Dokumentace k semestrální práci z KIV/ZOS

Vladimír Láznička

2015/2016

Cíl práce

Cílem práce je implementace řešení pro **kontrolu délky řetězu ve FAT souboru** (úkol č. 1) a následnou **defragmentaci bloků ve FAT souboru** (úkol č. 2). Práce bude naprogramována v jazyce C s využitím vícevláknového zpracování jednotlivých úkolů.

Návrh řešení zadaných úkolů

Víclevláknové zpracování úkolů bude řešeno principem *Farmer-Workers*, kdy **jedno hlavní vlákno** (*Farmer*) **bude přiřazovat jednotlivé soubory na žádost podřízených vláken** (*Workers*). Následně bude hlavní vlákno čekat (*join*) s další rutinou na dokončení prací jednotlivých podřízených vláken. Ty pak budou opakovaně žádat o práci a svůj běh ukončí, jakmile jim nebude už co přidělit.

Kontrola délky řetězu ve FAT souboru

Vláknům bude předávaný odkaz (*pointer*) na jednotlivé záznamy *Root Directory*, které v zásadě reprezentují jednotlivé soubory na disku. Mezi údaje záznamu *Root Directory* patří **délka souboru v bytech**. Toho lze využít k určení předpokládané velikosti souboru v clusterech, kdy vydělíme zmíněný údaj velikostí clusteru v daném FAT souboru (je uvedená v záznamu *Boot Record*) a zaokrouhlíme směrem nahoru.

Následně budeme v cyklu procházet řetěz daného souboru, kdy začínáme na **indexu daném**

proměnnou *„First Cluster“* ve structuře *Root Directory*, dokud nenarazíme na hodnotu **označující konec souboru** nebo reálný počet clusterů, který se bude inkrementovat s každým průběhem cyklu, **nepřesáhne předem vypočítaný předpokládaný počet clusterů**. V případě, že tyto počty souhlasí a zároveň na posledním procházeném indexu bude hodnota označující konec souboru, soubor má správnou délku, v opačném případě je délka chybná.

Defragmentace

Před samotnou defrgmentací bude proveden „výpočet“ umístění jednotlivých souborů. Pro každý soubor popsaný pomocí *Root Directory* bude spočítána velikost v clusterech (viz část řešení předchozího úkolu) a **na základě výsledných hodnot se nastaví plánované počáteční indexy**, které se uloží pro použití v samotné defragmentaci. Bude rovněž kontrolována přítomnost poškozených bloků – v takovém případě bude počáteční index pro konkrétní soubor **umístěn za tento poškozený blok**.

Vláknům se pak opět předá odkaz na jednotlivé záznamy *Root Directory* a navíc také **příslušné počáteční indexy bloků**, kam mají být soubory přemístěny. Vlákno bude na základě informací z FAT tabulky procházet jednotlivé bloky, přičemž bude jejich obsah **přemisťovat do bloků na indexech určených podle předaného počátečního indexu**. Zároveň s tím bude měnit obsah FAT tabulek, **aby odpovídal změně stavu na jednotlivých blocích**, a na indexech původních bloků bude nastaven stav *UNUSED*, aby tyto bloky mohly být k dispozici.

Pokud vlákno nebude moci obsah bloku přemístit, protože ten cílový je **stále obsazený původním obsahem**, bude tento blok společně s cílovým indexem a hodnotou, která se má uložit na tento index ve FAT tabulce, **odložen do fronty ještě nezpracovaných bloků daného vlákna**. Tyto bloky se poté přemístí na cílové místo, jakmile vlákno dokončí všechny dostupné práce předané hlavním vláknem. Tento mechanismus by měl předejít uvíznutí vláken v případě, že si budou **navzájem bránit přístupu k potřebným blokům**, ačkoliv bude také znamenat potřebu většího množství paměti.

Implementace

Soubory programu

Výsledný program se skládá ze **zdrojových** (`.c`) a **hlavičkových** (`.h`) souborů, **shellového skriptu** (`.sh`) provádějícího spuštění programu pro variabilní počet vláken a souboru `Makefile` sloužícím ke kompilaci programu. Hlavičkové soubory pak také obsahují **komentáře ke strukturám a funkcím**.

Zdrojové a hlavičkové soubory

- **`structures.h`** – definuje globálně používané struktury, do kterých jsou ukládány potřebné informace pro práci programu, včetně struktur pro uložení vstupního FAT souboru (*Boot Record*, *Root Directory*...).
- **`global_functions.h`** – definuje funkce spouštěné z hlavního vlákna programu (z funkce main)
- **`worker_functions.h`** – definuje funkce, které řeší činnosti podřízených vláken programu (žádost o práci od hlavního vlákna, defragmentace...)
- **`main.c`** – obsahuje hlavní (vstupní) funkci programu, která bude postupně iniciovat jednotlivé části programu
- **`global_functions.c`** – implementace funkcí hlavního vlákna
- **`worker_functions.c`** - implementace funkcí podřízených vláken

Globální struktury a hlavní funkce (main)

- **Struktura `boot_record`** - jedná se o strukturu pro uložení dat *Boot Record* záznamu ze souborového systému FAT.
- **Struktura `root_directory`** - jedná se o strukturu pro uložení dat *Root Directory* záznamu ze souborového systému FAT.
- **Struktura `suspend_cluster`** - struktura pro uložení obsahu clusteru, který nebylo možné při defragmentaci hned zpracovat. Tato struktura je zapojována do spojového seznamu, který si drží každé podřízené vlákno při defragmentačním procesu.

Hlavní funkce (main)

Program začne kontrolou vložených argumentů pomocí funkce `check_arguments()`, pokud tato proběhne v pořádku, pokračuje se voláním funkce `load_file()`, která načte obsah vstupního FAT souboru do připravených struktur (`boot_record* br`, `uint32_t** fat_tables`, `root_directory** rd_list` a `char** clusters`). V případě, že je načtení v pořádku, provede se alokace paměti pro podřízená vlákna (`worker_threads`).

Následně **proběhne nastavení několika potřebných parametrů pro zahájení kontroly délky souborů**. Důležitá je proměnná `bad_file_size_sum`, která značí počet souborů s chybnou velikostí (na začátku je 0). Poté se zahájí činnost vláken pomocí knihovní funkce `pthread_create()` a pomocí funkce `pthread_join()` se pozastaví činnost hlavního vlákna, dokud ta podřízená neskončí. V průběhu práce vláken se může inkrementovat proměnná `bad_file_size_sum`, pokud je vyhodnoceno, že nějaké soubory mají špatnou velikost, a na základě toho je rozhodováno o dalším postupu.

V případě, že žádné chybné velikosti nebyly objeveny, pokračuje se úlohou defragmentace. Nejprve jsou inicializovány mutexy v poli `cluster_mutex_array`, které slouží pro zamykání sekcí s clustery v průběhu defragmentace. Dále se nastaví parametry potřebné pro zahájení defragmentace a voláním funkce `prepare_fat_tables_for_defrag()` se připraví indexy v seznamu `defrag_indexes`, které jsou používány pro umisťování souborů na správná místa. Následně se opět spustí práce podřízených vláken a hlavní vlákno čeká na jejich dokončení.

Po skončení defragmentace se obsah FAT struktur uloží do výstupního souboru (program informuje, zda bylo uložení úspěšné) pomocí funkce `save_file()` a nakonec se provede vyčištění paměti držené strukturami funkcí `clear_structures()`. Při úspěchu program končí návratovou hodnotou **0**.

Funkce hlavního vlákna

Funkce `check_arguments()`

Nejprve je zkontrolován počet argumentů, pokud není správný (**program + 3 argumenty**), vypíše se chybová hláška a program se ukončí s hodnotou `EXIT_FAILURE`. V případě, že je počet argumentů v pořádku, **uloží se do připravených proměnných a provede se převod argumentu pro počet vláken z řetězce na integer** (rovněž kontrolováno, pokud se nejedná o celé kladné číslo, program skončí).

Funkce `load_file()`

Funkce se pokusí otevřít soubor v **režimu binárního čtení**, pokud dojde k chybě, program ji oznámí a skončí s hodnotou `EXIT_FAILURE`. V pozitivním případě se alokuje paměť pro jednotlivé struktury a knihovní funkcí `fread()` se do nich načte obsah ze souboru.

Funkce `save_file()`

Obdoba předchozí funkce, jen provádí zápis jednotlivých struktur do souboru pomocí knihovní funkce `fwrite()`.

Funkce `clear_structures()`

Postupně uvolní paměť používanou jednotlivými strukturami potřebnými pro FAT soubor.

Funkce `prepare_fat_tables_for_defrag()`

Funkce nejprve alokuje paměť pro plánované počáteční indexy (`uint32_t`) jednotlivých souborů – pole `defrag_indexes`. Následně se prochází jednotlivé záznamy *Root Directory* (seznam `rd_list`) a poměrem velikosti souboru v bytech (`file_size` ze struktury `root_directory`) a velikostí jednoho clusteru (`cluster_size` ze struktury `boot_record`) se zjistí počet clusterů, které soubor potřebuje. Ve vnitřní cyklu se pak procházejí indexy v první FAT tabulce a pokud se na některém z nich narazí na hodnotu `FAT_BAD_CLUSTER`, plánovaný první index souboru se přesune za něj (toto se opakuje, dokud není nalezeno dost místa pro právě

řešený soubor). Jednotlivé indexy se pak ukládají do zmíněného pole.

Funkce podřízených vláken

Funkce `do_work()`

Hlavní funkce podřízených vláken (používá se jako argument do funkce `pthread_create()`) – **obsahuje cyklus, ve kterém si vlákna říkají o další práci a pak ji vykonávají**. Jako argument této funkce se předává hodnota značící, jakou práci má vlákno vykonávat:

- kontrolu velikosti souboru – `CHECK_FILE_SIZE_JOB`
- defragmentaci souboru – `DEFRAGMENATATION_JOB`

Před zahájením cyklu si vlákno vytvoří počáteční prvek spojového seznamu struktury `suspend_cluster`, za který se budou případně přidávat clustery, **které není možné při defragmentaci ihned vyřešit**.

Následuje samotný cyklus, ve kterém se vykovává práce. Ten běží dokud návratová hodnota funkce `get_job()` je >= 0, tato hodnota pak slouží jako index do seznamu `rd_list` a v podstatě říká, jaký soubor má vlákno řešit. Vlákno pak na základě **aktuálně nastavené práce** volá buď funkci `size_check_func()`, jejíž výstup je posílán do funkce `increment_bad_file_size_sum()`, nebo `defragment_func()`, která vrací ukazatel na poslední prvek seznamu `suspend_cluster`, pokud byly za jejího průběhu nějaké vytvořeny a připojeny ke stávajícímu seznamu.

Jakmile cyklus skončí (všechny soubory byly obslouženy), provede se funkce `defragment_suspended_func()`, která vyřeší zbývající clustery (za podmínky, že nějaké jsou). Nakonec je uvolněna paměť zabraná seznamem struktur `suspend_cluster`.

Funkce `get_job()`

Tato funkce předává vláknu **index souboru, který má obsloužit, pokud ještě nějaké zbývají** (to je zajištěno porovnáním proměnné `processed_files` a proměnné `root_directory_max_entries_count` z *Boot Record* struktury). Pokud již žádné

nezbývají, funkce vrací **-1**. Ve funkci je použit mutex – `get_job_mutex` – který má za úkol zajistit přístup k proměnné `processed_files` pouze jednomu vláknu naráz.

Funkce `size_check_func()`

Funkce přejímá jako parametr ukazatel na strukturu `root_directory`, která popisuje zkoumaný soubor. **Z ní si načte potřebné parametry a projde v cyklu všechny indexy ve FAT tabulce, přičemž si inkrementuje proměnnou pro reálný počet clusterů**. Skončí v momentě, kdy narazí na hodnotu `FAT_FILE_END` nebo reálný počet počet clusterů přesáhne očekávaný počet.

Funkce vrací **0** v případě, že soubor má velikost v pořádku, a **1** v případě opačném.

Funkce `increment_bad_file_size_sum()`

Jednoduchá funkce, která má za úkol inkrementovat globální proměnnou `bad_file_size_sum` o v parametru předané množství. Používá mutex – `bad_file_size_sum_mutex` – aby zabránila současný přístup více vláken.

Funkce `defragment_func()`

Funkce obsluhující defragmentaci vybraného souboru. Jako parametry přejímá odkaz na strukturu `root_directory`, plánovaný počáteční index souboru (kam bude soubor umístěn) a poslední člen seznamu se strukturami `suspend_cluster`. Nejprve si zjistí, kolik clusterů soubor má a poté provádí cyklus pro všechny tyto clustery. V něm si nejprve určí **jeden z mutexů v poli cluster_mutex_array pro zdrojový cluster a druhý mutex pro cílový cluster** – tyto mutexy pak zamkne, aby se k nim žádné jiné vlákno nedostalo (vždy si pokryje **1/10 celkového množství clusterů v oblasti, kde se dané clustery nachází**)

Následuje přesun obsahu clusterů mezi dvěma místy a úprava údajů ve FAT tabulce podle provedené změny, pokud měl cílový cluster označení `FAT_UNUSED`. Pokud ne, **obsah clusteru, jeho cílový index a index následujícího clusteru** (pro budoucí vyplnění údaje do FAT tabulky), se uloží do nově vytvořené struktury `suspend_cluster`. Ta se pak přidá za poslední prvek v již existujícím seznamu. Jakmile se tyto operace pro daný cluster dokončí zabrané mutexy se odemknou a cyklus pokračuje dalším clusterem. Návratovou hodnotou funkce je odkaz na poslední člen seznamu struktur `suspend_cluster`.

Funkce `defragment_suspended_func()`

Tato funkce osluhuje seznam clusterů, **které nemohly být zpracovány během hlavní části defragmentace**. Pomocí funkce `cluster_unused()` si zjišťuje, zda je cílový index již volný (některá vlákna mohou teoreticky pořád řešit předané soubory), pokud ne, vlákno odevzdá zbytek přiděleného času pomocí `sched_yield()`. Jakmile se cílový cluster uvolní, vlákno si nad ním zamkne mutex z pole `cluster_mutex_array` a provede přesun obsahu na správné místo.

Funkce `cluster_unused()`

Funkce zjišťující, zda je cluster na předaném indexu označen ve FAT tabulce jako `FAT_UNUSED`. Využívá mutexu z pole `cluster_mutex_array` pro zajištění přístupu pouze jednoho vlákna naráz. Vrací pak výsledek porovnání mezi hodnotou ve FAT tabulce a konstantou `FAT_UNUSED`.

Funkce `clear_suspended_cluster_list()`

Funkce, která postupně uvolní paměť zabranou spojovým seznamem se strukturami `suspend_cluster`.

Spuštění programu

Program *FAT_Defragmentator* lze spustit samostatně z příkazové řádky (po kompilaci) předáním 3 parametrů:

- cesta ke vstupnímu FAT souboru
- cesta k výstupnímu FAT souboru
- počet podřízených vláken, které budou vytvořeny

Všechny ty to parametry musí být zadány, jinak program vypíše chybu a ukončí svůj běh. Počet vláken musí být zadán jako celé kladné číslo.

Dalším způsobem je spuštění pomocí přiloženého skriptu *FAT_Defragmentator_run.sh*, který spustí program **n-krát s 1...n vlákny**. Zároveň měří čas běhu programu a zapisuje do souboru FAT_Defragmentator.log jednotlivé časy. Potřebné parametry pro spuštění skriptu jsou:

- název programu ke spuštění (FAT_Defragmentator)
- cesta ke vstupnímu FAT souboru
- cesta k výstupnímu FAT souboru
- maximální počet vláken, které mají být použity (n)