**Politechnika Warszawska**
**Wydział Elektroniki i Technik Informacyjnych**
**Instytut Automatyki i Informatyki Stosowanej**

# Systemy operacyjne

## System plików
## Raport

**Zdający:**
**Jakub Sikora**

**Prowadzący:**
**mgr. inż. Aleksander Pruszkowski**

**Warszawa, 21 stycznia 2019**

# Spis treści

# 1. Treść zadania

## 1.1. Cel ćwiczenia

Należy napisać w środowisku systemu Minix program w języku C (oraz skrypt demonstrujący wykorzystanie tego programu) realizujący podstawowe funkcje systemu plików.

## 1.2. Funkcje programu

System plików należy zorganizowaæ w dużym pliku o zadanej wielkości, który będzie "wirtualnym dyskiem". Program powinien tworzyć dysk wirtualny, oraz dokonywać zapisów i odczytów w celu zrealizowania podstawowych operacji na dysku, związanych z zarządzaniem katalogiem, alokacją plików oraz utrzymywaniem unikalności nazw.

## 1.3. Zadanie do zrealizowania

W pliku na dysku należy zorganizować system plików z jednopoziomowym katalogiem. Elementem katalogu jest opis pliku, zawierający co najmniej nazwę, wielkość i sposób rozmieszczenia pliku na wirtualnym dysku. Należy zaimplementowaæ następujące operacje, dostępne dla użytkownika programu:

1. tworzenie wirtualnego dysku
2. kopiowanie pliku z dysku systemu Minix na dysk wirtualny
3. wyświetlanie katalogu dysku wirtualnego
4. usuwanie pliku z wirtualnego dysku
5. usuwanie wirtualnego dysku
6. wyświetlenie zestawienia z aktualną mapą zajętości wirtualnego dysku - czyli listy kolejnych obszarów wirtualnego dysku z opisem: adres, typ obszaru, rozmiar, stan (np. dla bloków danych: wolny/zajęty)

Program ma kontrolować wielkość dostępnego miejsca na wirtualnym dysku i pojemność katalogu, reagować na próby przekroczenia tych wielkości.

Nie trzeba realizować funkcji otwierania pliku ani czytania/pisania fragmentów pliku.

Nie trzeba realizować funkcji związanych z współbieżnym dostępem. Zakłada się dostęp sekwencyjny i wyłączny do wirtualnego dysku.

Należy przygotować demonstrację (zgrupowanie serii poleceń w postaci skryptu interpretera sh) prezentującą słabe i silne strony przyjętego rozwiązania w kontekście ewentualnych zewnętrznej i wewnętrznej fragmentacji.

# 2. Propozycja rozwiązania

## 2.1. System plików

System plików stworzę zgodnie z założeniami VSFS (Very Simple File System), przedstawionymi w rozdziale 40 książki *Operating Systems: Three Easy Pieces*, której autorami są Remzi H. Arpaci-Dusseau i Andrea C. Arpaci-Dusseau.

### 2.1.1. Wielkość i podział na bloki

Tworzony system plików będzie stworzony na wirtualnym dysku o pojemności 256kB. Zostanie on podzielony na bloki o rozmiarze 4kB, co daje nam 64 bloki.

### 2.1.2. Podział funkcjonalny systemu plików

#### User data

Zdecydowaną większość systemu plików będzie stanowiło miejsce na pliki użytkownika. Na tą funkcjonalność przeznaczę 56 bloków, z czego pierwszy blok user data będzie miał numer 8.

#### iNode table

Bloki od 3 do 7 zostaną przeznaczone na tablicę struktur `iNode` (po polsku i-węzły). Każdy iNode przechowuję informację o pliku takie jak jego nazwa, prawa dostępu, data ostatniego dostępu, data ostatniej modyfikacji i przede wszystkim wskaźnik/wskaźniki na block w regionie user data w którym znajduje się zawartość pliku. Sposób implementacji wskaźników może być różny. Najprostszą metodą jest przechowywanie jednego wskaźnika w strukturze iNode na początek pliku i jeśli plik jest większy niż jeden blok to wskaźnik na następny blok umieszczać na końcu tego bloku. Na takie też prymitywne rozwiązanie planuje się zdecydować. Wielkość iNodów dobiorę tak aby mogło ich być więcej niż bloków data.

#### Mapy zajętości bloków

Bloki 1 i 2 przeznaczę na mapy zajętości tablicy iNodów oraz obszaru user data. Zrealizuję ją za pomocą prostych map bitowych w której bit ustawiony będzie oznaczał zajętość danego bloku a nieustawiony blok wolny.

#### Superblock

Na pozycji 0 znajdzie się struktura super bloku. Będzie ona opisem funkcjonalnym całego systemu plików. Przechowywać będzie informację o liczbie struktur iNode, liczbie bloków danych. Dodatkowo, w strukturze znajdzie się wskaźnik na początek tablicy iNodów oraz początek sekcji danych, a także o rozmiarach poszczególnych struktur.

# 3. Program

## 3.1. Projekt programu

Tworzenie nowego systemu plików oraz dostęp do niego będzie realizował program `fs`. Będzie on realizował wszystkie wymienione funkcjonalności opisane w rozdziale pierwszym. Aby ułatwić pisanie skryptów testowych, z narzędzia będzie korzystało się podobnie jak z programów typu `git`, tj. na podstawie argumentu będzie wykonywane polecenie na systemie plików a następnie zakończy pracę.

### 3.1.1. Implementacja programu

Program zostanie w całości napisany w języku C.

### 3.1.2. Główne funkcjonalności

Narzędzie będzie realizowało następujące funkcjonalności:

— tworzenie i inicjalizacja wirtualnego dysku w aktualnej lokalizacji
— kopiowanie podanego pliku z dysku systemu Minix na dysk wirtualny z uwzględnieniem dostępnej pojemności
— wyświetlanie zawartości katalogu dysku wirtualnego
— wyświetlanie zawartości podanego pliku
— edycja wybranego pliku
— usuwanie wskazanego pliku z wirtualnego dysku
— usuwanie całego wirtualnego dysku
— wyświetlenie zestawienia z aktualną mapą zajętości wirtualnego dysku - czyli listy kolejnych obszarów wirtualnego dysku z opisem: adres, typ obszaru, rozmiar, stan (np. dla bloków danych: wolny/zajęty)

# 4. Testowanie rozwiązania

## 4.1. Realizacja testów

Testowanie rozwiązania zostanie przeprowadzone w sposób pół automatyczny za pomocą specjalnie przygotowanych skryptów `sh`.

## 4.2. Testowane funkcjonalności

### 4.2.1. Tworzenie i usuwanie dysku wirtualnego

Test powinien sprawdzać czy narzędzie poprawnie tworzy wirtualny dysk o zadanym wcześniej rozmiarze. Po stwierdzeniu poprawności kreacji dysku i jego inicjalizacji, dysk zostanie usunięty.

### 4.2.2. Tworzenie i usuwanie plików

Testy powinny również mieć możliwość sprawdzenia poprawności tworzenia plików i ich usuwania. W teście zostanie utworzonych kilkadziesiąt plików, celem sprawdzenia zachowania systemu przy przekroczeniu maksymalnej liczby plików w systemie. Test pozwoli również na sprawdzenie jak zwalniane są bloki pamięci dyskowej.

### 4.2.3. Pisanie i czytanie

Ważnym testem będzie sprawdzanie poprawności pisania do plików poprzez narzędzie lub kopiowanie plików z systemu MINIX. Poprawność testu zostanie stwierdzona poprzez odczytanie i porównanie zapisanej zawartości z oryginalną. Ważnym aspektem do przetestowania będzie zachowanie systemu w przypadku zwiększenia rozmiaru pliku już utworzonego w taki sposób że przestanie on się mieścić w oryginalnej ilości bloków, którą na początku zaalokował (tak zwane pisanie z alokacją).

### 4.2.4. Parametry last-accessed, last-modified

Każda operacja powinna skutecznie modyfikować parametry pliku. Test powinien za pomocą narzędzia sprawdzać daty ostatniego dostępu i ostatniej modyfikacji pliku za pomocą prostych operacji porównania.

### 4.2.5. Prezentacja zjawiska fragmentacji

Test powinien alokować miejsce na kilka dużych (zajmujących kilka bloków) i kilka małych plików, następnie usunąć część z nich i znowu stworzyć kilka dużych plików tak aby zaprezentować jak dany system plików radzi sobie ze zjawiskiem fragmentacji (z powodu wybranego sposóbu implementacji systemu plików, spodziewam się beznadziejnych wyników).

# 5. Realizacja rozwiązania

## 5.1. Instrukcja korzystania z programu

W celu wykonania zadania, stworzyłem program `fs`, który jest interfejsem do wirtualnego systemu plików `vsfs`. Poszczególne operacje wykonuje się poprzez podanie odpowiedniego argumentu.

**mkfs**

Aby stworzyć nowy wirtualny system plików należy uruchomić program w następujący sposób:

```
./fs mkfs
```

Polecenie tworzy nowy plik jeśli w aktualnym katalogu nie istnieje inny wirtualny system plików.

**rmfs**

Aby usunąć stary wirtualny system plików należy uruchomić polecenie:

```
./fs rmfs
```

W przypadku gdy w danym folderze nie ma systemu plików, polecenia wyrzuca błąd.

**ls**

Aby zobaczyć pliki znajdujące się w katalogu wirtualnego systemu plików należy uruchomić program w następujący sposób:

```
./fs ls
```

Wynikiem tego polecenia jest wypisanie listy wszystkich plików znajdujących się w folderze.

**fsinfo**

W celu wypisania informacji o systemie plików takich jak zajętość pamięci czy ilość plików, należy uruchomić program w następujący sposób:

```
./fs fsinfo
```

Wynikiem tego polecenia jest wypisanie mapy zajętości bitmap iNodów oraz bloków pamięci. Litera X oznacza że dany segment jest zajęty a litera O oznacza że jest wolny.

**touch**

Aby stworzyć nowy plik o nazwie `x`, należy uruchomić program w następujący sposób.

```
./fs touch <x>
```

W przypadku powodzenia, w systemie plików zostanie stworzony pusty plik o zadanej nazwie. Akcja może się nie powieść jeżeli w systemie znajduje się już plik o danej nazwie.

**cp**

Aby skopiować plik z systemu hostującego o nazwie `x` do systemu plików, należy uruchomić program w następujący sposób.

```
./fs cp <x>
```

W przypadku powodzenia, w systemie plików zostanie stworzony plik o zadanej nazwie z zawartością pliku oryginalnego. Akcja może się nie powieść jeżeli w systemie znajduje się już plik o danej nazwie.

**rm**

Aby usunąć plik z wirtualnego systemu plików o nazwie `x`, należy uruchomić program w następujący sposób.

```
./fs rm <x>
```

W przypadku powodzenia, z systemu plików zostanie usunięty plik o zadanej nazwie. Akcja może się nie powieść jeżeli w systemie nie ma takiego pliku.

**load**

W celu skopiowania pliku z wirtualnego systemu plików do systemu hostującego, należy uruchomić program w następujący sposób.

```
./fs load <x>
```

W przypadku powodzenia, w systemie plików hosta zostanie stworzony plik o zadanej nazwie z zawartością pliku taką jak w systemie wirtualnym. Akcja może się nie powieść jeżeli w systemie wirtualnym nie ma takiego pliku.

**cat**

Aby wypisać zawartość pliku z wirtualnego systemu na konsoli, należy uruchomić program w następujący sposób.

```
./fs cat <x>
```

W przypadku powodzenia, na konsoli zostanie wypisana zawartość pliku. Akcja może się nie powieść jeżeli w systemie wirtualnym nie ma takiego pliku. W celu poprawy pracy z wypisywanym tekstem należy przekazać wyjście na wejście programu `less` (`more` w przypadku systemu MINIX).