Dokumentacja Systemu Operacyjnego

ŻonkilOS

Projekt uniksopodobnego systemu operacyjnego, z użyciem języka programowania C#

Iwona Maraśkiewicz – Rok 2 Informatyki
na wydziale Elektrycznym, tryb stacjonarny

II. Opis rozwiązania

Poziom 1 – Zarządzanie procesorem i interpreter rozkazów

Zarządzenie procesorem jest najważniejszym mechanizmem w symulacji systemu operacyjnego. W przypadku systemu Unixopodobnego przydzielaniem odpowiedniego procesu do przetwarzania przez procesor zajmuje się algorytm karuzelowo-priorytetowy. Dla uproszczenia ilość kolejek priorytetowych została zmniejszona do 8. Każda z nich posiada odpowiedni przedział priorytetów (po 4 priorytety) mieszczących się w przedziale od 0 do 31.

1. Klasa Symulacja_procesora

Składnia

```
class Symulacja_procesora
{
  . . .
}
```

Pola

public int takt;	Pole przechowujące ilość taktów procesora (zwiększa się z każdym wykonanym rozkazem)
public int rekalk;	Pole przechowujące czas po jakim zostaną przeliczone priorytety dla wszystkich procesów
public int wywlaszcz;	Pole przechowujące czas po którym na nastąpić wywłaszczenie procesu
public TablicaKolejek main;	Pole przechowujące główną tablice kolejek priorytetowych
public Boolean blokada;	Pole odpowiedzialne za blokadę procesora (wykorzystywane, gdy proces został właśnie wykonany i wyszukiwany jest następny o najwyższym priorytecie)

Konstruktor

public Symulacja_procesora(int ptakt, int twywlaszcz, int prekalk)	Konstruktor tworzy automatycznie obiekt i przypisuje mu wartości (wartość początkową taktu procesora, czas po którym na nastąpić wywłaszczenie procesu i czas po jakim zostaną przeliczone priorytety dla wszystkich procesów) Konstruktor tworzy również tablice kolejek priorytetowych i przypisuje ją do pola main.

2. Klasa TablicaKolejek

Składnia

```
class TablicaKolejek
{
  …
}
```

Pola

public Boolean[] wektorbitowy = new Boolean[8];	Pole przechowujące 8- elementową tablice typu Boolean, która wskazuje czy jakieś elementy występują w odpowiadającej jej kolejce priorytetowej
public Kolejka[] tablicaWszystkich = new Kolejka[8];	Pole przechowujące 8- elementową tablice wszystkich kolejek priorytetowych
public Boolean coswpadlo = false;	Pole które zostaje ustawione na true w momencie, gdy nowy proces zostaje dodany do kolejki priorytetowej. Dzięki czemu następuje wywłaszczenie. Początkowo jest on ustawiony na wartość false.

Metody

public proces znajdz_zwroc_usun()	Funkcja odpowiedzialna za

	znalezienie procesu o najwyższym priorytecie, zwróceniu go i usunięciu z kolejki priorytetowej.
public proces znajdz_zworc()	Funkcja odpowiedzialna za znalezienie procesu o najwyższym priorytecie i zwróceniu go.
public void zmniejsziprzestaw()	Procedura dokonująca przeliczenia wszystkich priorytetów w kolejce priorytetowej i ustawienie w odpowiednich kolejkach.
public void info()	Procedura wyświetlająca tablice kolejek (PID i priorytet) a także stan wektora bitowego.
public void przestaw(proces p)	Procedura przestawiająca proces po wcześniej określonej liczbie cykli procesora.
public void przeliczdlaprocesu(proces p)	Procedura przeliczająca priorytet dla właśnie wywłaszczonego procesu.
public void dodaj(List<proces>listaprocgot)	Procedura dodająca procesy znajdujące się na liście procesów gotowych do odpowiednich kolejek priorytetowych
public void dodaj(proces proces)	Procedura dodająca proces do odpowiedniej kolejki priorytetowej
public proces dodajnowy(proces proces)	Funkcja dodająca nowy proces do odpowiedniej kolejki priorytetowej i zwracająca go.
public void usun(int kolejka)	Procedura usuwająca ostatni proces z kolejki.
public void wypiszStanRejestruProcesu(proces p)	Procedura wypisująca aktualny stan rejestrów procesu podanego jako parametr

3. Klasa Kolejka

Składnia

```
class Kolejka
{
 ...
}
```

Pola

public List<proces>listasp = new List<proces>();	Pole inicjujące właściwą kolejkę priorytetową

Metody

public void dodaj(proces pro)	Procedura odpowiedzialna za dodanie podanego jako parametr procesu do kolejki.
public void usun()	Procedura usuwająca ostatni element z kolejki.

4. Klasa Interpeter

Interpreter odpowiedzialny jest za analizę przekazanego mu kodu, który następnie zamienia na odpowiadający mu rozkaz i wykonuje. Zamienia 4-bajtowy kod składający się najczęściej z 2 części (kod rozkazu + dane). Niektóre rozkazy nie przewidują danych w przekazanym do interpretera kodzie. Przekazywany do interpretera jako parametr kod jest typu uint. Analiza i wykonanie rozkazu odbywa się bezpośrednio na rejestrach aktualnie przetwarzanego procesu.

Przykład :

Kod rozkazu:		**Czynność:**
1235	->	Wpisanie do rejestru R2 wartości 35
3212	->	Dodanie do rejestru R2 wartości 12
99	->	Zakończenie programu

Lista rozkazów

public uint MVI_R1=11	Wpisanie wartości do rejestru R1
public uint MVI_R2=12	Wpisanie wartości do rejestru R2
public uint MVI_R3=13	Wpisanie wartości do rejestru R3
public uint MVI_R4=14	Wpisanie wartości do rejestru R4
public uint MOV_R1_R2=15	Przeniesienie wartości z rejestru R2 do R1
public uint MOV_R1_R3=16	Przeniesienie wartości z rejestru R3 do R1
public uint MOV_R1_R4=17	Przeniesienie wartości z rejestru R4 do R1
public uint MOV_R2_R1=21	Przeniesienie wartości z rejestru R1 do R2
public uint MOV_R2_R3=22	Przeniesienie wartości z rejestru R3 do R2
public uint MOV_R2_R4=23	Przeniesienie wartości z rejestru R4 do R2
public uint MOV_R3_R1=24	Przeniesienie wartości z rejestru R1 do R3
public uint MOV_R3_R2=25	Przeniesienie wartości z rejestru R2 do R3
public uint MOV_R3_R4=26	Przeniesienie wartości z rejestru R4 do R3
public uint MOV_R4_R1=27	Przeniesienie wartości z rejestru R1 do R4
public uint MOV_R4_R2=28	Przeniesienie wartości z rejestru R2 do R4
public uint MOV_R4_R3=29	Przeniesienie wartości zrejestru R3 do R4
public uint ADD_R1=31	Dodawanie wartości do rejestru R1
public uint ADD_R2=32	Dodawanie wartości do rejestru R2
public uint ADD_R3=33	Dodawanie wartości do rejestru R3
public uint ADD_R4=34	Dodawanie wartości do rejestru R4
public uint ADD_R1_R2=35	Dodawanie wartości z rejestru R2 do rejestru R1
public uint SUB_R1=41	Odejmowanie wartości z rejestru R1
public uint SUB_R2=42	Odejmowanie wartości z rejestru R2
public uint SUB_R3=43	Odejmowanie wartości z rejestru R3
public uint SUB_R4=44	Odejmowanie wartości z rejestru R4
public uint SUB_R1_R2=45	Odejmowanie wartości podanej w

	rejestrze R2 od wartości w rejestrze R1
public uint MUL_R1=51	Mnożenie podanej wartości z wartością w rejestrze R1 i wpisanie jej do rejestru R1
public uint MUL_R2=52	Mnożenie podanej wartości z wartością w rejestrze R2i wpisanie jej do rejestru R2
public uint MUL_R3=53	Mnożenie podanej wartości z wartością w rejestrze R3i wpisanie jej do rejestru R3
public uint MUL_R4=54	Mnożenie podanej wartości z wartością w rejestrze R4i wpisanie jej do rejestru R4
public uint MUL_R1_R2=55	Mnożenie wartości rejestru R2 z wartością w rejestrze R1 i wpisanie jej do rejestru R1
public uint DIV_R1=61	Dzielenie wartości z rejestru R1 przez podaną wartość i wpisanie jej do rejestru R1
public uint DIV_R2=62	Dzielenie wartości z rejestru R2 przez podaną wartość i wpisanie jej do rejestru R2
public uint DIV_R3=63	Dzielenie wartości z rejestru R3 przez podaną wartość i wpisanie jej do rejestru R3
public uint DIV_R4=64	Dzielenie wartości z rejestru R4 przez podaną wartość i wpisanie jej do rejestru R4
public uint DIV_R1_R2=65	Dzielenie wartości z rejestru R1 przez wartość z rejestru R2 i wpisanie jej do rejestru R1
public uint INC_R1=71	Zwiększenie o 1 rejestru R1
public uint INC_R2=72	Zwiększenie o 1 rejestru R2
public uint INC_R3=73	Zwiększenie o 1 rejestru R3
public uint INC_R4=74	Zwiększenie o 1 rejestru R4
public uint DEC_R1=81	Zmniejszenie o 1 rejestru R1
public uint DEC_R2=82	Zmniejszenie o 1 rejestru R2
public uint DEC_R3=83	Zmniejszenie o 1 rejestru R3
public uint DEC_R4=84	Zmniejszenie o 1 rejestru R4
public uint JMP=91	Skok
public uint JNZ=92	Skok jeśli nie zero w R2

public uint RET=99	Zakończenie programu

Składnia

```
public class Interpreter
{
 …
}
```

Metody

public bool wykonajInstrukcje(proces p, byte[] tab)	Funkcja która przetwarza podany jako parametr rozkaz dla danego procesu, który również umieszczony jest jako parametr. Zwraca wartość bool, która ustawia się na true w momencie zakończenia programu. Początkowo ustawiona na false.

Poziom 2 – Zarządzanie Pamięcią

1. Klasa BlokPamieci

Klasa, w której znajduje się struktura bloków pamięci przydzielanych poszczególnym procesom. Ilość bloków pamięci, które mają zostać przydzielone na starcie procesowi jest ustalana przez użytkownika. W naszym systemie na starcie przydzieliliśmy każdemu procesowi po 3 bloki pamięci (tj. 48 bajtów).

Składnia

```
class BlokPamieci
{
  . . .
}
```

Pola

public int StronaPoczatkowa;	pole przechowujące adres strony początkowej bloku/bloków zajmowanego przez proces
public int Blokow;	pole przechowujące liczbę bloków zajmowanych przez proces
public BlokPamieci Nastepny;	pole wskazujące na następny blok pamięci

Konstruktor

```
public BlokPamieci(int stronaPoczatkowa, int blokow)
    {
        StronaPoczatkowa = stronaPoczatkowa;
        Blokow = blokow;
    }
```

Metody

public override string ToString()
{
 	
}
	Funkcja odpowiedzialna za ustalenie AdresPoczatkowy (StronaPoczatkowa*Pamiec.RozmiarBloku), Dlugosc (Blokow*Pamiec.RozmiarBloku) oraz AdresKoncowy (AdresPoczatkowy+Dlugosc) i wypisanie w konsoli listy zajętych bloków pamięci w formie : Od (AdresPoczatkowy) do (AdresKoncowy) Dlugosc: (Dlugosc) bajtow.

2. Klasa StronaPamieci

Klasa, która jest jest stroną w pamięci – posiada dane o stronie : jest rozmiar, adres, czy jest w pamięci fizycznej oraz czy jest na dysku.

Składnia

```
class StronaPamieci
{
  . . .
}
```

Pola

public const int Rozmiar = 16;	Pole przechowujące wielkość strony
public int AdresPoczatkowy;	Adres w pamięci fizycznej, w którym zaczyna się dana ramka
public bool JestZmapowanaWPamieciFizycznej;	Przyjmuje wartości : true – strona jest zmapowana w pamięci fizycznej, lub false – strony nie ma w pamięci fizycznej
public bool JestNaDysku;	Przyjmuje wartości : true – strona jest na dysku, lub false – strony nie ma na dysku

Metody

public override string ToString()
{
 	
}
	Zwraca tablicę stron w postaci : AdresPoczatkowy, JestZmapowanaWPamieciFizycznej oraz JestNaDysku

3. Klasa PlikStronicowania

Klasa, w której realizowane jest tworzenie pliku stronicowania na dysku oraz odczyt z pliku stronicowania, gdy strona znów jest potrzebna.

Składnia

```
class PlikStronicowania
{
  . . .
}
```

Pola

`private readonly List<StronaNaDysku> _stronyNaDysku = new List<StronaNaDysku>();`	Lista stron na dysku
`private struct StronaNaDysku { public short Numer; public byte[] Dane; }`	Struktura listy stron na dysku, która posiada takie pola jak : Numer strony oraz tablicę bajtów przechowującą dane (kod) procesu, który jest właścicielem tej strony.

Metody

`public byte[] Odczytaj(short numerStrony) { } `	Odczytuje stronę z pliku stronicowania z dysku, gdy jest ona znów potrzebna wyszukując ją po numerze strony. Dla każdej strony w _stronyNaDysku sprawdzane są pliki stronicowania po numerze strony i do tablicy `var copy = new byte[Pamiec.RozmiarBloku]` kopiowane są dane z pliku stronicowania : strona.Dane.CopyTo(copy, 0)
`public void Zapisz(short numerStrony, byte[] dane) {`	Zapisuje stronę do pliku stronicowania na dysku. Szukamy strony na dysku po numerze strony, jeśli była już utworzona to ją nadpisujemy : dane.CopyTo(strona.Dane, 0). Jeśli natomiast strony nie było na dysku

}	tworzymy nową : var nowaStrona = new StronaNaDysku {…}, zapisujemy do niej dane i ją dodajemy : _stronyNaDysku.Add(nowaStrona);

4. Klasa Pamiec

Klasa, w której mapujemy stronę do pamięci fizycznej, przeliczany jest adres wirtualny na fizyczny, odczytujemy i zapisujemy z/do pamięci fizycznej, alokujemy pamięć metodą First-Fit, zwalniamy pamięć fizyczną danego procesu oraz wyświetlamy stan całej pamięci : zajęte bloki, tablice stron oraz pamięć fizyczną w postaci tablicy.

Składnia

```
internal class Pamiec
{
  . . .
}
```

Pola

public const int RozmiarBloku = StronaPamieci.Rozmiar;	Pole przechowujące rozmiar bloku pamięci, który jest równy rozmiarowi strony, czyli tu 16 bajtów
private readonly byte[] _pamiecFizyczna;	Tworzenie tablicy bajtów będącej pamięcią fizyczną w programie
private readonly List<StronaPamieci> _strony;	Lista stron pamięci (tzw. Tablica – tu lista – stron), która przechowuje informacje o stronie : jej AdresPoczatkowy, czy JestZmapowanaWPamieciFizycznej oraz czy JestNaDysku
private readonly Queue<int> _stronyZmapowane;	Nowa kolejka stron zmapowanych w pamięci fizycznej
private readonly PlikStronicowania _swap;	Obiekt typu PlikStronicowania, potrzebny do wykorzystania metod Zapisz i Odczytaj z klasy PlikStronicowania
private readonly bool[] _zajeteStronyFizyczne;	Tablica bajtów zajętych stron w pamięci przyjmująca wartość true, jeśli dana ramka jest zajęta oraz false,

	jeśli jest wolna
private BlokPamieci _zajeteBlokiPamieci;	Obiekt typu BlokPamieci umożliwiający z korzystania z informacji dotyczących bloków pamięci, potrzebny do zaalokowania odpowiedniej ilości bloków pamięci oraz zwolnienia bloków pamięci danego procesu

Konstruktor

```
public Pamieci(int rozmiar)
{
    _pamiecFizyczna = new byte[rozmiar];
    _zajeteStronyFizycznej = new bool[rozmiar/RozmiarBloku];
    _strony = new List<StronaPamieci>();
    _stronyZmapowane = new Queue<int>();
    _swap = new PlikStronicowania();
}
```

Metody

private int ZmapujStrone(int numerStrony)
{
 	
}
	Mapuje stronę do pamięci fizycznej, numerStrony – adres w pamięci fizycznej, w której strona została zmapowana.
	Znajdujemy wolną stronę w pamięci : if(_zajeteStronyFizycznej[i] == false) i zmieniamy zmienną adres = i*RozmiarBloku oraz ustawiamy _zajeteStronyFizycznej[i] na false. Jeśli nie znaleziono wolnej strony, swapujemy jedną na dysk, żeby zwolnić miejsce, wykorzystujemy tutaj metodę _swap.Zapisz(…), jeśli natomiast strona jest na dysku ściągamy ją do pamięci fizycznej z pliku stronicowania wykorzystując metodę _swap.Odczytaj(…). Jeśli

	jednak strona istnieje, ale nigdy wcześniej nie była używana mapujemy ją w pamięci i zerujemy : _pamiecFizyczna[adres + i] = 0. Na końcu dodajemy zmapowaną stronę do _stronyZmapowane, zmieniamy pole JestZmapowanaWPamieciFizycznej na true oraz przypisujemy adres do AdresPoczatkowy strony. Zwracamy adres strony, którą zmapowaliśmy w pamięci.

private int AdresWirtualnyNaFizyczny(int adresWirtualny)
{
 	
}
	Przelicza adres wirtualny na fizyczny. Jeśli ZmapujStrone jest true, a adres nie jest aktualnie zmapowany to spróbuje go zmapować (rzuca wyjątek, jeśli odpowiednia strona nie istnieje na dysku). Zwraca indeks w tablicy _pamiecFizyczna odpowiadający podanemu adresowi wirtualnemu lub -1, jeżeli podany adres jest błędny. Adres wirtualny traktujemy jako rozmiarStrony * numerStrony + offset wewnątrz strony do bajtu w pamięci.

public byte Odczytaj(int adresWirtualny)
{
 	
}
	Pozwala odczytać z pamięci fizycznej zapisane dane procesu. Zwraca _pamiecFizyczna[adresFizyczny] lub 0, gdy nastąpi błąd – próba odczytu spod niezaalokowanego adresu.

public void Zapisz(int adresWirtualny, byte wartosc)
{
 	
}
	Metoda pozwalająca zapisać do pamięci fizycznej : _pamiecFizyczna[adresFizyczny] = wartosc;

public int Zaalokuj(int iloscBlokow)
{
 	
}
	Alokuje pamięć metodą First-Fit. Zaalokowane strony nie są natychmiast mapowane w pamięci fizycznej .
	iloscBlokow – ilość bloków do zaalokowania, każdy blok ma rozmiar Pamiec.RozmiarBloku. Zwraca adres początku zaalokowanego bloku w

	pamięci wirtualnej.

public void Zwolnij(int adresPoczatkowy)
{
 	
}
	Zwalnia bloki danego procesu w pamięci poprzez podanie jego adresu początkowego.
	Szukamy strony zaczynającej się pod podanym adresem i zapisujemy blok doZwolnienia(pole typu BlokPamieci), jeśli dostaliśmy zły adres albo blok został już zwolniony nie robimy nic.
	Następnie zwalniamy bloki z doZwolnienia : ustawiamy pola tego bloku : JestZmapowanaWPamieciFizycznej oraz JestNaDysku na false. Strona pozostanie w swap i w pamięci fizycznej, ale zostanie wyczyszczona i nadpisana po następnym zapisie do niej.

public override string ToString()
{
 	
}
	Wyświetlamy:
	- listę zajętych bloków
	- strony pamięci (tablicę stron)
	- pamięć fizyczną (zapisane do niej dane/rozkazy)

Poziom 3 – Zarządzanie procesami

1. Klasa proces

W tej kasie znajduje się blok kontrolny procesu oraz dwa konstruktory. Klasa nie posiada metod.

Pola

public proces rodzic	pole przechowujące referencje na deskryptor rodzica
public int PID	pole przechowujące ID danego procesu
public List<proces> potomkowie	lista potomków procesu
public int stan_procesu	pole przechowujące stan procesu (nowy, gotowy, czekający,

	zakończony)
public uint[] rej_stalo	tablica 4 elementowa przechowujące rejestry stało-przecinkowe
public int[] priorytet	tablica 3 elementowa pod indeksem 0 całkowity priorytet procesu, 1 priorytet statyczny, 2 priorytet dynamiczny
public int użytkownik	pole przechowujące ID użytkownika
public int l_rozk	pole przechowujące licznik rozkazów
public int heapAddrStart	adres początkowy w pamięci dla tego procesu

Konstruktory

public proces(int pid)	konstruktor przyjmujący jako argument PID nowego procesu, reszta pól jest kopiowana z procesu rodzica a następnie zamieniana(lub nie w zależności co proces ma robić.)
public proces(int pid, int fpriorytet)	konstruktor przyjmujący jako argument PID nowego procesu i priorytet całkowity, reszta pól jest kopiowana z procesu rodzica i w razie konieczności zamieniana

2. Klasa ZarzadcaProcesow

Klasa ta obejmuje funkcje zarządzania procesem tworzenie usuwanie, jest odpowiedzialna ze wykonanie przez proces programu oraz zmianę jego stanu, przyporządkowanie do odpowiedniej kolejki

Pola

public Kolejki kolejka	**instancja klasy Kolejki**

Metody

public proces utworz(proces rodzic, int name, int priorytet)	metoda tworząca nowy proces. Jako argumenty przyjmuje referencje na rodzica, PID i priorytet. Na początku sprawdzane jest czy PID(name) jest juz w użyciu jeśli tak metoda nie pozwoli na stworzenie nowego procesu. Zwracany jest nowo utworzony proces i od razu dodawany do kolejki wszystkich i nowych procesów
public void utwProDziecko(int PPID, int PID, int fprio)	metoda tworząca nowy proces, ale jako argumenty przyjmuje PID rodzica, PID(ten proces) i priorytet. Metoda sprawdza poprawność podanego PID i dodaje proces do kolejki wszystkich i nowych procesów. Nie jest zwracane nic
public void exec(proces pr)	metoda odpowiedzialna za przełączanie procesu w stan gotowości i dodanie do kolejki gotowych(jeżeli wszytko co potrzebuje proces zostało już przydzielone). Jako argument przyjmuje proces.
private void zmiensta(proces pr, int sta)	metoda odpowiedzialna za zmianę stanu procesu, wywoływana zaraz po metodzie exec.
public void wypisz()	metoda wypisująca wszystkie stworzone procesy wraz z initem. Podaje także rodzica danego procesu.

public void usun_z_k(int fPID)	metoda usuwająca proces z kolejki wszystkich procesów w przypadku gdy zostanie on zakończony normalnie lub poleceniem kill. Jeżeli po wywołaniu operacji kill proces zostanie bez ojca, zostaje adoptowany przez INIT
public proces znajdz(int fPID)	metoda znajdująca w której kolejce i gdzie znajduje się proces o podanym PID. Zwraca proces.

3. Klasa Kolejki

W tej klasie stworzone są kolejki niezbędne do przechowywania procesów oraz metody wykonujące operacje na tych kolejkach.

Pola

public List<proces> k_wszystkich	kolejka przechowujące wszystkie procesy.
public List<proces> k_nowy	kolejka procesów nowo utworzonych
public List<proces> k_gotowy	kolejka procesów gotowych do wykonania przez procesor.
public List<proces> k_oczekuj	kolejka procesów oczekujących, np. na przydział pamięci

Metody

public proces wyslijproces()	wysyła pierwszy dostępny proces, z kolejki procesów gotowych, procesorowi.
public void usunZKol(int a)	metoda usuwająca proces z kolejki wszystkich. Jako argument przyjmuje miejsce procesu w kolejce, wywoływana w metodzie usun_z_k.
public void usunZKol2(int a)	metoda usuwająca proces z kolejki procesów gotowych, jako argument przyjmuje miejsce procesu w kolejce. Wywoływana po zakończeniu wykonywania procesu przez procesor

public void usunZKolOczekuj(int a)	metoda usuwająca proces z kolejki procesów oczekujących. Jako argument przyjmuje miejsce procesu w kolejce
public void dodajdoOczekuj(proces a)	metoda dodająca proces do kolejki procesów oczekujących. Jako argument przyjmuje proces
public void wypiszkolejki()	metoda wypisująca wszystkie kolejki(procesy które w nich się znajdują)

Poziom 4 – Zarządzanie pliki

1. Klasa blok

Klasa ta reprezentuje bloki, z których składa się dysk.

Pola

public byte[] bl;	tablica bajtów, w której przechowywane są dane
public int nast;	indeks na nastepny blok używany min. do zarządzania przestrzenią na dysku

Konstruktor

public blok()
{
bl = new byte[32];
nast = -1;
}
	tablica ma 32 bajty, a indeks na następny blok domyślnie ustawiony jest na -1 co oznacza, że blok jest pusty

Metody

public static void Wyswietblok(ref pom x, int nr, string slo)	Metoda wyswietla odpowiedni numer bloku o numerze nr dla pliku o nazwie slo
public static void wolnebloki(ref pom x, ref int o)	Metoda wyświetla indeksy wolnych bloków na dysku oraz ich ilość.

	Parametro to ilość zajętych bloków w danym momencie
public static void zajetebloki(ref pom x, ref int o)	Metoda wyświetla indeksy zajętych bloków na dysku oraz ich ilość
public static void Wyswietlbloklubdysk(ref pom x, int nr, string slowo)	Metoda wyświetla dowolny blok o podanym numerze lub cały dysk.

Parametr o typie pom to struktura, która posiada wszystkie ważne stuktury, na których pracuje czyli dysk, tablicę i-węzłów oraz listę list wpisów katalogowych :

```
struct pom
{
        public blok[] dysk;
        public iwezel[] tabi;
        public List<List<WpisK>> LK;
}
```

Inicjowane są w konstruktorze klasy iwezel :

```
x.dysk = new blok[64];
for (int i = 0; i < x.dysk.Length; i++)
{
        x.dysk.SetValue(new blok(), i);
}
x.tabki = new iwezel[20];
x.LK = new List<List<WpisK>>();
```

2. Klasa WpisK

Klasa ta odpowiada za wpis katalogowy dla każdego pliku.

Pola

public string nazwa;	nazwa pliku
public byte nriwezla;	nr i-węzła pliku czyli indeks w tablicy i-węzłów

Konstruktor

```
public WpisK(string str, byte wezel)
{
        nazwa = str;
        nriwezla = wezel;
}
```

Metody

public static void wyswk1(string nazwa, ref pom x)	wyswietla zawartosc jednego katalogu o podanej nazwie
public static void wyswk2(ref pom x)	wyswietla zawartosc wszystkich katalogow

3. Klasa iwezel

Klasa ta odpowiedzialna jest za reprezentację i-węzła.

Pola

public string IDuzytkownika;	identyfikator użytkownika
public string IDgrupy;	identyfikator grupy
public char Typ;	typ pliku np. k to katalog itd.
public byte prawa1;	prawa dostępu do pliku dla użytkownika
public byte prawa2;	prawa dostępu do pliku dla grupy
public byte prawa3;	prawa dostępu do pliku dla grupy
public int rozmiar;	rozmiar pliku w bajtach

public int pierwszyblok;	indeks 1-ego bloku zajmowanego przez plik
public int drugiblok;	indeks 2-ego bloku zajmowanego przez plik
public int blokindeksowy;	indeks bloku indeksowego dla pliku
public DateTime datautw;	data utworzenia pliku
static int licz = 0;	licznik mówiący o ilości utworzonych plików w danym czasie

Konstruktor

```
public iwezel(ref pom x, char typ)
        {
            if (licz == 0)
            {
                x.dysk = new blok[64];
                for (int i = 0; i < x.dysk.Length; i++)
                {
                    x.dysk.SetValue(new blok(), i);
                }
                x.tabi = new iwezel[20];
                x.LK = new List<List<WpisK>>();
            }

            IDuzytkownika = "0";
            IDgrupy = "0";
            Typ = typ;
            prawa1 = 7;
            prawa2 = 3;
            prawa3 = 3;
            rozmiar = 0;
            pierwszyblok = -1;
            drugiblok = -1;
            blokindeksowy =-1;
            datautw = DateTime.Now;
            x.tabi[licz] = this;
        }
```

Podczas tworzenia katalogu Root inicjujemy dysk składający się z 64 bloków, tablicę i-węzłów, która może mieć max. 20 elementów oraz listę list na wpisy katalogowe. Inicjuje również pola dla każdego i-węzła domyślnymi wartościami. Typ nadaję podczas tworzenia pliku. Użytkownik posiada wszystkie prawa natomiast reszta prawa zapisu i odczytu.

Metody

public static void Wyswtabi(ref pom x)	wyswietlenie przykladowych pól dla tablicy iwęzłów
public static int wolnemiejsce(ref int o, ref pom x)	funkcja służąca do losowania kolejnych bloków dla pliku
public static int czyistnieje(ref pom x, string naz)	funkcja zwwraca nr i-wezla dla pliku w tablicy iwęzłów
public static int wyswzajblok(ref pom x, string naz)	funkcja wyświetla indeksy zajętych bloków na dysku przez plik
public static List<int> listazblok(ref pom x, int indeks)	zwraca liste z zajetymi blokami z której korzystam do odczytania danych z pliku
public static int open(ref pom x, string naz, flaga l, ref int o, ref string Out)	Out-zmienna potrzebna do zapisu odczytanego pliku l-flaga decyduje czy chcemy coś odczytać czy coś zapisać, funkcja służąca do odczytu lub zapisu
public static int create(ref pom x, string nazwa, char typ, string nazwfold)	funkcja służąca do utworzenia pliku o typie: "typ", o nazwie: "nazwa" w katalogu o nazwie: "nazwfold"
public static int delete(ref pom x, string nazwa, ref int o)	funkcja służąca do usuwania pliku
public static int chname(ref pom x, string co, string naco)	funkcja zmienia nazwę pliku z "co" na "naco"
public static int chacces(ref pom x, string nazwa, byte pd1, byte pd2, byte pd3)	funkcja służąca do zmiany praw dostępu do pliku
public static int chidu(ref pom x, string nazwa, string idu)	funkcja służąca do zmiany UID
public static int chidg(ref pom x, string nazwa, string idg)	funkcja służąca do zmiany GID

Poziom 5 – Zarządzanie kontami użytkowników i uprawnieniami

Jednym z podstawowych składników systemu zabezpieczeń są konta użytkowników. Moja warstwa umożliwia dodawanie i usuwanie użytkowników. Umożliwia zarządzanie kontami użytkowników. W tej sekcji zostały zestawione poszczególne elementy implementacji, służące zarządzaniu kontami użytkowników.

1. Klasa Uruchomienie

Jej jedynym celem jest sprawdzenie czy system jest uruchamiany po raz pierwszy.

Składnia

```
class Uruchomienie
{
  …
}
```

2. Klasa Logowanie

Składnia

```
class Logowanie
{
  …
}
```

Pola

string login, haslo;	Pola przechowujące nazwę i hasło wpisane podczas logowanie
string sciezka	Przechowuje ścieżkę do pliku tekstowego z wszystkimi nazwami użytkownika i hasłami

Metody

public void Log()	Pobiera od użytkownika login i hasło.
public int spr()	Sprawdza czy podane login i hasło są poprawne.

3. Klasa Administrator

Jest tworzona gdy do systemu logujemy się jako administrator.
Jest to interfejs użytkownika łączący pozostałe warstwy.

Składnia

```
class Admistrator
{
  …
}
```

Pola

string login, haslo;	Pola przechowujące nazwę i hasło.
pom p;	Obiekt przechowujący dysk systemu.
string slowo;	Zmienna przechowująca zawartość ostatnio otwieranego pliku.
string aktualnykatalog;	Zmienna przechowująca nazwę katalogu w którym użytkownik aktualnie się znajduje.
ZarzadcaProcesow zarzadcapr;	Obiekt zarządzający procesami.
int ile;	Zmienna przechowująca aktualną ilość procesów w kolejce procesów gotowych.

Pamiec pam;	Obiekt przechowujący pamięć operacyjną systemu.
proces init;	Tworzy proces inicjujący.
public string sciezka	Przechowuje ścieżkę do pliku tekstowego z wszystkimi nazwami użytkownika i hasłami.
public string grupa	Przechowuje ścieżkę do pliku tekstowego z wszystkimi grupami.

Konstruktor

public Admin(string log, string has,pom x, int ilo, string fff, ZarzadcaProcesow zzz, int il, Pamiec papa, proces ini)	Przypisuje wszystkim zmiennym w klasie ich odpowiedniki zainicjowane globalnie.

Metody

public int exit()	Powoduje zamknięcie systemu.
public void adduse(string n, string h, string gru)	Pozwala dodać nowego użytkownika.
public int removeuse(string n)	Pozwala usunąć istniejącego użytkownika.
public void dodajgru(string n)	Pozwala dodać nową grupę.
public int removegru(string n)	Pozwala usunąć istniejącą grupę.
public int wylogowanie()	Umożliwia wylogowanie z systemu.
public int polecenia()	Odczytuje wpisane przez użytkownika komendy i uruchamia odpowiednią funkcje.

4. Klasa Uzytkownik

Jest tworzona gdy do systemu logujemy się jako zwykły użytkownik.
Jest to interfejs użytkownika łączący pozostałe warstwy.

Składnia

```
class Uzytkownik
{
  …
}
```

Pola

string login, haslo;	Pola przechowujące nazwę i hasło.
string gru;	Pole przechowujące nazwę grupy danego użytkownika.
pom p;	Obiekt przechowujący dysk systemu.
string slowo;	Zmienna przechowująca zawartość ostatnio otwieranego pliku.
string aktualnykatalog;	Zmienna przechowująca nazwę katalogu w którym użytkownik aktualnie się znajduje.
ZarzadcaProcesow zarzadcapr;	Obiekt zarządzający procesami.
int ile;	Zmienna przechowująca aktualną ilość procesów w kolejce procesów gotowych.
Pamiec pam;	Obiekt przechowujący pamięć operacyjną systemu.
proces init;	Tworzy proces inicjujący.
public string sciezka	Przechowuje ścieżkę do pliku tekstowego z wszystkimi nazwami użytkownika i hasłami.

Konstruktor

public Uzytkownik(string log, string has, pom x, int ilo, string fff, ZarzadcaProcesow zzz, int il, Pamiec papa, proces ini)	Przypisuje wszystkim zmiennym w klasie ich odpowiedniki zainicjowane globalnie.

Metody

public int exit()	Powoduje zamknięcie systemu.
public int wylogowanie()	Umożliwia wylogowanie z systemu.
public void zmhaslo()	Umożliwia zmianę hasła użytkownikowi.
public int polecenia()	Odczytuje wpisane przez użytkownika komendy i uruchamia odpowiednią funkcje.

Poziom 6 – Komunikacja międzyprocesowa

Zadaniem tej warstwy systemu jest zapewnienie komunikacji międzyprocesowej za pomocą strumieni. W systemie Unix występują dwa typy łączy: łącza nienazwane (potoki) oraz łącza nazwane. Obydwa rodzaje łączy zostały zrealizowane w trybie jednokierunkowym. W tej sekcji zostały zestawione elementy implementacji odpowiedzialnej za komunikację między procesami.

1. Klasa Lacze

Klasa opisująca łącze nazwane.

Składnia

```
public class Lacze
{
  …
}
```

Pola

public string name;	Pole przechowujące nazwę łącza postaci 1-2 (dla łącza pomiędzy procesem o PID= 1 a łączem o PID=2.
public int source;	Pole przechowujące PID procesu piszącego do łącza.

public int destination;	Pole przechowujące PID procesu czytającego z łącza.
public string bufor;	Bufor, do którego zapisywany jest wysyłany komunikat/rozkaz.

2. Klasa Lacza

Klasa zawierająca metody obsługi komunikacji międzyprocesowej.

Pola

public List<String>ListaLaczy = new List<String>();	Lista przechowująca nazwy obiektów klasy Lacze reprezentujących łącza nazwane. Wykorzystywane do metod wypisywania i usuwania łączy nazwanych.

Metody

public void UtworzLacze(proces p1, proces p2)	Metoda tworząca łącze nazwane pomiędzy procesami (komunikacja jednokierunkowa). Za argumenty metoda przyjmuje dwa procesy, które identyfikowane są przez PIDy podane przez użytkownika.Metoda tworzy nowy obiekt klasy Lacze o nazwie złożonej z PIDów procesów oddzielonych znakiem '-' (np. 1-2), źródłem jest proces p1, celem proces p2. Przy tworzeniu łącza bufor pozostaje pusty.
public void UsunLacze(string lacze)	Usuwa łącze nazwane o nazwie podanej przez użytkownika (postaci 1-2).
public void CommunicationNamed(proces p1, proces p2, string rozkaz)	Metoda realizuję komunikację nazwaną- przekazuję rozkaz postaci string wczytany przez użytkownika pomiędzy procesami p1 i p2, których PIDy wczytywane są również przez użytkownika.

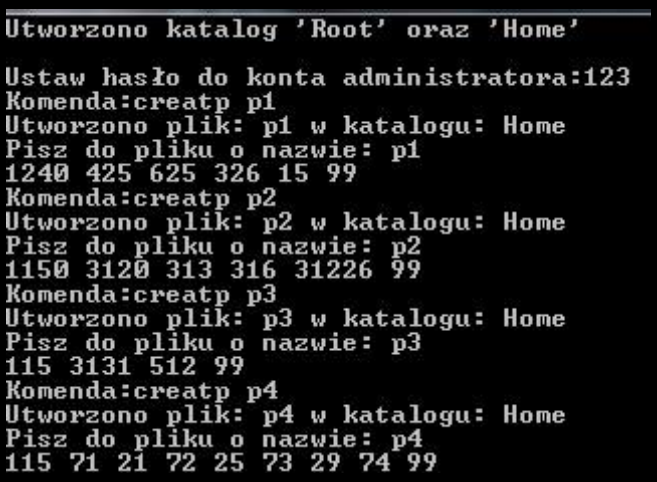
public void przekazrozkaz(proces a, proces b, int ktory)	Metoda przekazująca rozkaz pomiędzy procesami a i b. Realizowana jest zarówno za pomocą komunikacji nienazwanej jak i nazwanej.
public bool pokrewienstwo(proces p1, proces p2)	Funkcja sprawdzająca pokrewieństwo pomiędzy dwoma procesami. Zwraca true jeśli pokrewieństwo istnieje, w innym przypadku zwraca false.
public void CommunicationNoNamed(proces p1, proces p2, int ktory)	Metoda realizuję komunikację nienazwaną, pomiędzy procesami o pokrewieństwie ojciec-syn, syn-ojciec.
public void ClearOrder(proces p, string r)	Metoda usuwająca rozkaz z listy rozkazów procesu p1.
public void wypiszlacza()	Metoda wypisuję wszystkie dostępne łącza nazwane (ListaLaczy);

Klasa: public class proces z warstwy III została rozszerzona o listę przechowującą rozkazy: public List<String> rozkazy = new List<String>();

W metodach testujących użyję również funkcji klasy ZarzadcaProcesow: public void wypisz2(), która wypisuję listę wszystkich procesów oraz listy przesłanych do nich rozkazów/komunikatów.

III. METODY TESTUJĄCE

Symulacja działania systemu rozpoczyna się od ustawienia hasła użytkownika do konta administrator. Następnie w teście systemy stworzone zostały 4 pliki komendą *creatp nazwa_pliku* wraz z rozkazami (danymi), które do nich wpisano.



W kolejnym kroku komendą *proces nazwa_pliku* stworzono 4 procesy odpowiadające 4 różnym plikom, które stworzyliśmy wcześniej. Podczas tworzenia procesu podajemy PPID (PID rodzica procesu), PID (procesu) oraz priorytet nowo stworzonego procesu. Stworzone procesy :

PPID	PID	priorytet
1	2	12
1	3	5
1	4	12
1	5	12

Jak widać na poniższych zrzutach ekranu każdy blok początkowo zajmuje 3 bloki pamięci (tj. 16 bajtów * 3 = 48 bajtów). Lista bloków pamięci przedstawia adres początkowy oraz adres końcowy pamięci, którą zajmuje proces, a także długość w bajtach. Następnie widać tablicę stron, na której widać stworzone na potrzeby procesu strony, a wyświetlane parametry to : adres strony, czy jest ona zmapowana w pamięci fizycznej oraz czy jest na dysku (czy został stworzony plik stronicowania tej strony). Ostatnim wyświetlanym elementem modułu pamięci jest pamięć fizyczna, która jest tutaj przestawiona w postaci tablicy.

Rozkazy w wyświetlanej pamięci fizycznej różnią się od tych wpisanych do pliku, ponieważ podczas zapisu do pamięci następuje zamiana string na byte i w tablicy też możemy zobaczyć rozkazy w bajtach. Gdzie kończy się dany rozkaz można zauważyć po rozkazie 99, który ma tutaj nadal taką samą postać.

```
Komenda:proces p1
podaj PPID, PID i priorytet
1
2
12
Proces dodano
Podaj ile bloków powinien posiadac proces
Lista zajetych blokow:
Od:        0     Do:       48    Dlugosc:    48 bajtow
Strony pamieci:
Adres:         0 Zmapowany: True Na dysku: False
Adres:        16 Zmapowany: True Na dysku: False
Adres:         0 Zmapowany: False Na dysku: False
Pamiec fizyczna:
216 004 000 000 169 001 000 000 113 002 000 000 070 001 000 000
015 000 000 000 099 000 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
```

```
Komenda:proces p2
podaj PPID, PID i priorytet
1
3
12
Proces dodano
Podaj ile bloków powinien posiadac proces
Lista zajetych blokow:
Od:        0     Do:       48    Dlugosc:    48 bajtow
Od:       48     Do:       96    Dlugosc:    48 bajtow
Strony pamieci:
Adres:         0 Zmapowany: True Na dysku: False
Adres:        16 Zmapowany: True Na dysku: False
Adres:         0 Zmapowany: False Na dysku: False
Adres:        32 Zmapowany: True Na dysku: False
Adres:        48 Zmapowany: True Na dysku: False
Adres:         0 Zmapowany: False Na dysku: False
Pamiec fizyczna:
216 004 000 000 169 001 000 000 113 002 000 000 070 001 000 000
015 000 000 000 099 000 000 000 000 000 000 000 000 000 000 000
126 004 000 000 048 012 000 000 057 001 000 000 060 001 000 000
250 121 000 000 099 000 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
```

```
Komenda:proces p3
podaj PPID, PID i priorytet
1
4
5
Proces dodano
Podaj ile bloków powinien posiadac proces
Lista zajetych blokow:
Od:        0     Do:       48    Dlugosc:    48 bajtow
Od:       48     Do:       96    Dlugosc:    48 bajtow
Od:       96     Do:      144    Dlugosc:    48 bajtow
Strony pamieci:
Adres:         0 Zmapowany: True Na dysku: False
Adres:        16 Zmapowany: True Na dysku: False
Adres:         0 Zmapowany: False Na dysku: False
Adres:        32 Zmapowany: True Na dysku: False
Adres:        48 Zmapowany: True Na dysku: False
Adres:         0 Zmapowany: False Na dysku: False
Adres:        64 Zmapowany: True Na dysku: False
Adres:         0 Zmapowany: False Na dysku: False
Adres:         0 Zmapowany: False Na dysku: False
Pamiec fizyczna:
216 004 000 000 169 001 000 000 113 002 000 000 070 001 000 000
015 000 000 000 099 000 000 000 000 000 000 000 000 000 000 000
126 004 000 000 048 012 000 000 057 001 000 000 060 001 000 000
250 121 000 000 099 000 000 000 000 000 000 000 000 000 000 000
115 000 000 000 059 012 000 000 000 002 000 000 099 000 000 000
000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
```

```
Komenda:proces p4
podaj PPID, PID i priorytet
1
5
12
Proces dodano
Podaj ile bloków powinien posiadac proces
Lista zajetych blokow:
Od:        0     Do:       48    Dlugosc:    48 bajtow
Od:       48     Do:       96    Dlugosc:    48 bajtow
Od:       96     Do:      144    Dlugosc:    48 bajtow
Od:      144     Do:      192    Dlugosc:    48 bajtow
Strony pamieci:
Adres:         0 Zmapowany: True Na dysku: False
Adres:        16 Zmapowany: True Na dysku: False
Adres:         0 Zmapowany: False Na dysku: False
Adres:        32 Zmapowany: True Na dysku: False
Adres:        48 Zmapowany: True Na dysku: False
Adres:         0 Zmapowany: False Na dysku: False
Adres:        64 Zmapowany: True Na dysku: False
Adres:         0 Zmapowany: False Na dysku: False
Adres:         0 Zmapowany: False Na dysku: False
Adres:        80 Zmapowany: True Na dysku: False
Adres:        96 Zmapowany: True Na dysku: False
Adres:       112 Zmapowany: True Na dysku: False
Pamiec fizyczna:
216 004 000 000 169 001 000 000 113 002 000 000 070 001 000 000
015 000 000 000 099 000 000 000 000 000 000 000 000 000 000 000
126 004 000 000 048 012 000 000 057 001 000 000 060 001 000 000
250 121 000 000 099 000 000 000 000 000 000 000 000 000 000 000
115 000 000 000 059 012 000 000 000 002 000 000 099 000 000 000
115 000 000 000 071 000 000 000 021 000 000 000 072 000 000 000
025 000 000 000 073 000 000 000 029 000 000 000 074 000 000 000
099 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
```

Następnie rozpoczyna się właściwa symulacja procesora. Na starcie zostaje dodany tylko pierwszy proces P2 o priorytecie 12. Przy każdym następnym cyklu procesora zostaje dodany nowy proces do odpowiedniej kolejki priorytetowej. Co 4 wykonane rozkazy (procesu) wywłaszczony zostaje aktualnie wykonany proces. Następnie przeliczany priorytet ze wzoru (prio = baza + cpu/2) i wybierany kolejny proces do przetwarzania. Mechanizm jest cykliczny dla każdego procesu. Zgodnie ze wzorem (prio = cpu /2) co 15 taktów procesora priorytet jest przeliczany dla wszystkich procesów. Jest to współczynnik zaniku. Jeżeli w momencie dodania nowego procesu ma on priorytet wyższy niż aktualnie wykonywany przez procesor to ten wykonywany zostaje wywłaszczony, zostaje mu przeliczony priorytet i dodany do odpowiedniej kolejki priorytetowej procesów gotowych. Natomiast ten „nowo dodany" zostaje przetwarzany przez procesor.

Schemat budowy symulacji procesora:

- Po lewej stronie informacja o numerze kolejki priorytetowej procesów
- Obok niej umieszczony jest wektor binarny który informuje o tym czy dana kolejka jest pusta (ustawiony na 0) lub zawiera jakieś procesy (ustawiony na 1)
- Przy każdym wykonaniu rozkazu danego procesu pojawia się informacja o tym jaki rozkaz został wykonany, o jakim argumencie, dla jakiego procesu, stany rejestrów (R1 – rejestr pierwszy, R2 – rejestr drugi, R3 – rejestr trzeci, R4 – rejestr czwarty) a także licznik rozkazów

Zostaje wykonany pierwszy rozkaz: **1240** - Wpisanie do rejestru R2 liczby **40**

```
Komenda:procesor

Start Symulacji procesora!
(K0) 0:
(K1) 0:
(K2) 0:
(K3) 1: P:2 Prio:12
(K4) 0:
(K5) 0:
(K6) 0:
(K7) 0:

Wykonywany rozkaz to: 12 o argumencie: 40
Wkonywany proces P:2 Stan R1: 0 R2: 40 R3: 0 R4: 0 L. ROZK: 1
```

Po pierwszym wykonaniu rozkazu zostaje dodany proces P3 o priorytecie 12. Ma on ten sam priorytet co proces P2 więc zostaje dodany do odpowiedniej kolejki priorytetowej procesów gotowych.

Kolejny takt procesora to wykonanie rozkazu: **425** – Odejmowanie liczby o wartości 5 z rejestru R2

Po drugim takcie procesora zostaje dodany proces P4 o priorytecie 5. Więc aktualnie wykonywany proces P2 zostaje wywłaszczony, przeliczony zostanie mu priorytet i dodany do odpowiedniej kolejki priorytetowej. W 3 takcie procesora wykonany zostaje pierwszy rozkaz procesu P4, ponieważ ma on najmniejszy priorytet. Obrazuje to screen poniżej.

```
Dodano nowy proces do kolejki priorytetowej!

Wykonywany rozkaz to: 42 o argumencie: 5
Wkonywany proces P:2 Stan R1: 0 R2: 35 R3: 0 R4: 0 L. ROZK: 2


Dodano nowy proces do kolejki priorytetowej!

(K0) 0:
(K1) 1: P:4 Prio:5
(K2) 0:
(K3) 1: P:3 Prio:12     P:2 Prio:13
(K4) 0:
(K5) 0:
(K6) 0:
(K7) 0:

Wykonywany rozkaz to: 11 o argumencie: 5
Wkonywany proces P:4 Stan R1: 5 R2: 0 R3: 0 R4: 0 L. ROZK: 1
```

Po 3 takcie procesora zostaje dodany kolejny proces P5 o priorytecie 12 i dodany na końcu kolejki priorytetowej K3. Proces P5 ma priorytet niższy niż proces P4, więc nie będzie wywłaszczenia.

Wykonywane rozkazy dla procesu P4 to:
115 - Wpisanie liczby o wartości 5 do rejestru R1
3131 - Dodawanie liczby o wartości 31 do rejestru R1

Ponieważ żaden nowy proces nie zostanie dodany, więc kolejne 2 rozkazy zostają wykonane również dla procesu P4 (ma on największy priorytet).

Wykonywane rozkazy to:
512 - Mnożenie podanej liczby o wartości 2 przez wartość w rejestrze R2 i wpisanie jej do rejestru R2
99 - Zakończenie programu

Proces P4 został zakończony (a wraz z jego zakończeniem została zwolniona pamięć, co widać na liście zajętych bloków pamięci), więc jako następny do przetwarzania przez procesor zostaje wybrany ten, o najwyższym priorytecie. W tym przypadku będzie to proces P2 o priorytecie 13. Pomimo tego że ma on liczbowo priorytet wyższy (mniej ważny) to w obrębie danej kolejki procesy wykonywane są kolejno przydzielając im kwant czasu procesora, bez względu na indywidualny priorytet procesu.

```
Dodano nowy proces do kolejki priorytetowej!

Wykonywany rozkaz to: 31 o argumencie: 31
Wkonywany proces P:4 Stan R1: 36 R2: 0 R3: 0 R4: 0 L. ROZK: 2


Wykonywany rozkaz to: 51 o argumencie: 2
Wkonywany proces P:4 Stan R1: 72 R2: 0 R3: 0 R4: 0 L. ROZK: 3


Wykonywany rozkaz to: 99 o argumencie: 0
Wkonywany proces P:4 Stan R1: 72 R2: 0 R3: 0 R4: 0 L. ROZK: 4


Lista zajetych blokow:
Od:         0     Do:        48    Dlugosc:    48 bajtow
Od:        48     Do:        96    Dlugosc:    48 bajtow
Od:       144     Do:       192    Dlugosc:    48 bajtow
(K0) 0:
(K1) 0:
(K2) 0:
(K3) 1: P:2 Prio:13     P:5 Prio:12     P:3 Prio:12
(K4) 0:
(K5) 0:
(K6) 0:
(K7) 0:

Wykonywany rozkaz to: 62 o argumencie: 5
Wkonywany proces P:2 Stan R1: 0 R2: 7 R3: 0 R4: 0 L. ROZK: 3
```

Procesor przetwarza kolejny rozkazy procesu P2.
Są to odpowiednio:
625 – Dzielenie liczby o wartości 35 z rejestru R2 przez 5 i wpisanie jej do

rejestru R2
326 – Dodawanie liczby o wartości **6** do rejestru R2

Mijają 4 takty procesora dla procesu P2, więc zostaje on wywłaszczony i umieszczony na końcu odpowiedniej kolejki priorytetowej według wcześniej obliczonego priorytetu. Poszukiwany jest kolejny proces z najwyższym priorytetem. Będzie to proces P5 o priorytecie 12.

Ponieważ wszystkie pozostałe 3 procesy występują w tej samej kolejce to otrzymują one po 4 takty procesora. Widoczne to będzie poniżej.

Wykonywany jest proces P5. Procesor wykonuje na nim 4 rozkazy :
115 - Wpisanie liczby o wartości **5** do rejestru R1
71 - Zwiększenie o 1 rejestru R1
21 - Przeniesienie wartości z rejestru R1 do R2
72 - Zwiększenie o 1 rejestru R2

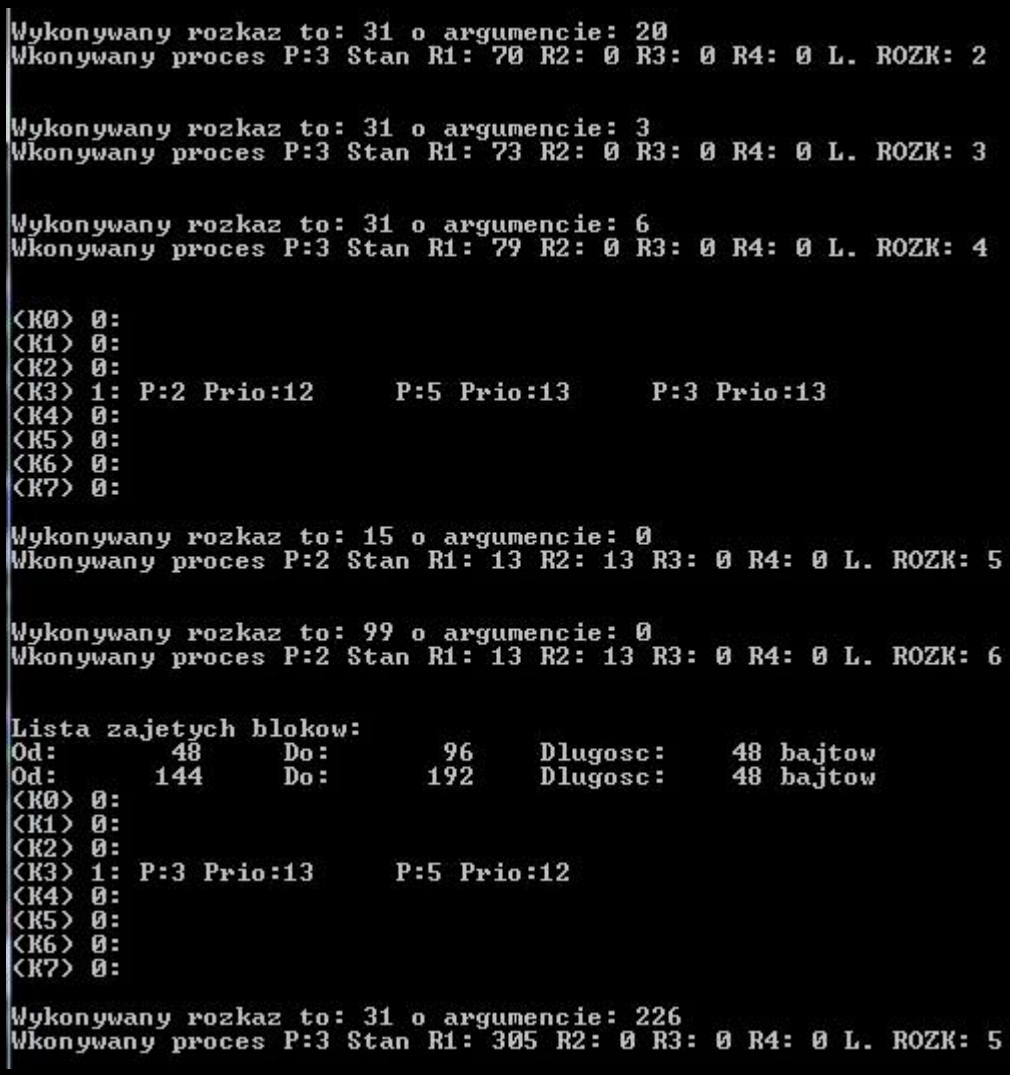
Mijają 4 takty procesora dla procesu P5, więc zostaje on wywłaszczony i umieszczony na końcu odpowiedniej kolejki priorytetowej według wcześniej obliczonego priorytetu. Poszukiwany jest kolejny proces z najwyższym priorytetem. Będzie to proces P3 o priorytecie 12.

```
Wykonywany rozkaz to: 32 o argumencie: 6
Wkonywany proces P:2 Stan R1: 0 R2: 13 R3: 0 R4: 0 L. ROZK: 4


(K0) 0:
(K1) 0:
(K2) 0:
(K3) 1: P:5 Prio:12     P:3 Prio:12     P:2 Prio:13
(K4) 0:
(K5) 0:
(K6) 0:
(K7) 0:

Wykonywany rozkaz to: 11 o argumencie: 5
Wkonywany proces P:5 Stan R1: 5 R2: 0 R3: 0 R4: 0 L. ROZK: 1


Wykonywany rozkaz to: 71 o argumencie: 0
Wkonywany proces P:5 Stan R1: 6 R2: 0 R3: 0 R4: 0 L. ROZK: 2


Wykonywany rozkaz to: 21 o argumencie: 0
Wkonywany proces P:5 Stan R1: 6 R2: 6 R3: 0 R4: 0 L. ROZK: 3


Wykonywany rozkaz to: 72 o argumencie: 0
Wkonywany proces P:5 Stan R1: 6 R2: 7 R3: 0 R4: 0 L. ROZK: 4


(K0) 0:
(K1) 0:
(K2) 0:
(K3) 1: P:3 Prio:12     P:2 Prio:13     P:5 Prio:14
(K4) 0:
(K5) 0:
(K6) 0:
(K7) 0:

Wykonywany rozkaz to: 11 o argumencie: 50
Wkonywany proces P:3 Stan R1: 50 R2: 0 R3: 0 R4: 0 L. ROZK: 1
```

Wykonywany jest proces P2. Procesor wykonuje na nim 4 rozkazy :
1150 – Wpisanie liczby o wartości **50** do rejestru R1
3120 – Dodawanie liczby o wartości **20** do rejestru R1
313 - Dodawanie liczby o wartości **3** do rejestru R1
316 - Dodawanie liczby o wartości **6** do rejestru R1

Mijają 4 takty procesora dla procesu P3, więc zostaje on wywłaszczony i umieszczony na końcu kolejki priorytetowej według wcześniej obliczonego priorytetu. Jest to kolejka oznaczona jako K3. Poszukiwany jest kolejny proces z najwyższym priorytetem. Będzie to proces P2 o priorytecie 12.

Procesor wykonuje tylko 2 rozkazy procesu P1 po czym zostaje on zakończony. Wykonane rozkazy to:
15 - Przeniesienie wartości z rejestru R2 do R1
99 – Zakończenie programu



Kolejny wybrany proces to P3 o priorytecie 13. Jak w poprzednim przypadku procesor wykonuje na nim tylko 2 rozkazy. Następnie zostaje on zakończony.

Wykonane rozkazy to:
31226 - Dodawanie liczby o wartości 226 do rejestru R1
99 – Zakończenie programu

Pozostał już tylko proces P5. Procesor wykonuje na nim kolejne 4 rozkazy:
25 - Przeniesienie wartości z rejestru R2 do R3
73 - Zwiększenie o 1 rejestru R3
29 - Przeniesienie wartości z rejestru R3 do R4
74 - Zwiększenie o 1 rejestru R4

Procesowi P5 zostaje odebrany procesor. Trafia on do odpowiedniej kolejki priorytetowej przy wcześniejszym ustaleniu priorytetu. Nie ma żadnego innego procesu o priorytecie wyższym więc zostaje on ponownie przekazany do przetwarzania przez procesor.

```
Wykonywany rozkaz to: 99 o argumencie: 0
Wkonywany proces P:3 Stan R1: 305 R2: 0 R3: 0 R4: 0 L. ROZK: 6


Lista zajetych blokow:
Od:      144     Do:      192    Dlugosc:    48 bajtow
(K0) 0:
(K1) 0:
(K2) 0:
(K3) 1: P:5 Prio:12
(K4) 0:
(K5) 0:
(K6) 0:
(K7) 0:

Wykonywany rozkaz to: 25 o argumencie: 0
Wkonywany proces P:5 Stan R1: 6 R2: 7 R3: 7 R4: 0 L. ROZK: 5


Wykonywany rozkaz to: 73 o argumencie: 0
Wkonywany proces P:5 Stan R1: 6 R2: 7 R3: 8 R4: 0 L. ROZK: 6


Wykonywany rozkaz to: 29 o argumencie: 0
Wkonywany proces P:5 Stan R1: 6 R2: 7 R3: 8 R4: 8 L. ROZK: 7


Wykonywany rozkaz to: 74 o argumencie: 0
Wkonywany proces P:5 Stan R1: 6 R2: 7 R3: 8 R4: 9 L. ROZK: 8
```

Ostatni rozkaz procesu P5: **99** - Zakończenie programu.

W kolejce priorytetowej procesów nie ma już żadnych programów do wykonywania przez procesor. Wszystkie zostały wykonane i usunięte z pamięci. Symulacja procesora zostaje zakończona. Sumaryczna ilość taktów procesora wynosi 25.

```
(K0) 0:
(K1) 0:
(K2) 0:
(K3) 1: P:5 Prio:14
(K4) 0:
(K5) 0:
(K6) 0:
(K7) 0:

Wykonywany rozkaz to: 99 o argumencie: 0
Wkonywany proces P:5 Stan R1: 6 R2: 7 R3: 8 R4: 9 L. ROZK: 9


Lista zajetych blokow:
Ilosc taktow procesora: 25
Wszystkie procesy zostaly wykonane!
```

Stany rejestrów :

Teoretycznie – przed wykonaniem symulacji	**Praktycznie – po wykonaniu symulacji**
R1 = 13, R2 = 13, R3 = 0, R4 = 0	**R1** = 13, **R2** = 13, **R3** = 0, **R4** = 0
R1 = 305, R2 = 0, R3 = 0, R4 = 0	**R1** = 305, **R2** = 0, **R3** = 0, **R4** = 0
R1 = 72, R2 = 0, R3 = 0, R4 = 0	**R1** = 72, **R2** = 0, **R3** = 0, **R4** = 0

R1 = 6, R2 = 7, R3 = 8, R4 = 9	**R1** = 6, **R2** = 7, **R3** = 8, **R4** = 9

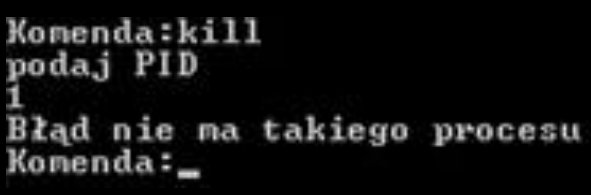
Teoretyczne stany rejestrów zgadzają się z praktycznymi, wykonanymi przez symulacje procesora. Można więc przyjąć, że zarówno algorytm karuzelowo-priorytetowy, interpreter i cała symulacja procesora działa poprawnie.

Następnie bardziej szczegółowo testowana była warstwa III (zarządzania procesami) i jej komendy.

Metoda polega na utworzeniu 5 procesów z czego rodzicem pierwszych dwóch będzie INIT procesu 3 proces 2, procesu 4 proces 3 a procesu 5 proces 4. Każdy proces dostanie unikatowy priorytet. Po zabiciu procesu 3 jego syn zostanie zaadoptowany przez proces INIT.
Operacją 'fork' tworzymy proces podając odpowiednio PID, PPID i priorytet.
Operacją 'kill' usuwamy proces podając PID.
Operacją 'wy' wyświetlamy aktualne procesy.

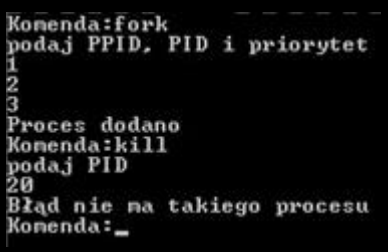
```
Komenda:fork
podaj PPID, PID i priorytet
1
2
10
Proces dodano
Komenda:fork
podaj PPID, PID i priorytet
1
3
12
Proces dodano
Komenda:fork
podaj PPID, PID i priorytet
2
4
20
Proces dodano
Komenda:fork
podaj PPID, PID i priorytet
3
5
17
Proces dodano
Komenda:fork
podaj PPID, PID i priorytet
4
6
1
Proces dodano
Komenda:wy
Proces PID< INIT >: 1
Proces PID: 2  priorytet:10 Rodzic PPID: 1
Proces PID: 3  priorytet:12 Rodzic PPID: 1
Proces PID: 4  priorytet:20 Rodzic PPID: 2
Proces PID: 5  priorytet:17 Rodzic PPID: 3
Proces PID: 6  priorytet:1 Rodzic PPID: 4
Komenda:kill
podaj PID
3
Komenda:wy
Proces PID< INIT >: 1
Proces PID: 2  priorytet:10 Rodzic PPID: 1
Proces PID: 4  priorytet:20 Rodzic PPID: 2
Proces PID: 5  priorytet:17 Rodzic PPID: 1
Proces PID: 6  priorytet:1 Rodzic PPID: 4
Komenda:
```

Metoda polega na usunięciu procesu systemowego INIT. System pokaże błąd, że taki proces nie istnieje, ponieważ nie jest on widoczny dla użytkownika. INIT ma PID =1.



Metoda polega na stworzeniu procesu o PID który już istnieje. Dwukrotnie podany PID=2, system pokaże błąd istnienia procesu o podanym PID

Metoda polegająca na usunięciu procesu który nie istnieje. System pokaże błąd nie istnienia takiego procesu



Następnie bardziej szczegółowo testowana była warstwa VI (komunikacja międzyprocesowa) i jej komendy.

Do przetestowania komunikacji nazwanej posłużą nam 4 przykładowe procesy. Komenda „wy” wypisuje na ekran konsoli wszystkie stworzone procesy oraz ich listy komunikatów.

```
Komenda:wy
Proces PID( INIT ): 1           Kom:
Proces PID: 2    Rodzic PPID: 1 Kom:
Proces PID: 3    Rodzic PPID: 1 Kom:
Proces PID: 4    Rodzic PPID: 3 Kom:
Komenda:
```

Wpisując komendę „com” otwieramy uproszczony panel sterowania komunikacji międzyprocesowej:

```
Komenda:com
Wybierz rodzaj Łącza
N -> Utwórz Łącze Nazwane
1 -> Komunikacja Nazwana
2 -> Komunikacja Nienazwana
3 -> Wypisz Dostepne Lacza
R -> Usuń Łącze Nazwane
P -> Przekaż rozkaz
W -> Wyczyść rozkaz
```

Najpierw przetestujemy komunikację nazwaną. Żeby było to możliwe musimy najpierw utworzyć łącze:

```
n
Podaj PID procesu nadawcy
2
Podaj PID procesu odbiorcy
3
Utworzono Lacze
```

Pomiędzy procesami o PID 2 i 3 zostało utworzone łącze o nazwie „2-3”. Łącze istnieje w systemie w postaci pliku- funkcja UtworzLacze tworzy nowy obiekt klasy Lacze, serializuję go oraz zapisuję na dysku w postaci pliku. Funkcja dodaje także do listy łączy jego nazwę. Stworzymy jeszcze dwa inne łącza:

```
n
Podaj PID procesu nadawcy
1
Podaj PID procesu odbiorcy
3
Utworzono Lacze
```

```
n
Podaj PID procesu nadawcy
4
Podaj PID procesu odbiorcy
1
Utworzono Lacze
```

Komenda „3” w panelu sterowania komunikacją pozwoli nam zobaczyć wszystkie dostępne łącza:

```
3

 Utworzone Łącza:

2-3  1-3  4-1
```

Możemy także usunąć niechciane łącze komendą „R”:

```
r
Podaj nazwe lacza postaci 1-2
4-1
Usunięto
```

```
3

 Utworzone Łącza:

1-3  2-3
```

Następnie skorzystamy z komendy „1”, która realizuję komunikację nazwaną przekazując pomiędzy procesami komunikat wczytany przez użytkownika. Wywołana funkcja najpierw sprawdza czy istnieją procesy o podanych PID, jeśli nie, wyświetlany jest komunikat:

```
1
PID procesu przekazującego komunikat
4
PID procesu przyjmującego komunikat
6
Napisz komunikat
Kom
Nie ma procesu o takim PID
```

Jeśli istnieją takie procesy ale nie istnieje pomiędzy nimi łącze wyświetlany jest komunikat:

```
1
PID procesu przekazującego komunikat
2
PID procesu przyjmującego komunikat
4
Napisz komunikat
kom
Nie ma takiego Łącza
```

Prawidłowy przekazanie komunikatu zostanie potwierdzone:

```
1
PID procesu przekazującego komunikat
1
PID procesu przyjmującego komunikat
3
Napisz komunikat
Komunikat1
Przekazano
```

Dodamy teraz więcej przykładowych komunikatów. Za pomocą rozkazu „wy” wyświetlamy procesy:

```
Komenda:wy
Proces PID( INIT ): 1           Kom:Komunikat3,
Proces PID: 2    Rodzic PPID: 1 Kom:
Proces PID: 3    Rodzic PPID: 1 Kom:Komunikat1, Komunikat2,
Proces PID: 4    Rodzic PPID: 3 Kom:
```

Teraz możemy skorzystać z komendy „przekaz”, która pobiera komunikat o numerze podanym przez użytkownika i przekazuję innemu procesowi.

```
Komenda:p
Podaj PID procesu, z którego pobrany ma zostać rozkaz.
3
Podaj PID procesu, do którego przekazany ma zostać rozkaz.
2
Podaj numer rozkazu.
2
Nie istnieje pokrewieństwo ani dostępne łącze nazwane pomiędzy tymi procesami.
```

W tej sytuacji nie możliwe jest przekazanie rozkazu ponieważ nie utworzyliśmy wcześniej łącza „3-2”. Gdy już to zrobimy:

```
Komenda:p
Podaj PID procesu, z którego pobrany ma zostać rozkaz.
3
Podaj PID procesu, do którego przekazany ma zostać rozkaz.
2
Podaj numer rozkazu.
1
Przekazano
Komenda:wy
Proces PID( INIT ): 1           Kom:Komunikat3,
Proces PID: 2    Rodzic PPID: 1 Kom:Komunikat1,
Proces PID: 3    Rodzic PPID: 1 Kom:Komunikat2,
Proces PID: 4    Rodzic PPID: 3 Kom:
```

Funkcja przekaż zanim sprawdzi czy istnieje łącze nazwane pomiędzy wskazanymi procesami, sprawdza ich pokrewieństwo. Dlatego możliwe będzie przesłanie rozkazu pomiędzy procesami
3-4, gdyż rodzicem 4. Procesu jest proces 3.

```
Komenda:p
Podaj PID procesu, z którego pobrany ma zostać rozkaz.
3
Podaj PID procesu, do którego przekazany ma zostać rozkaz.
4
Podaj numer rozkazu.
1
Przekazano
Komenda:wy
Proces PID( INIT ): 1           Kom:Komunikat3,
Proces PID: 2    Rodzic PPID: 1 Kom:Komunikat1,
Proces PID: 3    Rodzic PPID: 1 Kom:
Proces PID: 4    Rodzic PPID: 3 Kom:Komunikat2,
```

W tej sytuacji komunikacja ma charakter **komunikacji nienazwanej**.

Oprócz funkcji „przekaz” z komunikacji nienazwanej korzysta także komenda nr 2, która pozwala na bezpośrednie wczytanie komunikatu od użytkownika:

```
2
Podaj PID procesu nadawcy
3
Podaj PID procesu odbiorcy
4
Przysyłany komunikat
Komunikat4
Przekazano.
wy
Zły rozkaz
Komenda:wy
Proces PID( INIT ): 1           Kom:Komunikat3,
Proces PID: 2    Rodzic PPID: 1 Kom:Komunikat1,
Proces PID: 3    Rodzic PPID: 1 Kom:
Proces PID: 4    Rodzic PPID: 3 Kom:Komunikat2, Komunikat4,
```

W specjalnej sytuacji gdy jako PID procesu odbiorcy podamy PID procesu nieistniejącego, program spyta użytkownika czy chce utworzyć nowy proces będący dzieckiem procesu nadawcy i do niego przekazać komunikat:

```
2
Podaj PID procesu nadawcy
2
Podaj PID procesu odbiorcy
5
Przysyłany komunikat
Komunikat5
Nie ma procesu o takim PID. Czy chcesz utworzyć dziecko procesu o PID: 2 i przes
łać komunikat ? (Y/N)
y
Komunikat został przekazany dla potomka.
Komenda:wy
Proces PID( INIT ): 1           Kom:Komunikat3,
Proces PID: 2    Rodzic PPID: 1 Kom:Komunikat1,
Proces PID: 3    Rodzic PPID: 1 Kom:
Proces PID: 4    Rodzic PPID: 3 Kom:Komunikat2, Komunikat4,
Proces PID: 5    Rodzic PPID: 2 Kom:Komunikat5,
```

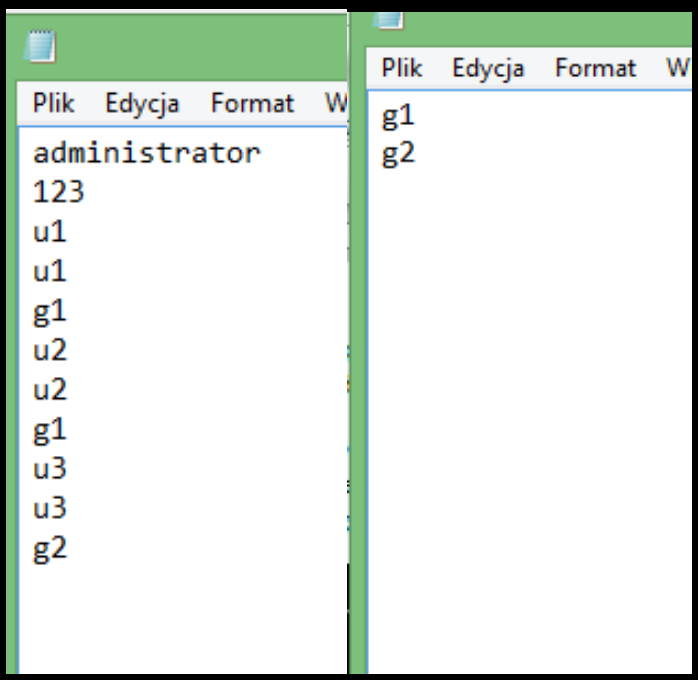
Następnie bardziej szczegółowo testowane były warstwa IV (pliki) oraz warstwa V (użytkownicy) i jej komendy.

System jest uruchamiany po raz pierwszy więc prosi nas o podanie hasło do administratora i automatycznie loguje nas jako administrator.

Na poczatek tworze 2 grupy. Próba stworzenia grupy i tej samej nazwie zakończyła się wyswietleniem komunikatu „taka grupa juz istnieje”. Poźniej tworzę 3 urzytkowników. Próba przypisania urzytkownika do nie istniejącej grupy konczy się niepowodzeniem.

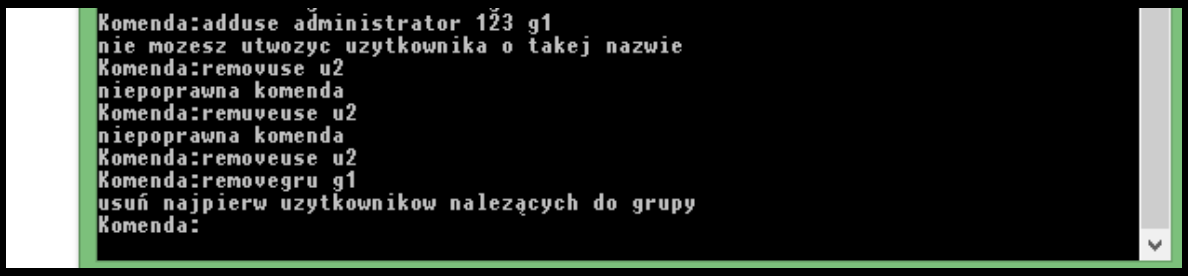
```
Utworzono katalog 'Root' oraz 'Home'

Ustaw hasło do konta administratora:123
Komenda:addgru g1
Komenda:addgru g2
Komenda:addgru g1
taka grua juz istnieje
Komenda:adduse u1 u1 g3
nie istnieje taka grupa
Komenda:adduse u1 u1 g1
Utworzono katalog: u1 w katalogu: Home
Komenda:adduse u2 u2 g1
Utworzono katalog: u2 w katalogu: Home
Komenda:adduse u3 u3 g2
Utworzono katalog: u3 w katalogu: Home
Komenda:_
```

Pliki tesktowe przechowujące loginy ,hasła i grupy.

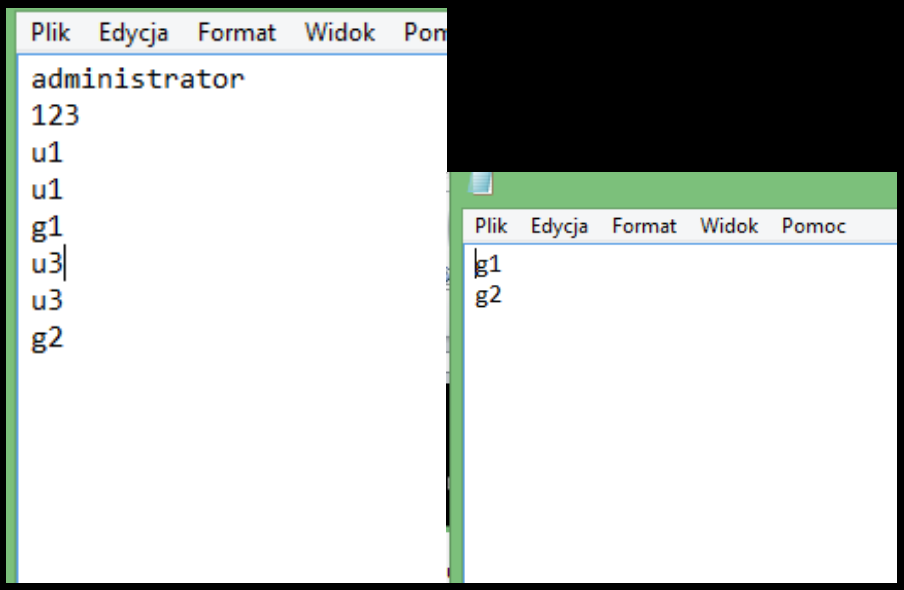


Nieudana próba utworzenia użytkownika o nazwie administrator. Wpisanie niepoprawnych komend.

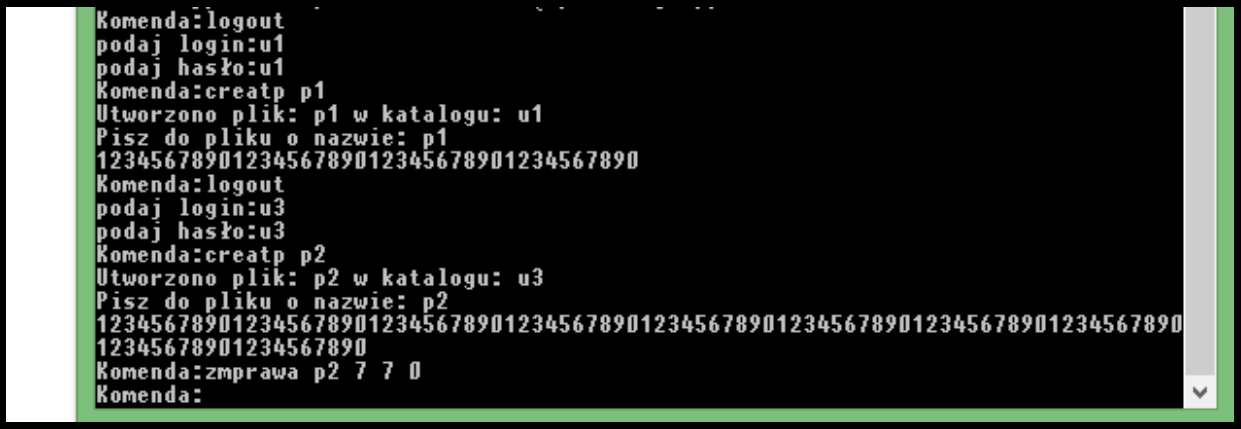
Usunięcie urzytkownika u2 i nieudana próba usunięcia grupy g1.



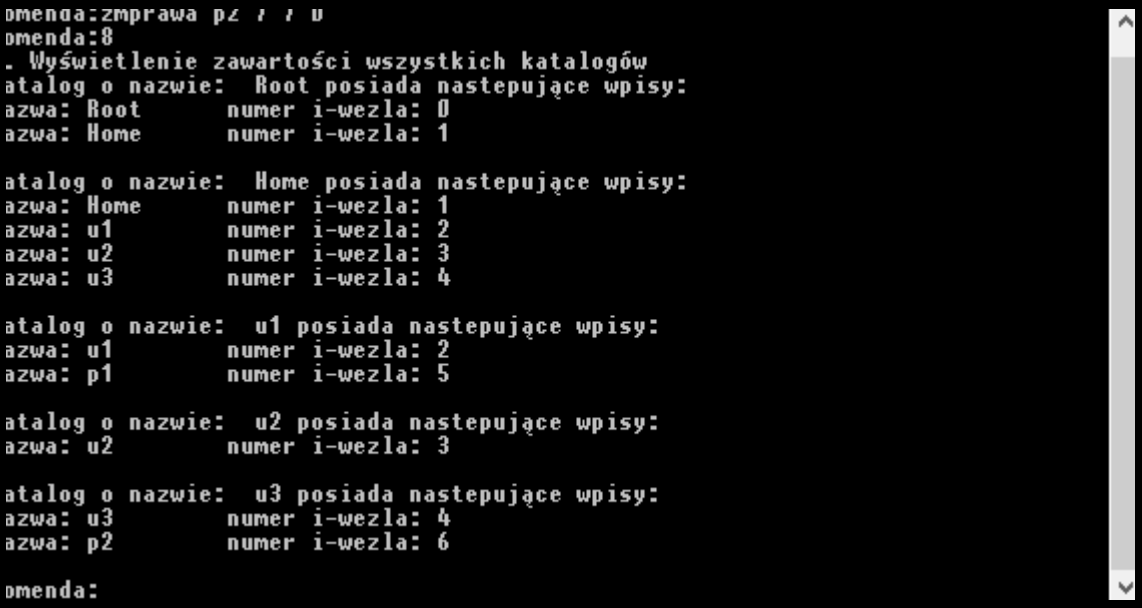
W pliku widzimy że urzytkownik u2 został usunięty a grupa g1 nie.



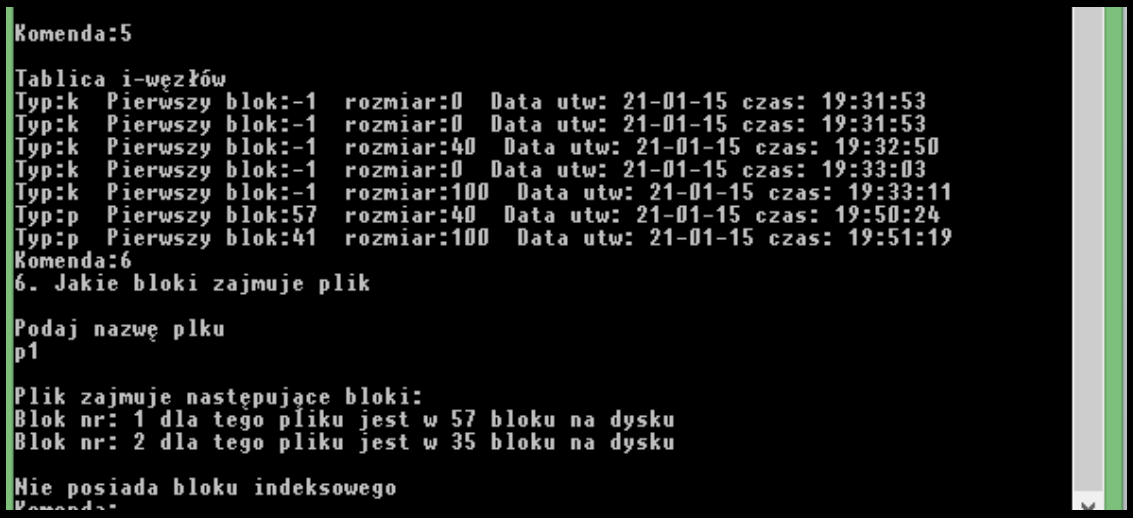
Wylogowanie z konta administratora i zalogowanie na użytkownika u1. Urzytkownik ten tworzy plik p1 o rozmiarze 40 Bajtów.Wylogowanie się i zalogowanie na konto użytkownika u3. Użytkownik ten tworzy plik p2 o rozmiarze 100 Bajtów i zmienia prawa dostepu dla tego pliku na 770.



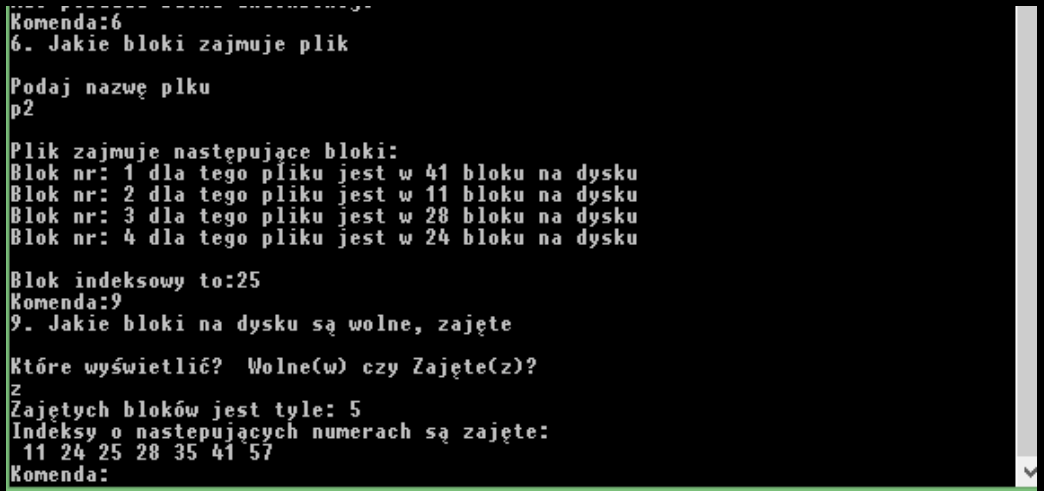
Wyświetlenie zawartości wszystkich katalogów



Wyświetlenie tablicy i-węzłów i bloków zajętych przez plik p1.



Wyświetlenie bloków zajętych przez p2 i wyświetlenie wszystkich zajętych bloków na dysku.



Użytkownik u1 próbuje usunąć plik p2 stworzony przez użytkownika u3. Wylogowanie i zalogowanie na użytkownika u3. Urzytkownik u3 usówa plik p2. Wyświetlenie bloków zajetych na dysku.

```
Komenda:delete p2
Nie mozesz tego usunac
Komenda:logout
podaj login:u3
podaj hasło:u3
Komenda:delete p2
usunięto plik o nazwie:p2
Komenda:9
9. Jakie bloki na dysku są wolne, zajęte

Które wyświetlić?  Wolne(w) czy Zajęte(z)?
z

Indeksy o nastepujących numerach są zajęte:
 35 57
Komenda:
```