Uniwersytet Warszawski
Wydział Matematyki, Informatyki i Mechaniki

Michał Malinowski
Nr albumu: 189420

System ciągłej ochrony danych

Praca magisterska
na kierunku INFORMATYKA

Praca wykonana pod kierunkiem
dr Janiny Mincer-Daszkiewicz
Instytut Informatyki

Październik 2006

Oświadczenie kierującego pracą

Potwierdzam, że niniejsza praca została przygotowana pod moim kierunkiem i kwalifikuje się do przedstawienia jej w postępowaniu o nadanie tytułu zawodowego.

Data Podpis kierującego pracą

Oświadczenie autora pracy

Świadom odpowiedzialności prawnej oświadczam, że niniejsza praca dyplomowa została napisana przeze mnie samodzielnie i nie zawiera treści uzyskanych w sposób niezgodny z obowiązującymi przepisami.

Oświadczam również, że przedstawiona praca nie była wcześniej przedmiotem procedur związanych z uzyskaniem tytułu zawodowego w wyższej uczelni.

Oświadczam ponadto, że niniejsza wersja pracy jest identyczna z załączoną wersją elektroniczną.

Data Podpis autora pracy

Streszczenie

W ramach niniejszej pracy został zaprojektowany i zaimplementowany system plików, realizujący założenia systemu ciągłej ochrony danych. W części pisemnej pracy opisano zasady działania takich systemów oraz zawarto porównania z tradycyjnymi sposobami tworzenia kopii zapasowych. Zostały przedstawione także różne podejścia do tworzenia systemów plików w przestrzeni użytkownika oraz możliwości wykorzystania sieciowych urządzeń blokowych. Na praktyczną część składa się realizacja przedstawionego projektu oraz wykonanie testów wydajności systemu w różnych konfiguracjach.

Słowa kluczowe

system ciągłej ochrony danych, archiwizacja danych, kopia zapasowa, system plików, FUSE, urządzenie blokowe, NBD, AoE, DRBD

Dziedzina pracy (kody wg programu Socrates-Erasmus)

11.3 Informatyka

Klasyfikacja tematyczna

D. Software
D.4. Operating Systems
D.4.3 File Systems Management

Tytuł pracy w języku angielskim

Continuous data protection system

Spis treści

Wprowadzenie

Problem archiwizacji i ochrony danych jest zagadnieniem znanym w informatyce od dawna. Wiąże się on z przechowywaniem ogromnej ilości informacji cyfrowych, która zwiększa się corocznie o kilkadziesiąt procent. Dodatkowo wciąż zaostrzane są wymagania co do sposobu i czasu przechowywania danych, w wielu przypadkach podyktowane regulacjami prawnymi.

Wiele firm i instytucji potrzebuje zgromadzonych informacji do prawidłowego funkcjonowania. Uzależnia swoje przetrwanie od ich bezpieczeństwa. Tymczasem zapewnienie danym bezpieczeństwa staje się z roku na rok coraz większym wyzwaniem. Im więcej osób na nich operuje, tym większe prawdopodobieństwo ich przypadkowego zniszczenia. Im większy jest zbiór danych, tym trudniej jest zapewnić mu należytą ochronę.

Pojęcie ochrony danych większości ludzi kojarzy się z ochroną przed wirusami i włamaniami hakerów. Niemniej jednak statystyki dowodzą, że tego typu zagrożenia stanowią stosunkowo niewielki odsetek wśród przyczyn zaistniałych przypadków utraty danych. Okazuje się, że dużo większym problemem są awarie sprzętowe oraz błędy użytkowników. W zależności od profilu działalności firmy oraz skali awarii takie niebezpieczeństwa mogą powodować tymczasowe utrudnienia w pracy lub prowadzić wręcz do upadku przedsiębiorstwa.

Choć obecnie istnieje wiele sposobów zabezpieczania danych, archiwizacja nie jest procesem ani łatwym, ani przyjemnym. Najczęściej spotykanym rozwiązaniem jest wykonywanie okresowych (np. codziennych) kopii zapasowych na nośnikach magnetycznych lub optycznych. Taki sposób ochrony danych jest bardzo skomplikowany, pociąga za sobą dość duże koszty oraz wymaga systematyczności i ogromnego nakładu pracy. Niestety mimo dużej czaso- i pracochłonności nie zapewnia pełnego bezpieczeństwa. W razie awarii wszelkie modyfikacje wykonane po ostatniej archiwizacji zostaną utracone.

Od kilku lat swoją popularność w archiwizacji danych zdobywają dyski twarde. Ze względu na lepszą wydajność i coraz niższą cenę w porównaniu z innymi urządzeniami i nośnikami, dyski twarde stwarzają nowe możliwości ochrony danych. Jedną z nich jest odmienne podejście do problemu wykonywania kopii zapasowych — system ciągłej ochrony danych.

System taki zapewnia automatyczną, nieprzerwaną archiwizację danych. Rejestruje on wszelkie modyfikacje na bieżąco. Takie rozwiązanie umożliwia dostęp do danych z dowolnego momentu przed awarią i pozwala zaoszczędzić czas użytkowników, zredukować zbędne koszty finansowe.

W ramach tej pracy powstał projekt ciągłej ochrony danych *BUFS*. W jego skład wchodzą m. in.: system plików `bufs-mt` oraz aplikacja odzyskująca utracone pliki `bufs-recovery`. Wykorzystano w nim zewnętrzne systemy, ułatwiające proces jego budowy. Jednym z nich jest *FUSE*, który umożliwia zaimplementowanie systemu plików w przestrzeni użytkownika, a tym samym unika problemów związanych z tworzeniem modułu do jądra systemu operacyjnego.

Zawartość pracy

Kolejne rozdziały obejmują:

- analizę przyczyn i kosztów utraty danych,
- opis koncepcji i założeń dotyczących systemu ciągłej ochrony danych,
- porównanie z tradycyjnymi metodami wykonywania kopii bezpieczeństwa,
- przykłady systemów plików działających w przestrzeni użytkownika,
- opis wykonanego w ramach tej pracy systemu plików *BUFS* realizującego założenia ciągłej ochrony danych,
- wyniki przeprowadzonych testów oraz ich analizę.

Choć problem bezpieczeństwa danych dotyczy także ochrony przed nieuprawnionym dostępem czy kradzieżą danych, w pracy rozważa się jedynie problemy ich utraty.

Rozdział 1

W oczekiwaniu na utratę danych

W czerwcu 2006 roku firma Ontrack ([18]), zajmująca się odzyskiwaniem danych, przeprowadziła sondaż na grupie 1,400 użytkowników komputerów, dotyczący ich nawyków i opinii dotyczących wykonywania kopii zapasowych. Badania wykazały, że mimo iż większość użytkowników uważa swe dane za cenne (blisko 80%), mało kto należycie dba o ich ochronę. Duża część respondentów nie zabezpiecza danych w ogóle (23%), inni często robią to w nieodpowiedni sposób.

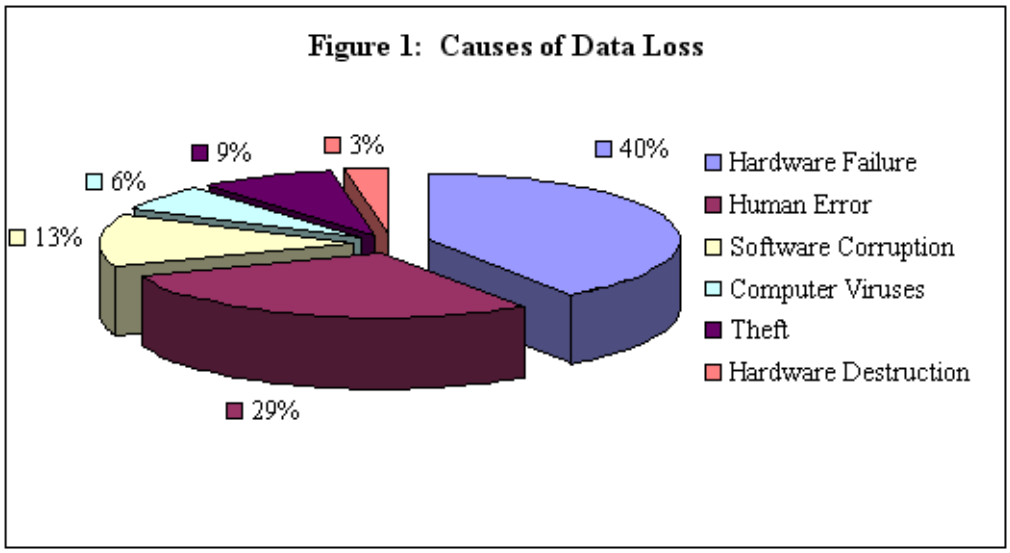
Duże przedsiębiorstwa na ogół stosują inną politykę ochrony danych niż zwykli użytkownicy komputerów. Podchodząc poważnie do problemu budują strategię ochrony danych: szkolą pracowników, kupują specjalistyczny sprzęt. Mimo to badania przeprowadzone w 2004 roku przez firmę IBM ([4]) pokazały, że nawet conocne wykonywanie kopii zapasowych nie zapewnia przedsiębiorstwom pełnego poczucia bezpieczeństwa.

1.1. Przyczyny utraty danych

Na zgromadzone w komputerach informacje czyha wiele niebezpieczeństw. Wśród powodów prowadzących do utraty danych należy wymienić:

- awarie sprzętu, głównie uszkodzenia dysków twardych i problemy z zasilaniem, a także błędy pamięci,
- pomyłki ludzkie, np. przypadkowe usunięcie pliku lub sformatowanie dysku,
- błędne działanie programów, systemów operacyjnych, które prowadzi do utraty niezapisanych informacji, zamazania danych lub uszkodzeń logicznych w strukturze plików,
- kradzież komputerów, laptopów, celowe niszczenie danych,
- wirusy i inne złośliwe programy,
- zniszczenia sprzętu spowodowane katastofami naturalnymi, takimi jak pożar, powódź, wyładowania atmosferyczne.

Powody utraty danych oraz częstotliwość ich występowania przedstawia rysunek 1.1. Statystyki dotyczą Stanów Zjednoczonych i zostały opracowane na podstawie informacji uzyskanych w firmach ubezpieczeniowych oraz firmach zajmujących się odzyskiwaniem danych. Widać, że aż blisko 70% przypadków utraty danych wiąże się z awariami sprzętu i błędami ludzkimi.



Rysunek 1.1: Przyczyny utraty danych. Źródło [22]

1.1.1. Awaria sprzętu

Najczęściej spotykanymi przyczynami awarii sprzętu są uszkodzenia mechaniczne oraz kłopoty z chłodzeniem. Do usterek mechanicznych należy zaliczyć na przykład zużycie obrotowych części napędów i wentylatorów. Innym stosunkowo często pojawiającym się problemem jest wadliwy styk elementów elektrycznych. Do takiej sytuacji może dojść w wyniku zanieczyszczenia lub korozji.

Kurz osadzający się wewnątrz komputera, a szczególnie na wentylatorze, może również znacząco obniżyć wydajność chłodzenia. Gdy wentylator przestanie poprawnie funkcjonować, podzespoły gwałtownie się zużywają i starzeją. Dwukrotne zwiększenie temperatury otoczenia, w jakiej pracuje urządzenie, skutkuje skróceniem żywotności o połowę. Jeżeli komputery działają w nieklimatyzowanych pomieszczeniach, to pierwsze problemy z dyskami twardymi pojawią się prawdopodobnie już po dwóch latach użytkowania, a nawet szybciej. Na zużycie podzespołów duży wpływ ma też ich rozmieszczenie w obudowie, np. ciasne ułożenie dysków twardych.

Dla danej serii dysków twardych na podstawie badań statystycznych można określić tzw. średni czas międzyawaryjny (ang. *Mean Time Between Failure*, w skrócie MTBF), mierzony w godzinach. Na podstawie tego parametru można obliczyć prawdopodobieństwo uszkodzenia nośnika w czasie roku użytkowania. Wartość MTBF podawana przez producentów nośników obecnie waha się zwykle pomiędzy 500.000 a 1.500.000, co przekłada się na dziesiątki lat. Okazuje się, że w praktyce dyski działają dużo krócej, głównie ze względu na niedpowiednie ich użytkowanie.

Problemy z zasilaniem nie należą do rzadkości i są podstawową przyczyną logicznych usterek prowadzących do utraty danych. Nawet chwilowa przerwa w dostawie prądu może uniemożliwić zapisanie modyfikowanych plików na trwałym nośniku. Gdy użytkownik zleci zapis modyfikacji, zmiany nie są od razu nanoszone na dysk twardy. Najpierw trafiają do podręcznego bufora programu, potem do bufora dysku, a dopiero później są trwale zapisywane. Awaria zasilania łatwo doprowadza do sytuacji, w której dane, pozornie zapisane, w rzeczywistości nigdy nie trafiają na dysk twardy. To wprowadza niespójność danych i powoduje ich najczęściej częściową, a niekiedy nawet całkowitą utratę.

1.1.2. Błędy ludzkie

Badanie, którego wyniki przedstawiono na rysunku 1.1 pokazało, że niemal 30% przypadków utraty danych wiąże się bezpośrednio z błędami użytkownika.

Omyłkowe usunięcie pliku, czy tym bardziej sformatowanie dysku może doprowadzić do utraty cennych zasobów. Zdarza się także upuszczenie lub przypadkowe strącenie laptopa, na którym jego użytkownik przechowywał ważne informacje.

1.1.3. Błędy oprogramowania

Nieprawidłowe wykonanie operacji przez program może doprowadzić do zamazania wcześniej zapisanych informacji lub pozostawić niespójne pliki na dysku. Zdarza się, że aplikacja nagle wykazuje brak interakcji z użytkownikiem, uniemożliwiając mu zapisanie modyfikowanego dokumentu. Błędy oprogramowania, szczególnie systemów operacyjnych, mogą prowadzić do zagubienia zawartości podręcznych buforów, wskutek czego użytkownik straci dane, których zapis zlecił chwilę wcześniej.

1.2. Koszt utraty danych

Za utratę danych często przychodzi zapłacić wysoką cenę. Koszty są spowodowane przez:

- konieczność odbudowy lub odzyskiwania utraconych danych,
- niedostępność informacji potrzebnych do kontynuacji pracy.

1.2.1. Koszty odbudowy lub odzyskiwania danych

Koszty awarii bezpośrednio wiążą się z wartością utraconych informacji. Choć na ogół trudno ją oszacować, można przyjąć, że jest ona tym wyższa, im więcej nakładów wymaga odbudowa lub odzyskanie danych. Przy założeniu, że istnieje papierowa dokumentacja źródłowa lub dane można odtworzyć w inny sposób, koszt jaki musi ponieść przedsiębiorstwo zależy od wynagrodzenia osób biorących udział w tym procesie oraz od czasu odbudowy danych. Im dłużej trwa odtwarzanie informacji, tym bardziej ich utrata jest dotkliwa.

Jeśli utrata danych nie jest trwała, tj. istnieje możliwość ich odzyskania, koszty zależą głównie od tego, czy może się tym zająć pracownik przedsiębiorstwa, w którym wystąpiła awaria, czy raczej należy to powierzyć zewnętrznej firmie. Niestety nie zawsze naprawa awarii jest możliwa na terenie przedsiębiorstwa. Dotyczy to głównie mechanicznych uszkodzeń dysków twardych, które mogą być odczytane jedynie w specjalnych laboratoriach. Niestety firmy odzyskujące dane wysoko cenią swoje usługi, dlatego przedsiebiorstwo dotknięte awarią ma dużo szczęścia, jeśli misję odzyskania danych może powierzyć własnemu pracownikowi.

1.2.2. Koszty niedostępności danych potrzebnych do kontynuacji pracy

Drugi istotny składnik kosztów wiąże się z czasem, w którym dane są potrzebne, ale niedostępne. Jest to tzw. koszt alternatywny, czyli zyski, które mogły być osiągnięte, gdyby nie doszło do awarii. W czasie naprawy skutków awarii część pracowników nie może efektywnie (lub wcale) wykonywać swoich obowiązków. Może to być spowodowane tym, że utracone dane są niezbędne do pracy, ale niedostępne lub tym, że muszą poświęcić czas na powtórne wykonanie pewnych zadań. Dlatego właśnie utrata danych wpływa niekorzystnie na wydajność pracy w firmach.

1.3. Podsumowanie

Tradycyjna, niezautomatyzowana ochrona danych jest zajęciem czaso- i pracochłonnym. Z tego powodu użytkownicy często ją zaniedbują, nie zastanawiając się nad przyszłością swoich plików. Zakładają tym samym — mniej lub bardziej świadomie — długowieczność urządzeń i własną nieomylność.

Przedsiębiorstwa prezentują zazwyczaj nieco inne podejście do tego problemu. Liczą się z tym, że ewentualna utrata danych pociągnie za sobą ogromne koszty i zniszczy ich reputację. Niemniej jednak nawet regularne wykonywanie kopii zapasowych nie zapewnia firmom i instytucjom pełnego bezpieczeństwa, dlatego idealnym dla nich rozwiązaniem wydają się być systemy ciągłej ochrony danych, o których będzie mowa w kolejnych rozdziałach.

Rozdział 2

Ciągła ochrona danych

Ciągła ochrona danych (ang. *continuous data protection*, w skrócie CDP) jest pojęciem dość nowym. W przeszłości wielu dostawców definiowało ją na własny sposób, często na potrzeby tworzonego produktu. Powodowało to pewne rozbieżności w określaniu systemów tym terminem.

W celu stworzenia jednoznacznej definicji systemu ciągłej ochrony danych, w ramach organizacji *SNIA*[1] w lutym 2005 roku powstała grupa *Continuous Data Protection Special Interest Group*, która zrzesza firmy i organizacje zainteresowane tą metodologią. Po kilku miesiącach zmagań udało się wypracować jednoznaczne stanowisko w tej sprawie.

2.1. Definicja ciągłej ochrony danych

System ciągłej ochrony danych można porównać do bazy danych ze specjalną transakcją, w której rejestrowane są wszystkie nowe zapisy danych, która pozwala na wycofanie zmian do dowolnego momentu z przeszłości i która posiada pełną historię wykonanych operacji.

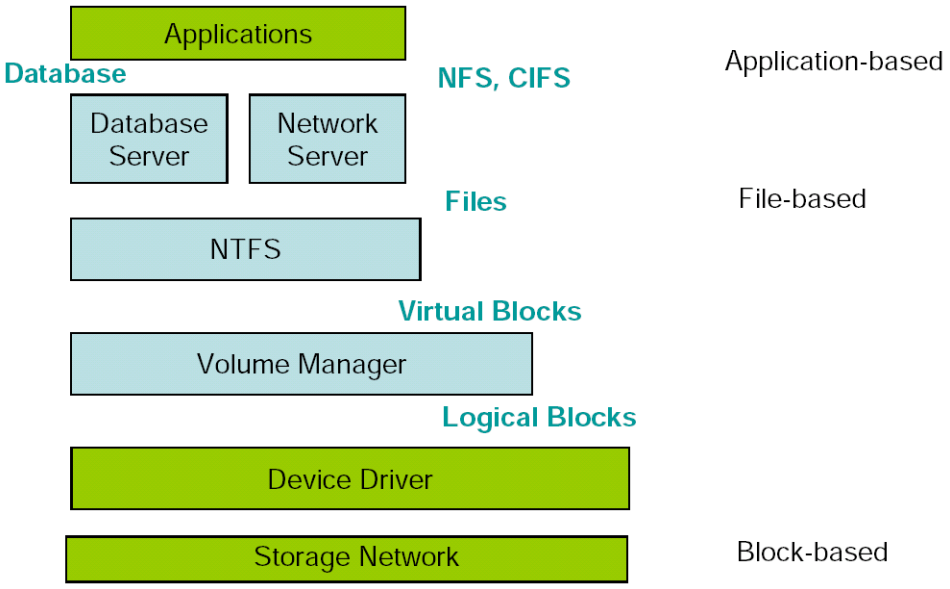
System ciągłej ochrony danych wykonuje automatycznie i natychmiastowo kopię zapasową każdej modyfikacji wykonanej przez użytkownika. Nie ogranicza się do odzyskiwania tylko najnowszej (aktualnej) wersji plików, ale potrafi odtworzyć dane z dowolnego punktu z przeszłości.

Charakterystyczną cechą takiego systemu jest dodatkowa warstwa kontroli nad informacjami zapisywanymi przez aplikację. Owa warstwa ochrony może być zlokalizowana w różnych miejscach na ścieżce zapisu danych. Wyróżnia się trzy typy ochrony danych, w zależności od sposobu rejestrowania zmian dokonywanych przez użytkownika:

- ochrona danych na poziomie bloków,
- ochrona danych na poziomie plików,
- ochrona danych na poziomie aplikacji.

Choć z punktu widzenia użytkownika wszystkie typy ochrony realizują te same założenia, systemy oparte na nich różnią się diametralnie pod względem budowy, a także funkcjonalności. Rysunek 2.1 obrazuje wszystkie wymienione poziomy ciągłej ochrony danych.

[1] Storage Networking Industry Association — stowarzyszenie producentów i konsumentów urządzeń masowej pamięci, działające w Stanach Zjednoczonych od 1997 roku.



Rysunek 2.1: Poziomy ciągłej ochrony danych. Źródło: [21]

2.1.1. Ochrona danych na poziomie bloków

Pierwszym rodzajem ochrony danych jest mechanizm oparty na blokach. Polega on na rejestracji każdej zmiany dokonywanej w urządzeniu fizycznym. Jednakże odbywa się to bez jakiejkolwiek ingerencji w strukturę danych (pliki, katalogi) zawartej na tym urządzeniu.

Ogromną wadą takich rozwiązań jest brak elastyczności. Ochrona danych jest ściśle powiązana z urządzeniem, a nie z plikami. Wymusza to tworzenie systemu plików na jednym dysku, bądź też zadbanie o to, aby wszystkie nośniki danych były chronione.

Pewnym problemem jest koszt odzyskiwania danych z przeszłości. Odtwarzanie pojedynczego pliku może pociągnąć za sobą konieczność odtworzenia całego dysku.

Systemy oparte na blokach pod pewnym względem przypominają lustrzane odbicia (RAID 1). Różnią się jedynie możliwością odtwarzania danych z przeszłości.

2.1.2. Ochrona danych na poziomie plików

Kolejny typ ciągłej ochrony danych jest oparty na integracji z systemem plików. W odróżnieniu od ochrony opartej na blokach, zmiany są rejestrowane oddzielnie dla każdego pliku i katalogu, a ochrona danych jest ściśle powiązana z systemem plików. Systemy oparte na plikach nie rozpatrują zawartości danych, które są zapisywane na dysku.

W przeciwieństwie do systemów opartych na blokach mogą pozwolić na ustalenie różnych strategii ochrony dla poszczególnych katalogów i grup plików. Są elastyczne — pozwalają na łatwe odtwarzanie poszczególnych plików, co jest bardzo naturalne z punktu widzenia użytkownika.

Ich przewaga w stosunku do systemów opartych na aplikacji polega na tym, że użytkownik zwykle potrzebuje ochrony różnych zasobów, nie tylko danych należących do wybranych aplikacji.

2.1.3. Ochrona danych na poziomie aplikacji

Jest to najbardziej zaawansowany sposób ciągłej ochrony danych. Sama aplikacja może spełniać warunki systemu ciągłej ochrony danych. Niektóre bazy danych, serwery poczty potrafią odzyskiwać skasowane informacje (np. usunięty rekord z tabeli lub list).

System ochrony może być bezpośrednio wbudowany w aplikację lub działać jako oddzielny agent, lecz zawsze opiera się na głębokiej znajomości struktury danych aplikacji. Z tego powodu jest on zawsze unikatowy, napisany specjalnie dla konkretnej aplikacji. Niektóre systemy ochrony danych, w szczególności te przeznaczone dla baz danych, są w pełni przetestowane, certyfikowane i wspierane przez producentów chronionych aplikacji.

2.2. Ciągła ochrona danych czy prawie ciągła

Istotną cechą ciągłej ochrony danych jest możliwość odtworzenia raz zapisanych informacji, bez względu na to czy awaria nastąpi za kilka dni, czy sekundę po zapisaniu. W wielu przypadkach firmy tworzące takie systemy chciały odstąpić od tej zasady i wprowadziły termin prawie ciągłej ochrony danych (ang. *near-continuous data protection*).

Bliska ciągłość charakteryzuje się niedługimi odstępami w czasie pomiędzy wykonaniem kolejnych kopii bezpieczeństwa. Pomimo niewielkich odstępów dane zapisane w ostatnich sekundach lub minutach przed awarią są nie do odzyskania.

2.3. Ciągła ochrona danych a macierz dysków RAID

Różnica pomiędzy ciągłą ochroną danych a macierzą dysków RAID polega na tym, że RAID może chronić jedynie przed skutkami awarii dysku, pozwalając odzyskać jedynie najświeższą wersję danych. Natomiast ciągła ochrona danych zabezpiecza również przed innymi awariami (np. oprogramowania), umożliwiając dostęp do wcześniejszych wersji plików.

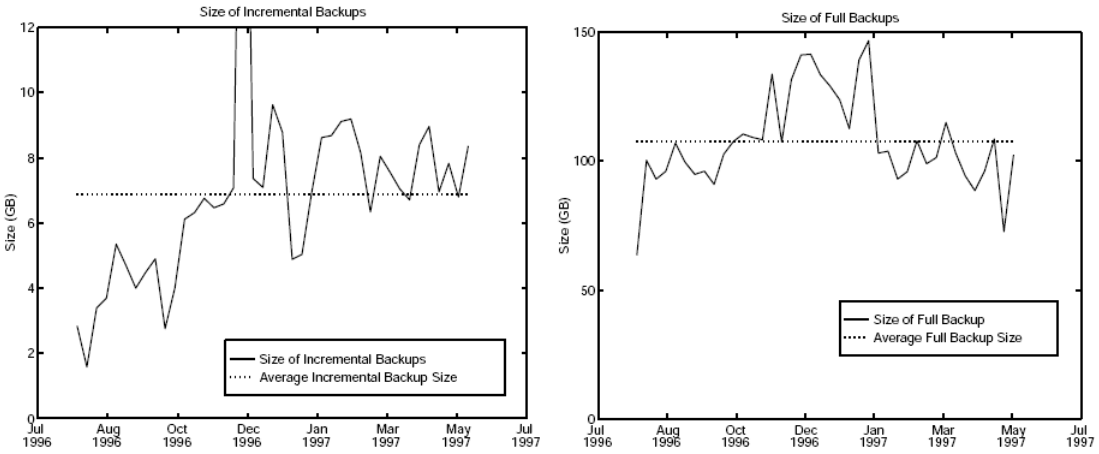
2.4. Miejsce przechowywania historii zmian

Historia zmian plików musi być przechowywana nie tylko na lokalnym komputerze, ale również na zdalnych serwerach. Czasami repozytoria danych mogą być replikowane pomiędzy wieloma zdalnymi serwerami, co dodatkowo podnosi stopień bezpieczeństwa chronionych informacji.

2.5. Rozmiar kopii zapasowej

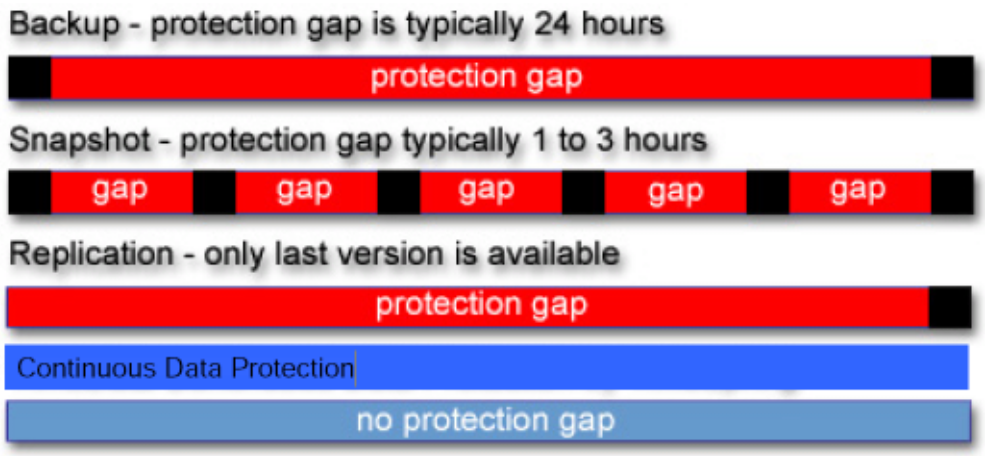
Rysunek 2.2 pokazuje różnice w rozmiarach pełnych i przyrostowych kopii zapasowych wykonywanych przez niecały rok w Georgia Tech's College of Computing. Archiwizacja dotyczyła około 400 stacji roboczych. Pełne kopie były wykonywane co tydzień, a przyrostowe codziennie. Jak widać na rysunku, średni rozmiar pełnych kopii był 15 razy większy niż rozmiar kopii przyrostowej.

Choć założenia ciągłej ochrony danych nie mówią wprost o tworzeniu przyrostowych kopii, wykonywanie pełnych kopii po każdej operacji na danych nie jest racjonalnym pomysłem. Dlatego rozmiar kopii zapasowej w przypadku ciągłej ochrony danych najczęściej wynosi kilka procent rozmiaru oryginalnych danych i jest znacznie mniejszy niż dla tradycyjnych sposobów ochrony danych, opartych na regularnej pełnej archiwizacji.



Rysunek 2.2: Rozmiar pełnej i przyrostowej kopii zapasowej. Źródło: [5]

2.6. Porównania z innymi sposobami ochrony danych



Rysunek 2.3: Porównanie długości przerw w ochronie danych charakterystycznych dla różnych sposobów zapewniania bezpieczeństwa. Źródło: [21]

Rysunek 2.3 ilustruje porównanie przerw w ochronie danych charakterystycznych dla różnych sposobów zapewniania bezpieczeństwa.

2.6.1. Regularna archiwizacja na zdalnym dysku

Łatwym i stosunkowo skutecznym sposobem ochrony danych jest ich regularna archiwizacja na zdalnym dysku. Można ją wykonywać np. za pomocą programów *rsync* i *rdiff-backup*. Ten sposób ochrony zabezpiecza dane przed skutkami awarii lokalnego dysku i jest dość mało pracochłonny w porównaniu z innymi metodami ochrony danych.

Jednak nawet częste wykonywanie kopii zapasowych nie chroni przed utratą cennych informacji zapisanych pomiędzy jedną a drugą archiwizacją.

Warto zauważyć, że regularne wykonywanie kopii zapasowych na żądanie, w przeciwieństwie do CDP, wymaga często dużego nakładu czasu — im bardziej aktualne kopie chcemy posiadać, tym częściej musimy je tworzyć i tym więcej czasu na to poświęcić.

2.6.2. Archiwizacja na nośnikach optycznych

Archiwizacja danych na nośnikach optycznych takich jak płyty CD czy DVD cechuje się — w porównaniu z archiwizacją na dysku — znacznie większą odpornością na awarie sprzętowe. Niemniej jednak wszelkie nośniki z cennymi danymi przechowywane w tym samym miejscu, w którym stoi nasza stacja robocza, są w niemal równym stopniu co ona narażone na kataklizmy takie jak powódź czy też pożar.

Kopiowanie danych na CD lub DVD wymaga jeszcze więcej nakładów czasowych niż archiwizacja na dysku twardym. Ze względu na niewielką pojemność nośników optycznych, wykonywanie kopii bezpieczeństwa dużej ilości danych jest mało wydajne. Osobnym problemem jest sposób przechowywania płyt oraz zbyt długi czas dostępu do zarchiwizowanych informacji.

2.6.3. Archiwizacja na taśmach

Używanie taśm jako nośników kopii zapasowych jest dość dobrym pomysłem, jeśli archiwizacji musi podlegać duża ilość danych. Tym niemniej, w przeliczeniu na gigabajt, są one obecnie droższe niż dyski twarde.

Niestety taśmy, podobnie jak małe nośniki optyczne, są narażone na ryzyko wystąpienia kataklizmów i związaną z tym utratę danych wraz z kopiami. Tak jak wszystkie wymienione wyżej sposoby ochrony danych, także i ten nie zapewnia, że w razie awarii podstawowego nośnika odzyskamy wszystkie cenne informacje. Wręcz przeciwnie — utracimy wszystko to, co zrodziło się pomiędzy jedną a drugą archiwizacją. Ma to niebagatelne znaczenie we wszystkich przedsiębiorstwach, które nieustannie produkują nowe dane i modyfikują stare.

2.7. Systemy ciągłej ochrony danych dostępne na rynku

Obecnie na rynku istnieje kilkanaście systemów ciągłej ochrony danych. Należą do nich między innymi:

- IBM Tivoli Continuous Data Protection for Files [10],
- Atempo LiveBackup [3],
- Mendocino RecoveryONE [15],
- Mimosa NearPoint for Microsoft Exchange [16],
- Revivio CPS 1200 [20],
- XOsoft Enterprise Rewinder [32],
- Symantec Backup Exec for Windows Servers [25],
- SonicWALL CDP [23].

Większość z nich jest przeznaczona wyłącznie dla różnych aplikacji firmy Microsoft (najczęściej MS Exchange, MS SQL) i w związku z tym pracuje w środowisku Microsoft Windows oraz wymaga przeglądarki Internet Explorer. Niemal wszystkie posiadają scentralizowane narzędzia administracyjne, pozwalające na łatwą obsługę wielu stacji roboczych.

Na uwagę zasługuje Revivio CPS 1200 [20], który zapewnia ochronę danych już na poziomie urządzenia blokowego. W odróżnieniu od wszystkich innych wymienionych tu systemów,

```
[user@machine] echo "This is the original foo.txt" > foo.txt
[user@machine] snapshot
Snapshot on .  1057845484
[user@machine] echo "This is the new foo.txt." > foo.txt
[user@machine] cat foo@1057845484
This is the original foo.txt.
[user@machine] cat foo
This is the new foo.txt.
```

Rysunek 2.4: Zapisywanie obrazu aktualnego stanu systemu plików i odwołanie do danych z przeszłości. Źródło: [19]

nie wymaga instalowania na każdym chronionym komputerze specjalnego oprogramowania. Mendocino RecoveryONE [15] oraz XOsoft Enterprise Rewinder [32] pracują nie tylko pod Windows, ale również w środowiskach uniksowych. Natomiast cechą wyróżniającą systemu Mimosa NearPoint for Microsoft Exchange [16] jest to, że nawet zwykłym użytkownikom oferuje szeroki dostęp do informacji. Mogą oni odzyskiwać swoje dane bez pomocy administratora.

Warto także zwrócić uwagę na LiveVault InSync [6], reklamowany jako system ciągłej ochrony danych. W rzeczywistości realizuje on jednak prawie ciągłą ochronę danych — około 100 razy dziennie generuje kopie chronionych plików.

Niestety, koncerny oferujące wyżej wymienione produkty nie udostępniają opisów sposobu działania ich aplikacji i realizacji zadań ciągłej ochrony danych.

2.8. Inne projekty powiązane z ciągłą ochroną danych

2.8.1. Ext3COW

Ext3cow, czyli trzeci rozszerzony system plików z kopiowaniem przy zapisie (ang. *third extended filesystem with copy-on-write*) to system plików z wersjonowaniem, zbudowany w oparciu o standardowy system plików w Linuksie — *ext3*. Został on zaprojektowany jako platforma spełniająca wymagania prawne stawiane przez ustawy prawa amerykańskiego[2].

Ext3cow nie jest systemem ciągłej ochrony danych, ponieważ historia plików jest zapisywana wyłącznie na żądanie użytkownika. Wersjonowanie plików odbywa się w nim poprzez kopiowanie starych bloków podczas zapisu nowych danych. Dzięki interfejsowi punktów czasowych (ang. *time-shifting interface*) umożliwia ciągły dostęp do danych z przeszłości w czasie rzeczywistym.

Zapis historii polega na wykonaniu "zdjęcia" (ang. *snapshot*), do którego później można się odwołać. Przykład użycia systemu plików *ext3cow* ukazuje rysunek 2.4. Więcej szczegółów dotyczących ext3cow można znaleźć w [19].

[2]Wymagania te dotyczą zapewnienia odpowiednich mechanizmów kontroli wewnętrznej spółek.

Rozdział 3

System plików w przestrzeni użytkownika

W systemie Linux wszystkie operacje na plikach i danych w nich zawartych mogą być wykonywane jedynie przez system operacyjny. Ze względu na możliwość obsługi wielu użytkowników jednocześnie, dane każdego z nich są chronione przed innymi użytkownikami systemu. Programy działające w przestrzeni użytkownika nie posiadają żadnych uprawnień do bezpośredniego odczytu bądź modyfikacji plików. Jedynie za pośrednictwem systemu operacyjnego mogą one zlecać różne operacje na plikach.

Do niedawna, jedynym sposobem na stworzenie systemu plików było napisanie nowego modułu do jądra Linuksa. Wiązało się to z różnymi trudnościami, gdyż, w odróżnieniu od zwykłych programów, pisanie modułów do systemu operacyjnego jest bardzo skomplikowanym i często czasochłonnym zadaniem. Programy uruchamiane w przestrzeni jądra mogą wyrządzić znacznie większe szkody w systemie, łącznie z zawieszeniem całego systemu i/lub zniszczeniem danych przechowywanych w komputerze. Napisany moduł jest trudno przetestować i wykryć w nim błędy.

Implementacja własnego modułu jest relatywnie prosta, jeśli tylko nie występują żadne problemy. Jednakże pierwszy błąd ujawnia złożoność przedsięwzięcia. Błędy spotykane w jądrze prowadzą często do zawieszenia systemu, przez co utrudniają prace nad ich wykryciem i usunięciem. Długi proces kompilacji, uruchamiania i testowania dodatkowo potęguje ten efekt.

Kolejną niedogodnością dla wielu programistów jest konieczność pisania modułów do jądra Linuksa w języku *C*. Choć na świecie dość często używany, nie jest to najprostszy język programowania. Inne języki oferują bardziej zaawansowane funkcje, takie jak automatyczne odśmiecanie pamięci, których brakuje w języku *C*.

Nie wszystkie elementy programu w równie łatwym stopniu implementuje się w jądrze. Kompresje plików czy komunikacje z internetem lepiej przeprowadzić w przestrzeni użytkownika. Ogromnym atutem zwykłych programów jest możliwość skorzystania z wielkiego zbioru bibliotek, których brakuje w jądrze.

3.1. Przykłady systemów plików w przestrzeni użytkownika

Powstało wiele projektów, których zadaniem było ułatwienie tworzenia nowych systemów plików. Choć w znacznej części były to osobiste projekty badawcze lub hobbistyczne programy, dziś mają coraz szersze zastosowanie. W dalszej części rozdziału przedstawię kilka z nich i krótko scharakteryzuję.

3.1.1. LUFS

Program *LUFS* pozwala na montowanie zdalnych systemów plików, które są dostępne przez protokoły sieciowe *FTP*, *SSH* i inne. Dostęp do zdalnych plików jest transparentny dla użytkownika i aplikacji. Wszystkie operacje wykonuje się tak jak na lokalnych plikach.

LUFS jest hybrydowym systemem plików, w skład którego wchodzą: program uruchamiany w przestrzeni użytkownika i moduł do jądra. Program realizuje zadania systemu plików i utrzymuje komunikację ze zdalnym węzłem. Część jądra przekazuje jedynie wszystkie żądania użytkownika do programu. Komunikacja pomiędzy modułem i programem jest dokonywana poprzez gniazda uniksowe (ang. *unix domain socket*). Dodatkowe informacje na ten temat można znaleźć w [14].

3.1.2. Podfuk

Autor w swoim projekcie przedstawił oryginalny pomysł na realizację nowego systemu plików. Zamiast tworzyć cały program od podstaw, wykorzystał istniejącą już implementację protokołu *NFS*[1] w Linuksie oraz bibliotekę wirtualnego systemu plików zawartą w programie *Midnight Commander*. Cała praca polegała na zmodyfikowaniu programu `rpc.nfsd`[2], w ten sposób, aby współpracował z wcześniej przygotowaną biblioteką. Szczegóły implementacji zostały opisane w [13].

Autor natknął się na spore problemy z wydajnością. Niezadowolony z szybkości systemu *NFS*, postanowił zamienić go na system plików *Coda*[3], który również jest zaimplementowany w jądrze Linuksa. Nowa wersja programu, jak przystało na świat Wolnego Oprogramowania, otrzymała nową nazwę — *Uservfs*.

3.1.3. UFO

UFO jest globalnym systemem plików, który udostępnia zasoby internetu w taki sposób, jakby znajdowały się na lokalnym dysku. Dostęp do plików i stron WWW odbywa się przez protokoły sieciowe *HTTP* lub *FTP* i jest transparentny dla programów użytkownika.

Autorzy projektu prezentują zupełnie inne podejście do problemu tworzenia i administracji systemów plików. W swojej pracy [1] wykazali, że realizacja systemu pliku nie wymaga interwencji w systemie operacyjnym. Zamiast wprowadzać nową obsługę operacji na plikach do jądra, modyfikację można wykonać w innych miejscach, m. in. w programie użytkownika lub bibliotece wejścia-wyjścia[4]. Rozwiązaniem jest wprowadzenie dodatkowej warstwy pośredniczącej między procesami a jądrem.

Pomysł autorów polega na wykorzystaniu istniejących narzędzi przeznaczonych do śledzenia procesów[5]. W celu udostępnienia światowych zasobów danych, system plików *UFO* podłącza się pod proces użytkownika, a następnie rozpoczyna przechwytywanie wywołań funkcji systemowych (ang. *syscall*). Odwołania do plików należących do *UFO* są wykrywane i realizowane bezpośrednio, z pominięciem jądra systemu operacyjnego. Schemat wykonania pojedynczej operacji prezentuje rysunek 3.1.

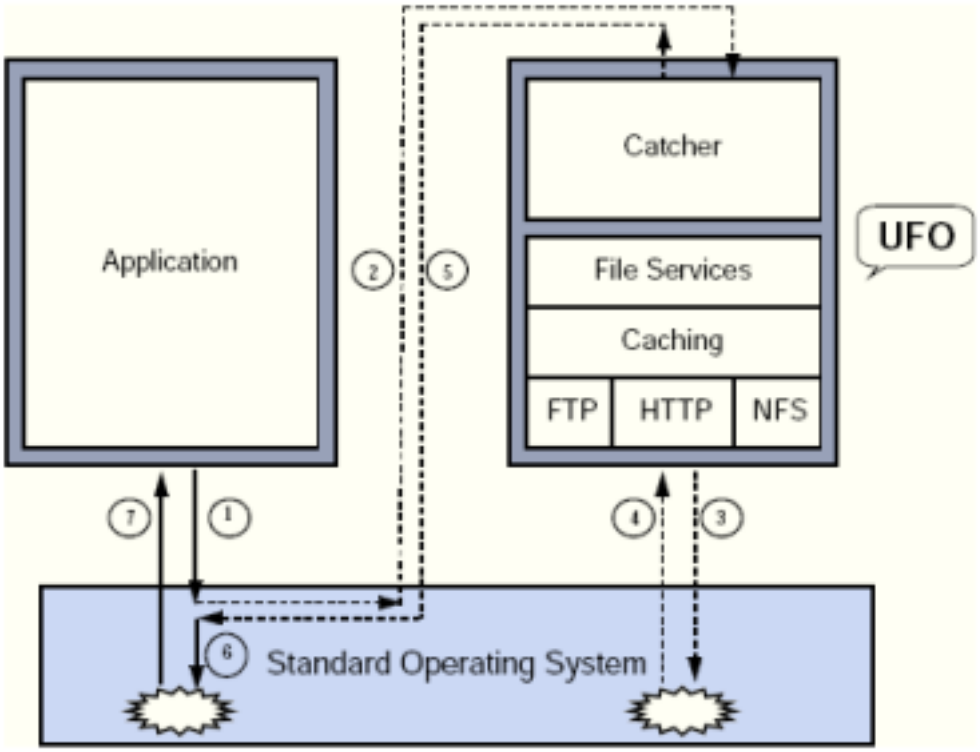
[1] Network File System — sieciowy system plików opracowany przez firmę Sun Microsystems. Jest on powszechnie używany w systemach uniksowych.

[2] Program świadczący usługę NFS w systemie Linux. Oparty jest na protokole *Remote Procedure Call*.

[3] Sieciowy, rozproszony system plików, wywodzący się z AFS (Andrew File System).

[4] W systemach z rodziny Unix najczęściej używana jest biblioteka *glibc*.

[5] Mechanizm śledzenia programów służy do wyszukiwania błędów w programach. Jest on wykorzystywany przez różne narzędzia diagnostyczne, takie jak *strace* czy *gdb*.



Rysunek 3.1: Architektura systemu plików *UFO*. Źródło: [1]

System plików *UFO* jest uniwersalnym rozwiązaniem. Nie wymaga modyfikacji sytemu operacyjnego, nie wymusza rekompilacji programów czy bibliotek. Można go uruchamiać bez uprawnień administratora. Jest to szczególnie istotne gdy posiadamy podstawowe konto, a zmiana konfiguracji serwera nie jest możliwa (np. ze względu na procedury bezpieczeństwa lub nieprzychylność administratora).

Niestety cena jaką przyjdzie zapłacić na uniwersalność jest duży narzut pracy spowodowany interpretacją wywołań systemowych, co wiąże się z dodatkowymi przełączeniami kontekstu pracy procesora. *UFO* jest projektem badawczym, który może znaleźć zastosowanie w aplikacjach, które w niewielkim stopniu wykorzystują operacje na plikach.

3.2. FUSE

Geneza projektu *FUSE* sięga systemu plików *AVFS* [26]. Od 2001 roku Miklos Szeredi rozwija system plików działający w przestrzeni użytkownika, który umożliwia programom zaglądanie do skompresowanych archiwów czy dostęp do zdalnych plików bez konieczności ich modyfikacji czy rekompilacji. Na potrzeby swojego projektu stworzył moduł do jądra Linuksa, który służy do komunikacji między warstwą *VFS*[6] w jądrze a systemem plików *AVFS*. Po trzech latach pracy Szeredi zdecydował się wydzielić część przeznaczoną dla systemu operacyjnego i rozpoczął osobny projekt *Filesystem in Userspace*.

FUSE umożliwia zaimplementowanie systemu plików, który będzie wykonywany w przestrzeni użytkownika. Przyczynia się do znaczącego uproszczenia jego budowy. Najprostszy system można napisać w stu linijkach kodu źródłowego.

[6]Virtual File System lub Virtual Filesystem Switch — jest abstrakcyjną warstwą systemu plików w jądrze. Dostarcza jednolity interfejs wspólny dla wszystkich systemów plików obsługiwanych przez jądro. Źródło: [31].

Ogromną bolączką projektu *FUSE* jest brak dobrej dokumentacji. Głównym źródłem informacji pozostaje kod źródłowy wraz z komentarzami. Pewnych informacji dostarcza strona domowa projektu [27], a także portal Wiki [27].

FUSE został włączony do głównej gałęzi rozwojowej Linuksa począwszy od wersji 2.6.14.

Projekt ten jest ciągle rozwijany. Obecnie w *FUSE* w wersji 2.6 autor pracuje nad mechanizmem blokowania plików. Od czasu włączenie do jądra *FUSE* zyskało sporą popularność. Pojawiły się także implementacje na innych systemach operacyjnych — np. w FreeBSD.

3.2.1. Założenia projektu *FUSE*

Głównym założeniem autora jest znaczące ułatwienie pisania nowych systemów plików.

Celem projektu *FUSE* jest umożliwienie stworzenia systemu plików, który:

- jest wykonywany w przestrzeni użytkownika,
- nie wymaga rekompilacji czy poprawiania jądra,
- może być uruchamiany bez praw administratora,
- jest efektywny.

3.2.2. Właściwości projektu *FUSE*

Wśród właściwości *FUSE* należy wymienić:

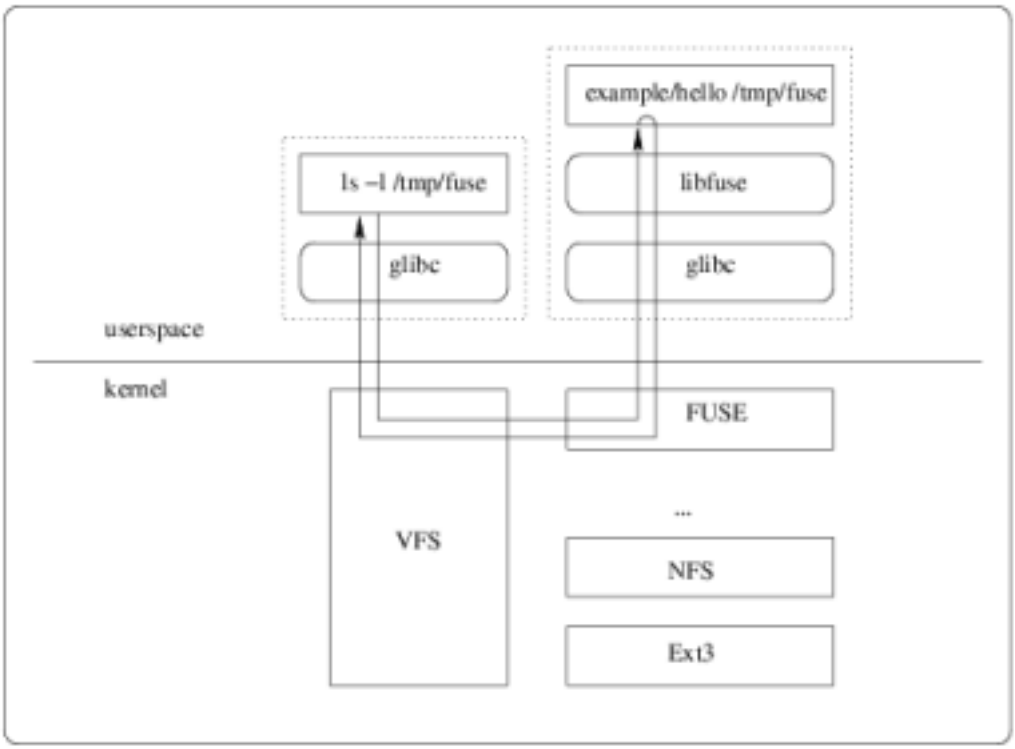
- jeden moduł w jądrze, który obsługuje wszystkie systemy plików,
- biblioteka użytkownika z prostym interfejsem,
- niewielki narzut pracy na obsługę operacji wejścia-wyjścia,
- bezpieczna implementacja (posiada zabezpieczenia przed złośliwym działaniem użytkownika),
- potwierdzone, stabilne działanie systemu.

3.2.3. Architektura systemu

FUSE składa się z trzech głównych części:

- modułu systemu operacyjnego,
- biblioteki programistycznej,
- pomocniczych programów do montowania i odmontowywania systemów plików.

Komunikacja między systemem operacyjnym a programem obywa się przez deskryptor pliku, przekazywany przy otwieraniu urządzenia znakowego `/dev/fuse`. Schemat realizacji pojedynczej operacji na pliku najlepiej ukazuje rysunek 3.2.



Rysunek 3.2: Architektura systemu *FUSE*. Źródło: [27]

3.2.4. Interfejs programisty

Biblioteka *libfuse* oferuje dwa interfejsy dostępu do systemu operacyjnego: niskopoziomowy i standardowy. Główna różnica pomiędzy nimi polega na kontroli nad rejestrowaniem systemu pliku w jądrze. Niskopoziomowy interfejs pozwala programiście na zarządzanie sesjami i kanałami komunikacji z jądrem.

W znacznej części odpowiada ona operacjom jakie można wykonać na plikach i katalogach.

Stworzenie nowego systemu plików jest równoznaczne z zaimplementowaniem funkcji opisanych w strukturze `fuse_operations`. *FUSE* nie wymaga implementacji wszystkich dostępnych operacji, można wybrać tylko pewien ich podzbiór. Ma to ogromne znaczenie dla wirtualnych systemów plików, gdzie nie wszystkie operacje muszą mieć sens (np. system plików oferujący tylko odczyt danych może pominąć operację zapisu czy modyfikacji atrybutów).

Właściwości interfejsu

Interfejs *FUSE* został napisany w języku *C*. W celu zapewnienia prawidłowego działania autor wprowadził kilka wymagań:

- nazwa pliku i katalogu jest wartością absolutną, zawiera pełną ścieżkę dostępu,
- parametry przekazywane do funkcji mogą być odczytywane lub modyfikowane tylko podczas wywołania, zabronione jest zapamiętywanie wskaźników,
- pierwsze odwołanie do pliku lub katalogu będzie poprzedzone żądaniem *getattr*.

Wywołanie funkcji *getattr* jest związane z pewnymi czynnościami jądra, takimi jak sprawdzenie praw dostępu do pliku.

Opis interfejsu

Oto podstawowe operacje udostępniane przez interfejs *FUSE*:

- `int getattr (const char * path, struct stat * st);`
- `int opendir (const char *, struct fuse_file_info *);`
- `int readdir (const char *, void *, fuse_fill_dir_t, off_t, struct fuse_file_info *);`
- `int releasedir (const char *, struct fuse_file_info *);`
- `int open (const char *, struct fuse_file_info *);`
- `int read (const char *, char *, size_t, off_t, struct fuse_file_info *);`
- `int write (const char *, const char *, size_t, off_t, struct fuse_file_info *);`
- `int truncate (const char *, off_t);`
- `int release (const char *, struct fuse_file_info *);`
- `int mknod (const char *, mode_t, dev_t);`
- `int mkdir (const char *, mode_t);`
- `int unlink (const char *);`
- `int rmdir (const char *);`
- `int rename (const char *, const char *);`
- `int chmod (const char *, mode_t);`
- `int chown (const char *, uid_t, gid_t);`
- `int utime (const char *, struct utimbuf *);`

3.2.5. Wsparcie dla różnych języków programowania

FUSE oferuje wsparcie dla następujących języków programowania:

- C (natywny interfejs),
- interfejs C++,
- interfejs dla języka Java, z wykorzystaniem JNI,
- interfejs dla języka C# (dla platform *.Net* i *Mono*),
- Perl,
- Python,
- Ocaml,
- i innych egzotycznych języków programowania.

3.2.6. Systemy plików bazujące na projekcie *FUSE*

Oto lista przykładowych systemów plików, które korzystają z *FUSE*.

EncFs System plików z szyfrowaniem, który opiera się na szyfrowaniu pojedynczych plików, w przeciwieństwie do szyfrowania całych urządzeń blokowych.

GmailFS System plików pozwalający na wykorzystanie konta pocztowego *GMail* jako nośnika danych.

gphoto2-fuse-fs System plików umożliwiający dostęp do zdjęć zawartych w cyfrowym aparacie.

sshfs Zdalny system plików korzystający z szyfrowanego protokołu *SSH*.

Flickrfs System plików umożliwiający korzystanie z portalu `flickr.com` jak z lokalnego dysku.

BlogFS System plików ułatwiający przeglądanie i redagowanie artykułów w elektronicznych pamiętnikach.

FUSEPod System plików posiadający dostęp do *iPoda*.

mysqlfs System plików zapisujący pliki w bazie danych *MySQL*.

3.2.7. Bezpieczeństwo

Bezpieczeństwo w projekcie *FUSE* oznacza ochronę użytkownika i przed użytkownikiem.

Ochronę użytkownika — ponieważ montując system plików, udostępnia on swoje pliki i katalogi. Domyślnym działaniem *FUSE* jest umożliwienie ich odczytu i modyfikacji jedynie przez procesy właściciela[7]. Wymienione opcje montowania zmieniają to zachowanie:

allow_other usuwa restrykcje dotyczące osoby uprawnionej do korzystania z plików,

allow_root pozwala na dostęp do plików tylko właścicielowi i administratorowi.

Ochronę przed użytkownikiem — gdyż system plików jest częścią systemu operacyjnego. Użytkownik musi posiadać pełne prawa zapisu do katalogu, w którym chce zamontować własny system plików. Dodatkowo *FUSE* montuje wszystkie systemy plików z flagami *nosuid* i *nodev*.

3.2.8. Współbieżność

Biblioteka *FUSE* oferuje dwa tryby pracy. Podstawowym i zalecanym trybem jest praca wielowątkowa. Poprawia ona wydajność sytemu plików w przypadku wykorzystywania go przez więcej niż jednego użytkownika. W przypadku programów nieprzystosowanych do pracy wielowątkowej należy skorzystać z trybu jednowątkowego.

Moduł jądra *FUSE* potrafi obsłużyć wiele systemów plików naraz. Autor deklaruje możliwość zamontowania ponad tysiąca systemów plików.

[7] Należy zaznaczyć, że administrator zawsze ma możliwość obejścia tego zabezpieczenia. Przykładem jest zalogowanie się na konto użytkownika poleceniem `su - nazwa_uzytkownika`.

3.2.9. Ograniczenia

Do najważniejszych ograniczeń *FUSE* należą:

- brak buforowania żądań w jądrze Linuksa,
- operacja zapisu jest przekazywana do systemu plików w blokach o maksymalnym rozmiarze 4 kB, co stanowi wielkość ramki pamięci,
- brak obsługi "magicznych" operacji na plikach *ioctl*.

3.3. Podsumowanie

W Linuksie wszelkie operacje na plikach są wykonywane w przestrzeni jądra. Procesy z przestrzeni użytkownika mogą odczytywać i modyfikować pliki jedynie za pośrednictwem systemu operacyjnego. W związku z tym jeszcze do niedawna tworzenie nowego systemu plików wiązało się z koniecznością napisania nowego modułu dla jądra Linuksa, co może być nie lada wyzwaniem.

Dziś by stworzyć nowy system plików, nie trzeba zmagać się z takimi trudnościami, bo na przestrzeni ostatnich lat powstało wiele modułów do jądra i bibliotek, na bazie których można opracować własny system plików działający w przestrzeni użytkownika. Należą do nich między innymi omówione w tym rozdziale *LUFS* i *FUSE*.

FUSE to projekt wciąż rozwijany, oferujący bardzo przejrzysty interfejs. Od niedawna jest zintegrowany z jądrem Linuksa i między innymi dzięki temu doskonale nadaje się do eksperymentów. Dlatego stał się podstawą autorskiego projektu *BUFS*, omówionego w kolejnym rozdziale.

Rozdział 4

Projekt *BUFS*

Centralną częścią pracy jest autorski system zapewniający ciągłą ochronę danych, *BUFS*. Nazwa *BUFS* to skrót od angielskiej nazwy *BackUp FileSystem*.

Na rynku brakuje uniwersalnych i tanich rozwiązań oferujących ciągłą ochronę danych. Te, które istnieją, są przeznaczone dla systemów operacyjnych z rodziny Microsoft, często dla wyznaczonych programów. Dodatkowo w wielu sytuacjach wysoki koszt jest barierą dla ich wdrożenia. Niniejsza praca stara się zapełnić tę lukę.

Całość została podzielona na trzy części. Każda z nich ma ściśle określoną rolę w systemie *BUFS*. Te części to:

- system plików `bufs-mt`,
- aplikacja odzyskująca utracone pliki `bufs-recovery`,
- zewnętrzne urządzenia blokowe.

W tym rozdziale zostaną przedstawione dwie pierwsze części projektu. Opis obejmie założenia, architekturę oraz implementację wymienionych programów. Udostępnianie zewnętrznych urządzeń blokowych zaprezentowane jest w rozdziale 5.

4.1. Założenia projektu

4.1.1. Ciągła ochrona danych

W założeniem projektu jest spełnienie wymagań ciągłej ochrony danych omówionej w rozdziale 2.1. Powinna być realizowana pełna ochrona, bez zbędnych odstępów czasowych pomiędzy kolejnymi archiwizacjami. Jest to równoznaczne z wykonywaniem kopii zapasowej dla każdej operacji zleconej przez użytkownika, jeszcze przed wysłaniem potwierdzenia jej realizacji. Kolejnym założeniem jest możliwość odzyskiwania wersji pliku, które istniały w przeszłości.

4.1.2. Koszt budowy

Celem projektu jest minimalizacja kosztów związanych z budową i utrzymaniem systemu ciągłej ochrony danych. Na koszt systemu mają wpływ:

- sprzęt komputerowy i
- oprogramowanie.

Wymagany sprzęt komputerowy powinien być dostępny powszechnie. Redukcja kosztów obejmuje także oprogramowanie, projekt *BUFS* powinien bazować na oprogramowaniu udostępnianym nieodpłatnie (np. na licencji *GPL*). Obniża to koszt całego przedsięwzięcia.

4.1.3. Bezpieczeństwo

System plików `bufs-mt` powinien oferować podobny poziom bezpieczeństwa i kontroli dostępu do plików i katalogów co inne systemy plików spotykane w Linuksie. Dotyczy to sprawdzania uprawnień uniksowych oraz obsługiwania listy kontroli dostępu[1].

Zakłada się, że system jest uruchomiony w prywatnej sieci (ewentualnie odgrodzony od internetu ścianą ogniową), a dostęp do katalogów, w których `bufs-mt` składuje informacje, posiadają jedynie odpowiednie osoby. *BUFS* nie posiada mechanizmów obronnych przed zewnętrzną ingerencją w jego struktury danych. Taka manipulacja może łatwo doprowadzić do niespójności zarchiwizowanych kopii. Również zakłócenia w komunikacji pomiędzy węzłami mogą być przyczyną utraty danych. Rolą administratora jest zapewnienie odpowiedniego zabezpieczenia komputerów i sieci, w której się znajdują.

4.1.4. Uniwersalizm i elastyczność

BUFS musi być uniwersalny, nie ograniczać się wyłącznie do wąskiej grupy aplikacji oraz być elastyczny, pozwalając na dostęp do wybranych danych bez konieczności pracochłonnego przeszukiwania kopii. Takie założenia może spełnić jedynie system oparty na plikach.

4.2. Architektura systemu *BUFS*

Projekt *BUFS* został stworzony z myślą o elastycznym budowaniu systemu ciągłej ochrony danych. Minimalne wymagania systemu spełnia każdy komputer wyposażony w dysk twardy i odpowiednie oprogramowanie.

Wydajność i odporność na awarie zależą w znacznym stopniu od konfiguracji systemu. Zadaniem administratora jest optymalizacja docelowej konfiguracji. Nieumiejętne skonfigurowanie systemu może doprowadzić do kilkukrotnego spadku wydajności oraz do utraty odporności na awarie sprzętu komputerowego.

Zalecana konfiguracja zawiera następujące elementy:

- dwa węzły (komputery),
- dwa dyski twarde, po jednym w każdym węźle,
- szybka sieć komputerowa łącząca węzły.

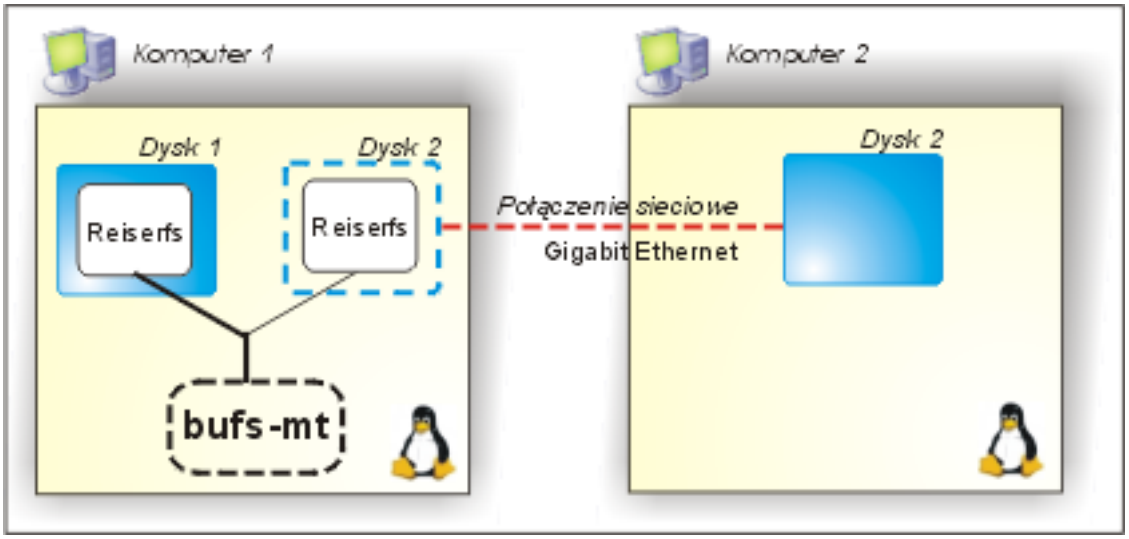
Na bazie wymienionego sprzętu tworzony jest system, w którym pierwszy komputer jest dominujący, a drugi udostępnia jedynie zasoby dyskowe. Użycie dwóch węzłów zostało podyktowane wymaganiami ciągłej ochrony danych. Pojedynczy komputer nie jest w stanie zapewnić ochrony m. in. przed skutkami awarii jednej maszyny.

Zamiast pojedynczych dysków dopuszczalne jest tworzenie macierzy RAID lub wolumenów LVM.

[1] Access Control List — lista kontroli dostępu. Wykorzystywana do definiowania uprawnień dostępu do pliku. Jest znacznie bardziej rozbudowana niż standardowe uprawnienia systemu plików w systemach uniksowych.

Interfejs sieciowy powinien oferować przepustowość zbliżoną do możliwości interfejsu ATA. Wśród niedrogich urządzeń dostępnych na rynku właściwość taką mają jedynie sieci Gigabit Ethernet[2].

Proponowana konfiguracja systemu *BUFS* została przedstawiona na rysunku 4.1.



Rysunek 4.1: Zalecana konfiguracja systemu ciągłej ochrony danych *BUFS*

Nie ma przeszkód do jednoczesnego budowania wielu systemów *BUFS* na tych samych węzłach. Należy się jednak liczyć ze znacznym spadkiem wydajności w przypadku wykorzystania tych samych dysków lub interfejsów sieciowych. Dodatkowo niekorzystny wpływ na wydajność rozwiązania ma współdzielenie procesora i szyny systemowej.

4.3. Architektura systemu plików `bufs-mt`

`Bufs-mt` jest wirtualnym systemem plików. Oznacza to, że w przeciwieństwie do standardowych systemów plików, takich jak *ext3* czy *reiserfs* nie potrafi on bezpośrednio wykorzystać urządzenia blokowego jako miejsca przechowywania informacji. `Bufs-mt` korzysta z pośrednictwa dowolnych, standardowych systemów plików udostępnianych w komputerze, w celu zapisywania danych.

Odporność projektu *BUFS* na awarie jest powiązana z wykorzystywanymi systemami plików. Dlatego polecane do współpracy z *bufs-mt* zalecane są systemy plików posiadające mechanizm księgowania operacji.

4.3.1. Podział na dwa katalogi

Podstawą działania systemu plików *bufs-mt* są dwa katalogi, w których przechowywane są wszystkie informacje potrzebne do jego działania. Miejsce zapamiętywania danych użytkownika zawiera:

- katalog ze źródłowymi plikami oraz
- katalog historii operacji.

[2]Sieć Gigabit Ethernet umożliwia przesyłanie danych z prędkością do ok. 100MB/s, co odpowiada interfejsowi ATA100.

Tworzenie kopii zapasowej będzie polegać na zapamiętywaniu historii operacji wykonywanych w systemie plików.

4.3.2. Realizacja projektu w przestrzeni użytkownika

System plików `bufs-mt` jest to program działający w przestrzeni użytkownika, który wykorzystuje projekt *FUSE* do komunikacji z jądrem Linuksa. Znacząco ułatwiło to proces tworzenia systemu a w późniejszym czasie wyszukiwania błędów. Jednakże zastosowanie *FUSE* wpływa dość niekorzystnie na wydajność systemu, co zostało wykazane w rozdziale 6.2.

4.4. Implementacja

System został zaimplementowany w języku C++. Podczas realizacji zadania wykorzystano następujące biblioteki:

- libfuse — 2.5.3,
- boost — 1.33.1,
- log4cxx — 0.9.7.

Biblioteki boost i log4cxx wykorzystano odpowiednio do synchronizacji wątków działających w *bufs-mt* oraz do logowania zdarzeń w programie.

4.4.1. Struktura plików i katalogów

Struktura katalogów w części historii zmian odpowiada plikom i katalogom utworzonym przed użytkownika. Dla każdego chronionego pliku `bufs-mt` zakłada katalog o identycznej nazwie, w którym przechowywana będzie historia zmian. Każdy taki katalog zawiera następujące pliki:

1. `.bufs.priv.oper`,
2. `.bufs.priv.nums`,
3. `.bufs.priv.data-XXX`, gdzie XXX to kolejne dodatnie liczby.

Pierwszy nich z jest zawsze tworzony niezależnie od tego czy odpowiada chronionemu plikowi czy katalogowi. Zawiera informacje dotyczące wykonanych operacji, takich jak utworzenie lub usunięcie pliku/katalogu.

Kolejne dwa pliki powstają w wyniku zapisywania danych przez użytkownika. Pliki `.bufs.priv.data-XXX` zawierają historię wszystkich operacji zapisu i obcięcia jakie były wykonane na chronionym pliku. Wielkość pliku jest konfigurowalna i uzależniona od decyzji administratora. Natomiast ich liczba jest powiązana z rozmiarem danych jakie były zapisane do chronionego pliku[3]. Gdy jeden plik osiągnie maksymalny rozmiar, tworzony jest następny, z wyższym numerem. Liczba stworzonych plików z historią jest zapisana w pliku `.bufs.priv.nums`.

Struktura plików i katalogów w części źródłowego systemu plików jest odpowiednikiem struktury zakładanej przez użytkownika. Oznacza to, że źródłowy i eksportowany system plików są identyczne.

[3]Liczy się łączny rozmiar danych zapisany do pliku, a nie tylko końcowy rozmiar pliku.

Ze względu na wykorzystanie słowa `.bufs.priv` jako prefiksu nazwy dla wewnętrznych plików, użycie go w eksportowanym systemie plików jest zabronione. Stworzenie pliku zawierającego podany prefiks zakończy się w najlepszym wypadku utratą historii wykonanych operacji.

4.4.2. Sesja otwartego pliku

Dla każdego otwartego przez użytkownika pliku system *bufs-mt* tworzy nową sesję. Przechowywane są w niej informacje dotyczące odpowiadającego mu pliku w źródłowym katalogu oraz bieżący plik z historią, do którego trafiają kolejne adnotacje o zapisach. W ramach sesji zapamiętywane są:

- deskryptor odpowiedniego pliku w źródłowym katalogu,
- deskryptor pliku z historią przeprowadzonych operacji,
- listy niezapisanych wyników, które dotyczą wcześniejszych operacji zapisu bloku.

Deskryptor pliku źródłowego pozostaje niezmienny przez cały czas życia sesji — aż do zamknięcia pliku przez użytkownika. W odmiennej sytuacji jest plik zawierający historię, który może być zmieniany. Lista niezapisanych wyników służy do poprawy wydajności.

Z uwagi na możliwość wykonywania jednocześnie kilku operacji na jednym pliku dostęp do zmiennych w sesji jest broniony przez semafor (mutex). Zniszczenie sesji następuje z chwilą zamknięcia ostatniego deskryptora (wcześniej otwartego) pliku, co jest równoznaczne z wywołaniem funkcji `release` przez *FUSE*.

4.4.3. Struktura historii

`Bufs-mt` implementuje wszystkie operacje interfejsu *FUSE*, które zostały przedstawione w rozdziale 3.2.4. Tylko niektóre z nich są rejestrowane i zapisywane w historii. Oto one:

- utworzenie pliku (mknod),
- utworzenie katalogu (mkdir),
- usunięcie pliku (unlink),
- usunięcie katalogu (rmdir),
- zmiana nazwy (rename),
- zapis (write),
- ustawienie długości pliku (truncate).

Choć pozostałe operacje nie podlegają rejestracji, to są wykonywane na źródłowym pliku.

Zapamiętywana operacja zawiera następujące pola:

1. rozmiar operacji (32 bity),
2. typ operacji (32 bity),
3. unikatowy numer sekwencyjny (64 bity),

4. czas wykonania (32 bity),

5. zmienne pole — dane zawarte w operacji (różna długość),

6. ponownie rozmiar operacji (32 bity).

Unikatowy numer sekwencyjny tworzony jest na podstawie numeru montowania systemu plików `bufs-mt` oraz numeru operacji. Czas zapisany w strukturze to bieżący czas otrzymania zlecenia. Zmienne pole jest uzależnione od rodzaju przeprowadzanej operacji i zawiera:

- prawa dostępu dla tworzenia pliku i katalogu,
- aktualną i nową nazwę pliku w przypadku jej zmiany,
- blok zapisu, w tym położenie, długość i dane w operacji zapisu,
- nowa długość pliku w przypadku ustawiania długości.

Długość struktury zawierającej kopię operacji jest większa o 48 bajtów od zlecnoej operacji.

Jednoczesny zapis tego samego bloku w pliku jest niezdefiniowany (tak jak w zwykłych systemach plików). Występuje w takim przypadku wyścig (ang. *race condition*) pomiędzy zapisami, a końcowy efekt nie jest przewidywalny. Także zapisanie historii operacji może dokonać się w odwrotnej kolejności.

4.4.4. Zapisywanie potwierdzeń

Po wykonaniu operacji na źródłowym pliku system *bufs-mt* otrzymuje jej wynik. Jest to bardzo ważna informacja, która określa czy dana operacja powiodła się, a w przypadku zapisu bądź odczytu, w jakim stopniu. Wynik operacji jest zapisywany do historii natychmiast po jej wykonaniu.

Struktura wyniku zawiera następujące pola:

1. rozmiar operacji (32 bity),

2. typ operacji — potwierdzenie (32 bity),

3. numer potwierdzanej operacji (64 bity),

4. rezultat wykonania (32 bity),

5. ponownie rozmiar operacji (32 bity).

Jedynym odstępstwem od zasady niezwłocznego zapisu wyniku jest operacja zapisu danych. W tym przypadku wynik odkładany jest do tymczasowej kolejki wyników i będzie zapisany wraz z nową operacją zapisu. Jeśli następną operacją będzie zamknięcie pliku, to wynik zostanie zapisany przed końcowym zamknięciem pliku.

Rozmiar struktury wyniku wynosi 48 bajtów. Sumując to z dodatkowym rozmiarem struktury zapisu daje to 96 dodatkowych bajtów przeznaczonych na opis jednej operacji. Przyjmując maksymalny rozmiar operacji zapisu 4 kB, narzut pamięci wynosi ok. 2,5%.

4.4.5. Wyrównanie zapisu historii

Choć narzut związany z zapisywaniem historii jest znikomy, to powoduje zmianę rozmiaru zapisywanego bloku. Systemy plików dość słabo radzą sobie z wykonywaniem operacji na niepełnych blokach. Zapis danych, których rozmiar nie jest wielokrotnością bloku w systemie plików, powoduje dodatkowe opóźnienia związane z koniecznością odczytania bloku, naniesienia zmian i zapisania bloku.

Rozwiązaniem opisanego problemu, które stosuje *bufs-mt*, jest wyrównywanie bloku do najmniejszej wielokrotności większej lub równej od rozmiaru danych. Dzięki temu unika się wąskiego gardła jakie stanowi zapis niepełnych bloków. Wadą rozwiązania jest dodatkowy koszt pamięci masowej, którą zużywa się w celu zwiększenia rozmiaru bloku. W najgorszym przypadku, gdy użytkownik zleci zapis danych o długości jednego bloku w systemie plików, to rozmiar historii dla tej operacji będzie odpowiadać podwójnemu blokowi.

Istotną sprawą jest wyrównanie zapisu do wielkości bloku systemu pliku a nie tylko do rozmiaru sektora na dysku.

4.4.6. Synchronizacja danych

Wszystkie pliki wykorzystywane do przechowywania kopii zapasowej są otwierane z flagą `O_SYNC`. Gwarantuje to natychmiastowe wykonywanie zapisów na tych plikach.

Jedną z właściwości *FUSE* jest brak buforowania zleceń przez jądro systemu operacyjnego. Umożliwia to synchronizację danych pomiędzy aplikacją użytkownika a systemem plików *bufs-mt*.

Osobnym problemem persystentnego zapisu danych jest bufor dyskowy. Pomaga on zwiększyć wydajność dysku poprzez umieszczenie zapisu jedynie w podręcznym buforze zamiast na talerzu. Oznacza to utratę danych w przypadku awarii zasilania. *Bufs-mt* nie zabezpiecza w żaden sposób przed takim niebezpieczeństwem. Jedynym rozwiązaniem jest zakup urządzeń podtrzymujących zasilanie (UPS).

4.4.7. Twarde i symboliczne dowiązania

Dowiązania w uniksowych systemach plików powodują pewne trudności w tworzeniu kopii zapasowej. Problem wiąże się z występowaniem dwóch różnych nazw odpowiadających temu samemu plikowi.

W przypadku dowiązań symbolicznych taki problem nie istnieje. W odróżnieniu od twardych dowiązań, operacje na dowiązaniach symbolicznych wykonywane są w rzeczywistości na plikach, na które wskazują, co sprawia, że są przezroczyste dla `bufs-mt`. System plików `bufs-mt` w pełni wspiera dowiązania symboliczne.

4.4.8. Operacja zapisu

Istnieje wiele koncepcji zapisu danych. Różnią się miedzy sobą głównie oferowaną wydajnością. Pomiędzy poszczególnymi wersjami projektu sposób zapisu danych ulegał istotnym zmianom. Przyczyną była zbyt niska wydajność, a także brak gwarancji odporności na awarię.

W `bufs-mt` operacja zapisu bloku danych przez użytkownika przebiega w kilku etapach:

1. odebranie zlecenia zapisu,

2. równoległe wykonanie operacji w lokalnym systemie plików oraz zapisanie jej w historii (z historią zapisywany jest także wynik poprzedniej operacji zapisu),

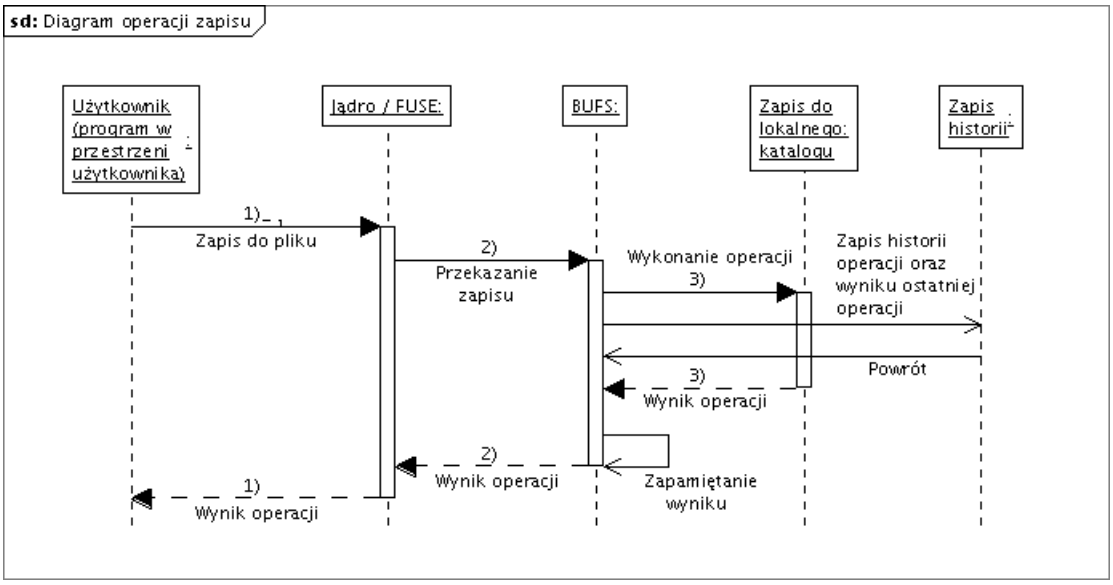
3. oczekiwanie na zakończenie operacji w lokalnym systemie plików,

4. zapamiętania wyniku operacji,

5. wysłanie potwierdzenia zapisu.

W celu równoległego wykonywania operacji zapisu bloku w lokalnym systemie plików i w historii, `bufs-mt` tworzy dodatkowe wątki. Ich liczba jest konfigurowalna przez administratora.

Przebieg operacji zapisu obrazuje rysunek 4.2.



Rysunek 4.2: Schemat dublowania zapisu bloku. Piewszy zapis wykonywany jest na lokalnym pliku, drugi wędruje do pliku z historią

4.5. Odzyskiwanie plików

Odzyskiwanie plików jest finalnym (jeśli chodzi o przetwarzanie historii zmian) narzędziem jakie oferuje projekt *BUFS*. Program `bufs-recovery` potrafi odtwarzać skasowane czy zamazane pliki.

Do odtwarzania danych wykorzystywany jest wyłącznie katalog zawierający historię wykonywanych operacji. Wybrane operacje są składane ponownie w celu odbudowania pliku.

Bufs-recovery stanowi niezależny program, który działa poza systemem plików `bufs-mt`. Ze względu na sposób wykonywania kopii zapasowej — dopisywanie nowych operacji do plików z historią — odzyskiwanie pliku możliwe jest podczas działania programu `bufs-mt`.

W celu odzyskania pliku należy podać numer ostatniej operacji (jeśli jest znany) lub datę i godzinę, z której pochodzi interesująca nas wersja pliku.

4.5.1. Sposób odtwarzania pliku

Znając operacje zapisu, jakie były przeprowadzone na danym pliku, można odtworzyć jego zawartość. W tym celu wystarczy złożyć bloki z danymi w jedną całość. Nie można jednak

dokonać tego w dowolny sposób, gdyż operacje mogą być ze sobą powiązane, tzn. mogą modyfikować bajty znajdujące się na tych samych pozycjach.

Dość logicznym sposobem na realizację odbudowy pliku jest ponowne wykonanie zarejestrowanych operacji. Choć jest on najłatwiejszy do zaimplementowania, to posiada niestety pewne wady. Pojedynczy plik może być wielokrotnie tworzony i usuwany, podczas gdy na na końcową zawartość pliku nie mają wpływu jego wcześniejsze wersje. Wykorzystanie całej historii do odbudowy ostatniej wersji może okazać się marnowaniem zasobów. Innym rozwiązaniem może być wyszukanie ostatniej adnotacji o utworzeniu pliku, ale także wymaga to dodatkowego przeczytania historii.

Kolejnym pomysłem na odtworzenie zawartości pliku jest wykonywanie w odwrotnej kolejności operacji zawartych w historii. Odczytując operację należy sprawdzić, czy na tej samym bloku danych nie została później wykonana inna operacja. W przypadku wykrycia kolizji należy pozostawić już odzyskane bajty oraz uzupełnić resztą bajtów zawartych w bieżącej operacji. Z operacji wykonych na danym bloku ostatnia decyduje o zawartości tego bloku — nowsze operacje przysłaniają starsze.

Strukturą bardzo pomocną w przeprowadzeniu opisanego procesu jest drzewo przedziałów. Można w nim przechować informacje o kolejnych zmianach dokonywanych na przedziałach bajtów. W programie *bufs-recovery* drzewo przedziałów zostało zaimplementowane za pomocą zbioru *STL*[4].

Spośród rejestrowanych operacji na plikach główne znaczenie mają operacje zapisu i ustawiania długości pliku. Pozostałe operacje — usuwanie bądź tworzenie plików i katalogów — są traktowane jak obcięcie pliku do zerowej długości.

Przykładowy scenariusz zapisu danych do pliku oraz proces odbudowy utraconych informacji zostały przedstawione na rysunkach 4.3 i 4.4.

Zgodnie z założeniami systemu ciągłej ochrony danych program `bufs-recovery` umożliwia odtworzenie danych z wybranego punktu w przeszłości. Aby uzyskać starszą (poprzednią) wersję pliku należy podać jako jedną z opcji datę i godzinę, z której pochodzi interesująca nas wersja pliku. Dostępna jest też możliwość precyzyjnego określenia numeru operacji, do której należy odbudować plik. Pełny wykaz opcji, wraz z opisem, znajduje się w załączniku A.3.

W przypadku skorzystania z opcji określających datę pliku, najnowsze operacje są pomijane, a odbudowa pliku rozpoczyna się od wskazanej operacji. Przykład odtwarzania danych z pewnego punktu w czasie przedstawia rysunek 4.5.

4.5.2. Zmiana nazwy pliku

Osobnym przypadkiem do rozpatrzenia jest operacja zmiany nazwy pliku. Występują dwie sytuacje w zależności od tego, czy przetwarzany plik został przeniesiony w inne miejsce czy też inny plik zmienił nazwę na obecny. Pierwszy przypadek jest traktowany jako usunięcie pliku (czyli obcięcie do zerowej długości), co kończy odbudowę danych.

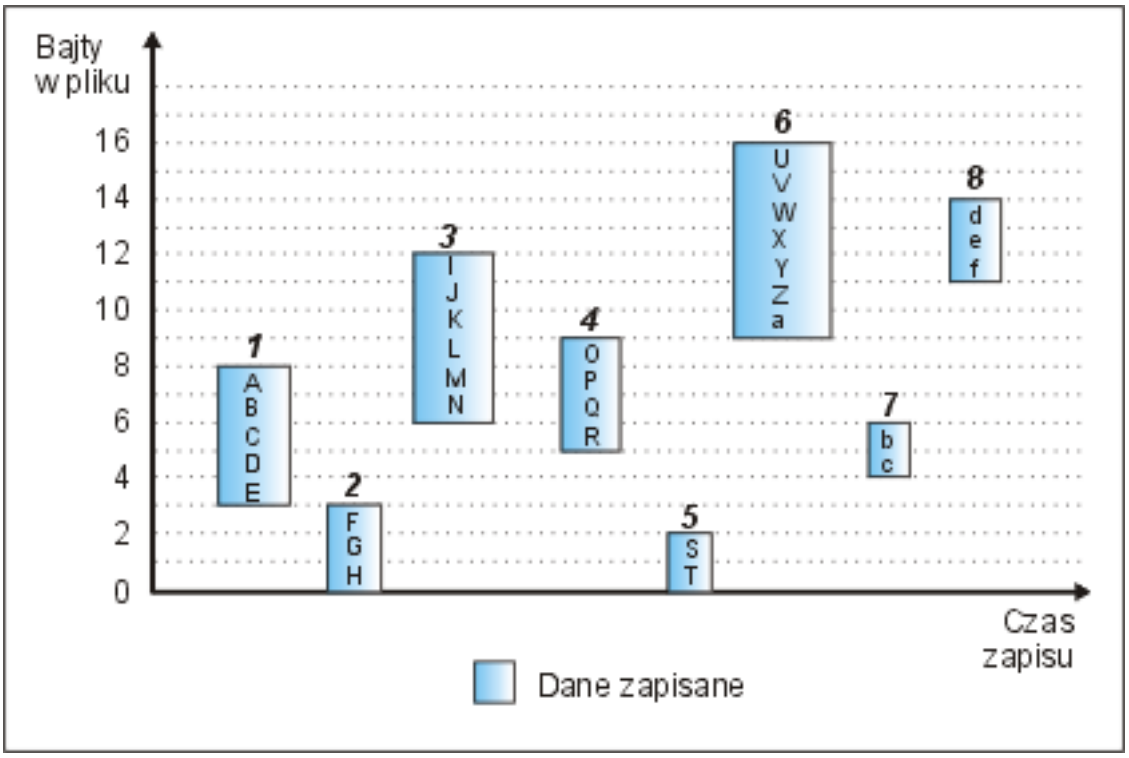
Drugi przypadek oznacza, że wcześniejsze operacje zapisane są w historii innego pliku. `Bufs-recovery` potrafi odnaleźć je odnaleźć i dokończyć odtwarzanie pliku.

W tym celu wykonuje on następujące czynności:

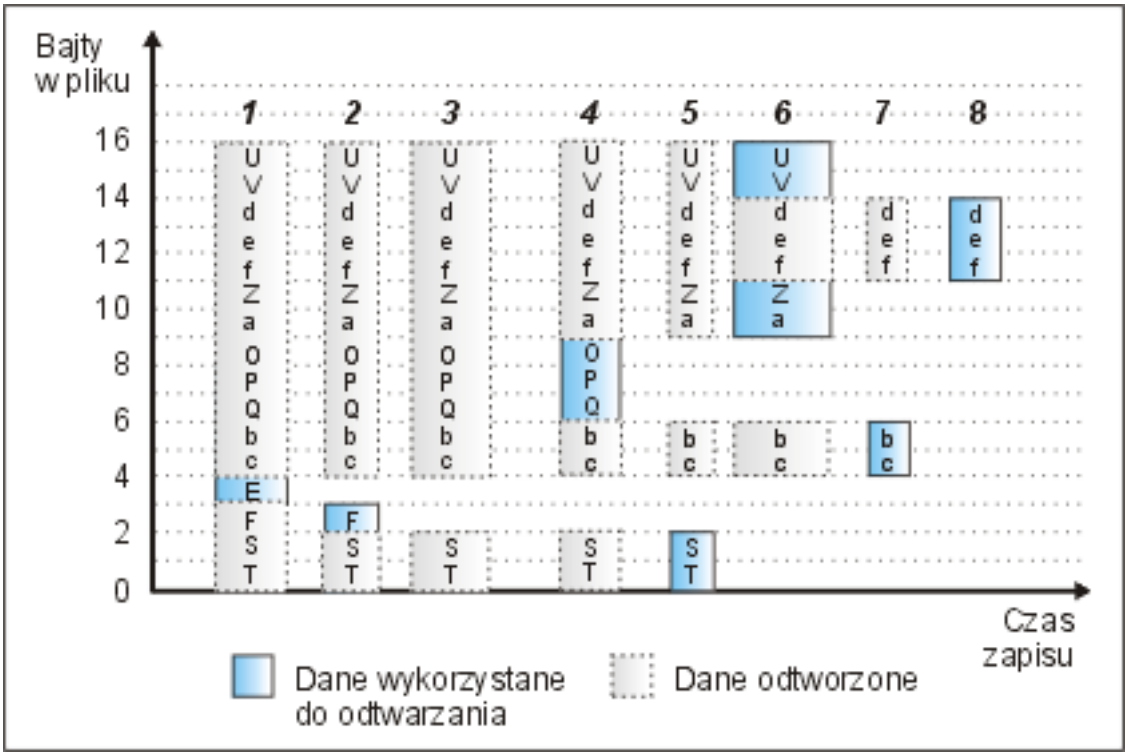
1. zapamiętuje numer sekwencyjny operacji zamiany pliku,

2. otwiera historię operacji zarejestrowanych dla poprzedniej nazwy pliku,

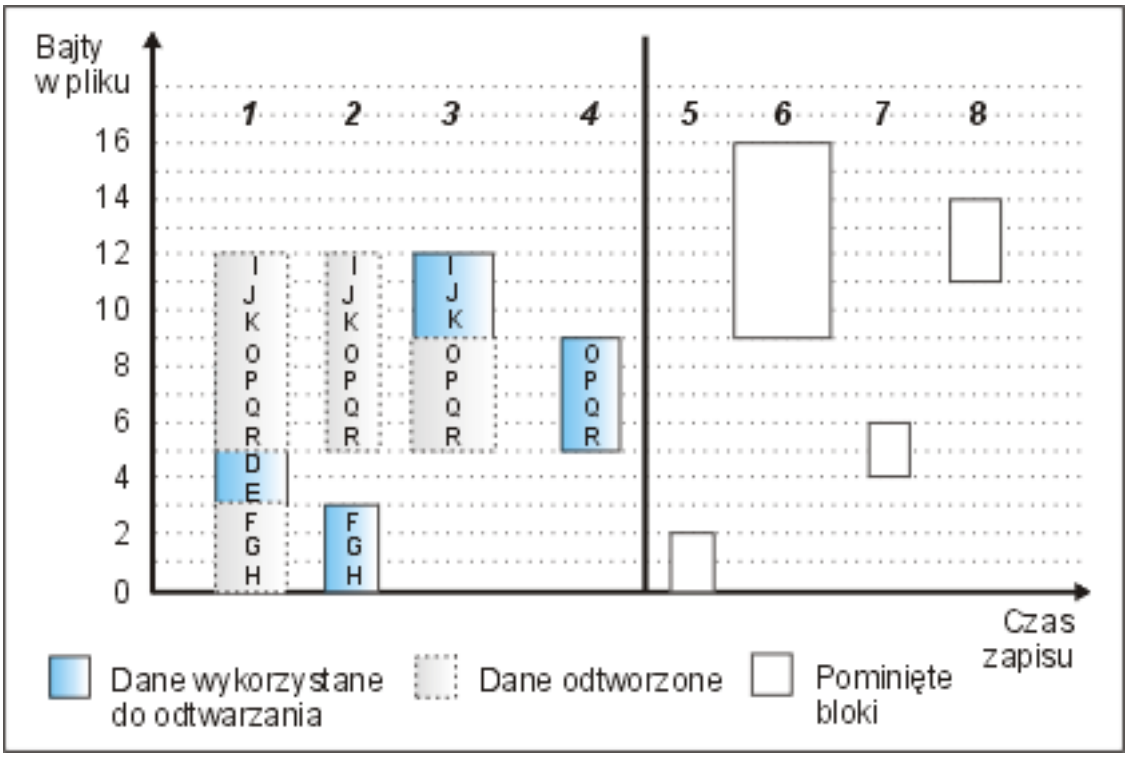
[4] *STL* jest standardową biblioteką C++ opartą na szablonach. Zbiór w *STL* jest zaimplementowany za pomocą drzew czerwono-czarnych.



Rysunek 4.3: Przykładowy scenariusz zapisu danych do pliku



Rysunek 4.4: Odbudowa pliku na podstawie dostępnej historii operacji opisanej w rysunku 4.3. Szarym kolorem zaznaczone są przedziały danych, które zostały już odtworzone



Rysunek 4.5: Odbudowa danych z punktu w czasie nr 4. Pozostałe bloki o numerze 5-8 zostały pominięte

3. pomija wszystkie operacje, których numer sekwencyjny jest równy bądź większy od zapamiętanego,
4. kontynuuje pracę odbudowy danych.

Dalsza praca programu przebiega w podobny sposób jak na początkowym zbiorze operacji. Dopuszczalna jest wielokrotna zmiana nazwy pliku, w każdym przypadku wykonywane są czynności opisane w punktach 1-4.

4.6. Kiedy kasować historię

Dyski twarde mają ograniczoną pojemność, podczas gdy rozmiar historii zwiększa się liniowo wraz z wykonywaniem nowych zapisów. Historii plików nie można zapamiętywać w nieskończoność, konieczne jest usuwanie najstarszych wpisów. Kwestia, jak długo historia powinna być przechowywana, jest rzeczą umowną i zależy od wymagań użytkownika oraz od pojemności zainstalowanych dysków.

Projekt *BUFS* jest eksperymentem, który ma za zadanie wykazać wydajność systemu ciągłej ochrony danych. Usuwanie danych z historii nie zostało zaimplementowane. Mogłaby się tym zajmować oddzielna aplikacja, która usuwałaby wpisy w plikach z historią starsze od określonej daty.

Rozdział 5

Zwiększanie niezawodności

Ciągła ochrona danych oznacza ich przetrwanie nawet po awarii maszyny, na której się znajdują. Ochrona nie może być stuprocentowa, gdy wszystkie kopie bezpieczeństwa znajdują się na jednym komputerze. Z pomocą przychodzą firmy sprzedające systemy pamięci masowej. Ich oferta jest bogata, ale ceny są często zaporowe dla wielu osób.

Sam system plików `bufs-mt` nie posiada mechanizmów replikacji danych na zdalnych maszynach. Potrafi jedynie zapisywać historię zmian do określonego katalogu. W celu zwiększenia jego niezawodności należy wykorzystać jeden z opisanych w tym rozdziale sposobów emulacji urządzenia blokowego. Dopiero w parze z zewnętrznym dyskiem system plików `bufs-mt` spełni wszystkie założenia ciągłej ochrony danych.

5.1. ATA over Ethernet (AoE)

AoE jest sieciowym protokołem umożliwiającym dostęp do zewnętrznych dysków twardych w lokalnej sieci komputerowej. Został zaprojektowany przez firmę Brantley Coile Company w celu budowy prostej i taniej pamięci masowej SAN[1].

AoE stanowi alternatywę dla skomplikowanych i drogich technologii, takich jak *iSCSI* czy *Fibre Channel*. Został on skonstruowany wyłącznie w oparciu o protokół sieciowy *Ethernet* i interfejs dyskowy *ATA*. Pominięto protokoły internetowe *IP*, *UDP* czy *TCP*, zrezygnowano z interfejsu *SCSI*. O prostocie protokołu może świadczyć specyfikacja techniczna, która została zamieszczona na ośmiu stronach (specyfikacja *iSCSI* zawiera 257 stron).

5.1.1. Protokół *AoE*

Protokół *AoE* służy do komunikacji między serwerem a zewnętrznym dyskiem. Polega on głównie na enkapsulacji poleceń dyskowych *ATA*. Żądania odczytu lub zapisu sektora są pakowane w komunikaty sieciowe (ramki) *Ethernet*, a następnie przesyłane do docelowego dysku. W identyczny sposób wracają odpowiedzi od zdalnego dysku. *AoE* nie ingeruje w treść przesyłanych żądań czy danych, zajmuje się jedynie ich transportem, gwarantując dostarczenie.

Choć *AoE* znajduje się w jednej warstwie sieciowej wraz z protokołem *IP*, nie posiada on tak rozbudowanych możliwości. Szczególnie brakuje trasowania pakietów w sieci, co zmusza do instalowania zdalnych dysków w obrębie lokalnej sieci komputerowej.

[1] Storage Area Network — rodzaj sieci służący do dostępu do zasobów pamięci masowej przez systemy komputerowe. Zobacz [30].

5.1.2. Sterownik *AoE*

Sterownik *AoE* udostępnia zewnętrzne dyski jako lokalne urządzenia blokowe. Z punktu widzenia systemu operacyjnego, działa on jako dodatkowy kontroler dysków znajdujący się w komputerze. Zdalne dyski można modyfikować w ten sam sposób jak zainstalowane w komputerze. Dopuszczalne jest partycjonowanie dysku, zakładanie systemu plików itp.

Sterownik *AoE* znajduje się jądrze Linuksa od wersji 2.6.11. Dostępne są też sterowniki do innych systemów operacyjnych — FreeBSD, Solaris, Mac i Windows.

5.1.3. Zewnętrzne dyski *AoE*

Jednym ze sposobów udostępnienia dysków twardych w sieci przy pomocy protokołu *AoE* jest zakupienie dodatkowego sprzętu. Urządzenia takie sprzedaje firma Coraid. Są to macierze dyskowe *EtherDrive*.

Kolejnym sposobem jest wykorzystanie programu *vblade*. Udostępnia on lokalne pliki i urządzenia blokowe jako dyski sieciowe.

5.2. Network Block Device (NBD)

Projekt *NBD* oferuje współdzielenie zasobów dyskowych pomiędzy różnymi komputerami. Został on zapoczątkowany w 1996 roku przez Pavla Macheka (autora systemu plików *Podfuk*, opisanego w rozdziale 3.1.2) i włączony do rozwojowej gałęzi Linuksa w wydaniu 2.1.101. Początkowo był to projekt eksperymentalny, pełen wad i niedociągnięć, lecz przez dziesięć lat wyeliminowano z niego wiele błędów, poprawiono znacznie wydajność. Pojawiła się obsługa większych dysków, procesorów 64-bitowych i wieloprocesorowości.

5.2.1. Architektura

W skład projektu wchodzą dwie aplikacje przeznaczone dla serwera i klienta oraz sterownika, który występuje tylko po stronie klienta. Aplikacja serwera, która udostępnia przestrzeń dyskową, może pracować w dowolnym systemie zgodnym z POSIX i wyposażonym w bibliotekę *GLib*[2]. Jest to niewielka aplikacja, której zadaniem jest akceptacja połączenia, a następnie w nieskończonej pętli wykonywanie poleceń zapisu i odczytu danych wysyłanych przez klienta.

Po stronie klienta komunikację z serwerem rozpoczyna aplikacja. Otwiera ona plik blokowy sterownika *NBD* (np. `/dev/nbd`), pod którym dostępny będzie zewnętrzny dysk. Po ustanowieniu połączenia wywołuje funkcję systemową *ioctl* na uprzednio otwartym pliku i zostaje zawieszona. Na tym kończy się rola aplikacji klienta, dalsze wykonywanie odbywa się w jądrze Linuksa.

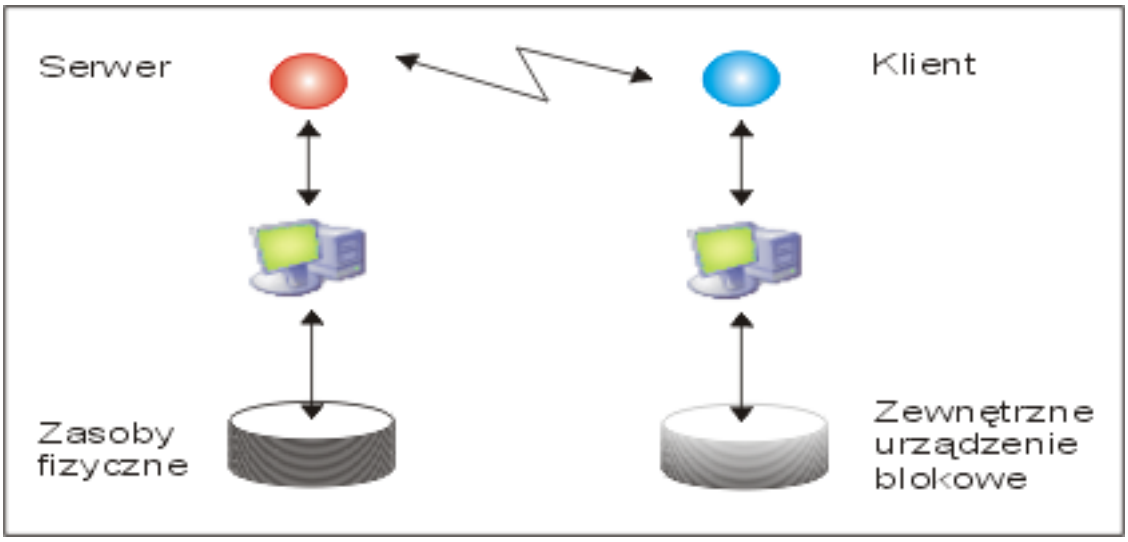
Wątek procesu aplikacji wykorzystywany jest przez sterownik *NBD* do komunikacji z serwerem. W pętli przyjmuje zlecenia od innych procesów, wysyła je do serwera, czeka na odpowiedź i przesyła odpowiedź. Schemat udostępnia zasobów obrazuje rysunek 5.1.

Sterownik *NBD* symuluje w systemie operacyjnym urządzenie blokowe, które odpowiada zewnętrznemu dyskowi. Pomimo komunikacji ze zdalnym komputerem, emulowane urządzenie traktowane jest na równi z lokalnymi, fizycznymi urządzeniami blokowymi. Można na nim założyć nowy system plików czy wykorzystać jako element macierzy RAID.

Jak dotąd jedyny stabilny sterownik *NBD* powstał dla jądra Linuksa[3].

[2]Istnieje także serwer *NBD* działający pod systemem Windows 2000 i XP. Zobacz [9].

[3]Sterownik dla systemów operacyjnych z rodziny BSD jest w fazie rozwoju. Zobacz [24].



Rysunek 5.1: Architektura systemu *NBD*. Źródło: [2]

5.2.2. Protokół komunikacji

NBD wykorzystuje protokół *TCP/IP* do komunikacji pomiędzy klientem i serwerem. Choć protokół *UDP* byłby lepszym wyborem, ze względu na mniejsze opóźnienia i narzut na komunikację, użycie *TCP/IP* pozwoliło znacznie zredukować kod źródłowy, jednocześnie zapewniając wysoką niezawodność połączenia.

5.3. DRBD

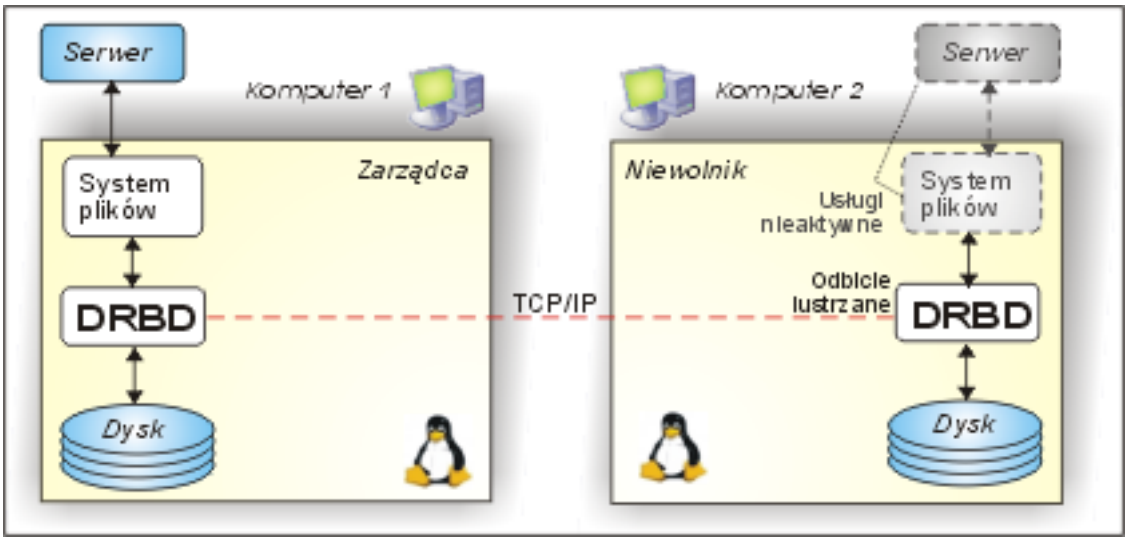
DRBD jest projektem oferującym wysoką niezawodność/dostępność[4] zgromadzonych informacji. Jest to realizowane dzięki replikacji danych w czasie rzeczywistym oraz braku pojedynczego punktu dostępu (ang. *single point of failure*). *DRBD* stanowi doskonały zamiennik dla drogich, sprzętowych macierzy dyskowych oferowanych przez różnych producentów[5]. Porównuje się go do macierzy RAID-1 działającej w sieci.

5.3.1. Architektura

System *DRBD* [7] składa się z dwóch węzłów[6] (komputerów) połączonych w sieci, które tworzą macierz dyskową. Węzły pracują w trybie zarządca-niewolnik (ang. *master-slave*), przy czym dowolny komputer może zostać zarządcą, ale nigdy oba naraz. Decyzja o wyborze zarządcy należy do administratora lub do zewnętrznego programu (np. *heartbeat*[7]). Dozwolona jest operacja zamiany ról pomiędzy węzłami.

Zlecenia odczytu lub zapisu bloku przyjmuje tylko węzeł dominujący. Jeśli jest to odczyt, to zlecenie jest przekazywane do lokalnego urządzenia. W przypadku zapisu, blok jest jednocześnie zapisywany na lokalnym dysku oraz wysyłany do drugiego węzła w celu replikacji

[4] High availability — cecha systemu określająca wysoki poziom niezawodności i dostępności usług. Zobacz [12] i [28].
[5] Przykładem takich macierzy mogą być produkty IBM Shark i EMC CLARiiON.
[6] Aktualnie projekt nie przewiduje większej liczby redundantnych węzłów.
[7] Heartbeat jest jednym z głównych komponentów projektu *Linux-HA*. Zobacz [12].



Rysunek 5.2: Architektura systemu *DRBD*. Źródło: [7]

danych. Dopiero po wykonaniu tych czynności użytkownik otrzymuje potwierdzenie zapisu. Schemat budowy systemu prezentuje rysunek 5.2.

Sterownik *DRBD*, podobnie jak poprzednie rozwiązania, udostępnia macierz dyskową w postaci urządzenia blokowego. Operacja kopiowania danych jest przezroczysta dla użytkownika. Najczęstszym sposobem wykorzystania urządzenia *DRBD* jest założenie na nim systemu plików. Proponowany jest system plików posiadający mechanizm księgowania (np. *reiserfs* czy *ext3*).

5.3.2. Synchronizacja danych

Synchronizacja danych jest wymagana w przypadku niedostępności jednego z węzłów, bądź problemów z siecią. W takiej sytuacji system *DRBD* jest zdegradowany do jednego węzła. Podczas pracy pojedynczego węzła, drugi nie otrzymuje nowych zapisów i następuje rozsynchronizowanie węzłów.

Na potrzeby synchronizacji *DRBD* utrzymuje mapę bitową, w której zawarte są informacje o niezreplikowanych blokach. Zastosowanie takiej struktury znacznie przyspiesza proces synchronizacji, gdyż nie wymusza kopiowania całego obrazu danych. Po ponownym podłączeniu drugiego węzła następuje synchronizacja danych, która polega na wysyłaniu wszystkich bloków z danymi, które zostały zapisane podczas jego nieobecności.

5.3.3. Protokół komunikacji

DRBD posiada trzy protokoły replikacji danych. Wszystkie trzy opierają się na *TCP/IP*, a dane są transmitowane w identyczny sposób. Różnica między nimi tkwi w poziomie odporności na awarię i charakteryzuje się momentem potwierdzenia zapisu bloku. Może to nastąpić gdy:

1. blok został zapisany na lokalnym dysku i trafił do wyjściowego bufora *TCP/IP*[8],
2. blok został zapisany na lokalnym dysku i docelowy węzeł otrzymał go,

[8]Nastąpił powrót z wywołania funkcji systemowej wysyłający blok danych.

3. blok został zapisany na lokalnym i zdalnym dysku.

Tylko trzeci przypadek może zagwarantować zwiększoną odporność na utratę danych i powinien być wykorzystywany. Pierwszy z nich ma ogromną wadę w przypadku awarii węzła zarządcy podczas dokonywania zapisu. Ostatnio potwierdzone bloki mogły nie dotrzeć do węzła niewolnika, czekając na wysłanie w buforze *TCP/IP*. W najlepszej sytuacji oznacza to utratę danych, w najgorszej zniszczenie transakcji i rozspójnienie systemu plików.

5.4. Ograniczenia

Wykorzystanie sieciowych urządzeń blokowych niesie za sobą następujące ograniczenia:

- tylko jeden użytkownik może wykorzystywać urządzenie blokowe w danej chwili (w przeciwieństwie do systemu plików *NFS*),
- wykorzystanie sieci obniża wydajność systemu, zwiększają się opóźnienia przy wykonywaniu operacji,
- istnieją przeciwskazania lub utrudnienia w montowaniu głównego systemu plików na takim urządzeniu.

5.5. NFS

Innym sposobem udostępnienia przestrzeni dyskowej jest wykorzystanie sieciowego systemu plików *NFS*. Choć posiada on znacznie szersze możliwości niż wymaga tego *bufs-mt* (np. obsługa wielu zdalnych użytkowników) to również może udostępniać mu przestrzeń dyskową.

5.6. Podsumowanie

Ciągła ochrona danych wymaga ich replikacji na innych maszynach, jednakże sam *bufs-mt* nie posiada odpowiednich do tego mechanizmów. Aby zapewnić w tym systemie zdalną replikację, można wykorzystać mechanizm udostępniania zdalnego dysk jako lokalne urządzenie blokowe. Połączenie *bufs-mt* z dowolnym z nich pozwoli mu zapisać historię zmian w zdalnym katalogu i dzięki temu spełnić wszystkie wymagania ciągłej ochrony danych.

Rozdział 6

Wydajność systemu plików `bufs-mt`

W tym rozdziale przedstawione zostaną wyniki testów wydajności przeprowadzonych na systemie plików `bufs-mt`. Testy przeprowadzono na domowym, prywatnym sprzęcie. Wykonano je w różnych konfiguracjach.

Zakres testów obejmował jedynie operacje wpływające na budowę historii, ze szczególnym uwzględnieniem zapisu danych. Operacje te badano na trzy sposoby: poprzez zapis dużego pliku, zapis czterech małych plików przez równoległe procesy oraz rozpakowanie archiwum ze źródłami jądra Linuksa.

Zbadano wydajność architektury *FUSE* oraz systemu `bufs-mt` z użyciem lokalnych, a następnie również zewnętrznych dysków. Dokładne wyniki przeprowadzonych testów są zawarte w załączniku B.

6.1. Środowisko testowe

6.1.1. Sprzęt

Do dyspozycji były dwa komputery — podstawowy oraz pomocniczy. Na pierwszym z nich był zakładany system plików `bufs-mt` oraz uruchamiane programy testujące wydajność. Drugi udostępniał jedynie dodatkową, zewnętrzną przestrzeń dyskową. Oto ich konfiguracja:

Parametr	Podstawowy komputer	Dodatkowy komputer
Procesor	Athlon 1.4 Ghz	Pentium 4 3.0 GHz
Pamięć	512 MB	Pentium 4 1024 MB
Dyski twarde	2x Western Digital ATA, 160 GB	Seagate Barracude ATA, 40 GB
Karty sieciowe	100 Mbit	100 Mbit
System operacyjny	Linux (Ubuntu 5.10)	Linux (Mandriva 2005)
System plików	ReiserFS ver. 3	ReiserFS ver. 3

Należy zwrócić uwagę na szybkość interfejsu sieciowego, który jest zainstalowany w komputerach. Prędkość 100 Mbit jest w tym wypadku niewystarczająca. Niestety, nie posiadam dostępu do kart sieciowych oferujących prędkość 1000 Mbit.

6.1.2. Oprogramowanie

Wersje oprogramowania, które zostały wykorzystane w testach:

- Fuse — 2.5.3,

- AoE — 6-36,
- vblade — 10,
- NBD — 2.7.3,
- DRBD — 0.7.21,
- NFS — v3.

6.1.3. Scenariusz testu

Scenariusz jednego testu obejmuje następujące etapy:

1. zapis jednego dużego pliku (100 MB) w blokach po 4 kB przez jeden proces,
2. zapis czterech mniejszych plików (po 30 MB każdy) w blokach po 4 KB przez cztery procesy równolegle,
3. rozpakowanie archiwum zawierającego źródła jądra Linuksa w wersji 2.0.

Test rozpoczynał się utworzeniem systemu plików *reiserfs* na dostępnych urządzeniach blokowych. Następnie każdy z etapów został przeprowadzony trzykrotnie w celu eliminacji wpływu zaburzeń na wyniki. Jak pokazały rezultaty zamieszczone w dodatku B, odchylenie standardowe otrzymanych wartości jest pomijalne. Ostatecznym wynikiem testu jest uśredniona prędkość zapisu dla każdego etapu. Pomiędzy poszczególnymi etapami wszystkie wygenerowane pliki były usuwane.

6.1.4. Konfiguracja testów

Na potrzeby testu zostały utworzone dwie partycje na dwóch dyskach znajdujących się w podstawowym komputerze oraz jedna partycja na pomocniczym komputerze. Oznaczmy je odpowiednio A, B i C. Pojemność każdej partycji wynosiła 4 GB.

Lokalny system plików `bufs-mt` był umieszczany wyłącznie na partycji A. Różnica pomiędzy wykonanymi testami polegała na umieszczeniu historycznych zmian w plikach odpowiednio na partycjach A, B i C lub ich kombinacjach. Wykorzystanie partycji C wiązało się z udostępnieniem jej na podstawowym komputerze z użyciem protokołów omówionych w rozdziale 5.

We wszystkich konfiguracjach testowany był, oprócz `bufs-mt`, przykładowy system plików *fusexmp*[1]. Jest on jedynie nakładką na rzeczywisty system plików, który znajduje się w komputerze. Za jego pomocą można zbadać narzut jaki niesie za sobą wykorzystanie *FUSE* i uruchamianie systemu plików w przestrzeni użytkownika.

Systemy plików *reiserfs* i *NFS* były montowane z dwiema flagami: *defaults* i *sync*. Druga z wymienionych flag powoduje synchroniczny zapis danych do pliku (zapisy nie są buforowane). Rozmiar bloku w systemie plików *reiserfs* był ustawiany na 1 kB.

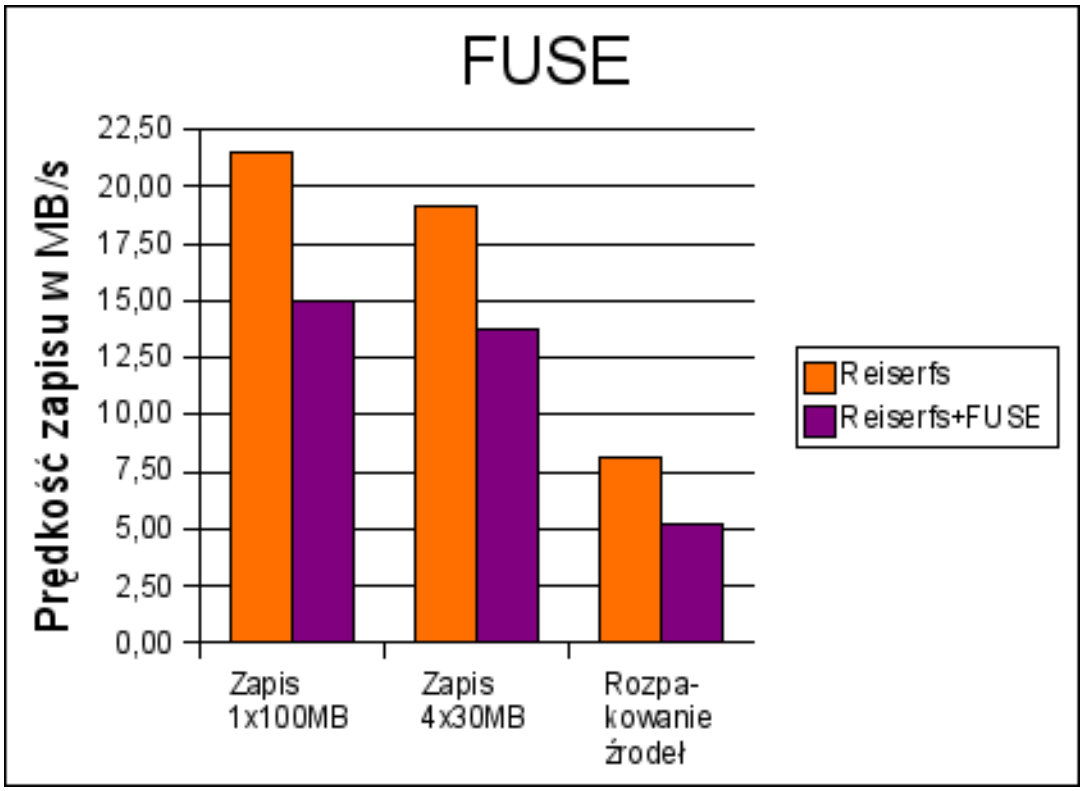
Podczas wykonywania testów w każdym dysku włączony był bufor zapisu bloków (*write-back cache*).

[1] System plików *fusexmp* dostarczany jest z projektem *FUSE* jako przykład systemu plików w przestrzeni użytkownika.

6.2. Wydajność *FUSE*

Pierwszym testem było zbadanie wydajności architektury *FUSE*. Ze względu na narzut związany z dodatkowym przełączaniem kontekstu pracy procesora i kopiowaniem bloków z operacjami zapisu, należałoby spodziewać się widocznego spadku wydajności w systemach plików wykorzystujących *FUSE*.

W teście został wykorzystany jeden dysk twardy, na którym utworzono system plików *reiserfs*. Po przeprowadzeniu testów opisanych w scenariuszu uruchomiony został przykładowy system plików *fusexmp* i testy powtórzono. Wyniki testów można obejrzeć na rysunku 6.1.



Rysunek 6.1: Porównanie wydajność systemu plików *reiserfs* oraz nakładki *FUSE*.

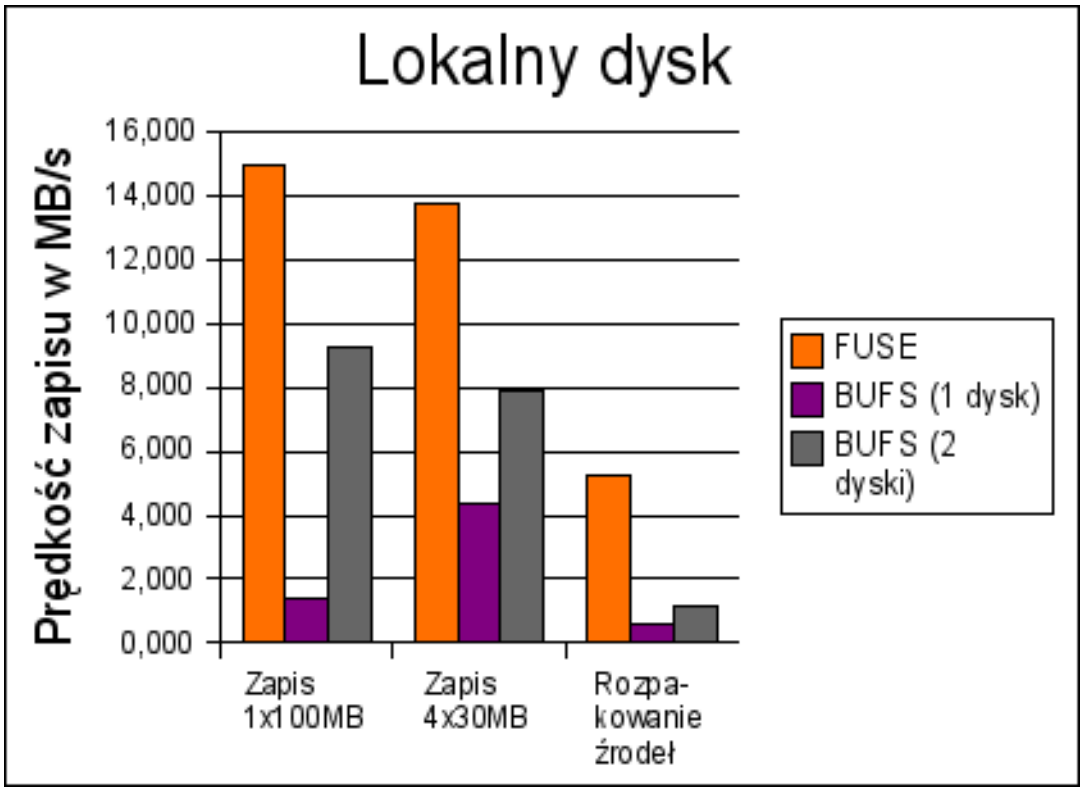
Przypuszczenia okazały się słuszne. Jak widać na wykresie, spadek wydajności wynosi około 30 procent dla zapisu plików oraz 35 procent przy rozpakowywaniu archiwum. Jest to zauważalna różnica, która wpływa niekorzystnie także na system plików `bufs-mt`. Z tego względu w dalszej części rozdziału `bufs-mt` będzie porównywany nie do podstawowego systemu plików *reiserfs*, ale do nałożonego na niego *FUSE*.

6.3. Wydajność `bufs-mt`

6.3.1. Lokalne dyski

Celem kolejnego testu było zaprezentowanie wydajności `bufs-mt` na lokalnych dyskach. Testy różniły się lokalizacją historii zmian w plikach. W pierwszym przypadku źródłowy system plików i historia zmian dzieliły między siebie jeden dysk. W drugim — historia zmian została umieszczona na osobnym dysku.

Umieszczenie dwóch katalogów `bufs-mt` na jednym dysku powinno skutkować obniżeniem wydajności przynajmniej o połowę. Jedna operacja zapisu do systemu plików `bufs-mt` zamienia się na dwie, kierowane do jednego dysku. Dodatkowo są to zapisy do różnych plików, które z dużym prawdopodobieństwem zostaną umieszczone w oddalonych sektorach. Wymusza to częsty ruch głowicy dysku i spowalnia zapis danych. Rezultaty z przeprowadzonych testów przedstawia rysunek 6.2.



Rysunek 6.2: Wyniki testów prędkości zapisu z wykorzystaniem tylko wewnętrznych dysków

Zgodnie z przewidywaniami podany wykres ukazuje wyraźną przewagę rozwiązania wykorzystującego dwa dyski. W porównaniu z jednym dyskiem uzyskany rezultat był od dwóch (rozpakowywanie źródeł i zapis do czterech plików) do sześć i pół (zapis do pojedynczego pliku) razy lepszy. Różnica jest ogromna, dlatego konfiguracja z jednym dyskiem nie ma racji bytu, co najwyżej nadaje się do testowania poprawności systemu.

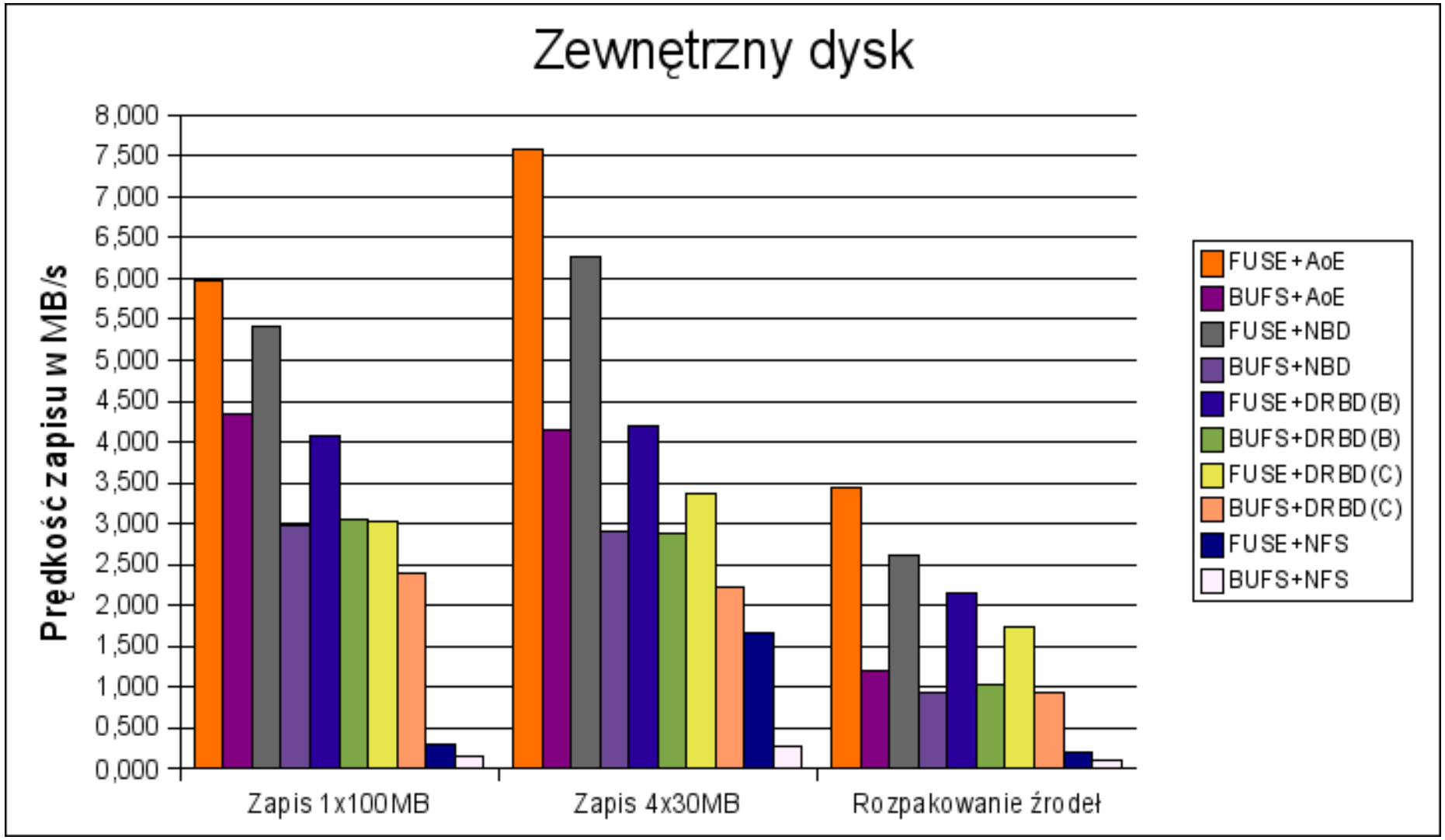
Wydajność `bufs-mt` w porównaniu do nakładki *FUSE* jest gorsza o ok. 40% w przypadku zapisu do plików. Choć nie jest to rewelacyjny wynik, to może on być zaakceptowany przez wielu użytkowników ceniących sobie bezpieczeństwo ponad wydajność. Niestety przy rozpakowywaniu źródeł jądra prędkość zapisu spada aż o 78%. Głównym powodem tego zjawiska jest koszt założenia nowego pliku w systemie *bufs-mt*.

6.3.2. Zewnętrzne dyski

Ostatnia grupa testów polegała na umieszczeniu historii zmian na zdalnym dysku. Jest to konfiguracja zalecana ze względu na unikanie sytuacji, w której jedyny węzeł ulegnie awarii. Udostępnianiem przestrzeni dyskowej zajmowały się kolejno programy przedstawione w rozdziale 5: *Aoe*, *NBD*, *DRBD* oraz *NFS*. Program *DRBD* został przetestowany w dwóch

konfiguracjach — przy użyciu protokołu B i C do komunikacji pomiędzy węzłami. Protokoły te zostały opisane w rozdziale 5.3.3.

Wyniki przeprowadzonych testów zostały zaprezentowane na rysunku 6.3.



Rysunek 6.3: Wyniki testów prędkości zapisu z wykorzystaniem zewnętrznego dysku.

AoE W przeprowadzonych testach prym wśród programów udostępniających zdalny dysk wiedzie *AoE*. Jest to spowodowane użyciem protokołu *ethernet* zamiast *TCP/IP*. Mniejszy narzut na komunikację między węzłami oznacza mniejsze opóźnienia w zapisie bloków, a to korzystnie przekłada się na przepustowość urządzenia blokowego i w efekcie także systemu plików `bufs-mt`.

NBD Kolejne miejsce zajął program *NBD*. Użycie protokołu *TCP/IP* wpłynęło niekorzystnie na prędkość zapisu. *NBD* wypadł gorzej od *AoE* o około 30%.

DRBD Wyniki konfiguracji *DRBD* okazały się być gorsze niż wyniki *NBD*. Jest to jak najbardziej zrozumiałe i akceptowalne. *DRBD* oferuje RAID1 poprzez sieć, co wiąże się z dodatkowymi kosztami zapisywania danych w sposób persystentny. Różnica pomiędzy wybranymi protokołami komunikacji wynosi około 20% na korzyść mniej bezpiecznego połączenia typu B (por. rozdział 5.3.3).

NFS Na uwagę zasługuje *NFS*, którego wydajność jest drastycznie niska. Porównując zapis jednego i czterech plików można dojść do wniosku, że operacja synchronizacji danych w tym systemie jest bardzo kosztowna. Nie bez przyczyny określa się go terminem "Not For Speed".

6.4. Przyczyny niskiej wydajności `bufs-mt`

Wydajność `bufs-mt` jest ewidentnie gorsza od wydajności tradycyjnych systemów plików. Prędkość zapisu na poziomie 3-4,5 MB/s jest na tyle niska, że obecnej postaci system ten nie posiada komercyjnych zastosowań.

Przyczyn niskiej wydajności można się doszukiwać w całej architekturze systemu. Największy wpływ na obniżenie przepustowości ma sposób zapamiętywania historii zmian w plikach. Po pierwsze, rozmiar zapisu danych do pliku z historią operacji jest powiększony o wyrównanie podane w konfiguracji. Po drugie operacja jest uznawana za wykonaną z chwilą zapisania danych w źródłowym systemie plików i w historii. Po trzecie zapis jest wykonywany w trybie synchronicznym.

Wymienione wyżej cechy sprawiają, że operacja zapisu w systemie plików `bufs-mt` trwa dłużej niż pojedynczy zapis bloku na dysku. Ze względu na znacznie większe opóźnienia w operacji zapisu spada wydajność całego systemu.

Osobnym problemem jest proces zakładania nowych plików. Na potrzeby zapamiętywania historii zmian tworzone są trzy pliki oraz jeden katalog. Powoduje to dodatkowy spadek wydajności w przypadku tworzenia dużej liczby nowych plików.

6.5. Możliwe usprawnienia

Rozwiązanie przedstawione w tym rozdziale nie jest doskonałe, ale stanowi jedynie eksperyment, projekt badawczy. Budując komercyjne narzędzie warto zastanowić się nad poprawieniem wydajności. W dalszej części przedstawiam propozycje usprawnienia systemu pliku `bufs-mt`.

6.5.1. Dodatkowy dysk dla tymczasowego zapisu historii operacji

Wąskim gardłem jest zapamiętywanie historii operacji zlecanych przez użytkownika. Dobrym pomysłem byłoby wykorzystanie dodatkowego dysku twardego jako bufora do tymczasowego

zapisu historii. Zapis do tego dysku byłby bezpośredni — z pominięciem systemu plików — do urządzenia blokowego. I na tym kończyłaby się rola programu `bufs-mt`.

Odbieraniem historii operacji z bufora i rozdzielaniem jej na poszczególne pliki mogłaby zajmować się osobna aplikacja. Szybkość dystrybucji historii powinna być regulowana w zależności od aktualnego obciążenia dysku-bufora w ten sposób, aby nie wpływać na wydajność `bufs-mt`. Dobrym rozwiązaniem byłoby przeprowadzanie tej czynności w nocy, gdy nikt nie korzysta w systemu plików, pod warunkiem, że bufor będzie posiadał wystarczająco dużo miejsca na zgromadzenie historii wszystkich operacji z jednego dnia.

6.5.2. Pominięcie bufora systemowego podczas zapisywania historii

Standardowo system operacyjny Linux oferuje buforowanie operacji wejścia-wyjścia. W wielu zastosowaniach wpływa to pozytywnie na wydajność systemu, szczególnie przy wielokrotnym odczycie lub zapisie tego samego bloku.

W przypadku programu `bufs-mt` mechanizm buforowania powoduje marnowanie zasobów — pamięci i czasu procesora. Wykonywanie kopii zapasowej w programie `bufs-mt` polega jedynie na ciągłym dopisywaniu danych do pliku z historią, co wyklucza ponowne wykorzystanie raz już zapisanego bloku.

Ulepszeniem może się okazać otwieranie plików z flagą `O_DIRECT`, która minimalizuje efekt związany z buforowaniem operacji.

6.5.3. Opracowanie nowego modułu do jądra

Wykorzystanie *FUSE* znacząco ułatwia tworzenie nowego systemu plików. Niestety niesie za sobą również duże pogorszenie wydajności. Rozwiązaniem dla komercyjnego produktu jest napisanie własnego modułu do jądra, który będzie realizował te same cele, co program `bufs-mt`.

Rozdział 7

Podsumowanie

Celem pracy było skonstruowanie niskobudżetowego systemu, który spełniałby wszystkie założenia ciągłej ochrony danych. Postawiony cel został osiągnięty. W ramach projektu *BUFS* powstał system plików `bufs-mt`, realizujący ochronę danych na poziomie plików oraz aplikacja odzyskująca utracone pliki *bufs-recovery*.

Dzięki wykorzystaniu możliwości projektu *FUSE*, system plików działa w całości w przestrzeni użytkownika. Poruszanie się jedynie w przestrzeni użytkownika i brak konieczności modyfikacji jądra w znaczącym stopniu ułatwiło opracowanie oprogramowania oraz przyspieszyło usuwanie błędów.

Ponadto w ramach pracy wykorzystane zostały sterowniki udostępniające zewnętrzne urządzenia blokowe na lokalnym komputerze. Przyczyniło się to do podniesienia bezpieczeństwa danych przez podniesienie liczby węzłów (pojedynczy węzeł byłby bardzo podatny na awarię).

Dodatkowo w pracy przedstawiłem różne aspekty ochrony danych i przeanalizowałem koszty finansowe wynikające z ich utraty. Omówiłem szczegółowo pojęcie ciągłej ochrony danych i porównałem ją z innymi popularnymi sposobami archiwizacji.

W pracy nie zabrakło testów wydajnościowych. Wykazały one spadek wydajności rzędu 40%-70% w zależności od konfiguracji i wykonywanych operacji. Wyniki nie są rewelacyjne, ale mogą być wystarczające dla wielu użytkowników, którzy nie mają krytycznych wymagań wobec wydajności systemu plików. Na szczególną uwagę zasługują wyniki testów z wykorzystaniem systemu *NFS*. Ujawniły one dość duże problemy z prędkością wykonywania operacji synchronicznych w tym systemie.

7.1. Przyszłość systemów ciągłej ochrony danych

Przyszłość systemów ciągłej ochrony danych wygląda optymistycznie. Wydaje się, że w ciągu kilku najbliższych lat wiele przedsiębiorstw, które na co dzień operują na ogromnych ilościach danych, zechce wdrożyć u siebie tego typu rozwiązania, by mieć pewność, że ich dane są rzeczywiście w stu procentach chronione. Wdrożenie systemów ciągłej ochrony danych pozwoli firmom zwiększyć wydajność pracy, dając alternatywę dla czasochłonnego i pracochłonnego regularnego wykonywania kopii zapasowych. Ponadto przedsiębiorstwa będą mogły natychmiast po awarii odzyskać dane w postaci, w jakiej istniały tuż przed jej wystąpieniem. Będą miały możliwość bezzwłocznie przywrócić do życia krytyczne aplikacje biznesowe. Nie sposób pominąć faktu, że wdrożenie systemów ciągłej ochrony danych przyniesie również znaczącą redukcję kosztów zapewnienia pełnego bezpieczeństwa istotnym informacjom.

7.2. Przyszłość projektu *BUFS*

BUFS jest projektem badawczym. Do jego pełnego, komercyjnego zastosowania jest jeszcze daleka droga. Głównie ze względu na swoją stosunkowo słabą wydajność oraz brak zintegrowanych narzędzi ułatwiających jego obsługę i konfigurację nie stanowi produktu, który mógłby być szeroko używany przez firmy i instytucje.

Pierwszym poważnym krokiem ku poprawie jego wydajności powinna być rezygnacja z *FUSE* jako podstawy systemu plików `bufs-mt`. Testy wykazały, że skorzystanie z *FUSE* powoduje spadek prędkości wykonywania operacji o około 30%. Dla polepszenia wydajności projektu *BUFS* warto rozważyć napisanie własnego modułu do jądra.

Mimo posiadanych wad projekt *BUFS* w pewnym stopniu wypełnia lukę na rynku systemów ciągłej ochrony danych. Obecnie istnieje bardzo mało tego typu programów, a ponadto prawie wszystkie dostępne komercyjnie rozwiązania są zintegrowane z określonymi aplikacjami, z kolei brak jest systemów chroniących dane na poziomie plików.

Dodatek A

Konfiguracja systemu *BUFS*

A.1. Opis konfiguracji

Systemu plików *BUFS* posiada następujące opcje konfiguracji:

LOCAL_DIR Katalog, który będzie podlegać ciągłej archiwizacji plików.

BACKUP_DIR Katalog (przyrostowej) kopii zapasowej.

MAX_FILE_SIZE_MB Maksymalny rozmiar plików wykorzystywanych w kopiach zapasowych (w megabajtach).

BACKUP_THREADS_NUM Liczba wątków wykonywujących operacje zapisu w kopii zapasowej.

BACKUP_BLOCK_ALIGN Wyrównanie bloku zapisu w kopii zapasowej do podanego rozmiaru.

A.2. Przykładowy plik konfiguracyjny

```
# Katalog do utrzymywania bieżącego systemu plików.
LOCAL_DIR = /bufs/local

# Katalog (przyrostowej) kopii zapasowej.
BACKUP_DIR = /bufs/backup

# Maksymalny rozmiar plików wykorzystywanych w kopiach zapasowych.
MAX_FILE_SIZE_MB = 64

# Liczba wątków wykonywujących kopię zapasową.
BACKUP_THREADS_NUM = 4

# Rozmiar bloku zapisywany w kopii zapasowej - wyrównanie (w bajtach)
BACKUP_BLOCK_ALIGN = 4096
```

A.3. Opcje programu odzyskującego pliki `bufs-recovery`

Program `bufs-recovery` posiada następujące opcje uruchamiania:

-b, –backup Katalog z kopią zapasową (używany przez system plików *BUFS*).

-f, –file Nazwa pliku wraz z absolutną ścieżką używaną w *BUFS* .

-t, –time Odzyskanie pliku z wyznaczonego punktu w czasie (format czasu: %Y-%m-%d %H:%M:%S).

-u, –unique Odzyskanie pliku z uwzględnieniem zapisów tylko do podanego numeru unikatowego (format: numer_montowania,numer_operacji).

-o, –out Nazwa pliku wynikowego.

-h, –help Krótka instrukcja w języku angielskim.

Jednoczesne pominięcie opcji `-t` oraz `-u` spowoduje odzyskanie najnowszej wersji podanego pliku.

Dodatek B

Wyniki testów

W poniższej tabeli zostały przedstawione dokładne wyniki przeprowadzonych testów wydajnościowych. Opis poszczególnych konfiguracji i typów testów znajduje się w rozdziale 6.

Konfiguracja	**Test**	**Wynik 1 (s)**	**Wynik 2 (s)**	**Wynik 3 (s)**	**Średnia (s)**	**Przepustowość MB/s**
Reiserfs	1x100MB	4,55	4,74	4,68	4,65	21,47
	4x30MB	6,39	6,30	6,11	6,26	19,15
	Kernel	2,57	2,75	2,48	2,60	8,13
FUSE	1x100MB	6,69	6,70	6,68	6,69	14,95
	4x30MB	8,80	8,63	8,70	8,71	13,78
	Kernel	3,90	4,02	4,18	4,03	5,24
BUFS	1x100MB	10,71	10,89	10,87	10,82	9,24
(2 dyski)	4x30MB	15,13	15,06	15,34	15,17	7,91
	Kernel	19,29	18,66	18,75	18,90	1,12
BUFS	1x100MB	69,43	70,13	71,52	70,36	1,42
(1 Dysk)	4x30MB	27,91	26,70	28,42	27,67	4,34
	Kernel	35,81	36,07	36,25	36,04	0,59
BUFS+	1x100MB	40,96	43,61	40,84	41,80	2,39
DRBD(C)	4x30MB	53,64	54,82	53,69	54,05	2,22
	Kernel	22,64	22,69	22,96	22,76	0,93
FUSE+	1x100MB	32,58	32,24	34,46	33,09	3,02
DRBD(C)	4x30MB	35,27	35,62	36,24	35,71	3,36
	Kernel	11,66	12,61	12,13	12,13	1,74
BUFS+	1x100MB	32,68	32,81	32,87	32,78	3,05
DRBD(B)	4x30MB	41,58	41,50	41,49	41,52	2,89
	Kernel	20,19	20,22	20,76	20,39	1,04
FUSE+	1x100MB	24,46	24,56	24,60	24,54	4,07
DRBD(B)	4x30MB	28,65	28,49	28,69	28,61	4,19
	Kernel	9,87	10,28	9,34	9,83	2,15
BUFS+	1x100MB	33,54	33,45	34,05	33,68	2,97
NBD	4x30MB	41,48	41,41	40,79	41,22	2,91
	Kernel	22,55	22,39	22,87	22,60	0,93
FUSE+	1x100MB	18,60	18,40	18,47	18,49	5,41
NBD	4x30MB	19,58	18,88	18,88	19,11	6,28
	Kernel	7,81	8,28	8,25	8,11	2,60

Konfiguracja	**Test**	**Wynik 1 (s)**	**Wynik 2 (s)**	**Wynik 3 (s)**	**Średnia (s)**	**Przepustowość MB/s**
BUFS+ AoE	1x100MB	23,00	23,03	22,90	22,97	4,35
	4x30MB	28,88	29,11	28,98	28,99	4,14
	Kernel	17,85	17,17	17,80	17,60	1,20
FUSE+ AoE	1x100MB	16,53	17,35	16,76	16,88	5,92
	4x30MB	15,95	15,04	15,85	15,61	7,69
	Kernel	5,72	6,25	6,15	6,04	3,50
BUFS+ NFS	1x100MB	631,79	628,15	635,66	631,86	0,16
	4x30MB	431,29	432,57	434,41	432,75	0,28
	Kernel	178,34	180,23	178,98	179,18	0,12
FUSE+ NFS	1x100MB	341,29	341,76	342,12	341,72	0,29
	4x30MB	72,83	72,01	72,16	72,33	1,66
	Kernel	99,50	105,39	105,56	103,48	0,20

Dodatek C

Zawartość płyty CD

Na załączonej płycie CD znajdują się:

- **Malinowski-Bufs.pdf** — niniejsza praca magisterska w formacie PDF,
- **Malinowski-Bufs-source/** — niniejsza praca magisterska w formacie LaTeX,
- **Bufs-mt/** — system plików *BUFS*,
- **Bufs-recovery/** — program odzyskujący pliki,
- **ExternalProjects/** — zewnętrzne projekty i biblioteki,
- **TestScripts/** — skrypty i programy wykorzystane do przeprowadzenia testów wydajności,
- **Papers/** — materiały opisane w bibliografii.

Bibliografia

[1] Albert Alexandrov, Maximilian Ibel, Klaus E. Schauser i Chris Scheiman, *Extending the Operating System at the User Level: the Ufo Global File System*, Department of Computer Science University of California, Santa Barbara, 1997

[2] P.T. Ares, *The Network Block Device*,
Linux Journal, http://www2.linuxjournal.com/article/3778, 2000

[3] Atempo, *Atempo LiveBackup*,
http://www.atempo.com/collateral/LiveBackup_Datasheet.pdf

[4] Brian Babineau, *Tivoli Welcomes CDP to Data Protection Family*,
ftp://ftp.software.ibm.com/software/tivoli/executive-brief/eb-enterprcdp.pdf

[5] Ann L. Chervenak, Vivekanand Vellanki i Zachary Kurmas *Protecting File Systems: A Survey of Backup Techniques*, http://www.isi.edu/ãnnc/papers/mss98final.ps

[6] Digital Iron Mountain, LiveVault Server Backup and Recovery for Small and Medium-sized Businesses and Remote Offices,
http://www.livevault.com/solutions/smb/datasheet.aspx

[7] *Distributed Replicated Block Device*,
http://www.drbd.org/

[8] Carl Greiner, *Continuous data protection: addressing timely data recovery and much more*, Ovum 2005

[9] Folkert van Heusden, *NBD-Server*, http://www.vanheusden.com/windows/nbdsrvr/

[10] IBM, *IBM Tivoli Continuous Data Protection for Files*,
http://www-306.ibm.com/software/tivoli/products/continuous-data-protection/

[11] Charles M. Kozierok, *The Risks To Your Data*,
http://www.pcguide.com/care/bu/risks.htm

[12] Linux HA-ckers, *The High-Availability Linux Project*, http://www.linux-ha.org/

[13] Pavel Machek, *POrtable Dodgy Filesystems in Userland*,
http://atrey.karlin.mff.cuni.cz/ pavel/podfuk/podfuk.html

[14] Florin Malita, *LUFS Intro*, http://lufs.sourceforge.net/lufs/intro.html

[15] Mendocino Software, *Optimized Recovery and Administration for Enterprise Applications*, http://www.mendocinosoft.com/assets/R1%20CDT%20DS.pdf

[16] Mimosa Systems, *The Mimosa Vision*, http://www.mimosasystems.com/html/about_overview.htm

[17] Brian J. Olson, *CDP Buyers Guide, First edition July 2005*, The Storage Networking Industry Association, SNIA DMF CDP Special Interest Group, 2005

[18] Ontrack, *Czekając na katastrofę*, http://media.ontrack.pl/notatka_64106.html

[19] Zachary N. J. Peterson, Randal Burns. *Ext3cow: A Time-Shifting File System for Regulatory Compliance*, Johns Hopkins University, 2005

[20] Revivio, *The Continuous Protection System — CPS 1200*, http://www.revivio.com/index.asp?p=prod_CPS_opt

[21] Michael Rowan, *Disk-Based Restoration Technologies*, http://www.snia.org/education/tutorials/2006/spring/data-management/Disk-based_Restoration_Technologies.pdf

[22] David M. Smith, *The Cost of Lost Data*, http://gbr.pepperdine.edu/033/dataloss.html

[23] SonicWALL, *Backup and Recovery Solutions*, http://www.sonicwall.com/products/cdp.html

[24] SourceForge, *Network Virtul Device*, http://sourceforge.net/projects/nvd

[25] Symantec, *Symantec Backup Exec for Windows Servers*, http://www.symantec.com/enterprise/products/overview.jsp?pcid=1018&pvid=57_1

[26] Miklos Szeredi, *AVFS - A Virtual File System*, http://www.inf.bme.hu/ mszeredi/avfs/

[27] Miklos Szeredi, *Filesystem in Userspace*, http://fuse.sourceforge.net/

[28] Wikipedia, Definicja terminu *High availability*, http://en.wikipedia.org/wiki/High_availability

[29] Wikipedia, Definicja terminu *Mean time between failure*, http://en.wikipedia.org/wiki/MTBF

[30] Wikipedia, Definicja terminu *Storage Area Network*, http://en.wikipedia.org/wiki/Storage_area_network

[31] Wikipedia, Definicja terminu *Virtual file system*, http://en.wikipedia.org/wiki/Virtual_file_system

[32] XOsoft, *Enterprise Rewinder Product Suite for Continuous Data Protection (CDP)*, http://www.xosoft.com/products/f_Rewinder.shtml