera wiersze typu Unit o długości allocUnitSize)
c. pf – pamięć wirtualnej o długości fileSize/allocUnitSize (zawiera wiersze typu Unit o długości allocUnitSize)
d. metoda call – wywołuje adres logiczny, tłumaczy go na numer strony, w razie potrzeby przenosi stronę do ramki, zlicza błędy stronicowania.
e. writeStatus – prezentuje zawartość ramek i stron, komunikacja komputer->użytkownik, wyświetla ostatnio użyty adres logiczne i zawartość strony w której się znajduje, wyświetla ilość błędów stronnicowania
f. load – wczytuje zawartość pliku wejściowego do pamięci wirtualnej

V. **PageMemoryEmulator –** spełnia rolę pośrednika między procesem (użytkownikiem) a zarządcą pamięci, spełnia rolę interfejsu użytkownika (komunikacja użytkownik -> komputer), tworzy strukturę emulującą zarządcę systemu, decyduje o rozmiarach pamięci wirtualnej.

a. **Metoda call**

```
1 void Manager::call(int logicAddres)
2 {
3       if(!pageList[logicAddres/allocUnitSize].pagefault)
4       {
5               lastAddresCalled=logicAddres;
6               lastFrameModified=logicAddres/allocUnitSize;
7       }
8       else
9       {
10              errorCounter++;
11
12
13              pageList[ramList[fifo->current()].page].pagefault=1;
14              r->load(pf->page[logicAddres/allocUnitSize],fifo->current());
15
16              pageList[logicAddres/allocUnitSize].pagefault=false;
17              pageList[logicAddres/allocUnitSize].memory=fifo->current();
18
19              ramList[fifo->current()].page=logicAddres/allocUnitSize;
20
21              lastAddresCalled=logicAddres;
22              lastFrameModified=logicAddres/allocUnitSize;
23              fifo->next();
24
25
26      }
27 }
```

1 metoda wywoływana z parametrem którym jest adres logiczny, do którego chcemy się odnieść

3 jeżeli dana strona jest już w pamięci zapisujemy wywołany adres logiczny (5) Następnie zapisujemy ostatnio zmienioną ramkę (6) (do późniejszego użycia w metodzie writeStatus)

Jeżeli dana strona nie rezyduje w ramce (PF = true) zwiększamy licznik błędów stronicowania (errorCounter)(10) w tablicy stron ustawiamy flagę PF na true w stronie którą usuwamy z ramki(13) kopiujemy nową stronę do ramki (14) usuwamy flaę PF z aktualnej strony(16) ponieważ znajduje się już w ramce o adresie (17).
Ustawiamy adres strony z której została pobrana zawartość do ramki (19) (do późniejszego wykorzystania przy czyszczeniu ramki, następnie wykonujemy rutynową procedurę opisaną w punktach (5) i (6), a następnie przesuwamy wskaźnik najdawniej użytej ramki (23) do realizacji algorytmu FIFO.

b. metoda writeStatus

```
1 void Manager::writeStatus()
2 {
3      cout << endl << "-------------------------------------"
4              << endl << "-------------------------------------"
5              << endl << "PAGEFILE:";
6      for(int i=0; i<pages ; i++)
7      {
8        cout << endl << "page "; if(i<10) cout << "0"; cout << i << " :" << pf->page[i]->datablock;
9      }
10
11     cout << endl << "-------------------------------------"
12             << endl << "RAM:";
13     for(int i=0; i<frames ; i++)
14     {
15       cout << endl << "frame "; if(i<10) cout << "0"; cout << i << " " << r->memory[i]->datablock;
16     }
17     cout << endl << "-------------------------------------"
18             << endl << "PageFaults: " << errorCounter
19             << endl << "Last addres used: " << lastAddresCalled
20             << " Data Frame: " << pf->page[lastFrameModified]->datablock
21             << endl << "-------------------------------------"
22             << endl << "-------------------------------------"
23             << endl;
24     ;
25 }
```

Wyświetla informacje o zawartości stron w pamięci wirtualnej (3 -> 9)
Wyświetla informacje o zawartości ramek pamięci RAM (11->16)
Wywietla:

(18)Liczbę wystąpień błędu stronnicowania
(19)Ostatni wywołany adres logiczny
(20)Zawartość ostatnio używanej strony

c. Uwagi

Dane do pamięci wirtualnej wczytywane są z pliku „dane.txt" który musi znajdować się w jednym folderze z plikiem wykoywalnym.

3. OPIS INTERFEJSU

Wczytanie od użytkownika ilości dostępnych ramek pamięci RAM i rozmiaru jej pojedynczego wersu(rekordu)

Wyświetenie zawartości pamięci wirtualnej, z załadowanymi przykładowymi danymi, wiersz „Elegia o chłopcu polskim” K.K.Baczyńskiego

Wyświetlenie zawartości pamięci RAM

Zademonstrowano wygląd tuż po starcie programu gdy wszystkie ramki są puste

Wyświetlenie liczby wystąpień błędów stron, ostatniego użytego adresu, zawartości ostatniej użytej ramki

Wczytanie od urzytkonika następnego adresu logicznego do wywołania

```
Rozmiar pliku: 561

podaj ilosc ramek pamieci RAM:

podaj rozmiar jednostki alokacji:

-------------------------------------

-------------------------------------

PAGEFILE:

page 00 :Oddzielili cię, syneczku, od snów, co jak motyl

page 01 :drżą,haftowali ci, syneczku, smutne oczy rudą kr

page 02 :wią,malowali krajobrazy w żółte ściegi pożóg wys

page 03 :zywali wisielcami drzew płynące morze.Wyuczyli c

page 04 :ię, syneczku, ziemi twej na pamięć,gdyś jej ście

page 05 :żki powycinał żelaznymi łzami.Odchowali cię w ci

page 06 :emności, odkarmili bochnem trwóg,przemierzyłeś p

page 07 :o omacku najwstydliwsze z ludzkich dróg.I wyszed

page 08 :łeś jasny synku, z czarną bronią w noc,i poczułe

page 09 :ś, jak się jeży w dźwięku minut - zło.Zanim padł

page 10 :eś, jeszcze ziemię przeżegnałeś ręką.Czy to była

page 11 : kula, synku, czy to serce pękło?

-------------------------------------

RAM:

frame 00

frame 01

frame 02

frame 03

-------------------------------------

PageFaults: 0

Last addres used: 0 Data Frame: Oddzielili cię, syneczku, od
snów, co jak motyl

-------------------------------------

-------------------------------------


Wywolaj adres :
```

4. TESTOWANIE APLIKACJI

Przeprowadzono testy programu pod względem poprawności działania, dla przykładowych dany (3 ramki, jednostka alokacji 32bajty, wywołanie kolejnych adresów zawartych w zadaniu) stwierdzono poprawność działania. Wydruk programu:

```
PAGEFILE:
page 00 :Oddzielili cię, syneczku, od snó
page 01 :w, co jak motyl drżą,haftowali c
page 02 :i, syneczku, smutne oczy rudą kr
page 03 :wią,malowali krajobrazy w żółte
page 04 :ściegi pożóg wyszywali wisielcam
page 05 :i drzew płynące morze.Wyuczyli c
page 06 :ię, syneczku, ziemi twej na pami
page 07 :ęć,gdyś jej ścieżki powycinał że
page 08 :laznymi łzami.Odchowali cię w ci
page 09 :emności, odkarmili bochnem trwóg
page 10 :,przemierzyłeś po omacku najwsty
page 11 :dliwsze z ludzkich dróg.I wyszed
page 12 :łeś jasny synku, z czarną bronią
page 13 : w noc,i poczułeś, jak się jeży
page 14 :w dźwięku minut - zło.Zanim padł
page 15 :eś, jeszcze ziemię przeżegnałeś
page 16 :ręką.Czy to była kula, synku, cz
page 17 :y to serce pękło?ÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ
-------------------------------------
RAM:
frame 00 i drzew płynące morze.Wyuczyli c
frame 01 ściegi pożóg wyszywali wisielcam
frame 02 w, co jak motyl drżą,haftowali c
-------------------------------------
PageFaults: 7
Last addres used: 180 Data Frame: i drzew płynące morze.Wyuczyli c
```

5. KOMPLETNY KOD ŹRÓDŁOWY

a. Pagememoryemulator.cpp (main)

```
// PageMemoryEmulator.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"
#include "Manager.h"
#include <stdio.h>
#include <iostream>

using namespace std;
int fsize(char * file)
{
	FILE * f;
	f = fopen(file, "rt");
	fpos_t temp;
	fseek(f,0,SEEK_END);
	fgetpos(f,&temp);
	fclose(f);
	return (int)temp;
}
int _tmain(int argc, _TCHAR* argv[])
{
	int logicAddres,frames,allocUnitSize;
	int fileSize = fsize("dane.txt");


	cout << "Rozmiar pliku: " << fileSize;
	cout << endl << "podaj ilosc ramek pamieci RAM: ";cin >> frames;
	cout << endl << "podaj rozmiar jednostki alokacji: "; cin >> allocUnitSize;


	FILE * f;
	f = fopen("dane.txt","rb");
	Manager main(fileSize,frames,allocUnitSize);
	main.load(f); /*Wczytywanie danych do stron*/
	fclose(f);

	main.writeStatus();
	for(;;)
	{
	cout << endl << "Wywolaj adres : ";
	cin >> logicAddres;
	if(logicAddres>fileSize)
		{
			cout << endl << "Adres nie istnieje" << endl;
		}
		else
		{
			main.call(logicAddres);
			main.writeStatus();
		}

	}
	system("pause");
	return 0;
}
```

b. Manager.h

```
#pragma once
#include "PageFile.h"
#include "Ram.h"
#include "FIFO.h"
#include <iostream>
#include <stdio.h>

class Manager
{
public:
    int errorCounter,frames,pages,allocUnitSize,lastAddresCalled,lastFrameModified;
    struct record
    {
            int page, memory; //page strona, memory komórka pamięci w której znajduje się
dana strona
            bool pagefault;
    };

    record * pageList;
    record * ramList;
    Ram * r;
    PageFile * pf;

    FIFO * fifo;
    Manager(int fileSize, int frames, int allocUnitSize);
    void call(int logicAddres);
    void load(FILE * f);
    void writeStatus();
    ~Manager(void);
};
```

c. PageFile

```
#pragma once
#include "Unit.h"
#include <string>
using namespace std;
class PageFile
{
public:
    Unit ** page;
    PageFile(int numberOfPages, int frameSize);
    char show(int frame);
    Unit moveToRam(int frame);
    ~PageFile(void);
};
```

d. RAM.h

```
#pragma once
#include "unit.h"
#include "PageFile.h"
#include <string>
using namespace std;
class Ram
{
public:
    Unit ** memory;
    Ram(int numberOfFrames, int frameSize);
    char show(int frame, int position);
    char show(int frame);
    void load(Unit * from, int frame);
    ~Ram(void);
};
```

e. FIFO.h

```
#pragma once
class FIFO
{
public:
    FIFO(int lastElement);
    int elementToChange;
    int lastElement;
    void next();
    int current();

    ~FIFO(void);
};
```

f. Unit.h

```
#pragma once
class Unit
{
public:
    char * datablock;
    Unit();
    Unit(int datablockSize);
    char show(int datablockPosition);
    char show();
    ~Unit(void);
};
```

g. Manager.cpp

```
#include "StdAfx.h"
#include "Manager.h"


using namespace std;
Manager::Manager(int fileSize, int frames, int allocUnitSize)
{
    this->frames = frames;
    this->pages = (fileSize/allocUnitSize)+1;
    this->allocUnitSize = allocUnitSize;
    pageList = new record[pages]; //po stronach
    ramList = new record[frames]; //po ramkach

    /*hardcoded test*/
    for(int i=0; i< pages; i++)
    {
           pageList[i].memory=0;
           pageList[i].page=0;
           pageList[i].pagefault = true;
    }
    for(int i=0; i< frames; i++)
    {
           ramList[i].memory=0;
           ramList[i].page=0;
           ramList[i].pagefault = true;
    }


    r = new Ram(frames,allocUnitSize);
    pf = new PageFile(pages,allocUnitSize);
    fifo = new FIFO(frames-1);
    lastFrameModified = NULL;
    lastAddresCalled = NULL;
    errorCounter = NULL;
}

void Manager::call(int logicAddres)
{
    if(!pageList[logicAddres/allocUnitSize].pagefault) //logicAddres/pages jako wynik
podaje stroę w której znajduje się adres logiczny
    {
           lastAddresCalled=logicAddres;
           lastFrameModified=logicAddres/allocUnitSize;
    }
    else
    {
           errorCounter++;


           pageList[ramList[fifo->current()].page].pagefault=1; //ustawienie flagi PF na
stronie do usunięcia
           r->load(pf->page[logicAddres/allocUnitSize],fifo->current()); //ładowanie
nowej strony do pamięci

           pageList[logicAddres/allocUnitSize].pagefault=false; //ustawienie flagi PF w
przeniesionej stronie
           pageList[logicAddres/allocUnitSize].memory=fifo->current();

           ramList[fifo->current()].page=logicAddres/allocUnitSize;
```

```
            lastAddresCalled=logicAddres;
            lastFrameModified=logicAddres/allocUnitSize;
            fifo->next();


    }
}

void Manager::load(FILE * f)
{
    for(int i=0; i < pages ; i++)
    {
            fread(pf->page[i]->datablock,sizeof(char),allocUnitSize,f);
    }
}

void Manager::writeStatus()
{
    cout << endl << "-------------------------------------"
             << endl << "-------------------------------------"
             << endl << "PAGEFILE:";
  for(int i=0; i<pages ; i++)
  {
     cout << endl << "page "; if(i<10) cout << "0"; cout << i << " :" << pf->page[i]-
>datablock;
  }

    cout << endl << "-------------------------------------"
             << endl << "RAM:";
  for(int i=0; i<frames ; i++)
  {
     cout << endl << "frame "; if(i<10) cout << "0"; cout << i << " " << r->memory[i]-
>datablock;
  }
    cout << endl << "-------------------------------------"
             << endl << "PageFaults: " << errorCounter
             << endl << "Last addres used: " << lastAddresCalled << " Data Frame: " << pf-
>page[lastFrameModified]->datablock /*r->memory[lastAddresCalled]->datablock*/
             << endl << "-------------------------------------"
             << endl << "-------------------------------------"
             << endl;
    ;
}

Manager::~Manager(void)
{
}
```

h. PageFile.cpp

```
#include "StdAfx.h"
#include "PageFile.h"

using namespace std;
PageFile::PageFile(int numberOfpages, int frameSize)
{
    page = new Unit* [numberOfpages];

for( int i=0 ; i<numberOfpages ; i++)
    {
            page[i] = new Unit(frameSize);
    }
}

char PageFile::show(int frame)
{
    return *page[frame]->datablock;
}


Unit PageFile::moveToRam(int frame)
{
    return *page[frame];
}

PageFile::~PageFile(void)
{
}
```

i. RAM.cpp

```
#include "StdAfx.h"
#include <array>
#include "Ram.h"

using namespace std;
Ram::Ram(int numberOfFrames, int frameSize)
{
    memory = new Unit* [numberOfFrames];
for( int i=0 ; i<numberOfFrames ; i++)
    {
            memory[i] = new Unit(frameSize);
            memory[i]->datablock = "";
    }
}
char Ram::show(int frame, int position)
{
    return memory[frame]->show(position);
}
char Ram::show(int frame)
{
    return *memory[frame]->datablock;
}

void Ram::load(Unit * from, int position)
{
    this->memory[position] = from;
}Ram::~Ram(void)
{
}
```

j. FIFO.cpp

```
#include "StdAfx.h"
#include "FIFO.h"


FIFO::FIFO(int lastElement)
{
    this->lastElement = lastElement;
    elementToChange = 0;
}

void FIFO::next()
{
    if(elementToChange == lastElement)
           elementToChange = 0;
    else
           elementToChange++;
}

int FIFO::current()
{
    return elementToChange;
}

FIFO::~FIFO(void)
{
}
```

k. UNIT.cpp

```
#include "StdAfx.h"
#include "Unit.h"


Unit::Unit(int datablockSize)
{
    datablock = new char[datablockSize+1];
    datablock[datablockSize] = NULL;
}

Unit::Unit()
{
}

char Unit::show(int datablockPosition)
{
    return datablock[datablockPosition];
}

char Unit::show()
{
    return *datablock;
}

Unit::~Unit(void)
{
}
```