Wojciech Celej
271248

Laboratorium 6. - Konstrukcja systemu plików

1. Proponowane rozwiązanie

Architekturą w jakiej skonstruowany jest wirtualny dysk to alokacja ciągła.

Możliwe jest utworzenie dysku o zadanym w blokach rozmiarze. Każdy blok to domyślnie 256 bajtów. Pierwsze 32 bitowe słowo pierwszego bloku zawiera deskryptor dysku:

- pierwsze 4 bajty zawierają liczbę bloków zawierających deskryptory pików; początkowo liczba ta wynosi 2
- następne 4 bajty zawierają pojemność całkowitą dysku w blokach
- następne 4 bajty zawierają liczbę bloków wolnych
- następne 4 bajty zawierają liczbę plików znajdujących się na dysku

Powyższe dane zgrupowane są w 32-bitowej strukturze *disc_info* zadeklarowanej w pliku nagłówkowym.
Pozostała przestrzeń pierwszych dwóch bloków zawiera kolejne deskryptory kolejnych plików przechowywanych na wirtualnym dysku. Deskryptor pliku jest również słowem 32-bitowym o następującej strukturze:

- pierwsze 4 bajty to adres pliku w blokach
- następne 4 bajty to rozmiar pliku w blokach
- następne 4 bajty to liczba bajtów niewykorzystywanych w ostatnim bloku w/w pliku
- pozostałe 20 bajtów zarezerwowane jest na nazwę pliku.

Pozostała przestrzeń pliku zostaje przydzielona plikom z dysku wirtualnego. Przy zapisie zostaje obliczona potrzebna wielkość przestrzeni dyskowej wyrażona w blokach. Jeśli jest ona mniejsza od ilości wolnych bloków oraz istnieje miejsce w blokach na deskryptory zapis przebiega następująco:

- procedura odczytuje adres ostatniego pliku, dodaje do niego jego rozmiar; otrzymaną wartość porównuje z całkowitym rozmiarem dysku: jeżeli ich różnica jest nie mniejsza od wymaganej ilości miejsca, to następuje zapis
- jeśli taka przestrzeń nie istnieje, wywoływana jest procedura relokacji: kolejne pliki przesuwane są w stronę początku dysku wirtualnego
- zapis następuje wtedy do przestrzeni począwszy od pierwszego wolnego bloku
- po udanym zapisie edytowane są odpowiednio dane zapisane w deskryptorze dysku oraz dodawany wpis o pliku w segmencie deskryptorów
- jeśli w blokach zawierających deskryptory plików brak miejsca, bloki zawierające pliki zostają przesunięte” o 2 bloki w stronę większych numerów, licznik bloków z deskryptorami plików zostaje zwiększony, a deskryptor zostaje wpisany w pierwsze możliwe miejsce; następnie zapis przebiega jak wyżej

Wszystkie powyższe procedury zabezpieczone są odpowiednio przed zapisywaniem do dysku, który nie istnieje, kopiowaniem pliku który nie istnieje (czytaniem/pisaniem do pustych strumieni).

Procedura zapisu pliku na dysk i zapisu z dysku jest zabezpieczona przed nadpisywaniem plików.

2. Implementacja

Plik nagłówkowy *filesystem.h*:

```
#define BLOCK 256            //rozmiar bloku w bajtach
#define MAX_LENGTH 20        //maksymalna długość nazwy pliku
struct disc_info
{
        int no_of_descriptor_blocks;
        int     size;
        int     free;
        int     f_count;
} disc_info;                 //struktura opisująca deskryptor dysku

struct descriptor
{
        int adress;
        int size;
        int free_bytes;
        char name[MAX_LENGTH];
} descriptor;                //struktura opisująca deskryptor pliku

/* procedury użytkownika */
int createDisc(char *name, int size);          //utwórz dysk o zadanej nazwie i rozmiarze w blokach
int deleteDisc(char *name);                    //usuń dysk o zadanej nazwie
int cpy2v(char* disc_name, char* file_name);       //kopiuj plik o zadanej nazwie na dysk
int cpy2l(char* disc_name, char* file_name);       //wyciągnij plik o zadanej nazwie z dysku
int deleteFile(char* disc_name, char* file_name);  //usuń plik z dysku
int ls(char *disc_name);                       //wylistuj pliki zapisane na dysku
int info(char* disc_name);                     //wyświetl informacje o stanie dysku
void help();                                   //wyświetl pomoc

/*procedury systemu*/
int reallocate(FILE* f);          //procedura realokacji
int add_descriptor_block(FILE* f);    //procedura dodania nowego bloku deskryptorów plików
int shift_right(FILE* f, int shift, int offset, int size);   //zrób miejsce po prawej stronie
int shift_left(FILE* f, int shift, int offset, int size);     //zrób miejsce po lewej stronie
```

3. Interfejs z poziomu użytkownika:

First argument is always name of disc. All available commands below:

c	<arg1> <arg2>	Create disc <arg1> of size <arg2>
d	<arg1>	Delete disc <arg1>
v	<arg1> <arg2>	Copy file <arg2> to disc <arg1>
u	<arg1> <arg2>	Copy file <arg2> from disc <arg2> to local user's folder
f	<arg1> <arg2>	Delete file <arg2> from disc <arg1>
l	<arg1>	Show list of files saved on disc <arg1>
i	<arg1>	Check disc <arg1> condition
h		Display help
q		Finish work with program

4. Wady i zalety rozwiązania

Największą wadą w/w rozwiązania jest konieczność realokacji danych w celu zapobiegnięcia fragmentacji zewnętrznej i umożliwienia zapisu w ogóle. W w/w rozwiązaniu pozwoliłem sobie uruchamiać powyższą procedurę jedynie w wypadku niemożności zapisu pliku na dysku (czyli wtedy, gdy pomimo że deskryptor dysku informuje, że wolnego miejsca jest wystarczająca ilość). Zaletami metody jest prostota oraz szybkość zapisu w sytuacji, gdy dysk dysponuje odpowiednią ciągłą ilością wolnej przestrzeni za ostatnim zapisanym plikiem. Bardzo szybko przebiega również usunięcie pliku. Gdybyśmy chcieli zmniejszyć stopień fragmentacji, należałoby procedurę realokacji uruchamiać również przy usuwaniu pliku. Pytanie, na czym bardziej nam zależy: czy na szybszym zapisie, czy na szybszym usuwaniu pliku?