# SOI LAB 6 – System plików

Igor Matynia

## Ogólny zarys systemu plików

Wirtualny dysk który zaimplementowałem opiera się na podzieleniu całej przestrzeni wirtualnego dysku na bloki wielkości 4096B. Zajętość każdego bloku jest zapamiętywana na bitmapie. Aby zmieścić duże pliki, bloki łączą się wskaźnikami. System plików obsługuje foldery jedno-poziomowe – pliki mogą znajdować się zarówno w folderze root, jak i podfolderach root'a.

## Realizacja systemu plików

### *Bitmapa*

Bitmapa wraz z jej header'em znajduje się na samym początku pliku. W header'ze znajdują się 2 wartości: Wielkość całej bitmapy w bajtach oraz ilość bloków pamięci dostępnych na całym dysku (wielkość bitmapy*8 != ilość bloków). Bitmapa sama w sobie jest ciągiem bajtów, gdzie każdy bit w bajcie reprezentuje po kolei 1 blok pamięci. Przy alokowaniu przestrzeni na pliki czy foldery to bitmapa ma decyduje o tym czy blok jest zajęty czy nie, bez względu na zawartość bloku.

### *Bloki pamięci*

Na dysku wystąpić mogą 4 rodzaje bloków:

0b00 → Blok typu null, bez zawartości.

0b01 → Blok typu header'a pliku

0b10 → Blok typu header'a folderu

0b11 → Blok danych

Każdy z tych bloków ma ustaloną wielkość – 4096 B – którą można zmienić w kodzie źródłowym. Każdy blok oprócz bloków null'owych zawiera na samym początku wartość typu block_id:

2 pierwsze bity to wcześniej opisany typ bloku, a następne 30 bitów to adres następnego bloku.

Bloki header'owe pliku oraz folderu mają na swoim początku tą samą strukturę – field_header_t. W strukturze tej na początku znajduje się oczywiście block_id. Następnymi zmiennymi są po kolei actual_size, block_size oraz name. Actual_size reprezentuje dokładną wielkość całego pola w bajtach, natomiast block_size mówi tylko o tym z ilu bloków pole to korzysta. Nazwa może mieć maksymalnie 64 znaki.

W przypadku nagłówków plików, reszta przestrzeni w bloku zajęta jest faktyczną zawartością pliku. Reszta danych znajduje się w blokach danych.

Nagłówki folderów są ograniczone do 1 bloku. Wpisy w folderach są reprezentowane jako block_id wskazujący na dany element folderu, którym może być plik lub folder. W tym wypadku pierwsze 2

bity reprezentują typ pola na które wskazuje adres. Z tego wynika, że maksymalna ilość elementów w folderze jest ograniczona wielkością pojedynczego bloku.

### *Alokacja pamięci*

Przy alokowaniu przestrzeni pod plik algorytm patrzy jedynie na pozycje bitmapy w których znajduje się 0 (blok niezajęty). Poszczególne bloki łączone są adresami tak jak opisano wcześniej. W przypadku bloku ostatniego, wskazuje on na NULL_BLOCK (wartość 0). Na koniec jedynie należy dopisać pozycję w nagłówku folderu oraz nadpisać w pliku bitmapę.

### *Usuwanie plików*

Przy usuwaniu jedyna podejmowana akcja to ustawianie właściwych pozycji bitmapy na 0 oraz usunięcie pozycji z nagłówka folderu. W przypadku usuwania folderów, program wymaga żeby folder był pusty.

## Zalety i wady rozwiązania

### *Zalety:*

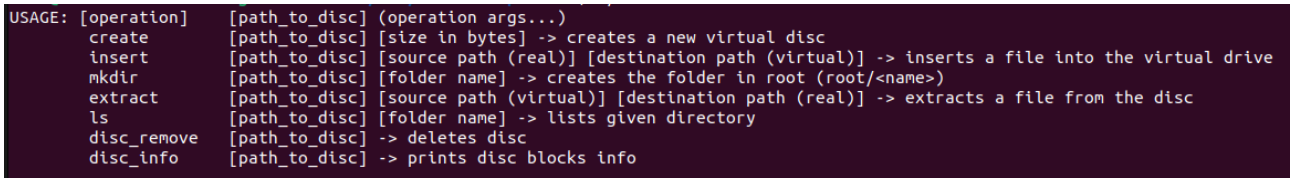
1. Wielkość całego dysku, czyli ilość bloków, ograniczona jest odgórnie przez maksymalną wartość adresu. Adresy bloków reprezentowane są 30-bitową liczbą oraz każdy blok ma 4KB, stąd maksymalny rozmiar dysku to aż 4TB.
2. Wielkość pojedynczego pliku jest w praktyce także ograniczona maksymalną wartością adresu, stąd na wirtualnym dysku można mieć plik o dowolnej wielkości, mniejszej niż 4TB.
3. Dzięki użyciu bloków, bitmapy i wskaźników bloków w plikach, system plików zawsze się będzie starał zapełnić cały dysk. Minimalizowana jest w ten sposób fragmentacja zewnętrzna.
4. System działa bezpośrednio na pliku - granicza to zużycie pamięci operacyjnej i pozwala na pracę z bardzo dużymi dyskami wirtualnymi.

### *Wady:*

1. Dostęp do danych jest sekwencyjny. Po skoczeniu do pierwszego bloku pliku, system musi skakać wedle wskaźników do poszczególnych bloków by załadować składowane dane. Jest to dość powolne, gdyż używana jest metoda fseek.
2. System działa bezpośrednio na pliku – operacje na danych zajmują dłużej niż gdyby znajdowały się one w pamięci operacyjnej.
3. Ilość plików w folderze jest odgórnie ograniczona przez wielkość bloku do 1002. Tak jak było to wspomniane wcześniej, foldery mogą być reprezentowane tylko przez 1 blok. (choć przy stosunkowo małym nakładzie pracy można to zmienić)
4. Bardzo małe pliki oraz foldery zawsze będą zajmowały 4KB, czyli 1 blok, co powoduje marnowanie części użytecznej przestrzeni. Powoduje to fragmentację wewnętrzną.

## Uruchamianie

Aby skompilować program oraz uruchomić wszystkie testy wystarczy użyć podstawowej opcji *make*. Aby tylko skompilować program należy użyć opcji *make compile.* Skompilowany program będzie plikiem o nazwie lab6.




Powyższy komunikat uruchomi się gdy użytkownik wpisze niepoprawną opcję.

Przy używaniu programu należy pamiętać że katalogiem podstawowym jest root, więc każda ścieżka dostępu, czy to do dodawania pliku, usuwania, czy listowania powinna się zaczynać od root/. Jedynym wyjątkiem tej reguły jest polecenie mkdir, które wymaga podania tylko nazwy folderu, bez ścieżki root/.

W folderze z rozwiązaniem znajdują się także skrypty testujące które można uruchomić poprzez Makefile.

## Wyjście skryptu testowego

Po wykonaniu podstawowej opcji make, na standardowe (i błędu) wyjście powinny zostać wypisane poniższe komunikaty:

```
sh ./init_disk.sh
==== INIT ====
Removed disc some_disc.vd

Creating disc some_disc.vd

sh ./mk_file_str.sh
==== Data insertion ====
Inserting small_file.txt into root/duplicate_name.txt

Creating directory folder

Listing directory root:

Type     | Real size      | Disc size      | Name
F        |           83B |         4096B | duplicate_name.txt
D        |            0B |         4096B | folder

Inserting small_file.txt into root/folder/delete_this_later

Inserting small_file.txt into root/folder/file_smo2.txt

Listing directory folder:

Type     | Real size      | Disc size      | Name
F        |           83B |         4096B | delete_this_later
F        |           83B |         4096B | file_smo2.txt

Removing root/folder/delete_this_later
```

*Tutaj mamy przykład eliminowania fragmentacji zewnętrznej:*

```
Block states:
D - directory header, F - file header, x - data block, . - empty block
DFD.F...........
................
................
................
................
Blocks occupied: 4/80 (16.0kB/320.0kB 5.00%)
2 files in 2 folders in total
```

```
Inserting big_file.txt into root/this_one_is_split.txt

Block states:
D - directory header, F - file header, x - data block, . - empty block
DFDFFxx.........
................
................
................
................
Blocks occupied: 7/80 (28.0kB/320.0kB 8.75%)
3 files in 2 folders in total
```

*Tutaj przykład sprawdzania unikalności nazw na poziomie folderu:*

```
Inserting big_file.txt into root/folder/duplicate_name.txt

Listing directory root:

Type    | Real size      | Disc size      | Name
F       |           83B |         4096B | duplicate_name.txt
D       |            8B |         4096B | folder
F       |        11360B |        12288B | this_one_is_split.txt

Listing directory folder:

Type    | Real size      | Disc size      | Name
F       |           83B |         4096B | file_smo2.txt
F       |        11360B |        12288B | duplicate_name.txt

Block states:
D - directory header, F - file header, x - data block, . - empty block
DFDFFxxFxx......
................
................
................
................
Blocks occupied: 10/80 (40.0kB/320.0kB 12.50%)
4 files in 2 folders in total

sh ./err_chk.sh
==== Error checking ====
Block states:
D - directory header, F - file header, x - data block, . - empty block
DFDFFxxFxx......
................
................
................
................
Blocks occupied: 10/80 (40.0kB/320.0kB 12.50%)
4 files in 2 folders in total

Inserting small_file.txt into root/duplicate_name.txt
This name is not unique!: Success
Inserting small_file.txt into
root/veryveryveryveryveryveryveryveryveryveryveryveryveryveryveryveryveryveryveryveryveryveryveryveryveryveryveryveryveryveryveryveryv
erylongname.txt
Name too long!: Success
Inserting small_file.txt into root/duplicate_name.txt/hmmm.txt
This is not a folder!: Success
Inserting small_file.txt into root/thisfolderisnotreal/file.txt
This is not a folder!: Success
Removing root/folder
You can only remove an empty folder: Success
Block states:
D - directory header, F - file header, x - data block, . - empty block
DFDFFxxFxx......
................
................
................
................
Blocks occupied: 10/80 (40.0kB/320.0kB 12.50%)
4 files in 2 folders in total

USAGE: [operation]    [path_to_disc] (operation args...)
        create        [path_to_disc] [size in bytes] -> creates a new virtual disc
        insert        [path_to_disc] [source path (real)] [destination path (virtual)] -> inserts a file into the
virtual drive
        mkdir         [path_to_disc] [folder name] -> creates the folder in root (root/<name>)
        extract       [path_to_disc] [source path (virtual)] [destination path (real)] -> extracts a file from the
disc
        ls            [path_to_disc] [folder name] -> lists given directory
        disc_remove   [path_to_disc] -> deletes disc
```

```
        disc_info     [path_to_disc] -> prints disc blocks info
```

```
sh ./data_extr.sh
==== Data extraction ====
Extracting file root/thisfiledoesnotexist
This is not a file!: Success
Extracting file root/folder
This is not a file!: Success
```

*W tych przykładach linia DIFF pokazuje między kwadratowymi nawiasami wynik polecenia diff out.txt oraz oryginalnego pliku.*

```
Extracting file root/duplicate_name.txt

DIFF: []
Extracting file root/this_one_is_split.txt

DIFF: []
Extracting file root/folder/duplicate_name.txt

DIFF: []
```