FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

# Zadání bakalářské práce

| | |
|---|---|
| **Název:** | Systém pro testování výkonnosti systémů souborů |
| **Student:** | Michal Pokorný |
| **Vedoucí:** | Ing. Jan Trdlička, Ph.D. |
| **Studijní program:** | Informatika |
| **Obor / specializace:** | Bezpečnost a informační technologie |
| **Katedra:** | Katedra počítačových systémů |
| **Platnost zadání:** | do konce letního semestru 2023/2024 |

## Pokyny pro vypracování

1) Porovnejte existující SW řešení pro testování různých systémů souborů.
2) Navrhněte a naimplementujte systém, který bude umět testovat výkonnost různých systémů souborů (např. čtení/zápis různě velkých souborů, paralelní zápis/čtení do souborů,...).
3) Otestujte váš systém na různých systémech souborů a porovnejte naměřené výsledky.

*Elektronicky schválil/a prof. Ing. Pavel Tvrdík, CSc. dne 14. ledna 2023 v Praze.*

Bakalářská práce

# SYSTÉM PRO TESTOVÁNÍ VÝKONNOSTI SYSTÉMŮ SOUBORŮ

**Michal Pokorný**

Fakulta informačních technologií
Katedra počítačových systémů
Vedoucí: Ing. Jan Trdlička, Ph.D.
11. května 2023

# Obsah

# Seznam obrázků



# Seznam tabulek



# Seznam výpisů kódu

*Rád bych poděkoval svému vedoucímu Ing. Janu Trdličkovi, Ph.D. za cenné rady v průběhu tvorby této bakalářské práce.*

# Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 2373 odst. 2 zákona č. 89/2012 Sb., občanský zákoník, ve znění pozdějších předpisů, tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené.

V Praze dne 11. května 2023 .....

# Abstrakt

Tato bakalářská práce se zabývá vytvořením nástroje pro testování výkonnosti systému souborů na operačním systému Linux. Součástí teoretické části je shrnutí existujících nástrojů a technik testování. Praktická část se věnuje návrhu a implementaci administrátorského nástroje pro mikro-testování různých systémů souborů. Návrh nástroje je vytvořen tak, aby jednotlivé systémy souborů byly porovnatelné mezi sebou přesnou metodou. Výsledkem práce je rozšiřitelný nástroj v jazyce C++, který testuje výkonnost základních operací na systému souborů a umožňuje lokální testování. V přiloženém médiu práce se nachází zdrojový kód tohoto nástroje.



**Klíčová slova** testování výkonnosti, systém souborů, nástroj, Linux, administrace, C++



# Abstract

This bachelor thesis deals with the creation of a tool for testing the performance of file systems on the Linux operating system. The theoretical part summarizes existing testing tools and techniques. The practical part focuses on designing and implementing an administrative tool for micro-testing different file systems. The tool design is created to make individual file systems comparable with each other using a precise method. The result of the work is an expandable tool in C++ language that tests the performance of basic operations on file systems and allows for local testing. The source code for this tool is included in the attached media of the thesis.



**Keywords** benchmarking, file system, tool, Linux, administration, C++

# Úvod

V dnešní době jsou systémy souborů základní součástí každého výpočetního systému. Je stále potřeba objektivně validovat, zda jsou používané systémy souborů opravdu nejlepším možným řešením vůči všem ostatním řešením. Systém pro testování souborových systémů je při volbě použitého souborového systému potřebný, vzhledem k tomu, že každý server může mít jiný styl ukládání/získávání dat z diskových medií, rozdílné požadavky na bezpečnost a další faktory, které mohou ovlivnit výkon systému. Proto je důležité, aby validace byla provedena při rozhodování o použitých technologiích, aby byly přesně a objektivně podloženy. Následně zvolený souborový systém může mít velký vliv na celkovou výkonnost systému, a to v případě horizontálního škálování může mít dokonce násobný vliv.

V této práci bude navržen a implementován nový nástroj pro testování výkonnosti souborových systémů na operačním systému Linux. Tento nástroj bude umožňovat provádět různé typy testů na různých souborových systémech a bude optimalizován pro co nejvyšší přesnost, škálovatelnost a znovupoužitelnost. Výsledky testů různých systémů souborů provedených pomocí tohoto nástroje budou zhodnoceny a porovnány.

V první části se věnuji uvedení do problematiky souborových systémů a vysvětlení, proč je důležité je testovat vůči sobě a jaké faktory mohou ovlivnit výkon systému. Také se zaměřím na pojmy, metody a kategorie kolem testování jako takového a srovnám nejpoužívanější softwarová řešení za poslední dekády. Na základě tohoto srovnání vytvořím požadavky na nový nástroj, který je cílem této práce. V druhé části se budu zabývat návrhem tohoto nástroje, popisem řešení jednotlivých vzniklých problému a shrnu implementační kroky. Následně tento nástroj využiji k otestování různých systémů souborů.

## Cíle práce

Hlavním cílem této práce je provést rozsáhlou analýzu nástrojů určených pro testování výkonnosti souborových systémů na operačním systému Linux. Dalším cílem práce i jeho hlavním výstupem je navrhnout a implementovat nový nástroj, který bude schopen provádět kvalitativní testování výkonnosti těchto souborových systémů. V neposlední řadě bude výstupem měření a porovnání výkonnosti několika souborových systémů.

V teoretické části práce bude nejprve podrobně představena problematika porovnávání výkonnosti souborových systémů. Zde budou uvedena již existující řešení, která se zabývají tímto tématem. Tyto metody budou pečlivě analyzovány a zhodnoceny z hlediska jejich přínosů a nedostatků. Díky výsledkům této analýzy bude možné přesně specifikovat vlastnosti implementace nového nástroje, včetně souhrnu kvalit a vlastností testování pro co největší přesnost, škálovatelnost a znovupoužitelnost.

Praktická část práce bude zaměřena na návrh a implementaci nového nástroje pro testování výkonnosti souborových systémů. Tento nástroj poté otestuje nejpoužívanější souborové systémy,

které se používají na serverech s operačním systémem Linux. Bude testováno dílčích testů výkonnosti v čistém a synteticky definovaném prostředí (mikro-výkonost). Výsledky těchto testů budou porovnány a zhodnoceny.

# Kapitola 1

# Souborové systémy

*V úvodu se čtenář seznámí se základními pojmy týkajícími se souborových systémů, včetně kontextu, kde se takové systémy používají, a komponent, se kterými tyto systémy spolupracují. Následně budou podrobněji popsány různé typy souborových systémů, včetně jejich klíčových vlastností a příkladů, jak se používají v praxi. Dále se zaměříme na různé vedlejší účinky, které souborové systémy mohou mít, jako je například žurnál, zabezpečení a cache. Tyto vedlejší účinky budou probrány podrobněji, aby čtenáři získali lepší představu o tom, jak mohou ovlivnit výkon a stabilitu souborového systému.*

## 1.1 Souborový systém

Systém souborů je klíčovým prvkem operačního systému, který umožňuje uživatelům ukládat, organizovat a spravovat data na jejich počítači. Administrátorům pak umožňuje zajistit bezpečnost dat, správu přístupových práv a monitorování toku dat. Systém souborů funguje tak, že data jsou uložena na pevném disku v různých souborech, které jsou organizovány do logické hierarchie složek [1]. Mezi základní vlastnosti systému souborů patří ochrana dat, zabezpečení přístupu k datům, rychlé vyhledávání a organizace dat.

Souborové systémy se liší svou strukturou a způsobem organizace dat. Dále se liší tím, jakým způsobem využívají metadata, jak velká data dokážou adresovat, nebo jaké operační systémy je podporují.

Linux/Unix [2]:

- ext2: Nejstarší ext souborový systém bez žurnálu a s malou ochranou proti ztrátě dat v případě nejhoršího scénáře.
- ext3: Standardem dlouhodobého používání v mnoha distribucích.
- ext4: Poslední standard ext od jádra verze 2.6.28.
- Btrfs: Nováček s vlastnostmi podobnými ZFS.
- XFS: Souborový systém s žurnálem vyvinutý společností SGI.
- ReiserFS: Souborový systém původně iniciovaný Hansem Reiserem.
- ZFS: Souborový systém od Sun Microsystems (nyní pod Oraclem), který je považován mnoha odborníky za nejvyspělejší.

Windows:

- FAT32, exFAT
- NTFS

## 1.2 Části souborových systémů

Souborové systémy se skládání z několika důležitých části [3]:

### 1.2.1 Metadata, pojmenování, žurnál

Souborové systémy využívají metadatové struktury, které umožňují ukládat a organizovat data, včetně informací o souborech, jako je název, velikost, datum a čas poslední úpravy, přístupová práva a mnoho dalších. Tyto metadata jsou uloženy na různých místech v systému v závislosti na konkrétním souborovém systému a použitém algoritmu. Například v ext2 souborovém systému jsou metadata uložena na začátku souboru, zatímco v NTFS jsou metadata uložena na konci souboru. Pojmenování souborů se také liší v závislosti na konkrétním souborovém systému a mohou mít různá omezení, jako je maximální délka názvu nebo určitý formát. Žurnál je mechanismus, který umožňuje souborovému systému zaznamenávat změny na disku a umožňuje snadnější obnovu po neočekávaném vypnutí počítače.

### 1.2.2 Ukládání na pevné disky, cachování

Bloky jsou přidělovány k souboru pomocí různých algoritmů, které určují, jak se data ukládají na pevný disk. Tyto algoritmy se liší v závislosti na konkrétním souborovém systému, ale jejich cílem je optimalizovat využití disku a minimalizovat fragmentaci dat. Souborové systémy používají cachování jako mechanismus pro ukládání dat do paměti tak, aby byl rychlejší přístup k těmto datům. Cachování se liší v závislosti na konkrétním souborovém systému a použitém algoritmu. V některých systémech se používá tzv. read-ahead cache, která načítá do paměti více dat, než je aktuálně potřeba, aby byl zajištěn rychlejší přístup k datům v budoucnu. V jiných systémech se používá write-back cache, která ukládá změny do paměti a později je zapisuje na disk.

### 1.2.3 Bezpečnost

Souborové systémy mohou být zabezpečeny různými úrovněmi zabezpečení, aby zajistily bezpečné ukládání a přístup k datům. Kromě omezení přístupu ke souborům a složkám mohou být soubory a složky také šifrovány, což zajišťuje ochranu dat před neautorizovaným přístupem třetích stran. Kontrolou integrity dat se zajistí, že data nebyla poškozena nebo ztracena během přenosu nebo ukládání. Tato opatření jsou důležitá zejména v případě citlivých dat, jako jsou osobní údaje nebo firemní údaje, které mohou být cílem útoků.

Kapitola 2

# Výkonnostní testování souborových systémů

*Tato kapitola se zaměřuje na výkonnostní testování souborových systémů. Vysvětlí se základní pojmy kolem testů výkonnosti, kategorizace a používané algoritmy a metody.*

## 2.1 Výkonnostní testování

Výkonnostní testování (ang. Benchmarking) je metoda umožňující objektivní a přesné srovnání pozorovaných měření. Výkonnostní testování souborových systémů tedy umožňuje objektivní a přesné srovnání různých souborových systémů. Měřením výkonu souborových systémů v různých scénářích, jako je čtení a zápis souborů různých velikostí nebo provádění paralelních operací čtení/zápisu, je možné určit, který souborový systém je nejefektivnější pro daný účel. Tato informace je cenná pro správce systému či vývojáře, kteří potřebují informovaně rozhodovat o tom, který souborový systém použít v daném kontextu. Navíc srovnání může pomoci identifikovat potenciální výkonnostní problémy a poskytnout náhled na to, jak vylepšit výkon souborového systému. Při použití výkonnostních testů je důležité vybrat vhodné algoritmy, které umožní správné měření výkonu a porovnání výsledků.

Zátěž (ang. Workload) je označení pro množství práce, kterou musí systém nebo aplikace zvládat v určitém časovém období. V případě testování výkonnosti souborových systémů se zátěží může být například množství paralelních operací čtení a zápisu souborů nebo velikost souborů, které jsou přenášeny.

## 2.2 Kategorie testů výkonnosti souborých systémů

Trager a kol. uvádějí následující klasifikace testů výkonnosti [4]:

- Makro-testy výkonnosti (ang. macrobenchmarks): Výkon je testován proti konkrétní zátěži, která má simulovat nějakou reálnou (produkční) zátěž.
- Opakování vzorků (ang. Trace Replays): Program opakuje operace, které byly zaznamenány v reálném scénáři, s nadějí, že tyto operace reprezentují reálnou pracovní zátěž.
- Mikro-testy výkonnosti (ang. microbenchmarks): Testují se několik základních I/O operací, aby se izolovaly jejich specifické režijní náklady v systému, např. čtení/zápis/vytvoření/mazání souboru. Zátěž je minimální, pouze pro potřeby testu. Jejich výhodou je rychlost běhu testu, jednodušší režie a jednoduchá zátěž.

## 2.3 Typy testů výkonnosti z pohledu dimenzí systému souborů

Z pohledu, jaké části systému souborů dané testy výkonnosti testují, můžeme rozlišit [5]:

- Testy výkonnosti vstupních/výstupních zařízení: šířka pásma a latence
- Testy výkonnosti ukládání na pevných discích
- Testy výkonnosti operací s metadaty (vytvoření souboru, změna atributu)
- Testy výkonnosti způsobu cachování: studená a horká
- Testy výkonnosti škálování systému souborů se zvyšující se zátěží

## 2.4 Způsob vyhodnocování a porovnávání výsledků

Tato kapitola se zaměřuje na požadavky pro testování výkonnosti souborových systémů, jak je představil Traeger a kol. [4] Vysvětlí se, jak vybrat konfigurace a prostředí pro testování a jak prezentovat výsledky. Tyto požadavky důrazně zdůrazňují, že je důležité podrobně vysvětlit, jaké kroky byly provedeny, a také zdůvodnit, proč byly provedeny.

### 2.4.1 Výběr konfigurace testu

Prvním krokem je evaluace systému, což zahrnuje porovnání výkonnosti souborového systému s podobnými systémy a zjištění, jak se systém chová pod zátěží. Dále je nutné určit baseline systému, konfiguraci systému a typy testů/operací, které se použijí při testování, aby bylo možné odpovědět na dané otázky. To produkuje trojici <prostředí systému, konfigurace, test>, která reprezentuje vstupní kontext testování.

### 2.4.2 Požadavky na testy

1. Testy souborového systému by měly být I/O závislé, nikoliv CPU závislé, resp. by CPU čas neměl být zahrnut do časovaných úseků. Výjimkou mohou být systémy souborů se značnou CPU výpočetní částí (kompresní, šifrovací systémy) [4].
2. Použít přesné měření času.
3. Test by měl být škálovatelný, aby testoval každý počítač stejným množstvím nezávisle na rychlosti hardwaru nebo softwaru.
4. Vícevláknovost může představovat více realistické scénáře a může pomoci saturaci systému požadavky.
5. Jednotlivé testy by měly být spouštěny několikrát pro zajištění statistické přesnosti.
6. Test by měl běžet alespoň tak dlouho, aby se systém pod testem dostal do ustáleného stavu.

### 2.4.3 Výběr prostředí systému

Stav systému při běhu testu může mít velký dopad na výsledky. Po zhodnocení přiměřeného stavu by se tento stav měl reportovat s velkou přesností. Hlavní vlivy ovlivňující test jsou stav cache, efekty ZCAV[1], stáří souborového systému a nepotřebné procesy, které běží při běhu testu.

Nežádoucí účinek je brán jako určitý efekt měřeného úseku, který může ovlivnit dobu trvání tohoto úseku a tím ovlivnit statistickou pravděpodobnost celkového výsledku. Prvním logickým účinkem může být cachování, souborové systémy používají různé mechanismy pro ukládání dat do paměti tak, aby byl rychlejší přístup k nim. Cachování je vlastnost systému souboru, která je

[1] zoned constant angular velocity, které využívají moderní disky k uložení dat. Cylindry jsou rozděleny do zón, kde počet sektorů v cylindru roste se vzdáleností od centra disku. Rychlost přenosu se liší zóna od zóny [6].

jeho součástí, takže testovat by se mělo s ním, může přijít zajímavé měření například rychlosti naběhnutí cachování.

Cache může ovlivnit kódové cesty, které jsou pod testem, a tím mohou ovlivnit výsledky. Cache se dělí na studenou a horkou: studená je například stav po restartu, připojení diskového média nebo mount systému souboru. Horká je pak cache, která je využívána v provozu a odráží tak reálné systémy. Pro neovlivnění časových výsledků a tím i míru spolehlivosti je nutné uvést cache do konzistentního stavu před každým testem a to stejným způsobem. Pro studenou cache se doporučuje čištění cache před každým testem, což lze docílit nejúčinněji pomocí restartu operačního systému [7]. Mezi další mechanismy, jak omezit účinky cachování patří např. Direct I/O[8], který sníží účinky cachování v rámci vyrovnávací paměti.

Pro minimalizaci účinku ZCAV lze vytvořit partition o nejmenší možné velikosti na okraji disku, ale nemusí být reálné (například pro long seek testy). Je ale nutné uvádět lokalitu testování partition.

Nepotřebné procesy: pro reprodukovatelnost výsledků je nutné vypnout všechny nepotřebné služby a procesy tak, aby prostředí bylo v kontrolovatelném stavu. Je doporučeno použít vícevláknovost pro více reálné výsledky, zamezit přihlášení uživatele a žádnou síťovou aktivitu [4].

Kromě toho existují další faktory, jako je mechanizmus prefetchingu, kde se při sekvenčním čtení dostane do cache i další bloky v sekvenci, tento počet bloků se zdvojnásobuje - to může zlepšit účinnost cache [9].

### 2.4.4 Prezentace výsledků

Při testování výkonnosti souborových systémů je důležité, aby výsledky byly zobrazovány přesně a aby měly statistickou relevantnost, například pro dělání závěrů nebo k použití výsledků k validaci či potvrzení. Výsledky by měly obsahovat i podrobný souhrn konfigurace, prostředí a kroků testů.

Je doporučeno použít intervaly spolehlivosti, nikoli pouze standardní odchylky. Standardní odchylka je míra variability mezi běhy. Polovina intervalu spolehlivosti udává s jakou šancí se pravá hodnota pohybuje od mediánu. Pro interval spolehlivosti s malým počtem běhů je dobré použít t-studentovo rozdělení (centrální limitní věta zde neobstojí s tak malým vzorkem) a směrodatná odchylka není známa, resp. je vypočítána ze vzorku. Pro přesnost t-studentova rozdělení, co nejvíce podobajícího se normálnímu rozdělení, je dobré použít 30 a více běhů v případě příliš velké chybovosti [4]. Vysoké intervaly spolehlivosti nebo ne-normální rozdělení by signalizovaly chybu softwaru nebo přímo testu výkonnosti. Polovina intervalu spolehlivosti by měla být 5% a méně (bráno z odhadovaného průměru). Krajní statistické případy by měly být zdůvodněny.

Kapitola 3

# Analýza existujících SW řešení pro testování systémů souborů

*V této kapitole popíši nejpoužívanější softwarová řešení pro jednotlivé kategorie testů výkonnosti. Zaměřuji se více na mikro-testy výkonnosti, které jsou předmětem následného vlastního řešení.*

## 3.1 Makro testy výkonnosti

Makro testy výkonnosti jsou testy, které simulují zátěž více podobnou reálným podmínkám. Zde popisuji tři řešení: Postmark, testy kompilace a The Andrew test.

Postmark je test výkonnosti, který testuje zátěž malých souborů s krátkou životností, zátěž typickou např. pro email [4]. Používá mix datových a metadatových operací nad systémem souborů [4]. Je intenzivní spíše na počet operací než na celkové přenášené množství dat a tím pádem testuje spíše operace systému souborů než I/O možnosti systému souborů či hardwaru, který je využíván systémem souborů [10].

Testy výkonnosti pomocí kompilace testují rychlost kompilace různých kódových repositářů (např. linux kernel, apache, openssh). Pro použití jako testu výkonnosti systému souborů je komplikace, že jsou intenzivní na CPU, což může mít význam pro systémy souborů s významnou CPU částí jako šifrování nebo komprese. Může být i problém přenositelnosti testu na různých systémech, které se mohou lišit jinou sadou nástrojů kompilace, jinými optimalizacemi či konfigurací [4].

Dalším populárním testem je The Andrew test. Testuje výkonost nad adresářovou strukturou nějakého zdrojového kódu. Skládá se z 5 částí: vytvoření stejné adresářové struktury v cílovém adresáři, zkopírování všech souborů, provedení `stat` operace nad všemi soubory v cílovém adresáři, přečtení všech dat a následné provedení kompilace pomocí `make`. Má podobné nedostatky jako kompilační testy, protože se jedná z podstaty o kombinaci kompilačního testu a mikro testu výkonnosti. Jeho největší slabinou je jeho nedostatečnost pro moderní systémy, protože využívá velmi malou zátěž, kdy se mohou všechny jeho kroky odehrát pouze v cache [4].

## 3.2 Mikro testy výkonnosti

V této části popisu jsou uvedeny 3 nejrozšířenější nástroje pro mikro testování výkonnosti: Bonnie, IOzone a FIO. Pro připomenutí, mikro testy výkonnosti jsou testy, které testují izolované jednotlivé I/O operace s velmi jednoduchou zátěží, která je syntetická a je v minimálním rozsahu tak, aby test bylo možné vůbec provést.

V této práci se budu věnovat primárně mikro-testům výkonnosti. Je to z důvodů jednoduchosti a toho, že na výstupu je jedno číslo (resp. interval spolehlivosti pro střední hodnotu), které dobře analyzuje testovaný systém souboru jednou metrikou a je pak v rámci této jedné metriky dobře porovnatelné s jinými systémy, či i s jinými parametry systému souboru (např. velikost bloku).

### 3.2.1 Bonnie a Bonnie++

Bonnie, který byl vyvinut v roce 1988 [11], provádí testy na jednotné velikosti souboru. Výchozí velikost souboru je 100 Kb, avšak sám autor navrhuje větší velikost pro moderní systémy. Jsou reportovány metriky jako bajty zpracované za sekundu, za CPU sekundu a uživatelský a systémový čas CPU. Testy zahrnují:

- Sekvenční výstup: Nejprve je soubor zapsán po znacích pomocí `putc`, a následně po blocích pomocí `write(2)`. Poté je každý blok dat přečten pomocí `read(2)`, nastaven dirty příznak a přepsán pomocí `write(2)`.
- Sekvenční vstup: Soubor je přečten po znacích a následně po blocích.
- Náhodný přístup: Je nastavena náhodná pozice v souboru, z níž se následně přečte blok a v 10 % případů je přepsán. Běží celkem 4 procesy paralelně, každý vykoná 4000x `lseek`. Tyto parametry jsou natvrdo v kódu.

Aby nebyly všechny dotazy uspokojeny z cache, je nutná dostatečná velikost souboru. Jinak se mohou všechny operace odehrát pouze s pomocí cache [4]. Doporučuje se použít alespoň velikost souboru 4x dostupné operační paměti [4].

Bonnie++ [12] je pak nástupcem Bonnie napsaný v C++ oproti C u Bonnie. Umožňuje testovat soubory větší než 2 Gb. Také je schopný testovat i jiné operace, jako jsou `create()`, `stat()` a `unlink()`. Dále umožňuje výstup v CSV i HTML.

Jedná se o celkově známé testy výkonnosti [10], nicméně se ve studii příliš nevyskytují [4]. To je možná z několika důvodů:

- Používají vestavěný OS pseudo-náhodný generátor, což ztěžuje objektivní testování různých OS. Z pohledu této práce je to však nepodstatný nedostatek, protože se zaměřuji pouze na operační systém Linux.
- Je zde velmi malá možnost parametrizace [10], lze nastavit pouze velikost souboru.
- Čtení a zápisové testy po jednom znaku nejsou příliš vypovídající testy na systém souboru, protože využívají bufferované stdio operace `getc` a `putc` bez následné operace flush.

### 3.2.2 IOzone

IOzone [13] je dalším nástrojem pro testování výkonnosti souborových systémů. Je napsán v ANSI C s použitím standardů POSIX, což zaručuje dobrou přenositelnost mezi různými operačními systémy. IOzone měří časy provádění operací, jako je čtení, zápis, opakované čtení a zápis s různými režimy operací, včetně náhodných, sekvenčních a zpětných. Výsledky z IOzone lze exportovat do Excelu.

Podporuje využití systémového rozhraní, nmap, mix souborového IO a nmap IO, direct IO a async IO. Direct IO využívá standardní flag při volání O_DIRECT. Async IO využívá funkce `aio_write` a `aio_read` z knihovny `aio.h`. Umožňuje zahrnout funkce `fsync()` a `fflush()` do měření [14].

IOzone nepodporuje kombinaci čtení a zápisu nebo kombinaci sekvenčních a náhodných přístupů v rámci jednoho měření. IOzone je charakterizován jako nástroj určený pro ladění souborového systému, nikoliv jako test výkonnosti, který dokáže modelovat pracovní zátěž souborového systému [10].

### 3.2.3 Fio

Fio (Flexible I/O tester) je dalším nástrojem pro testování výkonnosti I/O operací. Tento nástroj umožňuje velmi podrobnou konfiguraci jednotlivých testů a je vhodný pro testování různých scénářů zátěže, především díky velké možnosti nastavení statistických údajů celkového testu, včetně zvolení distribuční funkce a implementace pseudo-náhodného generátoru. Nástroj podporuje operace čtení a zápisu s možností nastavit sekvenční nebo náhodný přístup a různý mix typu operace a přístupu. Dále umožňuje zvolit použití vyrovnávací paměti nebo přímý přístup přes `O_DIRECT` flag. Fio také nabízí snadné měnění množiny funkcí systémového rozhraní, se kterými k I/O přistupuje (`sync`, `psync`, `libaio` či `nmap` a mnoho dalších). Podporuje vícevláknovost, výchozí nastavení používá `fork`, ale lze změnit na POSIX vlákna a volání `pthread_create`.

### 3.2.4 Porovnání

Bonnie, IOzone a Fio jsou tři nejznámější nástroje pro testování výkonu souborových systémů. Bonnie se zaměřuje na testy souborů o jednotné velikosti, IOzone na měření časů provádění různých operací a Fio umožňuje velmi podrobnou konfiguraci jednotlivých testů a je vhodný pro testování různých scénářů zátěže.

IOzone podporuje využití různých typů IO, jako je souborový IO a direct IO. Je vhodný pro ladění souborového systému. Bonnie++ umožňuje testovat větší soubory a operace, jako je `create()`, `stat()` a `unlink()`. Dále umožňuje výstup v CSV a HTML. Fio (Flexible I/O tester) umožňuje detailní konfiguraci testů pro různé scénáře zátěže. Podporuje vícevláknovost a umožňuje nastavení statistických údajů celkového testu.

Porovnání těchto nástrojů je komplikované, protože každý z nich se zaměřuje na jiné aspekty testování. Bonnie se zaměřuje na testování souborů o jednotné velikosti a je vhodný pro testování rychlosti přenosu dat. IOzone se zaměřuje na měření časů provádění různých operací a je vhodný pro testování rychlosti operací na souborovém systému. Fio umožňuje velmi podrobnou konfiguraci jednotlivých testů a je vhodný pro testování různých scénářů zátěže.

Celkově lze říci, že pro testování výkonu souborových systémů je nejlepší zvolit nástroj podle konkrétních potřeb a požadavků na testování.

## 3.3 Generátory zátěže

Za zmínku stojí i generátory zátěže, které pomáhají s generováním konfigurovatelné zátěže během testování výkonu. V praktické části nebudou použity, protože jsou spíše podkladem pro makro testy výkonu. Pro mikro testy výkonu není nutné simulovat reálnou zátěž.

Prvním generátorem zátěže je IOmeter, vyvinutý společností Intel v roce 1998 [15]. Autoři prohlašují, že umožňuje emulovat diskovou nebo síťovou I/O zátěž jakéhokoli programu nebo testu výkonu. Dokáže generovat zátěže pro několik systémů najednou (přes síť). Dokáže simulovat i jednotlivé parametry jako diskové a síťové řadiče, zpoždění sběrnice nebo rychlost síťového připojení.

Dalším generátorem je FileBench [16], který slouží k vygenerování zátěže podobné typickým serverům jako mailový, webový nebo databázový. Má velmi bohatou možnost specifikace pomocí konfiguračních skriptů. Lze jej použít jako podklad pro mikro testy zátěže, podobně jako Bonnie nebo Andrew [4].

Kapitola 4

# Návrh nového testovacího nástroje

*Tato kapitola se zaměřuje na návrh a popis implementace nového nástroje pro testování výkonnosti různých systémů souborů v operačním systému Linux v programovacím jazyku C++. V první části kapitoly budou popsány jednotlivé funkce a vlastnosti, které vycházejí z předchozí analýzy. Dále bude popsán způsob použití tohoto nástroje, tedy konfigurace a ovládání z příkazové řádky. Poté bude popsána implementace řešení, včetně jeho architektury, designu, způsobu testování a integrace do operačního systému. Celý projektový repozitář je součástí příloh, pro vývoj byl použit Git repozitář hostovaný na fakultním GitLabu. Projekt je pojmenován v rámci vývoje jednoduše jako FSBench a nástroj na příkazové řádce má název* **fsb**.

## 4.1 Motivace a zdůvodnění vytváření nového nástroje

Podle poznatků ze studií [4][3] není vytváření nového nástroje příliš žádoucí, a to z několika důvodů:

- jeho výsledky nemusí být porovnatelné s různými statistickými údaji z jiných studií
- jeho použití může být limitováno, protože nedokáže reprodukovat výsledky v jiných verzích kompilátorů nebo operačních systémů, například pokud používá knihovny operačního systému pro generování náhodných čísel jako třeba Bonnie [4].

Nový nástroj musí proto tyto problémy překonat a minimalizovat. Obecně lze tvrdit, že tento nástroj by měl být spíše jednoduchý, ale striktně dodržovat metodiku testování výkonnosti nastavenou například tak, aby nástroj bylo možné použít na jakémkoli operačním systému a jeho výsledky byly porovnatelné nejen s výsledky tohoto nástroje, ale i s výsledky ostatních nástrojů a studií [4].

Tento nástroj bude mít integrované základní modely statistické analýzy: průměr, směrodatnou odchylku, tak i výpočet intervalu spolehlivosti pro střední hodnotu. Nástroj tedy bude měřit několik vzorků stejného testu, ze kterých následně tyto údaje vypočítá. Kromě toho bude na výstupu poskytovat dané vzorky pro další statistickou analýzu.

## 4.2 Požadavky na nový nástroj

Nejprve je potřeba stanovit požadavky na ovládání, výstup a hlavní funkcionalitu. Z těchto požadavků bude následovat návrh jednotlivých technologií a architektury nového nástroje. Tento nástroj je jeden z primárních cílů této práce.

Z velmi jednoduchého pohledu tento nástroj dělá to, že jednu danou měřenou operaci (např. čtení souboru) provede v dostatečném počtu opakování a celkově tento proces změří a udá hodnotu průměrné doby trvání jedné operace (např. zde průměrnou dobu přečtení jednoho souboru). Tento jednoduchý pohled skýtá důležitý požadavek a to, aby toto iterování bylo co nejvíce *rychlé* a aby *nealokovalo zbytečně paměť* navíc než potřeba pro samotný běh testu.

Dalším důležitým poznatkem je, že tento nástroj by měl být *rozšiřitelný na ostatní operační systémy*. Tato práce je zaměřena jen na operační systém Linux, ale návrh nástroje by měl myslet na případné rozšíření na ostatní operační systémy. Také by měl umět jednoduše *měnit použité knihovny* na přístup k I/O operacím, tedy případné možnosti testovat i nativní knihovny daného systému souborů pro testování v rámci jednoho systému souborů.

Dále by nástroj měl umět provádět dané testy výkonnosti i v *paralelním režimu ve více vláknech*. Je to malý přesah do makro-testu výkonnosti, který umožní testovat paralelní (nezávislé) operace a tím dosáhnout co nejvyššího zatížení na systém souborů.

Nástroj by měl provádět měření několikrát pro získání dostatečného počtu vzorků. Počet těchto vzorků by měl být alespoň 30, případně více, pokud by chyba měření (standardní směrodatná odchylka) byla velká[4]. V případě měření I/O operací je tento požadavek nezbytný, protože tyto operace mohou mít odlehlé hodnoty[1]. Tyto hodnoty mohou vznikat jakoukoli komponentou vnitřní a mezi operačním systémem, systémem souborů a diskovým médiem.

Každý běh iterace (např. vytvoření 1000 souborů) by měl běžet nezávisle na ostatních testech a měl by se skládat z přípravného kódu, který není měřen, následně z měřené sekce a poté i částí pro vyčištení do původního stavu (např. smazat vytvořené soubory). Nezávislost je řešena na úrovni implementace každého dílčího testu, funkcionalitu testu by tedy měla být definována jako trojice funkcí příprava, běh a úklid.

Výsledky měření prováděním tímto nástrojem by měly být prezentovány jako výstupy jednotlivých cyklů jako vzorek měření. Tento údaj by měl být v přesném čase nejlépe v mikrosekundách, které jsou dostatečně přesné na všech moderních operačních systémech a hardwaru. V případě nanosekund může docházet k chybě měření při velmi drobných neiterativních operacích, protože frekvence měření nemusí být tak drobná.

Návrh je zaměřen pro administrátory a z pohledu používání nástroje je nutné zajistit možnost jednoduchého sestavení a používání v rámci konvence podobnému jiným nástrojům. Konfigurace by měla být přehledná a jednoduchá na úpravu. Jednoduchá příručka k použití by měla být integrována přímo v programu. Program by měl podporovat *základní protokol* o prováděných operací s možností zobrazení na standardních výstupech a/nebo do souboru.

Nástroj nad operačním systémem vykonává operace pouze nad soubory, které vytvořil tento program. To bude aplikovatelně zaručeno, nicméně pro zaručení bezpečnosti bude nástroj umožňovat sestoupení administrátorských oprávnění s možností zvolení Unix skupiny a uživatele pod jakým program poběží. Také by mělo být zaručeno start běhu v složce, která není vlastněna administrátorských účtem (např. v /tmp).

Shrnutí těchto požadavků na tento nástroj v bodech:

- Rychlé provádění operací kolem měření bez zbytečné alokace paměti.
- Nástroj je navržen tak, aby byl rozšiřitelný pro použití na různých operačních systémech.
- Nástroj provádí opakované měření dané operace v dostatečném počtu opakování, aby získal přesné výsledky.
- Nástroj by jednotlivé testy měl implementovat s možností přípravy na test a úklidu po testu.
- Nástroj by měl podporovat paralelní vykonávání iterací testu.
- Konfigurace nástroje je přehledná a jednoduchá na úpravu. Program obsahuje jednoduchou příručku k použití.
- Výsledky měření jsou prezentovány jako výstupy jednotlivých cyklů, které představují vzorek měření. Nástroj umožňuje přesné měření v mikrosekundách.
- Nástroj provádí operace pouze nad soubory, které vytvořil tento program.

[1] odlehlá hodnota bývá hodnota ležící několikanásobek vyběrové směrodatné odchylky.

## 4.3 Popis nového nástroje

Jako zvolený programovací jazyk je používán C++ ve verzi 17[17], neboť nabízí podporu pro všechny požadavky a dobře umí pracovat s operacemi nad operačním systémem. Je přenositelný do jiných operačních systémů. Další možností by bylo použití skriptovacího jazyka, např. BASH[18], nicméně pro jednodušší a škálovatelný vývoj se od této možnosti upouští.

Hlavním rozhraním pro uživatele je zvoleno rozhraní příkazového řádku (ang. CLI[2]). Další možnosti uživatelského rozhraní jsou zbytečné, neboť se předpokládá, že uživatel ovládá příkazový řádek.

Tento nástroj by měl být použitelný na jakémkoli systému souborů a na jakémkoli hardwaru. Nástroj testuje výkon systému souborů přes standardní Unixové rozhraní, jako jsou např. funkce `open`, `close`, `read` nebo `write`.

### 4.3.1 Vice vláknovost

Nástroj má možnost běžet vícevláknově a to pomocí paralelního rozdělení běhu jednotlivých testů (iterací) v pracovních vláknech, jejichž počet je konfigurovatelný. To umožní simulovat více reálného provozu jako vliv paralelních I/O operací. To bude implementováno pomocí C++ hlaviček pro práci s více vlákny `<thread>`.

### 4.3.2 Měření času

Pro dodržení přesnosti a stability měření je zapotřebí přesný způsob měření, který je přesný alespoň na řády mikrosekund. Funkce, která získává čas, by měla být nejlépe monotónní[19]. Monotonní hodiny nejsou ovlivněny změnou času v OS a jejich hodnota je vždy rostoucí.

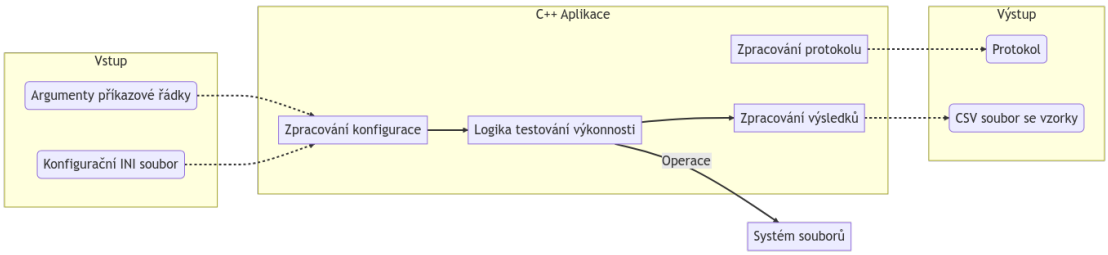
Hodiny měření mají i nějakou frekvenci snímání času, kde u rychlých operací může dojít k výraznému zkreslení měření. Proto pro výpočet běhu testu výkonnosti je podstatné ho nechat běžet tolikrát, aby se dalo vypočítat jeho rychlost v rozumných jednotkách (např. mikrosekundách). Toto kritérium lze zajistit tak, že se daný test provede vícekrát, bude se měřit uběhlý čas běhu za všechny iterace a následně se vypočítá průměrný čas jedné iterace. Pro mikro testy výkonnosti je dobré jeden test nechat běžet tolikrát, aby běžel celkově alespoň 100 ms až 1 sekundu [19]. Tento čas také zajistí, že test poběží v ustáleném stavu.

Dalším krokem, jak dodržet stabilitu, je analyzovat nahodilé chyby měření vzniklé převážně externím vlivem. Externí vlivy mohou být např. jiné procesy zápasící o prostředky, nedeterministické plánování I/O prostředku atd. [19]. Pro jejich analýzu lze použít porovnání mezi běhy v rámci intervalu spolehlivosti a tedy určení s jakou spolehlivostí jsou dva výsledky měření s významným rozdílem.

### 4.3.3 Vstupní konfigurace

Program bude mít možnost jednoduché konfigurace, jak z pohledu uživatele, tak i v případě dalšího rozšiřování programu. Konfigurace nástroje je nastavitelná přímo jako parametr na příkazové řádce nebo v konfiguračním souboru, který bude v INI formátu. Platí, že se hodnota se stejným klíčem jak v konfiguračním souboru, tak na příkazové řádce bere ta hodnota na příkazové řádce. Rozhraní argumentů na příkazové řádce se bude držet očekávaných konvencí, především při nastavení klíče bude použit prefix `--` oproti klíči v konfiguračním souboru, argument na příkazovém řádku pak bude oddělen od klíče mezerou. Touto konfigurací půjde nastavit všechny parametry používané v nástroji jako počet běhů, jaký test výkonnosti provést, parametry testu výkonnosti (počet souborů, velikost souborů, atd.) nebo adresář, kde test provádět a další.

[2] Command Line Interface




**Obrázek 4.1** Vnější architektrua

### 4.3.4 Výstup uživateli

Výstup souboru bude dvojí: první současný je protokol, označení časem, úrovní zprávy (chyba, informace) a vlastní zprávou. Tyto řádky protokolu bude možné zobrazit do standardního výstup tak i do souboru nebo jejich kombinace, konfigurovatelné budou cíle ukládání, minimální úroveň protokolu. Dalším výstupem bude CSV[3] soubor, kde budou výsledky jednotlivých vzorků měření. Lokalita tohoto souboru bude také konfigurovatelná.

### 4.3.5 Použité algoritmy

Většina funkcionalit jde implementovat triviálními algoritmy, například pro výpočet průměrné hodnoty nebo práci s řetězci při převádění konfigurace. Pro statistické výpočty jsou zajímavé použité algoritmy pro možnost výpočtu intervalu spolehlivosti pomocí t-studentova rozdělení, kde se využijí dva algoritmy pro výpočet tohoto intervalu spolehlivosti, resp. inverzní distribuční funkce t-studentova rozdělení [21][22].

### 4.3.6 Bezpečnost a spolehlivost

Nástroj bude umožňovat vytváření a mazání souborů i adresářů. Je tedy vhodné omezit, s jakými oprávněními se nástroj spouští, a tím omezit možné poškození adresářové struktury operačního systému. Nástroj bude umožňovat zvolit, pod jakým uživatelem a skupinou bude spuštěn. Na administrátorovi, který bude tento nástroj používat, bude, aby zajistil vytvoření uživatele s minimálními oprávněními, aby daný uživatel mohl provádět úpravy na souborech v testovaných adresářích.

## 4.4 Architektura nástroje

Dle požadavku a popisu tohoto nástroje bude navržena architektura, která nejlépe podpoří implementaci požadavků. Z celkového pohledu je na vstupu konfigurační soubor a argumenty z příkazové řádky. Na výstupu bude protokol (v STDOUT[4] nebo v textovém souboru) a CSV soubor s výsledky. Samotný nástroj se skládá z hlavních částí, jako je zpracování konfigurace, samotný běh testů a následné zpracování výsledků. Část logiky testování výkonnosti provádí operace nad systémem souborů.

[3] ang. Comma-separated values = hodnoty oddělené čárkami je formát souboru pro uložení tabulkových dat, kde každý řádek reprezentuje řádek tabulky a na každém řádku jsou texty tabulkových buněk odděleny čárkou[20]

[4] standardní výstup

### 4.4.1 Konfigurace

Konfigurace bude tvořena s možností rozšíření o nový typ vstupu. Konfigurace bude sestavena pomocí návrhového vzoru stavitele, kdy se přidají jednotlivé zdroje vstupu, které budou dědit společné rozhraní, které bude vracet seznam klíč-hodnota. Následně se instance konfigurace postaví tak, že se projdou všechny tyto zdroje a zavolá se jejich společné rozhraní a tím se získá sjednocení všech hodnot. Tato sestavená instance bude fungovat jako neměnná hodnota a bude jediná na celý program.

Z konfigurace bude možné získávat jednotlivé hodnoty pomocí pomocných metod pro získání hodnoty dle typu (např. číslo, text, pravdivostní hodnota). To se pak může využít u tříd, které budou popisovat konfiguraci nějaké dílčí komponenty programu.

### 4.4.2 Protokol

Instance protokolu se bude vytvářet po konfiguraci, neboť potřebuje sestavenou konfiguraci na vytvoření instance. Protokol bude definován konfigurací jako minimální úroveň protokolu a seznamu použití výstupních formátů. Nástroj podporuje výstupní formát do terminálu (STDOUT) a do souboru. Těmito informace se sestaví globální instance protokolu.

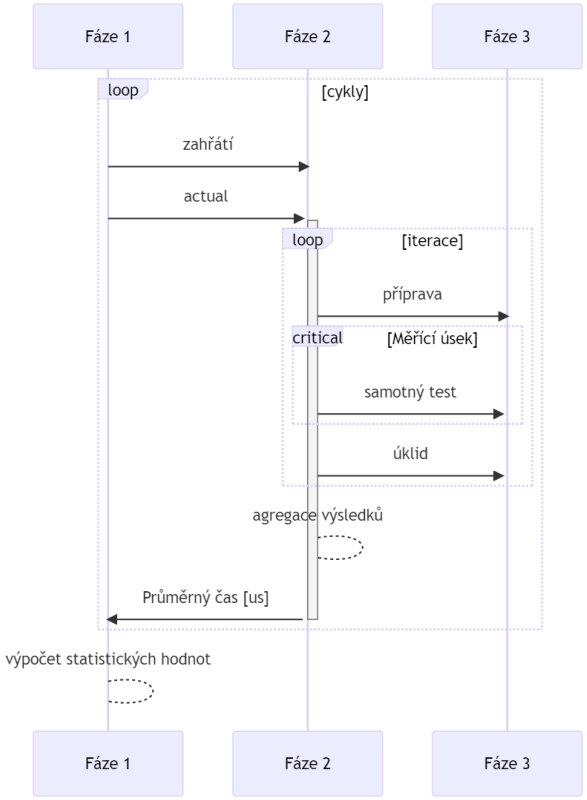
### 4.4.3 Logika testování výkonnosti

Samotný proces testování výkonnosti je rozdělen logicky do tří fází. Každá fáze je vnořena do další fáze. V jednoduchosti se jedná o dva až tři vnořené cykly. Fáze jsou (od vnější k vnitřní): cyklus (směr vzorku), iterativní běh (sekvenční nebo paralelní) a běh samotného testu výkonnosti.

Od nejvíce vnitřní fáze: samotný běh iterace, který bude definován abstraktní třídou reprezentující tento test (např. test vytváření souboru nebo test čtení souboru), který bude definovat tři funkce a to pro přípravu testu, samotný test a úklid po testu. Samotná funkce běhu se většinou bude skládat z iterace nad nějakou množinou souborů (tento počet bude uživatelsky nastavitelný) a právě doba jeho trvání by měla být ideálně stovky mikrosekund, z důvodu, že právě tento iterativní běh se měří. Tato fáze bude vracet rozdíl mezi časem před voláním funkce samotného testu a po tomto volání.

Další fáze je fáze iterativního běhu, tato mezifáze bude nabízet možnost běhu sekvenčně a nebo běhu paralelně ve více vláknech. Sekvenční běh bude další iterace v fázi samotného běhu. Paralelní běh rozdělí počet iterací mezi daný počet vláken (zde bude možnost použít indikaci počtu konkurence hodnotou nula pro použití výchozího nastavení počtu hardwarové konkurence) a spustí je (spustí N-1 vláken) a v hlavním volaném vlákně spustí také iteraci, následně počká na dokončení běhu a spojí výsledky. Tato fáze je zde z důvodu zajištění dostatečné doby běhu testu a dostání stavu měření do ustáleného stavu, pro ještě lepší statistickou přesnost, což by mělo být zaručeno během alespoň 100 ms, ideálně 1 s [19]. Na výsledku této fáze bude průměr běhu jedné iterace ze všech iterací.

Poslední fáze je opakování fáze iterativního běhu v několika cyklech (30 a více) pro nasbírání vzorků měření. Tato fáze se skládá ze dvou částí, první část je "zahřátí" testu a následně samotný běh iterativní fáze. První část je běh načisto, kdy se celkový iterativní běh několikrát provede, mělo by provést tolikrát tak, aby se ve vzorcích neobjevovali odlehlé hodnoty. Tato fáze zahřátí je potřeba, protože I/O operace často používají funkci skryté paměti. Tato část zahřívání lze konfiguračně vypnout, pro např. testování právě rychlosti bez využití skryté paměti. Druhá část je samotný běh, který jako takový je časován pro získání informace o rychlosti běhu fáze iterativního běhu. Výsledek této fáze je jako dvojice doby trvání celkového testu výkonnosti a výsledku (vzorku) z fáze iterativního běhu.




**Obrázek 4.2** Logika běhu testu

### 4.4.4 Zpracování výsledků

Z fáze sběru měření vznikne výsledek, který se skládá z dvojice dob měření a průměrných hodnot jednoho běhu testu. Z obou těchto metrik se vypočítají statické hodnoty jako průměrná hodnota, střední hodnota, standardní směrodatná odchylka a intervaly spolehlivosti 99% a 99,9%. Hodnota doby měření se předá pouze do protokolu, vzhledem k tomu, že nese jen informaci o běhu aplikace. Hodnota průměrné hodnoty běhu se uloží v CSV souboru pro případné další zpracování.

## 4.5 Integrace do systému

Nástroj bude předáván zdrojovým kódem a pro jeho kompilaci se využije nástroj `make`. Kompilace je prováděna pomocí kompilátoru GNU g++. Pro použití vícevláknovosti jsou potřebné knihovny nalinkovány pomocí argumentu g++ `-pthread`. Toto řešení je z důvodu jednoduchosti a případné možnosti změnit způsob nebo parametry kompilace. Nástroj je tak transparentní pro použití.

# Kapitola 5
# Implementace nástroje

Tato kapitola se bude věnovat implementaci nástroje pro testování výkonnosti souborů. Budou zde popsány jednotlivé kroky implementace dle navhrnuté architektury. Nejprve bude popsána implementace zpracování konfigurace a logiky testování výkonnosti. Dále bude uveden popis implementace zpracování výsledků testování. Uživatelská příručka je přiložena v příloze B.

## 5.1 Programovací jazyk C++

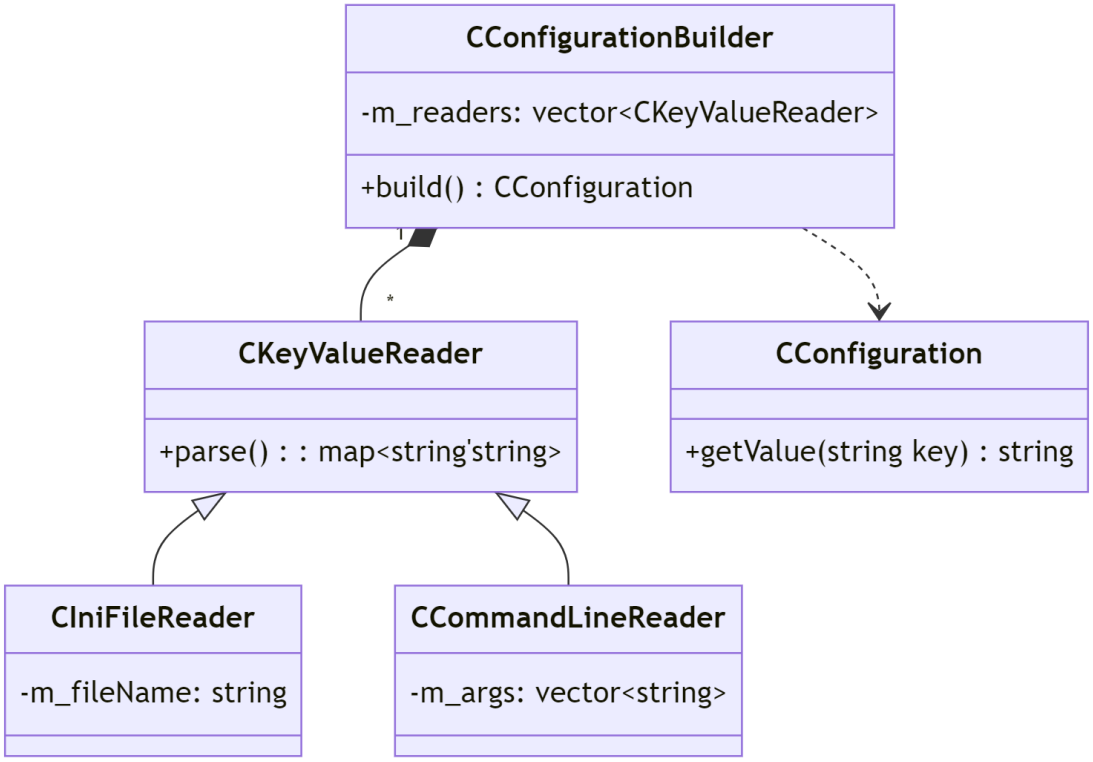
C++ byl zvolen jako jazyk pro implementaci nástroje pro testování výkonnosti souborů z několika důvodů. Za prvé, C++ je výkonný jazyk, což je důležité pro nástroj zaměřený na testování výkonnosti. Za druhé, C++ je široce používaný jazyk pro vývoj systémového softwaru, což znamená, že existuje mnoho knihoven a nástrojů pro práci se souborovým systémem a měření času. Za třetí, C++ nabízí možnost práce s vícevláknovým programováním, což je důležité pro implementaci paralelního zpracování testů. Celkově je C++ dobrou volbou pro nástroje pro testování výkonnosti a systémový software vůbec. Pro vývoj bude využita verze C++17.

### 5.1.1 Použité knihovny

Pro snímání času je použita hlavička `chrono`, která nabízí práci s přesnými systémovými hodinami a také umožňuje snadnou práci s časovými jednotkami. Dále jsou použity hlavičky pro zpracování pro více vláken jako `thread` a `mutex`. Pro jednoduchou práci se souborovým systémem je použita hlavička `filesystem`.

## 5.2 Konfigurace

Způsob sběru konfigurace a její použití v programu jsou velmi obecného designu. Konfigurace byla volně inspirována konfigurací v rámci .NET [23]. Hlavním třídou pro přístup ke konfiguraci je `common::CConfiguration`, která je tvořena jako databáze seznamu klíč-hodnota. Dalším důležitým mechanismem je přidávání datových zdrojů, ze kterých se tato databáze vytvoří. To je zajištěno děděnými třídami z abstraktní třídy `common::CKeyValueReader`. Program má implementováno čtení z příkazové řádky ve formátu `--klíč hodnota` v rámci `common::CCommandLineReader` a čtení z INI souboru pomocí `common::CIniFileReader`. Pro snadné vytvoření instance `common::CConfiguration` je potřeba použít stavitel `common::CConfigurationBuilder`, kterým lze přidávat jednotlivé datové zdroje, kde na jejich pořadí závisí a to v případě stejného čteného klíče, jeho hodnota se přepíše později přidaným zdrojem. V tomto programu je výchozí nastavení nejprve použití čtení z příkazové řádky a poté ze souboru INI s názvem `settings.ini`.




**Obrázek 5.1** Třídový diagram konfigurace

Pro snadné použití konfigurace v kódu třídy `common::CConfiguration` nabízí možnost převádět hodnoty do C++ primitivních typů jako `std::string`, `int` nebo `bool`. Toho se pak využívá ve třídách reprezentujících logické seskupení konfigurace (např. `common::CLoggerConfig` nebo z aplikace nastavení `bench::CBenchRunConfig`).

## 5.3 Protokol

Způsob sběru protokolu (angl. logu) je řešen pomocí třídy `common::CLogger`. Tato třída poskytuje funkce na zápis protokolu bez ohledu na použitou technologii (standardní výstup, soubor). Hlavní funkcí zápisu je `void common::CLogger::log(common::ELogLevel, std::string)`. Následně jsou zde pomocné funkce na zápis jednotlivých úrovní protokolu (nad `common::ELogLevel`).

Protokol nabízí konfiguraci formátu výstupu a možné výstupní formátory děděné třídy z `common::CLogSink`, které zprostředkovávají zápis do výstupního média. V tomto projektu jsou podporovány: výstup na standardní výstup `common::CTerminalLogSink` a do souboru `common::CFileLogSink`. Pomocí konfigurace je možné nastavit výstup na jeden i více těchto výstupních formátorů.

## 5.4 Způsob dodržení přesnosti měření

Pro dodržení přesnosti a stability měření je třeba dodržet několik zásad při implementaci. První zásadou pro dobré výsledky je použití přesného nástroje na měření času. V rámci knihovny `chrono` nabízí C++11 `high_resolution_clock`, `system_clock` a `steady_clock` jako způsoby vyhodnocení aktuálního času [24]. Použitý typ času závisí na kompilátoru i možnostech OS a hardwaru. Pokud jde o měření intervalů, nejlepší je použít monotónní hodiny [19], které se dají získat pomocí `steady_clock` z `chrono` API. Monotonní hodiny nejsou ovlivněny změnami času v OS a jejich hodnota je vždy rostoucí. Program získává délku běhu jako rozdíl dvou časů v nanosekundách z `std::chrono::steady_clock`.

Je nutné, aby měření času bylo nezávislé na přesnosti měření času operačním systémem či hardwarem. Jeden test výkonnosti by tedy měl trvat alespoň stovky mikrosekund s alespoň třemi platnými číslicemi.

## 5.5 Testy výkonnosti

Jednotlivé testy výkonnosti jsou implementovány jako odvozené třídy z abstraktní třídy `bench::CFileBenchmark`. Tato třída nabízí tři popsané metody pro běh testu a to `void setup()`, `void iteration()` a `void cleanup()`.

První vytvořený test je na metadatovou operaci vytvoření souboru, logika je obsažena v třídě `bench::CCreateFileBenchmark`. Tento test v přípravné fázi vytvoří názvy souborů pro vytvoření. V průběhu testu tyto testy vytvoří a v úklidové metodě tyto soubory smaže. Lze nastavit počet testů pomocí konfigurace *fileCount*.

Pro datové operace jsou zde implementovány dvě třídy `bench::CReadFileBenchmark` a `bench::CWriteFileBenchmark` pro test čtení souboru, resp. zápis do souboru. Oba tyto testy umožňují nastavit počet souborů (*fileCount*), velikost bloku v KB (*blockSize*), počet bloků (*blockCount*) a jestli použít náhodný přístup (*random*).

Tyto testy využívají Unixové funkce na operace se soubory z hlavičky `<unistd.h>`, testy využívají funkce jako `open`, `close`, `read`, `write` nebo `lseek64`.

Běh těchto testů je vykonáván tak, že se nejdříve vytvoří instance testu přes návrhový vzor továrna pomocí třídy `bench::CFileBenchmarkFactory`. Poté se zavolá metoda na přípravu testu, získá se aktuální čas pomocí `chrono::steady_clock` proběhne běh testu a opět se získá aktuální

čas. Následně se zavolá úklid po testu a vrátí se rozdíl koncového času od počátečního času. Tento celý kód je vykonáván v rámci iterativního běhu.

## 5.6 Iterační běh

Způsoby iterativního běhu jsou definovány abstraktní třídou `bench::CIterationExecution`, která obsahuje metodu pro spuštění, která vrací průměrný čas jedné iterace a jako parametr si bere ukazatel na funkci běhu iterace. Samotný iterativní běh je vytvořen z továrny `bench::CIterationExecutionFactory`, poté je spuštěna daný způsob provedení iterací. První způsob provedení iterací je triviální sekvenční způsob, kde se daná testovaná funkce provede v daném počtu iterací (*iterations*), tento způsob implementuje třída `bench::CSequentialExecution`. Výsledky se agregují jednoduše výpočtem jejich průměrné hodnoty.

Dalším způsobem běhu iterací poskytuje třída `bench::CParallelExecution`. Tento běh iterací je implementován jako paralelní běh dané testované funkce, počet iterací se rozdělí mezi požadovaný počet vláken (*threadCount*), pokud je nastaven tento počet jako 0, je použito výchozí nastavení systémové konkurence pomocí `std::thread::hardware_concurrency()`. Celkový počet iterací by v tomto stylu měl být rozumný, to znamená, že se použije tento paralelní běh pouze pokud je tento způsob požadován (*multiThreaded*) a zároveň, že počet iterací přesahuje 10. Jinak se použije sekvenční způsob provedení. V běhu každého vlákna je zaručeno, že se tvořené soubory nebudou překrývat, tím, že se změní pracovní adresář do podsložky `/tread{tid}`, kde *tid* značí číslo vlákna. Celkově se vytvoří o jedno vlákno méně, protože jedno rozdělení iterativního běhu se provede na hlavním vláknu. Po dokončení běhu hlavního vlákna se na ostatní vlákna zavolá metoda `join()`, kterou se zajistí, že se všechna vlákna dokončila. Jednotlivá měření z volané testované funkce se přidávají do seznamu těchto měření, seznam je chráněn pomocí `std::mutex` pro zajištění přístupu vždy z jednoho vlákna. Na konci této metody se ze všech výsledků vypočítá průměrná hodnota.

## 5.7 Běh cyklu

Poslední fáze je běh celkového testu několikrát pro získání dostatečného počtu vzorků. Tato funkce se, jak již bylo zmíněno, skládá ze dvou částí. První fáze je zahrátí celkového běhu provedením iterativního běhu několikrát, nastavitelné pomocí konfiguračního klíče *warmupCycleCount*. Je také možné tuto část přeskočit pomocí konfigurace *useWarmup*.

Další částí je samotný běh iterativního běhu, u kterého je měřen čas celkového doby běhu. Celková doba běhu je ukládána v milisekundách. Samotný průměr času z iterativního běhu je převeden z nanosekund do mikrosekund. Tyto dva výsledky jsou sbírány napříč běhu cykly.

### 5.7.1 Výpočet statistických hodnot

Pro práci se statistickými výpočty a jejich exportem je zavedena třída `bench::CStatistics`, která je definována názvem statistiky, vzorky měření a konfigurací pro nastavení exportu. Tato třída nabízí možnost výpočtu průměrné hodnoty, směrodatné odchylky, ale i percentilů (jako např. střední hodnota) a výpočet intervalů spolehlivosti. Následně poskytuje rozhraní pro vypsání výběru statistických údajů do protokolu pomocí funkce `void bench::CStatistics::print( const std::shared_ptr<common::CLogger> &)`, a rozhraní pro export do CSV souboru funkci `void bench::CStatistics::exportStatistics()`.

## 5.8 Bezpečnost

Nástroj umožňuje zabezpečit operační systém před poškozením, např. smazáním důležitých souborů nebo případně i přepisem souborů. Je tedy nutné na začátku běhu nástroje mít logiku, která se zřekne administrátorských privilegií. Pro tyto účely budou sloužit konfigurační hodnoty na nastavení identifikátoru uživatele (*uid*) a identifikátoru skupiny (*gid*), případně bude možnost použít `root` účet, ale to bude nutné vynutit nastavením konfigurace *forceRootUser* jako pravdivou hodnotu. Také pro zajištění, aby nástroj nezačal ve výchozím stavu používat aktuální složku jako výchozí výstupní složku, bude nastaven pracovní adresář na složku pro účely dočasného souboru v adresáři `/tmp`.

Implementace této logiky v rámci C++ a Unix prostředí je jednoduchá. Pro zřeknutí se práv stačí zavolat `setuid(uid_t)` a `setgid(gid_t)` pro nastavení identifikátoru uživatele, resp. skupiny. Nástroj neřeší případné zřeknutí ze všech skupin, které mohou mít administrátorská oprávnění. Předpokládá se, že administrátor používá bezpečnostní konvence. Pro změnu pracovního adresáře lze použít funkci `chdir`.

## 5.9 Možné rozšíření do budoucnosti

Nástroj nabízí základní operační možnosti pro testování výkonnosti systému souborů. Nabízí také mnoho prostoru pro zlepšení.

### 5.9.1 Nové benchmarky

V budoucnu by bylo možné se zaměřit na vytvoření benchmarků pro další druhy I/O operací. Také by mohly být nové testy specializovány na testování jednoho konkrétního systému souborů a použít jeho konkrétní API.

### 5.9.2 Další možnosti běhu

V současnosti se implementuje jednovláknový sekvenční model a vícevláknový paralelní model iterativního běhu. Lze přidat i další tyto modely, např. běh v více procesech nebo běh na vzdáleném počítači pomocí SSH.

### 5.9.3 Výstupní diagramy měření

Pro lepší vizualizaci výsledků měření by bylo možné vytvořit diagramy, např. grafy, které by zobrazovaly průběh času a rychlosti operací v závislosti na velikosti souboru nebo počtu iterací.

### 5.9.4 Výpočet heuristiky pro měření

Nástroj by také mohl umět vypočítat potřebný počet iterací a počet zahřívacích cyklů pro optimální statistické výsledky.

### 5.9.5 Podpora dalších operačních systémů

Momentálně je nástroj implementován pouze pro Unixové systémy. V budoucnu by bylo možné přidat podporu pro další operační systémy, jako například Windows nebo macOS.

### 5.9.6 Podpora automatického nastavení prostředí

Nyní, aby se nástroj dal použít, musí se prostředí nastavit jako připojení disku a formátování disku souborovým systémem. Nástroj by v budoucnosti mohl toto nastavení automatizovat např. automatickým přeformátováním a připojením diskového média.

### 5.9.7 Vylepšené uživatelské rozhraní

Nástroj nabízí základní operační možnosti pro testování výkonnosti systému souborů, nicméně existuje prostor pro jeho vylepšení v oblasti uživatelského rozhraní. V budoucnu by například bylo možné vytvořit grafické rozhraní pro snadnější ovládání a vizualizaci výsledků měření.

Kapitola 6

# Testování výkonnosti systémů souborů

*Tato kapitola popisuje proces testování výkonnosti pomocí nového nástroje. Nejprve popíšeme prostředí testování a jeho nastavení. Následně bude popsáno provedení jednotlivých testů nad systémy souborů. Nakonec bude provedeno porovnání výsledků jednotlivých testů napříč systémy souborů.*

## 6.1 Prostředí testování

Volba vhodného prostředí je důležitý krok při porovnání výkonnosti. Hlavním požadavkem je dle specifikace nástroje, aby se jednalo o Linuxový systém. Dále by prostředí mělo být dostatečně rychlé, aby nebylo nouze o prostředky počítače a test mohl probíhat bez omezení výpočetních prostředků. Jednou možností je využití lokálního počítače s operačním systémem Linux. Avšak se v případě lokálního počítače špatně zařizuje, aby daný disk nebyl zastaralý a celkově systém přehlcený.

Proto připadá v úvahu druhá možnost a to použití cloudového řešení. To umožní vždy testovat na čistém prostředí. Zde může být nevýhoda, že servery i disky jsou ve virtuálním prostoru, to by mohlo mít vliv pokud by se testovala rychlost samotného disku, pro test systému souborů je právě výhodné a důležité, že se testuje na jednom typu diskového média, což je v cloudovém prostředí z podstaty zajištěno. Pro cloudové řešení byla zvolena platforma Digital Ocean[25]. Digital Ocean je platforma velmi jednoduchá bez výrazné komplexity a umožňuje snadné spuštění virtuálních počítačů (v rámci Digital Ocean nazývány jako Droplet) Dále obsahuje jednoduché vytvoření rozšiřitelných diskových médií, které lze snadno připojit k vytvořeným virtuálním počítačům.

V následujících sekcích bude uveden způsob nastavení tohoto prostředí do stavu, kdy je možné na tomto prostředí spouštět výkonnostní testování.

### 6.1.1 Definice prostředí

Pro potřeby testování výkonnosti systému souborů jsou zapotřebí následující prostředky: virtuální počítač a několik diskových médií připojených k počítači. Pro definici cloudového prostředí (infrastruktury) bude využit nástroj pro jednoduchou správu tohoto prostředí v procesu infrastruktury jako kód[1], to z důvodu jednoduchého vytvoření této infrastruktury a jejího následného zrušení. Jako dobrá volba přichází v úvahu nástroj Terraform[27] od společnosti HashiCorp.

[1] infrastruktura jako kód (ang. Infrastructure as Code (IaC) je metodika správy datových center pomocí konfiguračních souborů[26]

Tento nástroj nabízí jednoduchý deklarativní zápis infrastruktury. Ve výpisu 6.1 lze vidět jeho základní nastavení. Skládá se z definice použité platformy (zde DigitalOcean) a následné definice prostředků a jejich napojení.

Konfigurace Terraformu je členěna do jednotlivých bloků, které definují vždy určitý aspekt infrastruktury. První dva bloky definují cílovou cloudovou platformu, následuje definice proměnných pro další použití. Základní bloky nastavení infrastruktury jsou `data` a `resource`, které přistupují k dané cloudové platformě pro získání, resp. úpravy na daném účtu, definovaném API klíčem. Hlavním rozdílem těchto dvou bloků je, že `data` se používají pro data jen pro čtení, které jsou potřeba v další konfiguraci a `resource` pro definice, jaké prostředí je potřeba založit na cloudu.

Před aplikací této infrastruktury je nutné mít staženou utilitu pro Terraform. Následně je dobré nastavit proměnnou prostředí `TF_VAR_do_token` pro definici API klíče pro Digital Ocean, jinak se bude na tuto proměnnou ptát při každém běhu. Následuje aplikování infrastruktury na cloud pomocí příkazu `terraform apply`, který zobrazí jaký plán vykoná na cloudové platformě a je nutné jej potvrdit. Pro následné zrušení celé infrastruktury lze použít příkaz `terraform destroy`. Po dokončení bude zobrazena IPv4 nového virtuálního počítače, toto zobrazení je definováno blokem `output` v Terraform konfiguraci.

Pro přihlášení do počítače lze využít příkaz `ssh root@<IP>`, přihlášení probíhá přes nastavený SSH klíč v Terraform konfiguraci.

### 6.1.2 Příprava disku pro testování

Nejdříve je potřeba hardwarově připojené disky nebo média zformátovat na požadovaný systém souborů a připojit je do adresářové struktury. Disky, které jsou připojené k počítači, lze zjistit příkazem `lsblk`. Ukázka je lze vidět ve výpisu 6.2. Pro zformátování a připojení do adresářové struktury lze vidět na příkladu 6.3. Je zde použitá konvence pro jména připojení do adresářové struktury jako `/<název_FS>`. Oproti standarní konvenci [28] není použitý adresář `/mnt` z důvodu větší jednoduchosti.

### 6.1.3 Nastavení počítače pro spuštění testu

Následně je potřeba daný počítač nastavit pro testování. Pro běh testu je potřeba doinstalovat balíčky nutné pro kompilaci nástroje: `make` a `g++`. Následně je stažen zdrojový kód z git repository a zkompilován. Příklad této konfigurace lze vidět ve výpisu 6.4.

Nyní je počítač připraven na testování novým nástrojem. Dále bude popsán způsob použití nástroje k testování a způsob, jakým bude probíhat sběr dat.

### 6.1.4 Založení uživatele pro běh testů

Pro zvýšení bezpečnosti při běhu testů bude vytvořen uživatel, který bude mít přístup k potřebám pro vykonání daného testu. Nástroj podporuje nastavení, pod jakým uživatelem spouští testování, takže i při spuštění pod `root` účtem se nástroj zřekne oprávnění a použije daný nastavený účet a skupinu. V 6.5 je příklad založení a nastavení uživatele pro systém souboru ext2. Nejdříve je potřeba založit Unix uživatele, poté ho nastavit jako vlastníka potřebného adresáře pro potřeby běhu testu a nastavit omezení pro možnost zápisu a čtení z tohoto adresáře.

### 6.1.5 Popis konfigurace HW

Pro testování byl použit virtuální počítač v cloudovém prostředí. Kompletní výpis konfigurace počítače je v přiloženém médiu. Základní údaje o použitém HW:

```
# Terraform konfigurace pro vytvoreni VM na Digital Ocean
terraform {
  required_providers {
    digitalocean = {
      source  = "digitalocean/digitalocean",
      version = "~> 2.0"
    }
  }
}

# API klic pro Digital Ocean
variable "do_token" {

}

# Konfigurace Digital Ocean providera
provider "digitalocean" {
  token = var.do_token
}

# Definice SSH klice
data "digitalocean_ssh_key" "SSH_KEY" {
  name = "MainKey"
}

# Definice VM
resource "digitalocean_droplet" "benchmark" {
  image    = "ubuntu-22-10-x64" # Ubuntu 22.10
  name     = "bench" # Nazev ciloveho pocitace
  region   = "fra1" # Region datoveho centra, zde Frankfurt
  size     = "s-8vcpu-16gb" # 8 vCPU, 16 GB RAM
  ssh_keys = [data.digitalocean_ssh_key.SSH_KEY.id] # nastaveni SSH klice
}

# Individualni diskove media a jejich pripojeni k VM

resource "digitalocean_volume" "main" {
  region = "fra1"
  name   = "main"
  size   = 100 # GiB
}

resource "digitalocean_volume_attachment" "bench_main" {
  droplet_id = digitalocean_droplet.benchmark.id
  volume_id  = digitalocean_volume.main.id
}

...

# Zobrazeni IP adresy VM
output "server_ip" {
  value = digitalocean_droplet.benchmark.ipv4_address
}
```

**Výpis kódu 6.1** Konfigurace Terraform

```
NAME    MAJ:MIN RM   SIZE RO TYPE MOUNTPOINTS
sda       8:0    0   100G  0 disk /ext4
sdb       8:16   0   100G  0 disk /ext2
sdc       8:32   0   100G  0 disk
sdd       8:48   0   100G  0 disk
 sdd1    8:49   0   100G  0 part /ntfs
```

**Výpis kódu 6.2** Ukázka výpisu příkazu lsblk

```
root@bench:~# mkfs.ext4 /dev/sda           <-- format disku na system souboru ext4
root@bench:~# mkdir /ext4                  <-- vytvoreni adresare pro pripojeni
root@bench:~# mount -t ext4 /dev/sda /ext4 <-- pripojeni disku do adresarove struktury
```

**Výpis kódu 6.3** Příklad formátování disku a mountu

```
root@bench:~# apt-get update
root@bench:~# apt-get install -y make g++ git
root@bench:~# git clone <GIT_REPO>
root@bench:~/fsbench# cd fsbench
root@bench:~/fsbench# make -B
```

**Výpis kódu 6.4** Příprava prostředí serveru

```
root@bench:~/fsbench# adduser bench                 <-- zalozeni uzivatele
root@bench:~/fsbench# mkdir /ext2/bench             <-- adresar pro test
root@bench:~/fsbench# chown bench:bench /ext2/bench <-- nastaveni uz. jako vlastnika
root@bench:~/fsbench# chmod 0700 /ext2/bench        <-- vlastnik ma prava cteni a zapisu
```

**Výpis kódu 6.5** Nastavení uživatele

- Virtuální počítač se systémem Linux Ubuntu 22.10 s 8 virtuálními jádry CPU a 16 GB RAM
- Připojené diskové média jsou disky s velikostí 100 GB a Digital Ocean udává jejich rychlost jako 7500 IOPS[2] s propustností 300 MB/s [29].

## 6.2 Testování

Po nastavení prostředí testování následuje samotné testování. Nejprve budou shrnuty všechny testy, které budou prováděny. Budou popsány, jaké systémy souborů se budou testovat a proč byly testovány. Nakonec budou přeneseny definované testy do konfigurace nástroje s jeho spuštěním.

### 6.2.1 Definice testů

Pro účely obecného vyhodnocení výkonnosti systému souborů vzhledem k možnostem poskytnutým novými nástroji. Prvním testem bude metadatová operace pro vytvoření souboru. Dalším testem bude datová operace čtení souboru s velikostmi bloku 512 KB a 4096 KB nad jedním 8 MB velkým souborem. Tento test poběží jednou v sekvenčním přístupu a podruhé v náhodném přístupu k testovanému souboru. Poslední skupinou testů bude nad datovou operací zápisu. Testování čtení poběží v sekvenčním iteračním módu, z důvodu, že jedna iterace je rychlá. Datové operace budou spušteny v paralelním výpočetním módu s max. možností hardwarové konkurence, v tomto případě je to 8. Define testů je následující:

- Vytváření 1000 souborů v sekvenčním provedení v počtu 10ti iterací
- Čtení souboru o velikosti 8 MB
  - Sekvenční čtení o velikosti bloku 512 KB
  - Sekvenční čtení o velikosti bloku 4096 KB
  - Náhodné čtení o velikosti bloku 512 KB
  - Náhodné čtení o velikosti bloku 4096 KB
- Zápis souboru o velikosti 8 MB
  - Sekvenční zápis o velikosti bloku 512 KB
  - Sekvenční zápis o velikosti bloku 4096 KB
  - Náhodný zápis o velikosti bloku 512 KB
  - Náhodný zápis o velikosti bloku 4096 KB

### 6.2.2 Testované systémy souborů

Pro testování byly zvoleny čtyři systémy souborů: ext2, ext4, btrfs a ntfs. Výběr je vzhledem k použití a vlastnostem velmi rozmanitý. Systém souborů ext2 byl zvolen pro jeho stáří v unixovém ekosystému a také pro chybějící žurnál, což by mělo znevýhodnit rychlost metadatových operací. Následně je zvolen standardní systém souborů ext4, poté novější typ Btrfs, který se vlastnostmi podobá např. systému souborů ZFS. A nakonec se otestuje systém souborů NTFS, který je nativní pro operační systém Windows. Na Linuxu lze použít, ale má omezení v podobě, že oproti Windows, kde je nativní přístup do NTFS systému souborů, na Linuxu se využívá implementace NTFS-3G, která však funguje na úrovni user space Linuxu a tím je z podstaty pomalejší [30].

[2] vstupně-výstupních operací za sekundu (ang. Input/output operations per second)

```
root@bench:~/fsbench# ./fsb --output /ext4/bench \
--exportOutput ./results/ext4/read-512-random.csv \
--uid `id -u bench` --gid `id -g bench` \
--fileCount 1 --iterations 10 \
--blockSize 512 --blockCount 65536 --random 1
```

**Výpis kódu 6.6** Spuštění testu

### 6.2.3 Spuštění testování pomocí nástroje

Pro jednoduché spuštění testu bude vytvořen skript pro snadné spuštění daných testů. Tento skript nastaví základní parametry jako výstupní složka a místo uložení exportního souboru, také vymaže soubor protokolu a po dokončení tento protokol přenese do společného adresáře s výsledky. Tento skript také nastaví daného uživatele, pod kterým test poběží. Další pomocné skripty pak definují jednotlivé parametry a typ testu. V obrázku 6.6 je příklad, jakým způsobem se volá nástroj při spuštění testu pro náhodné čtení na systému souborů ext4.

## 6.3 Analýza výsledků

V této kapitole uvedu souhrn naměřených výsledků a následně je analyzuji. Podrobné výsledky měření se vzorky budou v příloze C, podrobná data jako výpis hardwarových a softwarových parametrů měřeného systému a protokoly jednotlivých měření budou v rámci přiloženého média.

Nástroj poskytuje výsledky ve formátu doby trvání jedné iterace. Pro lepší čitelnost výsledků je potřeba tento formát přepočítat do srozumitelné interpretace. Například u testu vytvoření souboru nás bude nejvíce zajímat počet souborů za sekundu, zatímco u testu čtení opět velikost souboru za sekundu.

### 6.3.1 Výsledky

V následujících tabulkách 6.1, 6.2 a 6.3 je výpis jednotlivých naměřených průměrných hodnot. Následují grafy 6.1, 6.2 a 6.3, kde je vizualizováno porovnání jednotlivých operací napříč systémy souborů.

**Tabulka 6.1** Vytvoření (souborů/s)

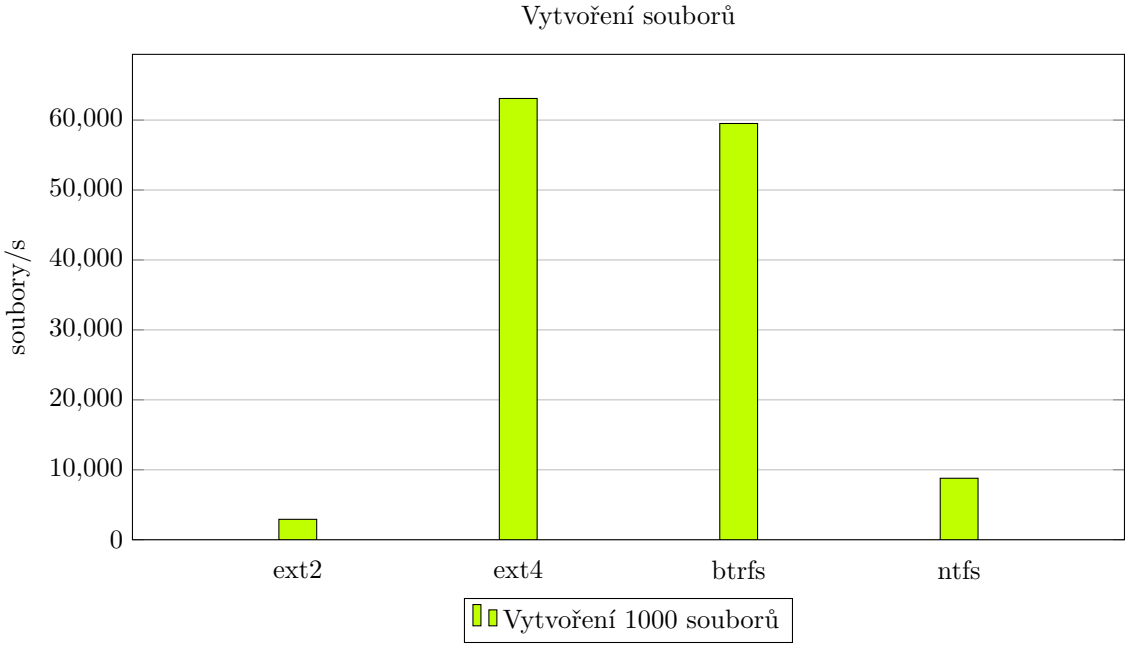
| | Vytvoření (souborů/s) |
|---|---|
| ext2 | 2918 |
| ext4 | 63088 |
| btrfs | 59513 |
| ntfs | 8789 |

**Tabulka 6.2** Rychlost čtení

| | Sekv. čtení 512 KB (MB/s) | Sekv. čtení 4096 KB (MB/s) | Náh. čtení 512 KB (MB/s) | Náh. čtení 4096KB (MB/s) |
|---|---|---|---|---|
| ext2 | 97.5 | 91.4 | 55.1 | 51.0 |
| ext4 | 95.2 | 94.5 | 53.8 | 50.5 |
| btrfs | 93.9 | 93.8 | 52.6 | 49.7 |
| ntfs | 84.2 | 2.9 | 54.5 | N/A |

**Tabulka 6.3** Rychlost zápisu

| | Sekv. zápis 512 KB (MB/s) | Sekv. zápis 4096 KB (MB/s) | Náh. zápis 512 KB (MB/s) | Náh. zápis 4096KB (MB/s) |
|---|---|---|---|---|
| ext2 | 24.6 | 9.8 | 20.9 | 14.4 |
| ext4 | 25.5 | 11.0 | 21.3 | 14.7 |
| btrfs | 14.8 | 10.4 | 14.0 | 10.9 |
| ntfs | 0.72 | 0.47 | 0.72 | 0.68 |

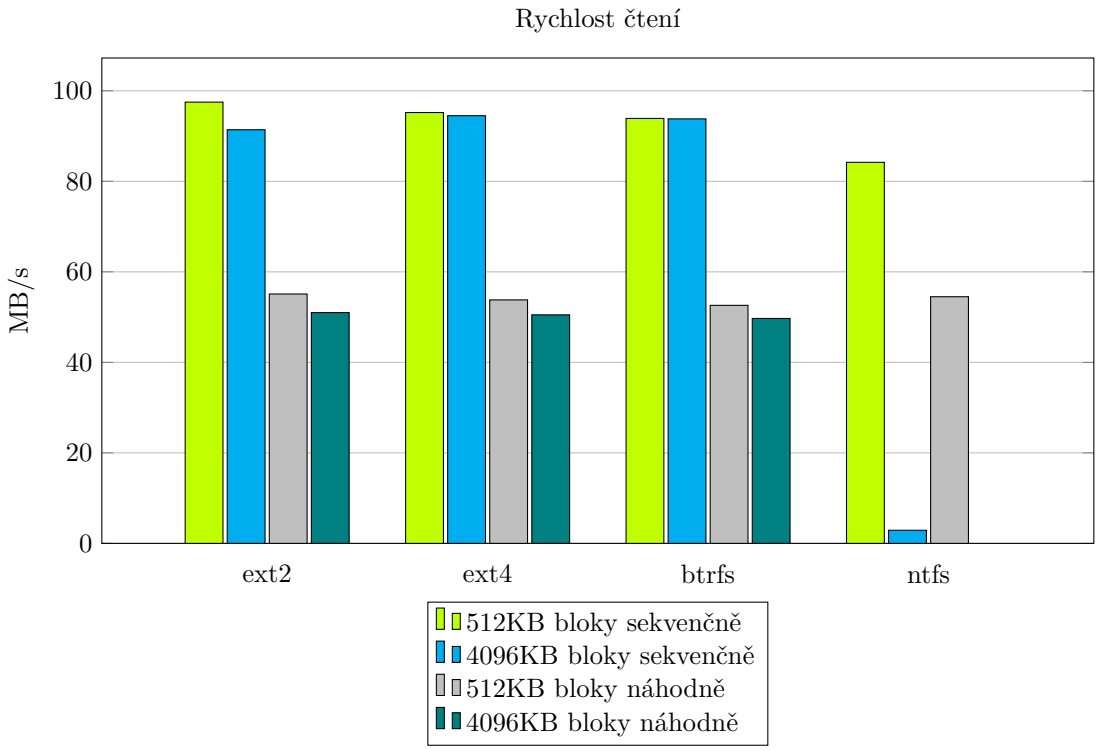



**Obrázek 6.1** Graf rychlosti čtení

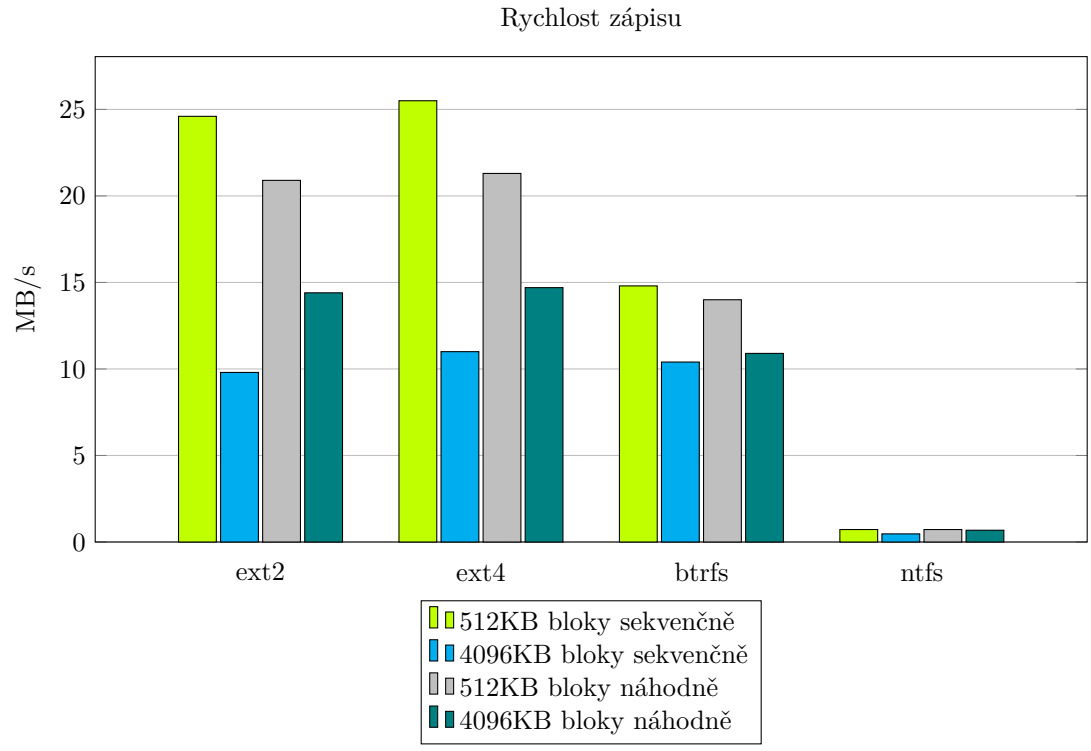
### 6.3.2 Analýza

Přesnost měření je zajištěna pomocí metadatových operací, což je obtížnější v systémech souborů s žurnálem, které jsou velmi rychlé a jsou vybaveny. U ext2, který nemá žurnál, je vidět velká odchylka při vytváření souborů.

Systém souborů NTFS má velmi pomalé čtení na Linuxu (při výchozím nastavení formátování). To by podtrhlo jeho degradaci kvůli jeho užití běhu v user-space. Zbytek měřeních je v porovnání velice podobných, to může být např. z důvodu silného užití cachování. Dále nehodné operace jsou pomalejší než sekvenční operace, což odpovídá technologickému omezení mechanických disků.




**Obrázek 6.2** Graf rychlosti čtení




**Obrázek 6.3** Graf rychlosti zápisu

# Závěr

V této bakalářské práci byl proveden test výkonnosti čtyř systémů souborů v Linuxovém prostředí s cílem získat podrobnější informace o výkonnosti jednotlivých systémů. Test byl proveden s použitím nového nástroje pro testování výkonnosti souborových systémů, který byl navržen a implementován také v rámci této práce.

V rámci testování výkonnosti byly zkoumány různé operace a velikosti bloků a výsledky ukázaly, že jednotlivé systémy souborů se liší v rychlosti v závislosti na konkrétní operaci a velikosti bloku. Například systém souborů ext2, který nemá žurnál, je pomalejší při vytváření souborů než ostatní systémy souborů. Naopak systém souborů NTFS je velmi pomalý na Linuxu a měl by být používán pouze v případě, že je nezbytně nutný.

Jedním z hlavních přínosů této bakalářské práce byl nový testovací nástroj, který se ukázal být úspěšným prostředkem pro testování výkonnosti souborových systémů. Poskytl důležité informace o výkonnosti testovaných systémů souborů, což může být užitečné pro vývojáře softwaru a správce systémů při výběru nejvhodnějšího systému souborů pro jejich potřeby.

V budoucnu lze tento nový testovací nástroj dále vylepšit a rozšířit o další funkce, aby mohl být ještě více užitečný pro uživatele. Například by mohly být přidány nové operace, které by byly dříve nedostupné, a výpočet počtu cyklů by mohl být ještě přesnější. Také by mohla být zavedena podpora pro další operační systémy, což by umožnilo více uživatelům využívat tento nástroj. Další možností by bylo vytvoření uživatelského rozhraní s intuitivními ovládacími prvky, což by zjednodušilo používání nástroje a zvýšilo jeho efektivitu. Všechna tato vylepšení by umožnila uživatelům dosáhnout lepších výsledků a zlepšit svou produktivitu.

Celkově lze tedy říci, že tato bakalářská práce přinesla nový nástroj pro testování výkonnosti souborových systémů v Linuxovém prostředí a poskytla užitečné informace o výkonnosti různých systémů souborů. Díky tomuto testovacímu nástroji mohou uživatelé získat více informací o výkonnosti svých systémů a vybrat si tak nejvhodnější systém souborů pro své specifické potřeby.

# Příloha A

# Seznam použitých zkratek

| | |
|---|---|
| ANSI | American National Standards Institute |
| API | Application Programming Interface |
| CLI | Command-line Interface |
| CPU | Central Processing Unit |
| CSV | Comma-Separated Values |
| FS | File System |
| HTML | HyperText Markup Language |
| I/O | Input/Output |
| POSIX | Portable Operating System Interface |
| OS | Operating System |
| STDOUT | Standard Output |

# Příloha B

# Uživatelská příručka

Nástroj podporuje testy pro operace jako čtení, zápis a vytváření souborů. Projekt je postaven na programovacím jazyce C++, a je dostupný pro Unixové systémy.

## B.1 Jak používat

Pro použití nástroje je třeba mít přístup k Unixovému systému a mít nainstalované C++17. Poté stačí stáhnout zdrojový kód a zkompilovat pomocí příkazu `make`. Nástroj se spouští pomocí příkazu `./fsb`. Použití nástroje lze dále konfigurovat pomocí přepínačů na příkazové řádce, nebo pomocí souboru konfigurace.

## B.2 Testy výkonu

Nástroj podporuje testy pro operace jako čtení, zápis a vytváření souborů. Testy jsou implementovány jako odvozené třídy z abstraktní třídy `CFileBenchmark`, která nabízí tři metody pro běh testu. Testy jsou dále rozděleny podle typu operace na `CCreateFileBenchmark`, `CReadFileBenchmark` a `CWriteFileBenchmark`. Tyto testy podporují různé konfigurace jako počet souborů a velikost bloku pro operaci.

## B.3 Bezpečnost

Nástroj obsahuje logiku pro zabezpečení operačního systému. Pro tyto účely slouží konfigurační hodnoty na nastavení identifikátoru uživatele a skupiny. Lze také použít `root` účet, ale to je nutné vynutit nastavením přepínaček `--forceRooUser`. Pro zajištění, aby nástroj nezačal ve výchozím stavu používat aktuální složku jako výchozí výstupní složku, bude nastaven pracovní adresář na složku pro účely dočasného souboru v adresáři `/tmp`.

## B.4 Konfigurace

Konfigurace nástroje se provádí pomocí souboru `settings.ini`. Soubor konfigurace obsahuje položky jako počet testovaných souborů, velikost bloků pro zápis a čtení a další.

### B.4.1 Seznam konfiguračních položek

Pro použití v konfiguračním souboru je nutné použít formát bez předpony `--`.

- `-help` vypíše nápovědu.

Protokol:

- `-minLogLevel` minimální úroveň logování (debug, info, warning, error)
- `-sink` log sinks (console, file) [odděleno čárkou pro více možností]
- `-logFile` log file path

Bezpečnost:

- `-uid` identifikátor uživatele pro změnu práv
- `-gid` identifikátor skupiny pro změnu práv
- `-forceRootUser` vynutit použití root uživatele

Běh cyklů:

- `-exportOutput` výstupního souboru pro export
- `-runCount` počet běhů benchmarku
- `-useWarmup` použít zahřívání
- `-warmupCycleCount` počet cyklů zahřívání

Iterační běh:

- `-iterations` - počet iterací benchmarku
- `-multiThreaded` - použít více vláken
- `-threadCount` - počet pracovních vláken

Benchmark:

- `-output` - výstupní adresář
- `-benchmark` - benchmark, který se má spustit (create, read, write)
- `-fileCount` - počet souborů, na kterých se má operovat
- `-blockSize` - velikost bloku v KiB (pouze pro read a write benchmark)
- `-blockCount` - počet bloků (pouze pro read a write benchmark)
- `-random` - použít náhodný přístup k blokům (pouze pro read a write benchmark)

Příloha C

# Naměřené výsledky testování

| # | Hodnota | Jednotka |
|---|---|---|
| 1 | 144738 | µs |
| 2 | 448067 | µs |
| 3 | 483218 | µs |
| 4 | 511281 | µs |
| 5 | 390795 | µs |
| 6 | 435737 | µs |
| 7 | 276525 | µs |
| 8 | 222143 | µs |
| 9 | 97563 | µs |
| 10 | 124194 | µs |
| 11 | 487005 | µs |
| 12 | 373788 | µs |
| 13 | 537924 | µs |
| 14 | 399808 | µs |
| 15 | 347365 | µs |
| 16 | 404122 | µs |
| 17 | 256116 | µs |
| 18 | 445343 | µs |
| 19 | 326976 | µs |
| 20 | 172625 | µs |
| 21 | 219898 | µs |
| 22 | 322204 | µs |
| 23 | 412335 | µs |
| 24 | 195622 | µs |
| 25 | 375259 | µs |
| 26 | 466236 | µs |
| 27 | 359706 | µs |
| 28 | 393940 | µs |
| 29 | 365123 | µs |
| 30 | 286719 | µs |

**(a)** ext2

| # | Hodnota | Jednotka |
|---|---|---|
| 1 | 17422 | µs |
| 2 | 15652 | µs |
| 3 | 15778 | µs |
| 4 | 14717 | µs |
| 5 | 15732 | µs |
| 6 | 17124 | µs |
| 7 | 14068 | µs |
| 8 | 15996 | µs |
| 9 | 14909 | µs |
| 10 | 15670 | µs |
| 11 | 18144 | µs |
| 12 | 16938 | µs |
| 13 | 16482 | µs |
| 14 | 15613 | µs |
| 15 | 14800 | µs |
| 16 | 15661 | µs |
| 17 | 19535 | µs |
| 18 | 16568 | µs |
| 19 | 15747 | µs |
| 20 | 16783 | µs |
| 21 | 16261 | µs |
| 22 | 17776 | µs |
| 23 | 14220 | µs |
| 24 | 17655 | µs |
| 25 | 13119 | µs |
| 26 | 13541 | µs |
| 27 | 15124 | µs |
| 28 | 13901 | µs |
| 29 | 16439 | µs |
| 30 | 14182 | µs |

**(b)** ext4

| # | Hodnota | Jednotka |
|---|---|---|
| 1 | 16027 | µs |
| 2 | 16471 | µs |
| 3 | 18673 | µs |
| 4 | 15635 | µs |
| 5 | 16879 | µs |
| 6 | 18196 | µs |
| 7 | 15517 | µs |
| 8 | 15761 | µs |
| 9 | 19112 | µs |
| 10 | 17003 | µs |
| 11 | 17092 | µs |
| 12 | 15562 | µs |
| 13 | 16694 | µs |
| 14 | 15715 | µs |
| 15 | 18188 | µs |
| 16 | 18313 | µs |
| 17 | 14617 | µs |
| 18 | 17442 | µs |
| 19 | 20387 | µs |
| 20 | 15432 | µs |
| 21 | 18558 | µs |
| 22 | 16024 | µs |
| 23 | 15971 | µs |
| 24 | 18985 | µs |
| 25 | 16206 | µs |
| 26 | 15396 | µs |
| 27 | 16589 | µs |
| 28 | 15272 | µs |
| 29 | 15773 | µs |

**(c)** btrfs

| # | Hodnota | Jednotka |
|---|---|---|
| 1 | 119157 | µs |
| 2 | 101534 | µs |
| 3 | 105033 | µs |
| 4 | 109499 | µs |
| 5 | 114564 | µs |
| 6 | 109051 | µs |
| 7 | 101254 | µs |
| 8 | 118148 | µs |
| 9 | 110404 | µs |
| 10 | 123542 | µs |
| 11 | 128519 | µs |
| 12 | 129118 | µs |
| 13 | 111820 | µs |
| 14 | 121765 | µs |
| 15 | 121427 | µs |
| 16 | 111862 | µs |
| 17 | 128318 | µs |
| 18 | 129713 | µs |
| 19 | 115181 | µs |
| 20 | 101313 | µs |
| 21 | 119833 | µs |
| 22 | 111572 | µs |
| 23 | 122174 | µs |
| 24 | 100447 | µs |
| 25 | 108864 | µs |
| 26 | 110244 | µs |
| 27 | 103278 | µs |
| 28 | 114778 | µs |
| 29 | 102976 | µs |
| 30 | 107811 | µs |

**(d)** ntfs

**Tabulka C.1** Vytváření 1000 souborů

| # | Hodnota | Jednotka |
|---|---|---|
| 1 | 11118 | µs |
| 2 | 10249 | µs |
| 3 | 10225 | µs |
| 4 | 10107 | µs |
| 5 | 10148 | µs |
| 6 | 10176 | µs |
| 7 | 10577 | µs |
| 8 | 9684 | µs |
| 9 | 10743 | µs |
| 10 | 9375 | µs |
| 11 | 9886 | µs |
| 12 | 10250 | µs |
| 13 | 9762 | µs |
| 14 | 10488 | µs |
| 15 | 9962 | µs |
| 16 | 10359 | µs |
| 17 | 9673 | µs |
| 18 | 10096 | µs |
| 19 | 10135 | µs |
| 20 | 10523 | µs |
| 21 | 9751 | µs |
| 22 | 11044 | µs |
| 23 | 9497 | µs |
| 24 | 10887 | µs |
| 25 | 10721 | µs |
| 26 | 10996 | µs |
| 27 | 10919 | µs |
| 28 | 9999 | µs |
| 29 | 10723 | µs |
| 30 | 9610 | µs |

**(a)** ext2

| # | Hodnota | Jednotka |
|---|---|---|
| 1 | 11067 | µs |
| 2 | 10806 | µs |
| 3 | 10580 | µs |
| 4 | 11825 | µs |
| 5 | 10571 | µs |
| 6 | 9647 | µs |
| 7 | 9969 | µs |
| 8 | 10250 | µs |
| 9 | 10787 | µs |
| 10 | 11154 | µs |
| 11 | 10366 | µs |
| 12 | 10365 | µs |
| 13 | 10522 | µs |
| 14 | 10808 | µs |
| 15 | 10808 | µs |
| 16 | 10950 | µs |
| 17 | 10541 | µs |
| 18 | 10092 | µs |
| 19 | 10840 | µs |
| 20 | 10174 | µs |
| 21 | 10925 | µs |
| 22 | 9555 | µs |
| 23 | 10770 | µs |
| 24 | 10403 | µs |
| 25 | 10962 | µs |
| 26 | 9785 | µs |
| 27 | 10897 | µs |
| 28 | 9350 | µs |
| 29 | 10034 | µs |
| 30 | 10182 | µs |

**(b)** ext4

| # | Hodnota | Jednotka |
|---|---|---|
| 1 | 10923 | µs |
| 2 | 9813 | µs |
| 3 | 9496 | µs |
| 4 | 9640 | µs |
| 5 | 10561 | µs |
| 6 | 9807 | µs |
| 7 | 9937 | µs |
| 8 | 10014 | µs |
| 9 | 10656 | µs |
| 10 | 11241 | µs |
| 11 | 11177 | µs |
| 12 | 12311 | µs |
| 13 | 11662 | µs |
| 14 | 11184 | µs |
| 15 | 10212 | µs |
| 16 | 11014 | µs |
| 17 | 11384 | µs |
| 18 | 11101 | µs |
| 19 | 10276 | µs |
| 20 | 10594 | µs |
| 21 | 10837 | µs |
| 22 | 10191 | µs |
| 23 | 10457 | µs |
| 24 | 11088 | µs |
| 25 | 10875 | µs |
| 26 | 10942 | µs |
| 27 | 10623 | µs |
| 28 | 10224 | µs |
| 29 | 10514 | µs |
| 30 | 10823 | µs |

**(c)** btrfs

| # | Hodnota | Jednotka |
|---|---|---|
| 1 | 10406 | µs |
| 2 | 10933 | µs |
| 3 | 11301 | µs |
| 4 | 12820 | µs |
| 5 | 13149 | µs |
| 6 | 10668 | µs |
| 7 | 12951 | µs |
| 8 | 12295 | µs |
| 9 | 11671 | µs |
| 10 | 11871 | µs |
| 11 | 13101 | µs |
| 12 | 12665 | µs |
| 13 | 11296 | µs |
| 14 | 12420 | µs |
| 15 | 11881 | µs |
| 16 | 11441 | µs |
| 17 | 11050 | µs |
| 18 | 11823 | µs |
| 19 | 12069 | µs |
| 20 | 10845 | µs |
| 21 | 9751 | µs |
| 22 | 12184 | µs |
| 23 | 14598 | µs |
| 24 | 11193 | µs |
| 25 | 12000 | µs |
| 26 | 12754 | µs |
| 27 | 12056 | µs |
| 28 | 11607 | µs |
| 29 | 11602 | µs |
| 30 | 11709 | µs |

**(d)** ntfs

**Tabulka C.2** Sekvenční čtení o velikosti bloku 512 KB

| # | Hodnota | Jednotka |
|---|---|---|
| 1 | 10440 | µs |
| 2 | 11505 | µs |
| 3 | 10737 | µs |
| 4 | 11217 | µs |
| 5 | 10760 | µs |
| 6 | 10301 | µs |
| 7 | 11146 | µs |
| 8 | 9706 | µs |
| 9 | 11380 | µs |
| 10 | 9924 | µs |
| 11 | 11223 | µs |
| 12 | 10588 | µs |
| 13 | 11107 | µs |
| 14 | 9896 | µs |
| 15 | 10571 | µs |
| 16 | 11128 | µs |
| 17 | 10568 | µs |
| 18 | 10945 | µs |
| 19 | 9899 | µs |
| 20 | 10770 | µs |
| 21 | 10129 | µs |
| 22 | 10896 | µs |
| 23 | 10201 | µs |
| 24 | 10129 | µs |
| 25 | 10696 | µs |
| 26 | 12536 | µs |
| 27 | 12188 | µs |
| 28 | 12030 | µs |
| 29 | 13704 | µs |
| 30 | 11975 | µs |

**(a)** ext2

| # | Hodnota | Jednotka |
|---|---|---|
| 1 | 10306 | µs |
| 2 | 10471 | µs |
| 3 | 9948 | µs |
| 4 | 10115 | µs |
| 5 | 10088 | µs |
| 6 | 11025 | µs |
| 7 | 10497 | µs |
| 8 | 11327 | µs |
| 9 | 9638 | µs |
| 10 | 10443 | µs |
| 11 | 10455 | µs |
| 12 | 11301 | µs |
| 13 | 10907 | µs |
| 14 | 10594 | µs |
| 15 | 11072 | µs |
| 16 | 10632 | µs |
| 17 | 10850 | µs |
| 18 | 11041 | µs |
| 19 | 10681 | µs |
| 20 | 10138 | µs |
| 21 | 10638 | µs |
| 22 | 10425 | µs |
| 23 | 11323 | µs |
| 24 | 10821 | µs |
| 25 | 10586 | µs |
| 26 | 10437 | µs |
| 27 | 10375 | µs |
| 28 | 9785 | µs |
| 29 | 10987 | µs |
| 30 | 10607 | µs |

**(b)** ext4

| # | Hodnota | Jednotka |
|---|---|---|
| 1 | 9906 | µs |
| 2 | 10813 | µs |
| 3 | 10049 | µs |
| 4 | 10499 | µs |
| 5 | 10176 | µs |
| 6 | 10880 | µs |
| 7 | 10634 | µs |
| 8 | 10774 | µs |
| 9 | 10961 | µs |
| 10 | 10861 | µs |
| 11 | 10375 | µs |
| 12 | 10512 | µs |
| 13 | 10410 | µs |
| 14 | 10879 | µs |
| 15 | 10116 | µs |
| 16 | 10642 | µs |
| 17 | 10388 | µs |
| 18 | 11283 | µs |
| 19 | 11010 | µs |
| 20 | 10744 | µs |
| 21 | 10684 | µs |
| 22 | 10767 | µs |
| 23 | 11076 | µs |
| 24 | 10406 | µs |
| 25 | 10757 | µs |
| 26 | 10819 | µs |
| 27 | 10613 | µs |
| 28 | 10988 | µs |
| 29 | 11205 | µs |
| 30 | 10663 | µs |

**(c)** btrfs

| # | Hodnota | Jednotka |
|---|---|---|
| 1 | 346578 | µs |
| 2 | 345798 | µs |
| 3 | 318499 | µs |
| 4 | 402520 | µs |
| 5 | 331370 | µs |
| 6 | 330701 | µs |
| 7 | 313810 | µs |
| 8 | 328151 | µs |
| 9 | 337990 | µs |
| 10 | 347551 | µs |
| 11 | 321761 | µs |
| 12 | 334341 | µs |
| 13 | 325532 | µs |
| 14 | 357743 | µs |
| 15 | 375143 | µs |
| 16 | 316326 | µs |
| 17 | 360176 | µs |
| 18 | 315646 | µs |
| 19 | 341485 | µs |
| 20 | 312104 | µs |
| 21 | 347569 | µs |
| 22 | 352756 | µs |
| 23 | 362279 | µs |
| 24 | 388077 | µs |
| 25 | 389391 | µs |
| 26 | 356644 | µs |
| 27 | 345916 | µs |
| 28 | 337239 | µs |
| 29 | 348655 | µs |
| 30 | 333075 | µs |

**(d)** ntfs

**Tabulka C.3** Sekvenční čtení o velikosti bloku 4096 KB

| # | Hodnota | Jednotka |
|---|---|---|
| 1 | 18040 | µs |
| 2 | 18265 | µs |
| 3 | 17089 | µs |
| 4 | 17362 | µs |
| 5 | 17866 | µs |
| 6 | 17180 | µs |
| 7 | 16188 | µs |
| 8 | 17568 | µs |
| 9 | 18992 | µs |
| 10 | 18922 | µs |
| 11 | 18922 | µs |
| 12 | 18311 | µs |
| 13 | 17509 | µs |
| 14 | 18779 | µs |
| 15 | 18805 | µs |
| 16 | 18729 | µs |
| 17 | 19284 | µs |
| 18 | 17782 | µs |
| 19 | 18643 | µs |
| 20 | 18918 | µs |
| 21 | 18033 | µs |
| 22 | 17548 | µs |
| 23 | 16326 | µs |
| 24 | 19403 | µs |
| 25 | 17308 | µs |
| 26 | 18997 | µs |
| 27 | 20053 | µs |
| 28 | 18318 | µs |
| 29 | 18148 | µs |
| 30 | 16730 | µs |

**(a)** ext2

| # | Hodnota | Jednotka |
|---|---|---|
| 1 | 18040 | µs |
| 2 | 18265 | µs |
| 3 | 17089 | µs |
| 4 | 17362 | µs |
| 5 | 17866 | µs |
| 6 | 17180 | µs |
| 7 | 16188 | µs |
| 8 | 17568 | µs |
| 9 | 18992 | µs |
| 10 | 18922 | µs |
| 11 | 18922 | µs |
| 12 | 18311 | µs |
| 13 | 17509 | µs |
| 14 | 18779 | µs |
| 15 | 18805 | µs |
| 16 | 18729 | µs |
| 17 | 19284 | µs |
| 18 | 17782 | µs |
| 19 | 18643 | µs |
| 20 | 18918 | µs |
| 21 | 18033 | µs |
| 22 | 17548 | µs |
| 23 | 16326 | µs |
| 24 | 19403 | µs |
| 25 | 17308 | µs |
| 26 | 18997 | µs |
| 27 | 20053 | µs |
| 28 | 18318 | µs |
| 29 | 18148 | µs |
| 30 | 16730 | µs |

**(b)** ext4

| # | Hodnota | Jednotka |
|---|---|---|
| 1 | 19295 | µs |
| 2 | 19144 | µs |
| 3 | 18140 | µs |
| 4 | 18215 | µs |
| 5 | 19412 | µs |
| 6 | 19294 | µs |
| 7 | 18433 | µs |
| 8 | 18683 | µs |
| 9 | 17415 | µs |
| 10 | 20277 | µs |
| 11 | 18543 | µs |
| 12 | 19822 | µs |
| 13 | 20149 | µs |
| 14 | 19527 | µs |
| 15 | 19626 | µs |
| 16 | 17653 | µs |
| 17 | 18215 | µs |
| 18 | 20128 | µs |
| 19 | 18586 | µs |
| 20 | 19929 | µs |
| 21 | 21453 | µs |
| 22 | 18312 | µs |
| 23 | 18188 | µs |
| 24 | 17891 | µs |
| 25 | 19659 | µs |
| 26 | 19234 | µs |
| 27 | 18454 | µs |
| 28 | 18588 | µs |
| 29 | 19558 | µs |
| 30 | 18245 | µs |

**(c)** btrfs

| # | Hodnota | Jednotka |
|---|---|---|
| 1 | 17213 | µs |
| 2 | 18132 | µs |
| 3 | 16489 | µs |
| 4 | 18774 | µs |
| 5 | 19306 | µs |
| 6 | 20878 | µs |
| 7 | 17568 | µs |
| 8 | 16523 | µs |
| 9 | 17023 | µs |
| 10 | 19404 | µs |
| 11 | 19776 | µs |
| 12 | 18381 | µs |
| 13 | 20890 | µs |
| 14 | 18013 | µs |
| 15 | 19150 | µs |
| 16 | 17841 | µs |
| 17 | 18142 | µs |
| 18 | 19473 | µs |
| 19 | 16978 | µs |
| 20 | 18806 | µs |
| 21 | 17949 | µs |
| 22 | 19227 | µs |
| 23 | 20416 | µs |
| 24 | 17132 | µs |
| 25 | 17844 | µs |
| 26 | 18777 | µs |
| 27 | 18382 | µs |
| 28 | 16872 | µs |
| 29 | 17415 | µs |
| 30 | 17225 | µs |

**(d)** ntfs

**Tabulka C.4** Náhodné čtení o velikosti bloku 512 KB

| # | Hodnota | Jednotka |
|---|---|---|
| 1 | 18603 | µs |
| 2 | 20186 | µs |
| 3 | 19615 | µs |
| 4 | 20910 | µs |
| 5 | 20123 | µs |
| 6 | 19610 | µs |
| 7 | 19608 | µs |
| 8 | 18254 | µs |
| 9 | 19852 | µs |
| 10 | 18346 | µs |
| 11 | 19138 | µs |
| 12 | 19913 | µs |
| 13 | 19441 | µs |
| 14 | 20343 | µs |
| 15 | 19584 | µs |
| 16 | 19286 | µs |
| 17 | 20543 | µs |
| 18 | 20006 | µs |
| 19 | 19848 | µs |
| 20 | 18237 | µs |
| 21 | 19362 | µs |
| 22 | 17780 | µs |
| 23 | 19178 | µs |
| 24 | 20568 | µs |
| 25 | 21397 | µs |
| 26 | 19973 | µs |
| 27 | 18539 | µs |
| 28 | 19866 | µs |
| 29 | 19347 | µs |
| 30 | 20449 | µs |

**(a)** ext2

| # | Hodnota | Jednotka |
|---|---|---|
| 1 | 18381 | µs |
| 2 | 19114 | µs |
| 3 | 19500 | µs |
| 4 | 20574 | µs |
| 5 | 19124 | µs |
| 6 | 20001 | µs |
| 7 | 20399 | µs |
| 8 | 21230 | µs |
| 9 | 19934 | µs |
| 10 | 19601 | µs |
| 11 | 22007 | µs |
| 12 | 21346 | µs |
| 13 | 18711 | µs |
| 14 | 19314 | µs |
| 15 | 18858 | µs |
| 16 | 19081 | µs |
| 17 | 19031 | µs |
| 18 | 19769 | µs |
| 19 | 20575 | µs |
| 20 | 19329 | µs |
| 21 | 22034 | µs |
| 22 | 19740 | µs |
| 23 | 20107 | µs |
| 24 | 20101 | µs |
| 25 | 18395 | µs |
| 26 | 21141 | µs |
| 27 | 20344 | µs |
| 28 | 18519 | µs |
| 29 | 18829 | µs |
| 30 | 18959 | µs |

**(b)** ext4

| # | Hodnota | Jednotka |
|---|---|---|
| 1 | 19600 | µs |
| 2 | 19477 | µs |
| 3 | 20558 | µs |
| 4 | 20632 | µs |
| 5 | 19025 | µs |
| 6 | 20618 | µs |
| 7 | 19979 | µs |
| 8 | 18533 | µs |
| 9 | 21169 | µs |
| 10 | 19584 | µs |
| 11 | 20964 | µs |
| 12 | 20322 | µs |
| 13 | 18832 | µs |
| 14 | 21478 | µs |
| 15 | 20558 | µs |
| 16 | 21112 | µs |
| 17 | 18580 | µs |
| 18 | 20606 | µs |
| 19 | 19994 | µs |
| 20 | 19997 | µs |
| 21 | 18562 | µs |
| 22 | 21994 | µs |
| 23 | 21062 | µs |
| 24 | 19618 | µs |
| 25 | 20240 | µs |
| 26 | 20793 | µs |
| 27 | 20628 | µs |
| 28 | 20069 | µs |
| 29 | 19432 | µs |
| 30 | 19252 | µs |

**(c)** btrfs

**Tabulka C.5** Náhodné čtení o velikosti bloku 4096 KB

| # | Hodnota | Jednotka |
|---|---|---|
| 1 | 35358 | µs |
| 2 | 50233 | µs |
| 3 | 47649 | µs |
| 4 | 39709 | µs |
| 5 | 38986 | µs |
| 6 | 39948 | µs |
| 7 | 35845 | µs |
| 8 | 47724 | µs |
| 9 | 38061 | µs |
| 10 | 39469 | µs |
| 11 | 40297 | µs |
| 12 | 39297 | µs |
| 13 | 50731 | µs |
| 14 | 36509 | µs |
| 15 | 35521 | µs |
| 16 | 50443 | µs |
| 17 | 37227 | µs |
| 18 | 38091 | µs |
| 19 | 38502 | µs |
| 20 | 38075 | µs |
| 21 | 44916 | µs |
| 22 | 37532 | µs |
| 23 | 37590 | µs |
| 24 | 40052 | µs |
| 25 | 37128 | µs |
| 26 | 44232 | µs |
| 27 | 42311 | µs |
| 28 | 39764 | µs |
| 29 | 39366 | µs |
| 30 | 38123 | µs |

**(a)** ext2

| # | Hodnota | Jednotka |
|---|---|---|
| 1 | 36011 | µs |
| 2 | 38004 | µs |
| 3 | 43012 | µs |
| 4 | 34683 | µs |
| 5 | 33184 | µs |
| 6 | 40826 | µs |
| 7 | 36492 | µs |
| 8 | 61409 | µs |
| 9 | 40575 | µs |
| 10 | 38705 | µs |
| 11 | 36944 | µs |
| 12 | 37535 | µs |
| 13 | 50990 | µs |
| 14 | 38429 | µs |
| 15 | 35786 | µs |
| 16 | 36782 | µs |
| 17 | 37412 | µs |
| 18 | 38011 | µs |
| 19 | 33832 | µs |
| 20 | 38161 | µs |
| 21 | 56645 | µs |
| 22 | 34994 | µs |
| 23 | 40211 | µs |
| 24 | 39666 | µs |
| 25 | 34991 | µs |
| 26 | 38130 | µs |
| 27 | 34951 | µs |
| 28 | 35591 | µs |
| 29 | 37761 | µs |
| 30 | 37581 | µs |

**(b)** ext4

| # | Hodnota | Jednotka |
|---|---|---|
| 1 | 67703 | µs |
| 2 | 65971 | µs |
| 3 | 65207 | µs |
| 4 | 59848 | µs |
| 5 | 65736 | µs |
| 6 | 66439 | µs |
| 7 | 78420 | µs |
| 8 | 63126 | µs |
| 9 | 64548 | µs |
| 10 | 69361 | µs |
| 11 | 58669 | µs |
| 12 | 72831 | µs |
| 13 | 66210 | µs |
| 14 | 66958 | µs |
| 15 | 71286 | µs |
| 16 | 73223 | µs |
| 17 | 67954 | µs |
| 18 | 70066 | µs |
| 19 | 65875 | µs |
| 20 | 59591 | µs |
| 21 | 83349 | µs |
| 22 | 67807 | µs |
| 23 | 60738 | µs |
| 24 | 66239 | µs |
| 25 | 62493 | µs |
| 26 | 81985 | µs |
| 27 | 64669 | µs |
| 28 | 60306 | µs |
| 29 | 66753 | µs |
| 30 | 68325 | µs |

**(c)** btrfs

| # | Hodnota | Jednotka |
|---|---|---|
| 1 | 1436217 | µs |
| 2 | 1422625 | µs |
| 3 | 1382581 | µs |
| 4 | 1355205 | µs |
| 5 | 1478154 | µs |
| 6 | 1365891 | µs |
| 7 | 1386506 | µs |
| 8 | 1489724 | µs |
| 9 | 1399461 | µs |
| 10 | 1430311 | µs |
| 11 | 1474325 | µs |
| 12 | 1333667 | µs |
| 13 | 1250289 | µs |
| 14 | 1501859 | µs |
| 15 | 1354595 | µs |
| 16 | 1354916 | µs |
| 17 | 1323530 | µs |
| 18 | 1418727 | µs |
| 19 | 1332247 | µs |
| 20 | 1383668 | µs |
| 21 | 1376068 | µs |
| 22 | 1325737 | µs |
| 23 | 1418775 | µs |
| 24 | 1380858 | µs |
| 25 | 1407534 | µs |
| 26 | 1320276 | µs |
| 27 | 1471126 | µs |
| 28 | 1457872 | µs |
| 29 | 1435004 | µs |
| 30 | 1464759 | µs |

**(d)** ntfs

**Tabulka C.6** Sekvenční zápis o velikosti bloku 512 KB

| # | Hodnota | Jednotka |
|---|---|---|
| 1 | 102921 | µs |
| 2 | 103124 | µs |
| 3 | 112043 | µs |
| 4 | 111437 | µs |
| 5 | 102977 | µs |
| 6 | 112932 | µs |
| 7 | 93002 | µs |
| 8 | 103912 | µs |
| 9 | 94250 | µs |
| 10 | 97123 | µs |
| 11 | 96041 | µs |
| 12 | 103020 | µs |
| 13 | 99813 | µs |
| 14 | 102548 | µs |
| 15 | 101204 | µs |
| 16 | 99743 | µs |
| 17 | 97190 | µs |
| 18 | 104666 | µs |
| 19 | 98372 | µs |
| 20 | 102638 | µs |
| 21 | 95141 | µs |
| 22 | 102551 | µs |
| 23 | 104083 | µs |
| 24 | 103620 | µs |
| 25 | 103650 | µs |
| 26 | 110665 | µs |
| 27 | 112241 | µs |
| 28 | 100608 | µs |
| 29 | 99635 | µs |
| 30 | 99045 | µs |

**(a)** ext2

| # | Hodnota | Jednotka |
|---|---|---|
| 1 | 92142 | µs |
| 2 | 95517 | µs |
| 3 | 84529 | µs |
| 4 | 90590 | µs |
| 5 | 85442 | µs |
| 6 | 88369 | µs |
| 7 | 92197 | µs |
| 8 | 92390 | µs |
| 9 | 94636 | µs |
| 10 | 89794 | µs |
| 11 | 92006 | µs |
| 12 | 92944 | µs |
| 13 | 88707 | µs |
| 14 | 100936 | µs |
| 15 | 94653 | µs |
| 16 | 86481 | µs |
| 17 | 95313 | µs |
| 18 | 92356 | µs |
| 19 | 91349 | µs |
| 20 | 91650 | µs |
| 21 | 93242 | µs |
| 22 | 92072 | µs |
| 23 | 89869 | µs |
| 24 | 90391 | µs |
| 25 | 89154 | µs |
| 26 | 88123 | µs |
| 27 | 87953 | µs |
| 28 | 88833 | µs |
| 29 | 87120 | µs |
| 30 | 90688 | µs |

**(b)** ext4

| # | Hodnota | Jednotka |
|---|---|---|
| 1 | 94368 | µs |
| 2 | 94392 | µs |
| 3 | 92479 | µs |
| 4 | 98542 | µs |
| 5 | 92084 | µs |
| 6 | 96160 | µs |
| 7 | 87876 | µs |
| 8 | 100548 | µs |
| 9 | 94880 | µs |
| 10 | 102171 | µs |
| 11 | 96483 | µs |
| 12 | 105552 | µs |
| 13 | 96621 | µs |
| 14 | 90671 | µs |
| 15 | 88703 | µs |
| 16 | 88016 | µs |
| 17 | 103593 | µs |
| 18 | 96656 | µs |
| 19 | 93324 | µs |
| 20 | 100994 | µs |
| 21 | 104297 | µs |
| 22 | 95916 | µs |
| 23 | 94563 | µs |
| 24 | 88815 | µs |
| 25 | 95427 | µs |
| 26 | 92202 | µs |
| 27 | 104072 | µs |
| 28 | 94272 | µs |
| 29 | 97767 | µs |
| 30 | 95805 | µs |

**(c)** btrfs

| # | Hodnota | Jednotka |
|---|---|---|
| 1 | 2257182 | µs |
| 2 | 2122717 | µs |
| 3 | 2054744 | µs |
| 4 | 2003574 | µs |
| 5 | 2124221 | µs |
| 6 | 2264544 | µs |
| 7 | 2044501 | µs |
| 8 | 2057117 | µs |
| 9 | 2386669 | µs |
| 10 | 2231194 | µs |
| 11 | 1989732 | µs |
| 12 | 2017016 | µs |
| 13 | 2102771 | µs |
| 14 | 2012901 | µs |
| 15 | 2403199 | µs |
| 16 | 2034606 | µs |
| 17 | 2204092 | µs |
| 18 | 2139132 | µs |
| 19 | 2362759 | µs |
| 20 | 1938428 | µs |
| 21 | 2150374 | µs |
| 22 | 2213609 | µs |
| 23 | 2116620 | µs |
| 24 | 2150108 | µs |
| 25 | 2145138 | µs |
| 26 | 2246355 | µs |
| 27 | 1935394 | µs |
| 28 | 1984312 | µs |
| 29 | 1923011 | µs |
| 30 | 2040999 | µs |

**(d)** ntfs

**Tabulka C.7** Sekvenční zápis o velikosti bloku 4096 KB

| # | Hodnota | Jednotka |
|---|---|---|
| 1 | 46273 | µs |
| 2 | 46614 | µs |
| 3 | 49816 | µs |
| 4 | 48777 | µs |
| 5 | 48767 | µs |
| 6 | 48130 | µs |
| 7 | 51258 | µs |
| 8 | 47833 | µs |
| 9 | 46520 | µs |
| 10 | 51527 | µs |
| 11 | 45780 | µs |
| 12 | 52904 | µs |
| 13 | 47910 | µs |
| 14 | 45177 | µs |
| 15 | 44904 | µs |
| 16 | 43113 | µs |
| 17 | 45571 | µs |
| 18 | 41862 | µs |
| 19 | 42069 | µs |
| 20 | 57827 | µs |
| 21 | 49039 | µs |
| 22 | 44340 | µs |
| 23 | 47521 | µs |
| 24 | 44365 | µs |
| 25 | 45420 | µs |
| 26 | 49339 | µs |
| 27 | 51232 | µs |
| 28 | 51720 | µs |
| 29 | 52527 | µs |
| 30 | 49391 | µs |

**(a)** ext2

| # | Hodnota | Jednotka |
|---|---|---|
| 1 | 39189 | µs |
| 2 | 41235 | µs |
| 3 | 43440 | µs |
| 4 | 49658 | µs |
| 5 | 50376 | µs |
| 6 | 41890 | µs |
| 7 | 46391 | µs |
| 8 | 47410 | µs |
| 9 | 48148 | µs |
| 10 | 59469 | µs |
| 11 | 45425 | µs |
| 12 | 47260 | µs |
| 13 | 45847 | µs |
| 14 | 40654 | µs |
| 15 | 57684 | µs |
| 16 | 42276 | µs |
| 17 | 42169 | µs |
| 18 | 50204 | µs |
| 19 | 45645 | µs |
| 20 | 47718 | µs |
| 21 | 43811 | µs |
| 22 | 45930 | µs |
| 23 | 62856 | µs |
| 24 | 45881 | µs |
| 25 | 44249 | µs |
| 26 | 47558 | µs |
| 27 | 45003 | µs |
| 28 | 52068 | µs |
| 29 | 43534 | µs |
| 30 | 46876 | µs |

**(b)** ext4

| # | Hodnota | Jednotka |
|---|---|---|
| 1 | 64444 | µs |
| 2 | 66624 | µs |
| 3 | 65952 | µs |
| 4 | 72236 | µs |
| 5 | 78606 | µs |
| 6 | 69235 | µs |
| 7 | 66015 | µs |
| 8 | 69224 | µs |
| 9 | 65506 | µs |
| 10 | 62721 | µs |
| 11 | 77605 | µs |
| 12 | 79435 | µs |
| 13 | 73380 | µs |
| 14 | 96399 | µs |
| 15 | 66463 | µs |
| 16 | 67556 | µs |
| 17 | 69115 | µs |
| 18 | 72799 | µs |
| 19 | 71442 | µs |
| 20 | 66873 | µs |
| 21 | 79292 | µs |
| 22 | 65791 | µs |
| 23 | 69674 | µs |
| 24 | 75564 | µs |
| 25 | 69788 | µs |
| 26 | 74076 | µs |
| 27 | 69989 | µs |
| 28 | 72897 | µs |
| 29 | 69432 | µs |
| 30 | 68539 | µs |

**(c)** btrfs

| # | Hodnota | Jednotka |
|---|---|---|
| 1 | 1364208 | µs |
| 2 | 1351684 | µs |
| 3 | 1467335 | µs |
| 4 | 1469288 | µs |
| 5 | 1397689 | µs |
| 6 | 1360011 | µs |
| 7 | 1409921 | µs |
| 8 | 1379665 | µs |
| 9 | 1457440 | µs |
| 10 | 1464409 | µs |
| 11 | 1362040 | µs |
| 12 | 1360571 | µs |
| 13 | 1483942 | µs |
| 14 | 1329682 | µs |
| 15 | 1361201 | µs |
| 16 | 1556262 | µs |
| 17 | 1384028 | µs |
| 18 | 1444360 | µs |
| 19 | 1478994 | µs |
| 20 | 1365256 | µs |
| 21 | 1369241 | µs |
| 22 | 1260064 | µs |
| 23 | 1517914 | µs |
| 24 | 1390977 | µs |
| 25 | 1303321 | µs |
| 26 | 1360775 | µs |
| 27 | 1321874 | µs |
| 28 | 1323081 | µs |
| 29 | 1456838 | µs |
| 30 | 1362817 | µs |

**(d)** ntfs

**Tabulka C.8** Náhodný zápis o velikosti bloku 512 KB

| # | Hodnota | Jednotka |
|---|---|---|
| 1 | 71677 | µs |
| 2 | 68777 | µs |
| 3 | 72214 | µs |
| 4 | 69558 | µs |
| 5 | 66009 | µs |
| 6 | 67593 | µs |
| 7 | 75773 | µs |
| 8 | 76167 | µs |
| 9 | 68361 | µs |
| 10 | 63690 | µs |
| 11 | 66498 | µs |
| 12 | 65290 | µs |
| 13 | 66962 | µs |
| 14 | 68663 | µs |
| 15 | 66131 | µs |
| 16 | 73265 | µs |
| 17 | 70440 | µs |
| 18 | 71372 | µs |
| 19 | 63510 | µs |
| 20 | 69682 | µs |
| 21 | 71591 | µs |
| 22 | 71118 | µs |
| 23 | 78500 | µs |
| 24 | 67605 | µs |
| 25 | 63567 | µs |
| 26 | 73684 | µs |
| 27 | 73391 | µs |
| 28 | 66793 | µs |
| 29 | 66616 | µs |
| 30 | 65820 | µs |

**(a)** ext2

| # | Hodnota | Jednotka |
|---|---|---|
| 1 | 66478 | µs |
| 2 | 65916 | µs |
| 3 | 65249 | µs |
| 4 | 71965 | µs |
| 5 | 66697 | µs |
| 6 | 65471 | µs |
| 7 | 65320 | µs |
| 8 | 68606 | µs |
| 9 | 69463 | µs |
| 10 | 77438 | µs |
| 11 | 72010 | µs |
| 12 | 72631 | µs |
| 13 | 70810 | µs |
| 14 | 72482 | µs |
| 15 | 64839 | µs |
| 16 | 60069 | µs |
| 17 | 66821 | µs |
| 18 | 74529 | µs |
| 19 | 68451 | µs |
| 20 | 63969 | µs |
| 21 | 66174 | µs |
| 22 | 70008 | µs |
| 23 | 68546 | µs |
| 24 | 66341 | µs |
| 25 | 65443 | µs |
| 26 | 65375 | µs |
| 27 | 69183 | µs |
| 28 | 73636 | µs |
| 29 | 65978 | µs |
| 30 | 67006 | µs |

**(b)** ext4

| # | Hodnota | Jednotka |
|---|---|---|
| 1 | 87793 | µs |
| 2 | 86261 | µs |
| 3 | 83264 | µs |
| 4 | 87032 | µs |
| 5 | 89266 | µs |
| 6 | 91416 | µs |
| 7 | 94513 | µs |
| 8 | 83806 | µs |
| 9 | 87401 | µs |
| 10 | 82773 | µs |
| 11 | 94971 | µs |
| 12 | 96537 | µs |
| 13 | 114674 | µs |
| 14 | 95593 | µs |
| 15 | 92775 | µs |
| 16 | 92894 | µs |
| 17 | 97346 | µs |
| 18 | 92812 | µs |
| 19 | 93309 | µs |
| 20 | 84329 | µs |
| 21 | 83058 | µs |
| 22 | 93946 | µs |
| 23 | 90137 | µs |
| 24 | 104868 | µs |
| 25 | 92968 | µs |
| 26 | 98062 | µs |
| 27 | 89126 | µs |
| 28 | 97948 | µs |
| 29 | 85539 | µs |
| 30 | 91717 | µs |

**(c)** btrfs

| # | Hodnota | Jednotka |
|---|---|---|
| 1 | 1431643 | µs |
| 2 | 1347863 | µs |
| 3 | 1425919 | µs |
| 4 | 1543602 | µs |
| 5 | 1408376 | µs |
| 6 | 1277650 | µs |
| 7 | 1483950 | µs |
| 8 | 1494200 | µs |
| 9 | 1413579 | µs |
| 10 | 1432545 | µs |
| 11 | 1702222 | µs |
| 12 | 1520898 | µs |
| 13 | 1641776 | µs |
| 14 | 1362734 | µs |
| 15 | 1321507 | µs |
| 16 | 1382686 | µs |
| 17 | 1487452 | µs |
| 18 | 1357885 | µs |
| 19 | 1509794 | µs |
| 20 | 1425603 | µs |
| 21 | 1404658 | µs |
| 22 | 1478820 | µs |
| 23 | 1665378 | µs |
| 24 | 1469351 | µs |
| 25 | 1579716 | µs |
| 26 | 1539330 | µs |
| 27 | 1514301 | µs |
| 28 | 1628753 | µs |
| 29 | 1391052 | µs |
| 30 | 1418580 | µs |

**(d)** ntfs

**Tabulka C.9** Náhodný zápis o velikosti bloku 4096 KB

# Bibliografie


1. TRDLIČKA, Jan. *Operacní systémy Systémy souborů I* [online]. CTU, Faculty of Information Technology, [b.r.]. Dostupné také z: `https://courses.fit.cvut.cz/BI-OSY/lectures/biosy-p11-FS_Disk-01.pdf`.
2. KROMER, Michael. *Linux Filesystem Performance Tests* [online]. Linux Magazine, [b.r.]. Dostupné také z: `https://www.linux-magazine.com/Online/Features/Filesystems-Benchmarked`.
3. BHAT, Wasim; QUADRI, Syed. BENCHMARKING CRITERIA FOR FILE SYSTEM BENCHMARKS. *International Journal of Engineering Science and Technology*. 2011, roč. 3.
4. TRAEGER, Avishay; ZADOK, Erez; JOUKOV, Nikolai; WRIGHT, Charles. A nine year study of file system and storage benchmarking. *TOS*. 2008, roč. 4. Dostupné z DOI: `10.1145/1367829.1367831`.
5. TARASOV, Vasily; BHANAGE, Saumitra; ZADOK, Erez; SELTZER, Margo. Benchmarking file system benchmarking: it *IS* rocket science. In: 2011, s. 9–9.
6. VAN METER, Rodney. Observing the Effects of Multi-Zone Disks. In: *Proceedings of the Annual Conference on USENIX Annual Technical Conference*. Anaheim, California: USENIX Association, 1997, s. 2. ATEC '97.
7. WRIGHT, Charles; JOUKOV, Nikolai; KULKARNI, Devaki; MIRETSKIY, Yevgeniy; ZADOK, Erez. Auto-pilot: A Platform for System Software Benchmarking. In: 2005, s. 175–188.
8. *IBM Benefits of direct I/O* [online]. [B.r.]. Dostupné také z: `https://www.ibm.com/docs/en/aix/7.2?topic=io-benefits-direct`.
9. SHRIVER, Elizabeth; SMALL, Christopher; SMITH, Keith A. Why Does File System Prefetching Work? In: *Proceedings of the Annual Conference on USENIX Annual Technical Conference*. Monterey, California: USENIX Association, 1999, s. 6. ATEC '99.
10. BRYANT, Ray; RADDATZ, Dave; SUNSHINE, Roger. PenguinoMeter: A New File-I/O Benchmark for Linux®. In: *Proceedings of the 5th Annual Linux Showcase & Conference - Volume 5*. Oakland, California: USENIX Association, 2001, s. 10. ALS '01.
11. BRAY, Tim. *Bonnie* [online]. 1990. Dostupné také z: `https://www.textuality.com/bonnie/`.
12. TIM BRAY, Russell Coker. *Bonnie++* [online]. 1999. Dostupné také z: `https://www.coker.com.au/bonnie++/readme.html`.
13. NORCOTT, WilliamD. IOzone Filesystem Benchmark [online]. 2003. Dostupné také z: `https://www.iozone.org/`.

14. *iozone(1) - Linux man page* [online]. [B.r.]. Dostupné také z: https://linux.die.net/man/1/iozone.
15. *Iometer* [online]. [B.r.]. Dostupné také z: http://www.iometer.org/.
16. *filebench(1) - Linux man page* [online]. [B.r.]. Dostupné také z: https://linux.die.net/man/1/filebench.
17. *C++17* [online]. 2023. Dostupné také z: https://en.cppreference.com/w/cpp/17.
18. WIKIPEDIE. *Bash — Wikipedie: Otevřená encyklopedie* [online]. 2022. Dostupné také z: https://cs.wikipedia.org/w/index.php?title=Bash&oldid=21439872.
19. AKINSHIN, Andrey. *Pro .NET Benchmarking: The Art of Performance Measurement*. 2019. ISBN 978-1-4842-4940-6. Dostupné z DOI: 10.1007/978-1-4842-4941-3.
20. WIKIPEDIE. *CSV — Wikipedie: Otevřená encyklopedie* [online]. 2023. Dostupné také z: https://cs.wikipedia.org/w/index.php?title=CSV&oldid=22546689.
21. IBBETSON, D. Algorithm 209: Gauss. *Commun. ACM*. 1963, roč. 6, č. 10, s. 616. ISSN 0001-0782. Dostupné z DOI: 10.1145/367651.367664.
22. HILL, G. W. ACM Algorithm 395: Student's t-Distribution. *Commun. ACM*. 1970, roč. 13, č. 10, s. 617–619. ISSN 0001-0782. Dostupné z DOI: 10.1145/355598.355599.
23. *Configuration in .NET* [online]. 2023. Dostupné také z: https://learn.microsoft.com/en-us/dotnet/core/extensions/configuration.
24. *Date and time utilities* [online]. [B.r.]. Dostupné také z: https://en.cppreference.com/w/cpp/chrono.
25. *Digital ocean* [online]. [B.r.]. Dostupné také z: https://www.digitalocean.com/.
26. WIKIPEDIE. *Infrastructure as Code — Wikipedie: Otevřená encyklopedie* [online]. 2022. Dostupné také z: https://cs.wikipedia.org/w/index.php?title=Infrastructure_as_Code&oldid=22271471.
27. *Terraform by HashiCorp* [online]. [B.r.]. Dostupné také z: https://www.terraform.io/.
28. *Linux Filesystem Hierarchy - 1.12. /mnt* [online]. [B.r.]. Dostupné také z: https://tldp.org/LDP/Linux-Filesystem-Hierarchy/html/mnt.html.
29. *Volume Limits* [online]. [B.r.]. Dostupné také z: https://docs.digitalocean.com/products/volumes/details/limits/.
30. WIKIPEDIA CONTRIBUTORS. *NTFS-3G — Wikipedia, The Free Encyclopedia* [online]. 2023. Dostupné také z: https://en.wikipedia.org/w/index.php?title=NTFS-3G&oldid=1146825320.

# Obsah přiloženého média

- `readme.txt` ..... stručný popis obsahu média
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF
- `benchmark` ..... naměřené výsledky měření