I. MECHANIZMY SYNCHRONIZACJI

Celem synchronizacji jest ograniczenie swobody przeplotu tak, żeby dopuszczalne stały się tylko przeploty zgodne z intencją programisty.

Synchronizacja na najniższym poziomie polega na wykonaniu określonych (często specjalnych) instrukcji, które powodują blokowanie wykonywania odpowiednich instrukcji synchronizujących w innych strumieniach (wątkach).

Synchronizacja na wyższym poziomie polega na przygotowaniu specjalnych konstrukcji programotwórczych, które gwarantują zachowanie określonych własności w przeplocie instrukcji, zgodnie z intencją programisty.

PODSTAWOWE PROBLEMY SYNCHRONIZACJI

Blokada i zagłodzenie

Zbiór procesów znajduje się w stanie blokady, jeśli każdy z tych procesów jest wstrzymany w oczekiwaniu na zdarzenie, które może być spowodowane tylko przez jakiś inny proces z tego zbioru.

Blokada jest przykładem niespełnionej własności bezpieczeństwa. Możliwość wystąpienia blokady nie oznacza, że wystąpi ona w każdym wykonaniu programu współbieżnego. Dlatego testowanie nie jest dobrą metodą jej wykrywania. Zagłodzenie występuje wtedy, gdy proces nie jest wznowiony, mimo że zdarzenie, na które czeka występuje dowolną liczbę razy. Zagłodzenie jest przykładem niespełnionej własności żywotności.

Problem producenta i konsumenta

Polega na zsynchronizowaniu dwóch procesów: producenta, który cyklicznie produkuje porcje danych, a następnie przekazuje je do konsumpcji, i konsumenta, który cyklicznie te dane pobiera i konsumuje.

Producent musi być wstrzymany, jeśli nie może przekazać wyprodukowanych porcji, a konsument, jeśli nie może ich pobrać. Porcje powinny być konsumowane w kolejności ich wyprodukowania (to nie zawsze jest wymagane).

Rozwiązania:

Synchronizacja przez N elementowy bufor (N od 1 do nieskończoności).

Efektywność rozwiązania zależy od wielkości bufora i zależności czasowych realizacji produkcji i konsumpcji.

Problem ten jest abstrakcją wielu rzeczywistych problemów pojawiających się w protokołach komunikacyjnych i systemach czasu rzeczywistego.

Problem czytelników i pisarzy

Polega na zsynchronizowaniu dwóch grup cyklicznych procesów konkurujących o dostęp do wspólnej czytelni.

Proces czytelnik odczytuje informacje zgromadzone w czytelni i może to robić razem z innymi czytelnikami.

Proces pisarz cyklicznie zapisuje informacje i może to robić tylko sam przebywając w czytelni.

Czytanie i pisanie trwa skończenie długo. Problem ten jest abstrakcją problemu synchronizacji procesów korzystających ze wspólnej bazy danych.

Rozwiązanie z możliwością zagłodzenia pisarzy:

Czytelnik może wejść do czytelni, gdy jest pusta lub są w niej inni czytelnicy.

Gdy w czytelni jest pisarz czytelnik jest wstrzymany.

Gdy czytelnia nie jest pusta, to pisarz jest wstrzymany.

Rozwiązanie z możliwością zagłodzenia czytelników:

Jeśli pisarz czeka na wejście do czytelni, to nie może już do niej wejść żaden nowy czytelnik

Rozwiązanie bez zagłodzenia: do czytelni wpuszczani są na przemian czytelnicy i pisarze. Z tym, że wraz z każdym czytelnikiem wchodzą wszyscy inni oczekujący czytelnicy.

Problem pięciu filozofów

Polega na zsynchronizowaniu działań pięciu filozofów, którzy siedzą przy okrągłym stole i cyklicznie myślą i jedzą, korzystając z talerza i dwóch widelców. Przed każdym filozofem stoi talerz, a pomiędzy każdymi dwoma talerzami leży jeden widelec. Zakłada się, że jeśli filozof podniósł oba widelce, to w skończonym czasie je odłoży.

Rozwiązanie gwarantuje, że każdy filozof, który poczuje się głodny, będzie mógł się w końcu najeść.

Ponadto wszyscy filozofowie powinni być traktowani jednakowo. Jest to wzorcowy przykład obrazujący zjawiska zagłodzenia i blokady.

Rozwiązanie z możliwością blokady: głodny filozof czeka, aż będzie wolny lewy widelec, podnosi go i czeka na prawy i także go podnosi. Po zjedzeniu odkłada oba widelce.

Rozwiązanie z możliwością zagłodzenia: filozof czeka, aż oba widelce będą wolne i wtedy podnosi je jednocześnie.

Rozwiązanie poprawne: wykorzystany jest zewnętrzny arbiter (lokaj), który zapewnia, że jednocześnie nie więcej niż czterech (4) filozofów chciałoby jeść, to znaczy konkuruje o widelce. Jeśli pięciu (5) filozofów chciałoby jeść naraz, to lokaj powstrzyma jednego z nich.

METODY SYNCHRONIZACJI

Sekcja krytyczna
Sekcja krytyczna (angielskie critical section), fragment kodu, który powinien być wykonany z zachowaniem niepodzielności, tj. jednoetapowo, bez przerwań. Brak ochrony wykonywania sekcji krytycznej może powodować nieokreślone skutki w działaniu oprogramowania. Obsługę sekcji krytycznej organizuje się w procesach np. za pomocą semaforów, które z kolei są wspomagane sprzętowo, np. przez wyłączanie przerwań na czas wykonywania sekcji krytycznej albo za pomocą specjalnych, niepodzielnie wykonywanych rozkazów: zamień (swap) lub testuj i ustaw (test-and-set). Pierwszy z nich zamienia zawartość dwóch komórek pamięci, drugi bada stan komórki i określa jej nową wartość.

Mutex
Zamek (z angielskiego mutex, lock), obiekt synchronizacji wątków. Definiują się tu dwie operacje: lock (zamknij) i unlock (otwórz). Wątek próbujący zamknąć zamek już zamknięty zostaje zablokowany. Operacja otwarcia zamku odblokowuje jeden z zablokowanych wątków. Spotyka się też operację trylock (spróbuj zamknąć), która nie blokuje wątku, powiadamiając go o niemożliwości zamknięcia zamku.

Semafory
Nazwa dobrze oddaje naturę semaforów -- jest to mechanizm synchronizacji idealnie pasujący do rozwiązywania problemu sekcji krytycznej, a jego działanie przypomina działanie semafora kolejowego. Wyobraźmy sobie wielotorową magistralę kolejową, która na pewnym odcinku zwęża się do *k* torów. Pociągi jeżdżące magistralą to procesy. Zwężenie to uogólnienie sekcji krytycznej. Na zwężonym odcinku może znajdować się na raz maksymalnie *k* pociągów, każdy na oddzielnym torze. Podobnie jak w przypadku sekcji krytycznej, każdy oczekujący pociąg powinien kiedyś przejechać i żaden pociąg nie powinien stać, jeżeli jest wolny tor. Semafor to specjalna zmienna całkowita, początkowo równa *k*. Specjalna, ponieważ (po zainicjowaniu) można na niej wykonywać tylko dwie operacje:

- P -- Jeżeli semafor jest równy 0 (semafor jest opuszczony), to proces wykonujący tę operację zostaje wstrzymany, dopóki wartość semafora nie zwiększy się (nie zostanie on podniesiony). Gdy wartość semafora jest dodatnia (semafor jest podniesiony), to zmniejsza się o 1 (pociąg zajmuje jeden z *k* wolnych torów).
- V -- Wartość semafora jest zwiększana o 1 (podniesienie semafora). Jeżeli jakiś proces oczekuje na podniesienie semafora, to jest on wznawiany, a semafor natychmiast jest zmniejszany.

Istotną cechą operacji na semaforach jest ich niepodzielność. Jeżeli jeden z procesów wykonuje operację na semaforze, to (z wyjątkiem wstrzymania procesu, gdy semafor jest opuszczony) żaden inny proces nie wykonuje w tym samym czasie operacji na danym semaforze. Aby zapewnić taką niepodzielność, potrzebne jest specjalne wsparcie systemowe lub sprzętowe.
Rozwiązanie problemu sekcji krytycznej za pomocą semaforów jest banalnie proste. Wystarczy dla każdej sekcji krytycznej utworzyć odrębny semafor, nadać mu na początku wartość 1, w sekcji wejściowej każdy proces wykonuje na semaforze operację P, a w sekcji wyjściowej operację V. Wymaga to jednak od programisty pewnej dyscypliny: Każdej operacji P musi odpowiadać operacja V i to na tym samym semaforze. Pomyłka może spowodować złe działanie sekcji krytycznej.
Jeżeli procesy mogą przebywać na raz więcej niż w jednej sekcji krytycznej, to musimy zwrócić uwagę na możliwość *zakleszczenia*.

Kolejki komunikatów
Kolejki komunikatów przypominają uporządkowane skrzynki na listy. Podstawowe dwie operacje, jakie można wykonywać na kolejce, to wkładanie i wyjmowanie komunikatów (pakietów informacji) z kolejki. Komunikaty są uporządkowane w kolejce na zasadzie FIFO (ang. *first in, first out*). Włożenie powoduje dołączenie komunikatu na koniec kolejki, a pobranie powoduje wyjęcie pierwszego komunikatu z kolejki. Jeżeli kolejka jest pusta, to próba pobrania komunikatu powoduje wstrzymanie procesu w oczekiwaniu na komunikat.
Możliwych jest wiele różnych implementacji kolejek komunikatów, różniących się zestawem dostępnych operacji i ew. ograniczeniami implementacyjnymi. Pojemność kolejki może być ograniczona. Wówczas próba włożenia komunikatu do kolejki powoduje wstrzymanie procesu w oczekiwaniu na pobranie komunikatów. Niektóre implementacje mogą udostępniać nieblokujące operacje wkładania/pobierania komunikatów -- wówczas zamiast wstrzymywania procesu przekazywana jest informacja o niemożności natychmiastowego wykonania operacji. Możliwe jest też selektywne pobieranie komunikatów z kolejki.
Kolejki komunikatów są równie silnym mechanizmem synchronizacji, co semafory. Mając do dyspozycji semafory można zaimplementować kolejki komunikatów -- przekonanie się o tym pozostawiamy jako ćwiczenie dla

czytelnika. Odwrotna konstrukcja jest również możliwa. Rozważmy kolejkę komunikatów, do której wkładamy tylko puste, nic nie znaczące komunikaty. Wówczas kolejka taka będzie zachowywać się jak semafor. Chcąc zainicjować ją na określoną wartość, należy na początku włożyć do niej określoną ilość pustych komunikatów.

FUNKCJE SYSTEMOWE

fork()

Zadaniem funkcji fork() jest tworzenie nowego procesu, będącego potomkiem procesu wywołującego. Wszystkie procesy w systemie powstają za jej pomocą, oprócz procesu o pidzie 0, który jest tworzony wewnętrznie przez jądro przy ładowaniu systemu do pamięci.

exec()

zmienia kontekst poziomu użytkownika danego procesu na kopie programu wykonywalnego umieszczonego w pliku dyskowym. Jeżeli funkcja zostanie wykonana bezbłędnie, to poprzedni kontekst procesu znika i już nigdy nie zostanie odtworzony. Warto zwrócić uwagę, że funkcja exec nie powoduje powstania nowego procesu, lecz jedynie dokonuje wymiany kontekstu istniejącego procesu (wywołującego tę funkcję), który otrzymuje nowy segment danych, kodu i stosu.

exit()

W systemie Linux procesy kończą się wykonując funkcję systemową exit. Proces wywołujący exit przechodzi do stanu ZOMBIE, zwalnia swoje zasoby i oczyszcza kontekst, z wyjątkiem pozycji w tablicy procesów. Sposób wywołania funkcji jest następujący: exit (status); przy czym status jest wartością całkowitą przekazywaną procesowi macierzystemu do zbadania. Przyjmuje się zasadę, że liczba zero oznacza poprawne zakończenie się procesu, a wartość statusu różna od zera wskazuje na wystąpienie błędu. Procesy mogą wołać exit jawnie bądź niejawnie na zakończenie programu; funkcja inicjująca dołączana do wszystkich programów w C wywołuje exit po powrocie programu z funkcji main. Z drugiej strony jądro może wywołać funkcję exit na potrzeby procesu po nadejściu nieprzechwytywanego sygnału. W tym przypadku wartością status jest numer sygnału.

wait()

Wstrzymanie procesu do momentu zakończenia działania określonego potomka.

II. KOMUNIKACJA MIĘDZYPROCESOWA (IPC) W WINDOWSIE

KOMUNIKACJA MIĘDZY PROCESAMI

Procesy, które chcą komunikować się między sobą, muszą mieć możliwość zwracania się do siebie. Mogą to robić za pomocą komunikacji bezpośredniej lub komunikacji pośredniej.

Komunikacja bezpośrednia

W komunikacji bezpośredniej każdy proces, który chce nadać lub odebrać kominikat musi jawnie nazwać odbiorcę lub nadawcę, który uczestniczy w tej wymianie informacji.
Łącze komunikacyjne w tym schemacie ma następujące właściwości:

- Jest ustanawiane automatycznie między parą procesów, które mają komunikować się. Do wzajemnego komunikowania się wystarczy, aby procesy znały swoje identyfikatory.
- Dotyczy dokładnie dwu procesów.
- Między każdą para komunikujących się procesów istnieje dokładnie jedno łącze.
- Jest ono dwukierunkowe.

Komunikacja pośrednia

W komunikacji pośredniej komunikaty są nadawane i odbierane poprzez skrzynki pocztowe (nazywane także portami). Abstrakcyjna skrzynka pocztowa jest obiektem, w którym procesy mogą umieszczać komunikaty i z którego komunikaty mogą być pobierane. Każda skrzynka pocztowa ma jednoznaczną identyfikację. Proces może komunikować się z innym procesem za pomocą różnych skrzynek pocztowych. Możliwość komunikacji między dwoma procesami istnieje tylko wówczas, gdy mają one jakąś wspólną skrzynkę pocztową.

BUFOROWANIE

Łącze ma pewną pojemność, która określa liczbę komunikatów mogących w nim czasowo przebywać. Cecha ta może być postrzegana jako kolejka komunikatów przypisanych do łącza. Są trzy podstawowe metody implementacji takiej kolejki:

- Pojemność zerowa - maksymalna długość kolejki wynosi 0. Tzn., że łącze nie dopuszcza, by czekał w nim

jakikolwiek komunikat. W tym przypadku nadawca musi czekać aż odbiorca odbierze komunikat. Aby można było przesyłać komunikaty, oba procesy muszą być zsynchronizowane. Synchronizację taką nazywamy rendezvous.

- Pojemność ograniczona - kolejka ma skończoną długość n. Więc może w niej pozostawać maksymalnie n komunikatów. Jeśli w chwili nadania nowego komunikatu kolejka nie jest pełna, to nowy komunikat będzie w niej umieszczony i nadawca może kontynuować działanie bez czekania. Lecz gdy łącze, z powodu ograniczonej pojemności ulegnie zapełnieniu, wtedy nadawca będzie musiał być opuźniany, aż zwolni się miejsce w kolejce.
- Pojemność nieograniczona- kolejka ma potencjalnie nieskończoną długość. Dzięki temu może w niej oczekiwać dowolna liczba komunikatów. Nadawca nigdy nie jest opóźniany.

MECHANIZMY KOMUNIKACJI

DDE – *Dynamic Data Exchange*
Dynamiczna wymiana danych - specjalna metoda wymiany danych pomiędzy dwoma lub więcej programami pracującymi w Windows; jeśli chcemy jej użyć, musimy wpierw otworzyć wszystkie programy, pomiędzy którymi mamy zamiar wymieniać informacje; następnie wydajemy Windows polecenie, by utworzone zostało połączenie pomiędzy dwoma wybranymi programami; może to być na przykład edytor tekstu i arkusz kalkulacyjny, pomiędzy którymi chcemy dokonać wymiany danych; technika DDE jest obecnie obsługiwana przez większość programów pracujących w Windows.

OLE (Object Linking and Embedding)
Łączenie i wstawianie obiektów (OLE), jest cechą systemu Microsoft Windows, która umożliwia integrację obiektów pochodzących z różnych aplikacji. Obiektem są np. informacje pochodzące z arkusza kalkulacyjnego, rysunek utworzony przez program graficzny lub cyfrowy zapis dźwięku.
Jedną z głównych cech systemu OLE jest możliwość wyświetlania przez dowolną (zgodną z nim) aplikację, danych pochodzących z innej aplikacji. Dokument wyświetlający informacje poprzez technikę OLE zwie się dokumentem złożonym (compound document). Przypomina on kontener, który przechowuje obiekty. Podobne właściwości ma wyświetlana przez przeglądarkę internetową strona WWW. Może ona na przykład zawierać obrazy graficzne, które wstawiane są w tekst na podstawie odnośników w pliku źródłowym zapisanym w języku HTML.
Obiekty w dokumentach złożonych mogą być łączone (linked) lub wstawiane (embedded).
Obiekt łączony. Nie stanowi on integralnej części dokumentu, lecz przechowywany jest oddzielnie. Obiekt złożony zawiera łącze, które definiuje lokalizację obiektu. Obiekt wyświetlany jest w dokumencie złożonym, gdy jest on otwierany. Dzięki temu, iż obiekt wyświetlany jest w dokumencie złożonym dopiero po otwarciu rozmiar dokumentu złożonego jest niewielki. Jednakże jeżeli dokumenty złożone zostaną przeniesione w inne miejsce, wraz z nimi trzeba także przenieść obiekty łączone. Łącza w dokumencie będą poprawne, jeżeli pomiędzy nową lokalizacją obiektu złożonego a obiektami łączonymi zachowane jest trwałe połączenie sieciowe, łącza w dokumencie mogą zachować swoją poprawność.
Obiekt wstawiany. W przeciwieństwie do obiektów łączonych, obiekty wstawione (zwane również obiektami osadzonymi lub zanurzonymi) przechowywane są bezpośrednio w dokumencie złożonym. Przy przenoszeniu dokumentu złożonego przemieszczane są wraz z nim obiekty wstawione. Rozmiar dokumentu rośnie wraz z rozmiarem wielkością osadzanych w nim obiektów.
Jedną z zalet osadzania obiektów jest to, że wszystkie składniki dokumentu przechowywane są w jednym pliku. Jeżeli przenosimy go w inne miejsce, wszystkie zawarte w nim informacje przechodzą wraz z nim. W przypadku obiektów łączonych podczas przemieszczania dokumentu mogą pojawić się kłopoty. Może okazać się, że należy zaktualizować łącza lub przenieść obiekty łączone wraz z dokumentem głównym.

POTOKI NAZWANE (NAMED PIPES)
Potoki nazwane (zwane też potokami etykietowanymi lub mianowanymi) są interfejsem wysokiego poziomu, przeznaczonym do wymiany danych pomiędzy procesami realizowanymi na oddzielnych komputerach działających w sieci. Są sieciową wersją usług łączności międzyprocesowej IPC (interprocess communication), które tworzą interfejs pomiędzy procesami uruchomionymi pod nadzorem jednego, wielozadaniowego systemu operacyjnego. Potoki nazwane były pierwotnie pomyślane jako rozszerzenie systemu operacyjnego OS/2 i zostały zaimplementowane w sieciowych systemach operacyjnych Microsoft LAN Manager i IBM LAN Server. Po pewnych modyfikacjach wykorzystano je w środowisku systemu Windows. Potoki nazwane są zorientowane na połączenie. Ich implementacja opiera się na wywołaniach procedur interfejsu OS/2 API, który w Windows NT został powiększony o usługi asynchroniczne oraz dodatkowe funkcje zwiększające bezpieczeństwo.

GNIAZDA WINDOWS (WINSOCKS)
Windows Sockets (winsocks) obejmuje takie funkcje
- Obsługa operacji wejścia – wyjścia typu „scatter – gather" (rozproszenie – zebranie) i aplikacji asynchronicznych
- Konwencje jakości usług (QoS). Dzięki temu aplikacje mogą uzgadniać wymagania w zakresie czasu zwłoki i szerokości pasma, gdy sieć obsługuje QoS
- Rozszerzalność. Dzięki temu Winsock może być używana z protokałami innymi niż wymagane do obsługi przez Windows 2000
- Obsługa zintegrowanych przestrzemi nazw

MAILSLOTS
Innym mechanizmem komunikacji międzyprocesowej jest tzw. mailslots, który umożliwia przesyłanie komunikatów w trybie "zapisz i wyślij" (store and forward). Przesyłanie komunikatów odbywa się poprzez kolejki, przy czym nie gwarantuje się dostarczenia ich do adresata. Zakłada się bowiem, że wykorzystywana sieć jest wysoce niezawodna, a ewentualne błędy zostaną zlokalizowane przez protokoły wysokiego poziomu. System mailslots wykorzystuje się często do lokalizacji komputerów lub usług dostępnych w sieci, a także do rozsyłania zawiadomień. Zarówno potoki nazwane, jak i program mailslots implementowany jest jako system plików dzielący wspólne właściwości (np. buforowanie) z innymi systemami plików.

WM_COPYDATA
Komunikat WM_COPYDATA umożliwia wysyłanie danych z jednej aplikacji do innej. Aplikacja odbierająca pobiera dane w strukturze COPYDATASTRUCT. Gdy dane są przesyłane pomiędzy aplikacją 16-bitową i aplikacją opartą o Win32, system zamienia wszystkie wskaźniki.
Aplikacja wysyła komunikat WM_COPYDATA aby przesłać dane do innej aplikacji. Aby wysłać ten komunikat należy wywołać funkcję SendMessage z następującymi parametrami (nie wywoływać funkcji PostMessage).

Funkcja systemowa SendMessage()
SendMessage((HWND) *hWnd*, // uchwyt docelowego okna WM_COPYDATA, // komunikat do wysłania (WPARAM) *wParam*, // uchwyt okna (HWND) (LPARAM) *lParam* // dane (PCOPYDATASTRUCT));
- *wParam* - Uchwyt okna wysyłającego dane.
- *lParam* -Wskaźnik do struktury COPYDATASTRUCT, która zawiera dane do przesłania.

Jeśli aplikacja docelowa odbierze komunikat, funkcja powinna zwrócić TRUE; w innym przypadku powinna zwrócić FALSE.

Uwagi
Wysyłane dane nie może zawierać wskaźników ani innych referencji do obiektów nie dostępnych dla aplikacji odbierającej dane. Kiedy ten komunikat będzie wysyłany, dane nie mogą być zmieniane przez inny wątek wysyłającego procesu. Aplikacja odbierająca powinna rozważać dane tylko do odczytu. Parametr *lParam* jest poprawny tylko podczas wysyłania komunikatu. Aplikacja odbierająca nie powinna zwalniać pomięci na którą wskazuje *lParam*. Jeśli aplikacja musi mieć dostęp do danych po powrocie z SendMessage, to musi ona skopiować dane do bufora lokalnego.

CLIPBOARD (SCHOWEK)
Obiekt Clipboard zapewnia dostęp do systemowgo schowka. Obiekt Clipboard jest wykorzystywany do manipulowania tekstem i grafiką schowka systemowego. Można wykorzystać ten obiekt aby umożliwić użytkownikowi kopiowanie, wycinanie i wklejanie tekstu lub grafiki poprzez schowek. Przed skopiowaniem do schowka, należy wyczyścić jego zawartość metodą Clear.
Obiekt Clipboard jest wykorzystywany przez wszystkie aplikacje Windows, więc jego zawartość może się zmieniać po przełączeniu na inną aplikację.
Obiekt Clipboard może zawierać wiele danych w różnych formatach. Dane w Clipboard zostaną utracone, gdy zostaną zapisane do niego inne dane o tym samym formacie. Można umieścić obraz bitmapowy w formacie vbCFDIB przez wykorzystanie metody SetData a następnie umieścić tekst w formacie vbCFText metodą SetText. Potem można pobrać tekst metodą GetText i obraz metodą GetData
Metody:
- Clear – czyści zawartość obiektów
- GetData – pobiera grafikę z Clipboard
- GetFormat – zwraca format zawartości Clipboard
- GetText – pobiera tekst z Clipboard

- SetData – kopiuje grafikę do Clipboard
- SetText – kopiuje tekst do Clipboard

ATOMS
- atom table - tablica ciągów znaków i ich identyfikatorów
- typy
    - tablic:
        - globalne - dostępne dla wszystkich aplikacji
        - lokalne - dostępne w obrębie jednej aplikacji, InitAtomTable()
    - danych:
        - ciągi znaków
        - liczby całkowite (od 1 do MAXINTATOM-1)

Użycie
- AddAtom(), DeleteAtom(), FindAtom(), GetAtomName()
- GlobalAddAtom(), GlobalDeleteAtom(), GlobalFindAtom(),
- GlobalGetAtomName()

III. KOMUNIKACJA MIĘDZYPROCESOWA (IPC) W LINUKSIE

POTOKI JEDNOKIERUNKOWE (PIPES)
Najprościej, potok jest metodą przekierowywania standardowego wyjścia jednego procesu na standardowe wejście innego. Potoki są najstarszym narzędziem IPC, istnieją od najwcześniejszych wersji systemu operacyjnego UNIX. Są szeroko używane, nawet w linii poleceń powłoki. Kiedy proces tworzy potok, jądro tworzy dwa deskryptory na jego użytek. Jeden umożliwia zapis do potoku, drugi czytanie z niego. Na tym poziomie, potoki są mało użyteczne, gdyż tylko proces tworzący potok może z niego czytać. Pod Linuksem, potoki są obecnie prezentowane wewnętrznie jako prawidłowe i-węzły (inodes). Oczywiście taki i-węzeł przebywa wewnątrz jądra, nie na jakimś fizycznym systemie plików. Takie podejście otwiera nam ładną, poręczną furtkę We/Wy, o czym przekonamy się później. Aby wysłać dane do potoku używamy wywołania systemowego *write()*, natomiast aby odczytać dane musimy posłużyć się wywołaniem *read().* Pamiętaj, że niskopoziomowe We/Wy do pliku działa z jego deskryptorem!
Tworzenie potoków wraz z C może być troszeczkę trudniejsze niż to wynika z naszych doświadczeń z powłoki. Aby utworzyć prosty potok w C musimy posłużyć się wywołaniem systemowym pipe(). Podaje się jeden argument, który jest tablicą dwóch liczb całkowitych, jeżeli nasze wywołanie powiodło się macierz zawiera dwa deskryptory plików, które urzywamy do operacji na potoku. Najczęściej po takiej operacji proces tworzy swojego potomka, który dziedziczy wszystkie otwarte deskryptory plików. Pierwsza liczba całkowita w macierzy (element 0) jest ustawiana i otwierana w celu odczytu, natomiast drugi element służy do pisania. Unaoczniając, wyjście fd1 staje się wejściem dla fd0. Oczywiście wszelkie dane przesyłane przez potok są przesyłane przez jądro. Jeżeli rodzic chce otrzymywać dane od procesu potomnego powinien zamknąć fd1, potomek powinien to uczynić z fd0. Jeżeli ma być odwrotnie, rodzic zamyka fd0, a potomek zamyka fd1. Ponieważ deskryptory są wspólne dla potomka i rodzica musimy pamiętać aby zamknąć końcówkę, której nie będziemy używać, gdyż jeżeli tego nie zrobimy nigdy nie otrzymamy EOF. Po ustaleniu kierunku przepływu możemy używać deskryptorów jako zwykłe deskryptory plików.
Często deskryptory w procesie potomnym są duplikowane na standardowe wejście lub wyjście, następnie proces potomny może wykonać - *exec()* inny program, który dziedziczy standardowe strumienie. Istnieje inne wywołanie systemowe - *dup2()*, którego można również używać. To wywołanie oryginalnie powstało w 7 Wersji Unixa i zostało przeniesione do BSD, obecnie wymagane jest przez standard POSIX. Dzięki temu wywołaniu mamy duplikację deskryptora i zamknięcie poprzedniego w jednym. Dodatkowo, mamy zapewnione atomowe(atomic) działanie, które oznacza, że nie zostanie ono przerwane przez nadchodzący sygnał. Cała operacja zostanie wykonana z wyłączeniem przesyłania sygnałów przez jądro. Używając oryginalnego *dup()* programiści musieli zamknąć deskryptor przed użyciem wywołania. Przez co tworzył się pewien słaby punkt pomiędzy tymi wywołaniami - jeżeli sygnał dotarł pomiędzy wywołaniami *close()* i *dup()* duplikacja deskryptorów mogła zawieść. *Dup2()* rozwiązuje ten problem.
Prostą metodą na tworzenie potoku jest funkcja *popen()*. Ta standardowa funkcja biblioteki tworzy potok wywołując wewnętrznie *pipe()*. Po tym forkuje proces potomny, uruchamia powłokę Bourne'a oraz uruchamia argument "komenda" korzystając z powłoki. Kierunek przepływu danych określany jest przez argument "typ", który może przybierać wartości "r" (odczyt) lub "w" (zapis). Nie można ich użyć jednocześnie. Jeżeli cię

podkusi i podasz "rw" pod Linuxem potok zostanie otwarty w trybie wskazanym przez pierwszą literę, czyli w tym wypadku w trybie odczytu. Potoki otwierane za pomocą *popen()* muszą zostać zamknięte funkcją *pclose()*.

Uwagi na temat jednokierunkowych potoków:

- Dwukierunkowy potok może zostać stworzony poprzez otwarcie dwóch potoków oraz odpowiednie przemianowanie deskryptorów w potomku.
- Wywołanie *pipe()* musi zostać uruchomione PRZED wywołaniem *fork()*, w innym wypadku potomek nie odziedziczy deskryptorów! (dotyczy również *popen()*).
- Używając jednokierunkowych potoków należy pamiętać, iż wszystkie procesy przyłączone do potoku muszą mieć wspólnego przodka. Dzieje się tak dlatego, że jądro nie zezwala na adresację pamięci, w której znajduje się potok, jeżeli nie jest się potomkiem twórcy potoku. W przypadku nazwanych potoków nie zachodzi to ograniczenie.

POTOKI NAZWANE FIFO (NAMED PIPES)

Nazwany potok pracuje podobnie jak normalny ale znacząco różni się.

- Nazwane potoki istnieją w systemie plików jako plik urządzenia.
- Procesy różnych rodziców mogą dzielić dane poprzez nazwany potok.
- Po skończeniu wszystkich operacji nazwany potok pozostaje w systemie plików w celu późniejszego użycia.

Istnieje kilka sposobów aby utworzyć nazwany potok. Dwie pierwsze – *mknod* i *mkfifo* - mogą zostać wykonane bezpośrednio z powłoki. Powyższe dwie komendy wykonują to samo zadanie, z jednym wyjątkiem. Komenda *mkfifo* umożliwia zmianę atrybutów utworzonego FIFO. Natomiast po użyciu *mknod* będziemy musieli w tym celu uruchomić chmod. FIFO może być szybko zidentyfikowane poprzez literkę 'p' w listingu.
Aby utworzyć FIFO za pomocą C musimy posłużyć się wywołaniem systemowym *mknod()*. Operacje We/Wy na FIFO są właściwie takie same jak dla normalnych potoków, z jednym wyjątkiem - musimy użyć wywołania lub funkcji 'open' w celu fizycznego otwarcia kanału do potoku. Używając potoków nie musieliśmy tego robić, gdyż potok rezydował w jądrze a nie w systemie plików. W naszych przykładach będziemy traktować potoki tak jak strumienie - otwieramy je za pomocą *fopen()*, a zamykamy za pomocą *fclose()*. Normalnie, FIFO blokuje wykonywanie programu. Dla przykładu jeżeli otworzymy FIFO do czytania wykonywanie programu zostanie zatrzymane do czasu gdy inny proces otworzy FIFO do pisania. Oczywiście działa to również w drugą stronę. Jeżeli nie podoba nam się takie działanie musimy przekazać znacznik O_NONBLOCK wywołaniu *open()*.

KOLEJKI WIADOMOŚCI (MESSAGE QUEUES)

Kolejki wiadomości najlepiej opisać jako wewnętrznie połączoną listę w obszarze adresowym jądra. Wiadomości mogą być przesyłane do kolejki po kolei oraz odbierane z kolejki na kilka różnych sposobów. Każda kolejka posiada unikalny identyfikator.

Bufor wiadomości – struktura msgbuf

Pierwszą strukturą, której się przyjrzymy jest *msgbuf.* Ta struktura może być postrzegana jako szablon dla wiadomości. Sprawą programisty jest zdefiniowanie struktury tego typu, jest ważne abyś zrozumiał, iż istnieje struktura typu *msgbuf*. Jest ona zadeklarowana w linux/msg.h.
Struktura msgbuf ma dwóch członków:

- mtype - typ wiadomości reprezentowany jako liczba dodatnia. To musi być liczba dodatnia!
- mtext - wiadomość.

Możliwość nadawania określonej wiadomości typu dostarcza ci narzędzia do tworzenia różnorodnych wiadomości w jednej kolejce. Dla przykładu: procesy typu klient mogą rozpoznawać specjalny numer oznaczający wiadomość od serwera, a serwer będzie obsługiwał inny numer oznaczający, że jest to wiadomość od klienta. Inny scenariusz: aplikacja może znakować wiadomości o błędach jedynką, żądania dwójką, masz nieograniczone możliwości.
Nie daj się zwieść przez nazwę drugiego elementu - mtext. Pole to nie jest zmuszone do przechowywania tablicy znaków, możesz przesyłać cokolwiek. Samo to pole może zniknąć z wiadomości, gdyż cała struktura może zostać przedefiniowana przez programistę.

Struktura msg

Jądro przechowuje każdą wiadomość wewnątrz ramki jaką jest struktura msg. Jest ona zdefiniowana w linux/msg.h

- msg_next - jest to wskaźnik do następnej wiadomośći. Są one przechowywane jako pojedyńczo łączona lista wewnątrz przestrzeni adresowej jądra.
- msg_type - jest to typ wiadomości ustalony w strukturze msgbuf przez użytkownika.
- msg_spot - wskaźnik do początku ciała wiadomości.

- msg_ts - długość ciała lub tekstu wiadomości.

Struktura msqid_ds
Każdy z trzech typów obiektów IPC posiada wewnętrzną strukturę danych utrzymywaną przez jądro. Dla kolejek wiadomości jest to struktura msqid_ds. Jądro tworzy, przechowuje i utrzymuje kopię tej struktury dla każdej kolejki wiadomości utworzonej w systemie. Jest ona zdefiniowana w linux/msg.h

- msg_perm - struktura ipc_perm, która jest zdefiniowana w linux/ipc.h. Zawiera ona informację na temat praw kolejki, włączając prawa dostępu oraz informacje o twórcy kolejki (uid, etc).
- msg_first - wskaźnik do pierwszej wiadomości w kolejce (początku listy).
- msg_last - wskaźnik do ostatniej wiadomości (końca listy).
- msg_stime - czas (time_t) kiedy wysłano ostatnią wiadomość.
- msg_rtime - czas ostatniego odebrania wiadomości z kolejki.
- msg_ctime - czas ostatniej 'zmiany' dokonanej na kolejce (więcej dowiesz się później).
- wwait i rwait - wskaźniki do kolejki oczekiwania (wait queue) jądra. Używane są gdy operacja na kolejce uważa, że proces śpi (np. kolejka jest pełna i proces czeka na otwarcie).
- msg_cbytes - suma rozmiarów wszystkich wiadomości w kolejce.
- msg_qnum - liczba wiadomości w kolejce.
- msg_qbytes - maxymalna liczba bajtów dla kolejki.
- msg_lspid - PID procesu, który jako ostatni wysłał wiadomość.
- msg_lrpid - PID procesu, który jako ostatni odebrał wiadomość.

Wywołanie systemowe msgget()
W celu utworzenia nowej kolejki wiadomości lub dostępu do istniejącej używamy wywołania msgget(). Pierwszym argumentem msgget() jest wartość klucza (w naszym przypadku podana przez ftok()). Wartość ta porównywana jest do wartości kluczy kolejek istniejących wewnątrz jądra, w tym momencie otwarcie lub otworzenie kolejki zależy od zawartości argumentu msgflg:

- IPC_CREAT - utwórz kolejkę jeżeli ona nie istnieje.
- IPC_EXCL - użyte razem z IPC_CREAT zwraca błąd jeżeli kolejka istnieje.

Jeżeli używasz samo IPC_CREAT msgget() zwraca identyfikator nowo utworzonej kolejki lub istniejącej, której wartość klucza jest taka sama. Jeżeli używasz IPC_EXCL razem z IPC_CREAT zostanie utworzona nowa kolejka lub wywołanie zwróci błąd. Użycie tej flagi gwarantuje nam, że nie istenieje kolejka z otwartym dostępem. Możesz również z'OR'ować dodatkowy ósemkowy tryb z argumentem, gdyż każdy obiekt IPC posiada prawa podobne w działaniu do praw pliku z systemu plików Unixa.

Wywołanie systemowe msgsnd()
Kiedy posiadamy identyfikator kolejki możemy przeprowadzać operacje na niej. Aby wysłać wiadomość do kolejki możemy posłużyć się wywołaniem msgsnd. Pierwszy agrument (msgsnd) jest identyfikatorem zwróconym przez msgget. Drugi (msgp) to wskaźnik do naszego bufora wiadomości. Argument msgsz zawiera rozmiar wiadomości w bajtach, wyłączając długość typu (4 bajty).
Argument msgflg może być ustawiony na 0 (ignorowany) lub na IPC_NOWAIT - jeżeli kolejka jest pełna wiadomość nie jest zapisywana, kontrol powraca do procesu wywołującego. Jeżeli nie podana proces będzie czekał na możliwość zapisu.

Wywołanie systemowe msgctl()
Aby kontrolować kolejkę używamy wywołania systemowego msgctl(). Wywołując msgctl() z odpowiednio dobranymi poleceniami możesz manipulować jednostki, które zbyt łatwo nie doprowadzą systemu do upadku.

SEMAFORY
Semafory można opisać jako liczniki używane do kontroli dostępu do dzielonych zasobów przez różne procesy. Najczęściej używa się ich do blokowania dostępu do zasobu podczas gdy jakiś proces przeprowadza na nim operacje. Uważa się, iż najtrudniejszymi do zrozumienia wśród obiektów IPC są semafory. W celu pełnego zrozumienia ich działania zaczniemy od krótkiego opisu, dopiero po tym przejdziemy do wywołań systemowych i sposobu użycia.
Semafor - urządzenia sygnalizacyjne przy torze kolejowym, podające określone sygnały, zapewniajacę prawidłowość w ruchu pociągów. To samo możemy powiedzieć na temat prostego semafora. Jeżeli jest on włączony (ramiona w górze) zasób jest dostępny (możemy przejechać). Gdy semafor jest wyłączony (ramiona w dole) zasób jest niedostępny (musimy czekać).

Ten prosty przykład ukazyje jedynie podstawy, należy sobie zdać sprawę, iż semafory zaimplementowane są jako układy. Oczywiście układ może zawierać tylko jeden semafor, tak jak w naszym przykładzie.
Możemy również podejść do zagadnienia semaforów w inny sposób - jako liczniki zasobów. Wyobraźmy sobie inny rzeczywisty scenariusz. Wyobraź sobie zarządcę drukowania (spooler) obsługującego wiele drukarek, każda drukarka obsługuje wiele żądań wydruku. Nasz hipotetyczny zarządca drukowania będzie obsługiwał układ semaforów w celu monitorowania dostępu do każdej z drukarek.
Załużmy, że mamy 5 dostępnych drukarek. Nasz zarządca tworzy zestaw semaforów zawierający 5 semaforów, po jednym dla drukarki. Każda z drukarek jest zdolna do fizycznego drukowania jednego zadania, każdy z semaforów zainicjujemy wartością 1 oznaczającą, że drukarka jest podłączona i gotowa na żądanie.
Jaś wysyła żądanie wydruku do zarządcy. Zarządca spogląda na zestaw semaforów i wyszukuje pierwszego semafora, który równa się 1. Przed wysłaniem żądania do drukarki zarządca zmniejsza o jeden używany semafor. Teraz wartością semaforu jest 0. W świecie Systemu V semafor ustawiony na 0 oznacza 100% użycia. W naszym przykładzie żadne inne żądanie nie może zostać przesłane do tej drukarki aż do czasu zmiany wartości.
Kiedy zadanie Jasia zostanie wykonane zarządca zinkrementuje wartość semaforu odpowiadającego za użytą drukarkę. Wartościa semaforu staje się 1 - drukarka jest dostępna. Kiedy wszystkie semafory są ustawione na 1 oznacza to brak dostępnych drukarek.
Nie daj się zwieść inicjalizacji semaforu jedynką. Kiedy używamy semaforów jako liczników zasobów możemy je inicjować każdą dodatnią wartością . Gdyby każda z drukarek potrafiła obsłużyć 10 zadań wydruku jednocześnie, ustawilibyśmy semafory wartością 10. W następnym rozdziale odkryjesz powiązania semaforów z segmentami dzielonej pamięci, które będą działać jako wartownicy zapobiegający wielu zapisom jednocześnie.

Struktura semid_ds w jądrze
Tak jak w przypadku kolejek wiadomości jądro utrzymuje specjalną strukturę dla każdego zestawu semaforów, istnieje ona wewnątrz obszaru adresowego jądra. Struktura ta jest typu semid_ds, który zdefiniowany jest w linux/sem.h
Tak jak w przypadku kolejek wiadomości, operacje na tej strukturze przeprowadzane są przez specjalne wywołanie systemowe i tylko przez nie, nie powinieneś zabawiać się :) z tą strukturą w inny sposób. Opis niektórych pól:

- sem_perm - Jest to struktura typu ipc_perm, który jest zdefiniowany w linux/ipc.h. Przechowuje prawa dostępu, oraz informacje o twórcy i właścicielu zestawu.
- sem_otime - Czas ostaniej operacji semop() (więcej o tym za chwilkę)
- sem_ctime - Czas ostatniej zmiany struktury (np.: zmiana praw, itp.)
- sem_base - Wskaźnik do pierwszego semaforu w tablicy (zobacz następną strukturę)
- sem_undo - Liczba żądań odwołania (undo) dla tej tablicy (więcej za chwilę)
- sem_nsems - Liczba semaforów w zestawie (tablicy).

Struktura sem w jądrze
W strukturze semid_ds istnieje wskaźnik do podstawy tablicy semaforów. Każdy członek tej tablicy jest strukturą typu sem, która jest zdefiniowana w linux/sem.h

- sem_pid - PID (ID procesu), który przeprowadził ostatnią operację
- sem_semval - Obecna wartość semaforu
- sem_semncnt - Liczba procesów oczekujących dostępu do zasobu
- sem_semzcnt - Liczba procesów oczekujących na 100% użycie zasobów

Wywołanie systemowe semget()
W celu utworzenia nowego zestawu semaforów, lub dostępu do istniejącego, używamy wywołania systemowego semget(). Pierwszym argumentem dla semget() jest wartość klucza (w naszym przypadku zwrócona przez ftok()). Wartość ta porównywana jest do istniejących kluczy. Teraz operacje otworzenia lub dostępu zależą od zawartości argumentu semflg:

- IPC_CREAT - Utwórz zestaw semaforów jeżeli jeszcze nie istnieje w jądrze.
- IPC_EXCL - Użyte z IPC_CREAT zwraca błąd jeżeli zestaw już isnieje.

Jeżeli używamy samego IPC_CREAT semget() zwraca identyfikator zestawu, który został utworzony lub identyfikator już istniejącego, który posiada podaną wartość klucza. Jeżeli podano również IPC_EXCL zostanie utworzony nowy zestaw lub wywołanie zwróci -1. Podanie tylko IPC_EXCL jest bezużyteczne, jednak w połączeniu z IPC_CREAT gwarantuje nam, iż nie używamy istniejącego semaforu.
Tak jak dla każdej innej formy IPC Systemu V możemy podać ósemkowe prawa dostępu, które zostaną z'OR'owane z umaską aby utworzyć prawa dostępu dla zestawu.

Argument nsems określa liczbę semaforów, które zostaną utworzone dla nowego zestawu. Reprezentuje to liczbę drukarek w naszym przykładzie. Maxymalna liczba semaforów jest zdefiniowana w linux/sem.h.

Wywołanie systemowe semop()
Pierwszym argumentem dla semop() jest klucz (w naszym przypadku zwrócony przez semget). Drugi argument (sops) jest wskaźnikiem do tablicy operacji , które mają zostać przeprowadzone na grupie semaforów. Trzeci argument (nsops) jest liczbą operacji w tablicy.
Argument sops wskazuje tablicę typu sembuf, który jest zdefiniowany w linux/sem.h.

- sem_num - Numer semafora którym jesteś zainteresowany
- sem_op - operacja do wykonania (dodatnia, ujemna lub zero)
- sem_flg - Flagi operacji

Jeżeli sem_op jest ujemny jego wartość zostaje odjęta od semafora. Odpowiada to używaniu zasobów kontrolowanych lub monitorowanych przez semafor. Jeżeli nie podano IPC_NOWAIT proces wywołujący śpi do czasu aż wymagana liczba zasobów jest dostępna (inny proces zwolnił jakieś).
Jeżeli sem_op jest dodatni jego wartość jest dodawana do semafora. Odpowiada to zwolnieniu zasobów. Zasoby powinny być zawsze zwalniane jeżeli nie są dłużej używane!
Jeżeli sem_op jest zerem proces będzie spał do czasu aż semafor będzie miał wartość zera. Odpowiada to 100% użycia. Dobrym przykładem jest demon działający z prawami roota, który będzie dynamicznie zmieniał rozmiar zbioru semaforów jeżeli ten będzie w 100% użyty.
W celu wytłumaczenia wywołania semop przywołajmy nasz przykład z drukarkami. Załużmy, iż mamy tylko jedną drukarkę, która potrafi drukować tylko jedną pracę. Utworzymy zestaw tylko z jednym semaforem (tylko jedna drukarka) i zainicjujemy go wartością jeden (tylko jedna praca).
Za każdym razem gdy chcemy wysłać pracę do drukarki musimy upewnić się czy jest ona dostępna. Robimy to próbując otrzymać jedną jednostkę z semafora.

Wywołanie systemowe semctl()
Wywołanie systemowe semctl używane jest do kontrolowania zestawu semaforów. Jest ono analogiczne do wywołania msgctl, które jest opdowiednie dla kolejek. Jeżeli porównasz listę argumentów obu wywołań zauważysz, że są one nieco odmienne. Przypomnij sobie, iż aktualnie semafory są zaimplementowane jako zestawy, rzadziej niż pojedyńcze jednostki. Do operowania semaforem potrzeba nie tylko klucz IPC, ale także numer semafora w zestawie.
Oba wywołania systemowe posiadają argument cmd używany do określenia operacji. Ostatnią różnicą jest krańcowy argument. Dla msgctl jest to kopia wewnętrznej struktury używanej przez jądro. Przypomnij sobie, że używaliśmy ją do odbierania lub ustawiania informacji o prawa dostępu i właścicielu kolejki wiadomości. Przy semaforach, dodatkowe komendy operacyjne są dozwolone co wymaga bardziej złożonej struktury. Użycie union myli wielu początkujących programistów. W celu zapobieżenia temu przeanalizujemy tę strukturę dokładnie.
Pierwszym argumentem dla semctl() jest klucz (zwrócony przez semget). Drugi argument (semun) to numer semaforu, którym jesteśmy zainteresowani. Jest to index elementu w zestawie, pierwszy element ma index o wartości 0.
Argument cmd przedstawia komendę, która ma zostać wykonana. Jak widzisz, mamy znane komendy IPC_STAT/IPC_SET, jak również wiele dodatkowych specyficznych dla zestawów semaforów:

- IPC_STAT - Pobiera strukturę semid_ds i zachowuje ją w pod adresem buf argumentu semun.
- IPC_SET - Ustawia wartość członka ipc_perm struktury semid_ds. Wartości pobiera z buf unii semun.
- IPC_RMID - Usuwa zestaw z jądra.
- GETALL - Używany do pobierania wartości wszystkich semaforów z zestawu Wartości całkowite przechowywane są w tablicy unsigned short int wkazywanej przez array - członka unii.
- GETNCNT - Zwraca ilość procesów oczekujących na zasoby.
- GETPID - Zwraca PID procesu, który jako ostatni wywołał semop.
- GETVAL - Zwaraca wartość pojedyńczego semaforu z zestawu.
- GETZCNT - Zwraca ilość procesów czekających na 100% użycia zasobu.
- SETALL - Ustawia wartości wszystkich semaforów wartościami z tablicy array zawartej w unii.
- SETVAL - Ustawia wartość jednego semaforu z zestawu. Wartość pobierana z val zawartego w unii.

Argument arg jest elementem typu semun. Ta unia zadeklarowana jest w linux/sem.h

PAMIĘĆ DZIELONA (SHARED MEMORY)
Pamięć dzieloną można najlepiej opisać jako obszar pamięci (segment), który będzie dzielony między kilkoma procesami. Jest to najszybsza forma IPC, ponieważ nie istnieje żaden pośrednik (potok, kolejka wiadomości). Zamiast tego informacja jest bezpośredno umieszczana w segmencie pamięci, w obszarze adresowym procesu

wywołującego. Segment może zostać utworzony przez jeden proces, a wykorzystywać go może dowolna liczba procesów.

Struktura shmid_ds w jądrze
Tak jak w przypadku kolejek wiadomości i zestawów semaforów, jądro przechowuje specjalną wewnętrzną strukturę dla każdego segmentu pamięci dzielonej. Struktura ta jest typu shmid_ds, który zdefiniowany jest w linux/shm.h
Operacje na tej strukturze wykonywane są przez specjalne wywołanie systemowe, nie powinieneś jej zmieniać inną metodą. Oto opis ważniejszych pól:
shm_perm - Jest to struktura ipc_perm, jest ona zdefiniowana w linux/ipc.h. Przechowuje ona informacje na temat praw dostępu, twórcy (uid, itd.).
- shm_segsz - Rozmiar segmentu (w bajtach).
- shm_atime - Czas ostatniego podpięcia segmentu.
- shm_dtime - Czas ostatniego odpięcia segmentu.
- shm_ctime - Czas ostatniej zmiany tej struktury (zmiana trybu, itp.).
- shm_cpid - PID procesu tworzącego.
- shm_lpid - PID procesu, który jako ostatni operował na segmencie.
- shm_nattch - Liczba procesów podpiętych do segmentu.

Wywołanie systemowe shmget()
W celu utworzenia nowego (lub otworzenia) segmentu pamięci używamy wywołania systemowego shmget(). Pierwszym argumentem jest klucz (w naszym wypadku zwrócony przez ftok()). Wartość ta porównywana jest do innych kluczy segmentów pamięci. Teraz akcja zależy od zawartości argumentu shmflg:
- IPC_CREAT - Utwórz segment jeżeli ten nie istnieje.
- IPC_EXCL - Użyte z IPC_CREAT zwraca błąd jeżeli segment już istnieje.

Jeżeli podano tylko IPC_CREAT shmget() zwraca identyfikator segmentu istniejącego lub utworzonego. W połączeniu z IPC_EXCL tworzony jest nowy segment lub wywołanie zwraca błąd. Gwarantuje nam to, iż to my tworzymy segment pamięci.
I tym razem możesz również z'OR'ować z maską ósemkowy tryb.

Wywołanie systemowe shmat()
Jeżeli argument addr = 0 jądro próbuje znaleźć niemapowany region. Jest to zalecany sposób użycia. Jednakże można również podać adres, jest to używane aby ułatwić dostęp do sprzętu, lub rozwiązać konflikty z innymi programami. Flaga SHM_RND może być z'OR'owana z argumentem shmflg aby wymusić stronicowe wyrównanie adresu (zaokrąglenie w dół do najbliższego rozmiaru strony).
Dodatkowo, jeżeli podano flagę SHM_RDONLY segment zostanie wmapowany jako "tylko do oczytu".
Jest to chyba najłatwiejsze wywołanie. Po poprawnym przyłączeniu segmentu otrzymujemy wskaźnik do początku segmentu, użycie segmentu jest po prostu użyciem wskaźnika. Pamiętaj aby zachować wartość początkową wskaźnika! Jeżeli tego nie zrobisz nie będziesz miał dostępu do początku segmentu.

Wywołanie systemowe shmctl()
Wywołanie to jest utworzone na podobieństwo msgctl.

Wywołanie systemowe shmdt()
Gdy segment nie jest już potrzebny przez proces powinno nastąpić odłączenie. Jak wspomniano wcześniej nie jest to równoznaczne z usunięciem segmentu z jądra. Pod odłączeniu element shm_nattch struktury shmid_ds jest dekrementowany. Jeżeli jego wartość osiągnie 0 segment zostanie fizycznie usunięty przez jądro.

GNIAZDA UNIX (UNIX SOCKETS)
Gniazdka sieciowe (ang. sockets) są chyba najbardziej uniwersalnym spośród mechanizmów IPC (ang. Inter Process Communication - komunikacja międzyprocesowa). Ich implementacja w systemie Linux wzorowana jest na kodzie pochodzącym z systemu BSD (Berkeley System Distribution) w wersji 4.4. Jeśli dwa procesy mają się między sobą komunikować, każdy z nich tworzy po swojej stronie jedno gniazdo. Parę takich gniazd można więc określić mianem końcówek kanału komunikacyjnego. Gniazd używa się głównie do komunikacji z odleglym procesem za pośrednictwem sieci, jednak można je zastosować także w przypadku wymiany informacji między procesami działającymi w obrębie jednej maszyny. Ta uniwersalność zastosowań jest zapewniona dzięki istieniu różnych odmian gniazd. Gniazdo jest opisywane za pomocą kombinacji trzech atrybutów: domeny adresowej, sposobu komunikacji i protokołu sieciowego. Niezależnie od tego, do jakich specyficznych zastosowań

zamierzamy wykorzystać gniazda, za każdym razem będziemy zaczynali od tego samego - utworzenia naszej (klienta lub serwera) końcówki połączenia (czyli właśnie jednego z pary gniazd). Gniazdowe API (ang. Application Programmer's Interface - interfejs programisty) systemu BSD desygnuje do tego celu funkcję socket(). Funkcja oczekuje trzech parametrów:

- domain - Tzw. domena adresowa, oznacza domenę w której nastąpi komunikacja poprzez to gniazdo. Jest to konieczne aby określić, jaki rodzaj adresu będziemy przypisywać gniazdu (o tym potem). Wszystko stanie się bardziej jasne, kiedy ujrzymy kilka spośród dopuszczalnych wartości tego parametru:
  - PF_LOCAL (PF_UNIX) - wspomniana wcześniej komunikacja w obrębie jednej maszyny
  - PF_INET - Internet, czyli używamy protokołów z rodziny TCP/IP PF_IPX - protokoły IPX/SPX (Novell)
  - PF_PACKET - niskopoziomowy interfejs do odbierania pakietów w tzw. surowej (ang. raw) postaci

  Wszystkie dopuszczalne wartości znajdują się w pliku bits/socket.h. Zaglądajac tam spostrzeżemy, że zamiennie używa się notacji PF_xxx (ang. PF - Protocol Family) oraz AF_xxx (ang. AF - Address Family). Nie zdziwmy się więc, kiedy w jednym kodzie zobaczymy PF_INET, a w innym AF_INET - to praktycznie jedno i to samo.

- type - Sposob komunikacji. Do wyboru m.in.:
  - SOCK_STREAM - dwukierunkowa komunikacja zorientowana na połączenia
  - SOCK_DGRAM - komunikacja bezpołączeniowa korzystająca z tzw. datagramów
  - SOCK_RAW - dostęp do surowych (ang. raw) pakietów na poziomie warstwy sieciowej (ang. network layer) warstwowego modelu TCP/IP
  - SOCK_PACKET - ten typ był używany do obsługi surowych pakietów ale z warstwy fizycznej (ang. link layer), czyli o jedną warstwę "niżej" od SOCK_RAW; obecnie nie należy stosować tego typu - jego rolę przejęła oddzielna domena PF_PACKET

  Najczęściej używanymi typami są trzy pierwsze. Wszystkie wartości są zdefiniowane w pliku bits/socket.h.

- protocol - Określa konkretny protokół, którego będziemy używać. Zazwyczaj w obrębie jednej domeny adresowej i sposobu komunikacji istnieje tylko jeden protokół do ich obsługi. Na przykład, jeśli korzystamy z domeny PF_INET oraz SOCK_STREAM to nie mamy dużego wyboru ponieważ możemy skorzystać tylko z protokołu TCP. W takim wypadku parametrowi protocol nadajemy wartość 0 - system sam określi, jakiego protokołu używać. Czasami jednak zachodzi możliwość wyboru spośród kilku różnych protokołów. Rozważmy przykład gniazda PF_INET ale typu SOCK_RAW. W tym przypadku dzięki parametrowi protocol możemy określić, jakie konkretnie pakiety nas interesują.

Wszystkie dopuszczalne wartości dla domeny PF_INET znajdziemy w linux/in.h.
Funkcja socket() zwraca wartość -1 w przypadku błędu albo deskryptor gniazda, jeśli udało się stworzyć dane gniazdo. Wspomniany deskryptor gniazda spełnia analogiczną rolę, jak deskryptor pliku. Dzięki temu do obsługi gniazd możemy wykorzystywać funkcje typowe dla obsługi plików (np. write(), read(), close()).

REMOTE PROCEDURE CALL (RPC)
Remote Procedure Call (RPC) to protokół zdalnego wywoływania procedur, stworzony przez Suna i swego czasu dość popularny na Uniksach, obsługiwany w bibliotekach języka Java, a współcześnie wypierany przez bardziej rozbudowane protokoły takie jak CORBA czy XML-RPC.
Standard RPC zdefiniowany jest w RFC 1057. RPC używa do kodowania danych formatu XDR (eXternal Data Representation) zdefiniowanego w RFC 1832.
Protokoły tego typu jak (RPC, CORBA, DCOM, czy XML-RPC) mają na celu ułatwienie komunikacji pomiędzy komputerami. Zazwyczaj wyglądało to tak:

- Serwer (czyli program oferujący usługi, np. drukowania) przez cały czas nasłuchuje na wybranym porcie, czy ktoś się z nim nie łączy.
- Klient (czyli program który potrzebuje jakiejś usługi od serwera na innym komputerze) nawiązuje z nim łączność poprzez sieć komputerową.

Klient wysyła swoje dane we wcześniej ustalonym przez programistów klienta i serwera formacie.
Serwer realizuje usługę i odsyła potwierdzenie lub kod błędu.

Protokoły powyższe same zapewniają cały powyższy mechanizm, ukrywając go przed klientem. Może on nawet nie wiedzieć, że łączy się z innym komputerem - z punktu widzenia programisty zdalne wywołanie procedury

serwera wygląda jak wywołanie dowolnej innej procedury z programu klienta.

POLECENIA SYSTEMOWE ZWIĄZANE Z IPC
ipcs udostępnia informacje o urządzeniach IPC, do których ma dostęp proces wywołujący.
Opcja -i umożliwia podanie konkretnego identyfikatora (id) zasobu. Wtedy drukowane będą tylko dane dotyczące tego id.
Zasoby mogą być określane następująco:

-m segmenty dzielonej pamięci
-q kolejki komunikatów
-s tablice semaforów
-a wszystko (domyślnie)

Format wyjściowy może być określony następująco

-t czas
-p pid
-c twórca
-l limity
-u ogólnie

ipcrm usuwa zasób IPC wskazywany przez podane mu ID

kill wysyła podany sygnał do danego procesu. Jeżeli nie podano numeru sygnału, wysyłany jest sygnał TERM. Sygnał TERM kończy te procesy, które go nie przechwytują. Dla innych procesów niezbędne może się okazać użycie sygnału KILL(9), ponieważ nie może on zostać przechwycony. Większość nowoczesnych powłok posiada wbudowane polecenie kill.
OPCJE

- pid - Lista procesów, które mają być zakończone. Każda opcja pid może być jedną z następujących:
  - nazwa procesu, wówczas sygnał zostanie wysłany do nazwanego procesu. n, gdzie n jest liczbą większą od zera. Sygnał zostanie wysłany do procesu, którego pid jest równy n. -1, w którym to przypadku sygnał zostanie wysłany do wszystkich procesów o numerach pid o wartościach od MAX_INT do 2, jeżeli pozwalają na to uprawnienia użytkownika.
  - -n, gdzie n jest większe od 1; sygnał zostanie wysłany do wszystkich procesów należących do grupy o numerze n. Jeżeli n jest ujemne, musi przed nim wystąpić numer sygnału - w przeciwnym wypadku numer grupy zostanie potraktowany jako numer sygnału do wysłania.
- -s - Rodzaj wysyłanego sygnału. Może być podany jako nazwa lub numer.
- -p - Polecenie wypisze jedynie numery procesów (pid), do których zostałby wysłany sygnał, zamiast go rzeczywiście wysyłać.
- -l - Wypisz listę nazw sygnałów. Znajduje się ona w pliku /usr/include/linux/signal.h

IV. ZARZĄDZANIE PAMIĘCIĄ W SYSTEMIE LINUX

Podsystem zarządzania pamięcią jest jedną z najważniejszych części systemu operacyjnego. Jego zadaniem jest obsługa pamięci wirtualnej, a co za tym idzie stronicowania. Odpowiada on za tłumaczenie adresów wirtualnych na fizyczne i na odwrót. W Linuksie nazywa się on jednostką zarządzania pamięcią (ang. *Memory Management Unit*), w skrócie MMU.

Usługi dostarczane przez menadżera pamięci.
Rezultaty działania MMU:

- Większa przestrzeń adresów. System operacyjny działa tak, jakby w systemie było dużo więcej pamięci niż jest faktycznie, ponieważ pamięć wirtualna może być wielokrotnie większa niż pamięć fizyczna.
- Ochrona pamięci. Każdy proce w systemie ma swoją własną przestrzeń adresów. Jest ona odizolowana od innych więc inne procesy nie mogą sobie wzajemnie przeszkadzać. Mechanizmy pamięci wirtualnej umożliwiają ochronę przed zapisem wybranych obszarów pamięci. Dzięki czemu nie dochodzi do nadpisania nieswoich obszarów pamięci.
- Mapowanie pamięci. Czyli odwzorowywanie. Używane jest do odwzorowania obrazu i plików danych do przestrzeni adresów wirtualnych. Podczas mapowania procesu zawartość pliku jest linkowana (czyli wiązana) bezpośrednio do przestrzeni adresów wirtualnych procesu.
- Sprawiedliwy przydział pamięci. Podsystem zarządzania pamięcią umożliwia każdemu działającemu

procesowi w systemie sprawiedliwy podział pamięci fizycznej w systemie. Umożliwia to algorytm wymiany.
- Współdzielenie pamięci. Jeśli jest to potrzebne umożliwia kilku procesom korzystanie z tego samego obszaru pamięci. Np. w przypadku bibliotek dynamicznych.
- Wymiana informacji między procesami. Czyli mechanizm Inter Process Communication (IPC), który umożliwia dwóm lub większej ilości procesom wymianę informacji przez wspólną dla nich wszystkich pamięć.

Model pamięci wirtualnej.
W systemie pamięci wirtualnej wszystkie adresy są wirtualne, a nie fizyczne. Czyli system pracuje na adresach wirtualnych, które są odniesieniami do adresów fizycznych. Zarówno fizyczna jak i wirtualna pamięć podzielona jest na specjalne kawałki, takiej wielkości aby łatwo było konwertować pamięć wirtualną na fizyczną i odwrotnie. Kawałki te nazywane są stronami. Strony te są takich samych rozmiarów. Linux na systemie Alpha AXP używa stron 8 kilobajtowych, a na systemie Intel x86 4 kilobajtowych. Każda z tych stron posiada unikalny numer, tzw. numer ramki strony (ang. *page frame number*). W skrócie PFN. Pozwala to na sprawniejsze zarządzanie pamięcią.

Adresy wirtualne.
Adresy wirtualne w przeciwieństwie do fizycznych są „sztuczne", z punktu widzenia sprzętu nie istnieją. Adresy te są generowane przez procesor i przeliczane na fizyczne przez MMU. W ich skład wchodzą dwa elementy. Pierwszym z nich jest numer ramki strony, używany jako indeks w tabeli stron. A drugim przesunięcie strony, dzięki któremu można określić o jaki adres w danej stronie fizycznej nam chodzi.

Tablica stron.
Każdy proces posiada własną tablice stron. Zawiera ona adresy każdej ramki strony w fizycznej pamięci używanej przez dany proces. Dzięki temu procesor wykonując go może, posiadając adres wirtualny, przetłumaczyć go sobie na adres fizyczny. Każdy indeks w tablicy posiada tzw. wejście. W wejściu zapisane są informacje o dostępie do ramki strony, jej ważności (czyli czy istnieje obecnie w pamięci fizycznej czy nie) oraz rodzaju (np. tzw. brudna strona). Tablica stron przechowywana jest w pamięci operacyjnej.

Układ przestrzeni adresów.
Układy przestrzeni adresów stron fizycznych i wirtualnych są od siebie niezależne. Ramki strony fizycznej nie przypisuje się na stałe określonej ramki strony wirtualnej. Dla różnych procesów jedna ramka strony fizycznej może odpowiadać różnym ramką ich stron wirtualnych. Dla jednego procesu może istnieć za każdym razem w innym miejscu.

Współdzielenie pamięci.
Istnieje możliwość współdzielenia tej samej pamięci przez kilka procesów. Ta sama ramka strony fizycznej dla różnych procesów jest odzwierciedlana przez różne ramki wirtualne. Współdzielona strona fizyczna nie musi istnieć w tym samym miejscu w pamięci wirtualnej dla jakiegokolwiek lub wszystkich procesów współdzielących go. A dla różnych procesów w różnych miejscach dla każdego z nich.

Odwzorowanie pamięci.
Odwzorowywaniem, czyli mapowaniem, pamięci nazywamy utworzenie połączenia (odniesienia) pomiędzy stronami wirtualnymi a stronami fizycznymi. Procesor nie pracuje tak naprawdę na zwartości pamięci wirtualnej a na zawartości pamięci fizycznej. Napotykając na adres wirtualny odnosi się do zawartości adresu fizycznego. Proces nie wie, że wykorzystywane przez niego strony znajdują się w pamięci fizycznej, ponieważ zna on tylko swoją przestrzeń adresów wirtualnych, która jest w rzeczywistości jedynie odwzorowaniem pamięci fizycznej.

Translacja adresów.
Wirtualne adresy są przekonwertowywane na fizyczne przez procesor za pomocą jednostki zarządzania pamięcią. Bazuje on na informacjach zawartych w zestawie tablic zachowanych przez system operacyjny. Z adresu wirtualnego odczytuje numer ramki i przesunięcie. Numeru ramki strony wirtualnej używa jako indeksu do tablicy stron procesu. Z tablicy odczytuje znów numer ramki strony, ale już nie wirtualnej a fizycznej, a następnie mnoży go przez rozmiar strony. Otrzymuje dzięki temu adres bazy (czyli początku) strony w fizycznej pamięci. Ostatecznie doda przesunięcie, aby otrzymać szukany adres.

Tablica stron w Linuksie.
Linux używa max. trzech poziomów tablicy stron. Tzn. tablica pierwszego poziomu przetwarza zawartość tablic drugiego poziomu, a każda tablica drugiego poziomu przetwarza zawartość tablic poziomu trzeciego. Procesory

Alpha umożliwiają trzypoziomowe stronicowanie, natomiast procesory Intel x86 jedynie dwupoziomowe.

Translacja przy dwupoziomowej tablicy stron.
W takim przypadku, w systemie 32-bitowym, gdzie rozmiar strony wynosi 4 kilobajty i adres wirtualny jest rozmiaru 1 bajt. 12 bitów przeznaczone jest na przesunięcie, a 20 na numery ramek stron (po 10 bitów na poziom stronicowania). W ten sposób możliwe jest dość łatwe i szybkie przetłumaczenie adresu wirtualnego na fizyczny. Dwupoziomowa tablica stron powoduje także iż ilość pamięci wirtualnej w systemie wynieść może nawet 4 gigabajty.

Ochrona pamięci.
Mechanizmy pamięci wirtualnej umożliwiają ochronę przed zapisem wybranych obszarów pamięci. Dzięki czemu nie dochodzi do nadpisania nieswoich obszarów. Każde wejście w tablicy stron zawiera informacje dotyczące kontroli dostępu. Gdy procesor "wchodzi" do tablicy, chcąc przetłumaczyć stronę wirtualną na fizyczną, automatycznie pobiera te informacje i może sprawdzić czy proces wykorzystuje pamięć zgodnie z prawem dostępu. Czyli czy np. nie nadpisuje swoimi danymi kod wykonawczy innego procesu, albo czy nie próbuje wykonywać jakichś danych jako kodu. Takie działanie powoduje wystąpienie błędu.

Wymiana (Swapping).
Jeżeli proces potrzebuje sprowadzić stronę wirtualną do pamięci fizycznej i tam nie ma wolnej dostępnej strony fizycznej system operacyjny musi zrobić wolną przestrzeń, dla tej strony, poprzez odrzucenie innej strony z pamięci fizycznej. Jeżeli wirtualna strona, będąca kandydatem do odrzucenia pochodzi z obrazu lub pliku danych i nie została zmodyfikowana od czasu swojego pojawienia się do tej chwili, to można ją bezpiecznie usunąć, ponieważ może być w każdej chwili odtworzona. Jeśli jednak została zmodyfikowana Linux musi zachować jej zawartość. Robi to zachowując ją na dysku twardym w tzw. pliku wymiany. Stronę taką nazywamy tzw. brudną stroną (ang. d*irty page*).
Plik wymiany jest specjalnie wydzielonym obszarem dysku twardego, na którym zapisane są wirtualne strony nie mieszczące się w pamięci fizycznej, czyli takie które nie mają przydzielonej ramki strony fizycznej. Dostęp do pliku wymiany w porównaniu z dostępem do pamięci fizycznej jest bardzo długi. Dlatego to Linux musi decydować czy powinien przenieść daną stronę na dysk i pozbawić się szybkiej do niej dostępu, czy może zachować ją w pamięci fizycznej i przenieść inną.

Algorytm wymiany
Jest to algorytm zapewniający sprawną wymianę odpowiednich stron, w taki sposób, że system operacyjny nie musi zbyt często sięgać do stron zapisanych w pliku wymiany, gdy w tym czasie inne nieużywane przez dłuższy czas strony zalegają niepotrzebnie w pamięci fizycznej.
Niewydajny algorytm powoduje szamotanie (ang. *thrashing*). Te same strony są ciągle zapisywane i czytane z pliku wymiany. W ten sposób system operacyjny traci cenny czas na operacje, które nie przynoszą widocznych rezultatów, a jedynie hamują wykonywanie procesów. Strony, których stale potrzebuje dany proces nazywane są jego zestawem roboczym (ang. w*orking set*). Aby proces wykonywał się szybko nie powinny one znajdować się w pliku wymiany a w pamięci fizycznej.
Linux używa algorytmu Najmniejszy Ostatnio Użyty (ang. Least Recently Used - LRU). Wprowadza konieczność zapisywania wieku każdej strony. A działa on w taki sposób, że do pliku wymiany przenoszone są strony najstarsze, natomiast najmłodsze zachowywane są w pamięci fizycznej. Ponieważ najmłodsze strony są aktualnie wykorzystywane dlatego przysługuje im pierwszeństwo w dostępie do pamięci fizycznej. W ten sposób jeśli tylko dana strona używana jest wystarczająco często może nigdy nie zostać z niej przeniesiona. Natomiast strony najstarsze, prawdopodobnie używane najrzadziej, są jako pierwsze usuwane do pliku wymiany. Algorytm ten bierze także pod uwagę rodzaj strony. Jeśli jest to brudna strona musi ją zapisać w pliku wymiany. Jeśli nie może poprostu skasować. Dlatego też, aby oszczędzić sobie długiego zapisywania na dysku brudne strony odrzuca w ostateczności.

Błędy stron.
Błąd strony to brak możliwości zrealizowania przez CPU zadania bezpośredniego dostępu do pamięci fizycznej. Błędy stron są zgłaszane przez MMU w postaci przerwań w pewnych ściśle określonych sytuacjach. Nie jest to błąd w sensie niepoprawnie działającego procesu, a jedynie w sensie braku możliwości zrealizowania przez procesor zadania bezpośredniego dostępu do pamięci fizycznej.

Powody dla z powodu których występuje błąd stron:
- ***brak strony*** - Błąd ten pojawia się w momencie, gdy żądamy dostępu do strony pamięci wirtualnej, której

zawartość nie istnieje w pamięci fizycznej adresowanej przez procesor. Informacja czy strona istnieje w pamięci fizycznej czy nie znajduje się w tablicy stron procesu.
Funkcja odpowiedzialną za obsługą przerwania braku strony jest:
void do_no_page() - Funkcja ta sprawdza informacje dotyczące bieżącego procesu, adresu który spowodował błąd oraz flagę informującą czy dane miały być pisane czy czytane.

Przyczyny błędu "brak strony":

- *is_present* - strona znajduje się w pamięci, lecz jest przydzielona innemu procesowi. Wtedy uaktualnione zostają dane dotyczące przydziału stron poszczególnym procesom.
- *swap_page* - strona dostępna na urządzeniu wymiany. Czyli brak strony w pamięci fizycznej i trzeba pobrać ją z pliku wymiany.
- *anonymous_page* - strona niezainicjalizowana. Tzn. stronie wirtualnej, mimo iż istnieje w tabeli stron, nie przydzielono jeszcze strony fizycznej. Robi się to w tym momencie.
- *sig_bus* - strona niezidentyfikowana. Czyli odwołano się do strony która nie istnieje w tabeli stron. Wysyłany jest sygnał SIGBUS.

- ***naruszenie praw ochrony strony*** *- z* tym błędem mamy do czynienia w przypadku gdy zapis danych do strony pamięci jest zabroniony, bądź zapis ten wiąże się z koniecznością wykonania dodatkowych czynności na poziomie programowym. Informacja o tym znajduje się w tablicy stron procesu.
  Funkcją odpowiedzialną za obsługę błędu ochrony strony jest:
  void do_wp_page()

  Naruszenie ochrony zapisu na stronie może nastąpić gdy:
  - proces próbuje pisać do własnej przestrzeni kodu, chcąc go zmienić.
  - proces próbuje pisać do specjalnie zdefiniowanej przestrzeni danych tylko do odczytu. Chociażby próbuje zmienić stałą globalną.
  - proces próbuje pisać do strony pamięci dzielonej z innym procesem, nie mając przy tym praw zapisu do tej strony.

Przykładem wykonania dodatkowych czynności na poziomie programowym jest przypadek, gdy strona pamięci może nie być dzielona, a jednocześnie może w pewnych okolicznościach zawiera dane wspólne dla kilku procesów. Wtedy strona taka zostanie skopiowana na użytek procesu, który spowodował błąd ochrony i dopiero prywatna kopia będzie mogła zostać zmodyfikowana. Jest to mechanizm dość często wykorzystywany, ponieważ umożliwia szybkie kopiowanie stron.

Wpływ błędów stron na wydajność systemu.
Błędy stron potrafią w znacznym stopniu spowolnić system. Powodem tego jest zajmowanie się obsługą błędów stron, zamiast rzeczywistymi czynnościami, przynoszącymi jakieś wymierne rezultaty i postęp w wykonywaniu procesów.

V. SYSTEM PLIKÓW NTFS

NTFS (New Technology File System) został stworzony specjalnie dla systemów operacyjnych Nowej Technologii. Jest zaawansowanym system plików przystosowanym do pracy z bardzo dużą ilością informacji, dlatego jest skierowany do wymagających użytkowników, którzy potrzebują bardzo szybki dostęp do dużej ilości małych plików jak i do fragmentów plików o znacznym rozmiarze. NTFS min. cechują takie funkcje jak kompresja, szyfrowanie, określanie list dostępu do danych oraz przydziały. System plików NT ma wbudowany system ochrony przed uszkodzeniem danych (np. na skutek błędów wynikłych podczas pracy dysku) o nazwie journaling, który jest charakterystyczny dla nowoczesnych systemów operacyjnych.

Nowe rozwiązania w stosunku do FAT32
• Przypisanie praw dostępu do poszczególnych plików oraz folderów, które umożliwiają wyszczególnienie, kto ma, jakiego rodzaju prawo dostępu do danego pliku lub folderu. NTFS oferuje więcej możliwości nakładania praw dostępu oraz pozwala na ustalenie praw dla poszczególnych użytkowników lub grup użytkowników.
• Odzyskiwanie dla NTFS jest zaprojektowane na tak wysokim poziomie, iż bardzo rzadko użytkownik będzie potrzebował użyć narzędzi do naprawy dysku na woluminie NTFS.W przypadku awarii i zawieszenia się systemu, system plików NTFS stosuje plik wydarzeń (log file) oraz informacje z punktów kontrolnych w celu automatycznego odzyskania spójności systemu plików.
• Struktura drzew B powoduje, ze dostęp do plików w folderach bardziej obszernych jest szybszy niż dostęp do

plików w folderach o podobnej wielkości na woluminie FAT.
• Istnieje możliwość kompresji poszczególnych plików oraz folderów na woluminie FAT. Kompresja NTFS umożliwia odczyt oraz zapis plików podczas dokonywania kompresji, bez potrzeby uprzedniego uszycia programu do dekompresji. To czy plik jest kompresowany czy nie ustala się za pomocą atrybutu. Warunkiem możliwości kompresji jest rozmiar klastra, który musi być większy niż 4KB.
• Obsługa długich nazw plików.
• Przechowywanie informacji o ostatnim czasie dostępu (FAT przechowuje informacje tylko o dacie).
• Obsługa do 2^64bajtów (16 exabajtów).

Budowa wewnętrzna
Podstawową jednostką systemu NTFS jest wolumin. Wolumin jest tworzony przez program administrowania dyskiem systemu NT; u jego podstaw leży logiczny podział dysku. Wolumin może zajmować część dysku lub cały dysk, może też rozciągać się na kilka dysków. System NTFS nie ma do czynienia z poszczególnymi sektorami dysku. Zamiast nich używa klastrów. System NTFS używa w charakterze adresów dyskowych logicznych numerów klastrów (LCN). Przypisuje je poprzez ponumerowanie klastrów od początku dysku do jego końca. Za pomocą tego schematu system może wyliczyć fizyczną odległość na dysku (w bajtach), mnożąc numer LCN przez wielkość klastra. Plik w systemie NTFS nie jest zwyczajnym strumieniem bajtów, lecz jest obiektem strukturalnym złożonym z atrybutów. Każdy atrybut jest niezależnym strumieniem bajtów, który podlega tworzeniu , usuwaniu, itp.. Niektóre atrybuty są standardowe dla wszystkich plików, wliczając w to nazwę pliku, czas jego utworzenia, itp. Większość tradycyjnych plików danych ma beznazwowy atrybut danych, mieszczący wszystkie dane pliku. Każdy plik w systemie NTFS jest opisany przez jeden lub więcej rekordów przechowywanych w specjalnym pliku o nazwie główna tablica plików (master file table - MFT). Rozmiar rekordu jest określony podczas tworzenia systemu plików i waha się w granicach od 1 do 4 KB. Małe atrybuty przechowuje się w samym rekordzie MFT i nazywa rezydentnymi. Wielkie atrybuty, takie jak nienazwana masa danych - określone mianem nierezydentnych - są przechowywane w jednym lub większej liczbie ciągłych rozszerzeń na dysku, do których wskaźniki przechowuje się w rekordzie MFT. W przypadku małych plików w rekordzie MFT może się zmieścić nawet atrybut danych. Jeżeli plik ma wiele atrybutów lub jeśli jest on mocno pofragmentowany i wymaga zapamiętania wielu wskaźników pokazujących wszystkie jej części, to jeden rekord w tablicy MFT może okazać się za mały. W tym przypadku plik jest opisany przez rekord o nazwie: podstawowy rekord pliku (base file record), który zawiera wskaźniki do rekordów nadmiarowych, przechowujących pozostałe wskaźniki i atrybuty. Każdy plik w woluminie systemu NTFS ma niepowtarzalny identyfikator zwany odsyłaczem do pliku. Odsyłacz do pliku jest wielkością 64-bitową, składającą się z 48-bitowego numeru pliku i 16-bitowego numeru kolejnego. Numer pliku jest numerem rekordu w strukturze MFT opisującej plik. Numer kolejny jest zwiększany za każdym razem, gdy następuje powtórne użycie w tablicy MFT. Zwiększenie to umożliwia systemowi NTFS wykonywanie wewnętrznej kontroli spójności - na przykład wyłapywanie nieaktualnych odwołań do usuniętego pliku po użyciu wpisu MFT na nowy plik. Każdy katalog stosuje strukturę danych zwaną B+-drzewem, w którym zapamiętuje indeks swoich nazw plików. Każdy wpis w katalogu zawiera nazwę pliku i odsyłacz do niego oraz kopię znacznika czasu uaktualnienia i rozmiaru pliku - pobranej z atrybutów pliku rezydujących w tablicy MFT. Kopie tych informacji są przechowywane w katalogu, co przyspiesza wyprowadzanie jego zawartości - nazwy wszystkich plików , i ich rozmiary i czasy uaktualnień są obecne w samym katalogu, więc nie potrzeba ich zbierać na podstawie wpisów w tablicy MFT każdego z plików. Wszystkie metadane woluminu systemu NTFS są przechowywane w plikach. Pierwszym z takich plików jest jest tablica MFT. Drugi plik, używany do działań naprawczych w przypadku uszkodzenia tablicy MFT, zawiera kopię pierwszych szesnastu pozycji tablicy MFT. Oprócz tego istnieje jeszcze kilka innych specjalnych plików.

Zalety NTFS
Formatowanie woluminów Windows 2000 za pomocą systemu plików NTFS zamiast FAT umożliwia korzystanie z zaawansowanych funkcji, do których zaliczają się:
- Odzyskiwanie systemu plików. Użytkownik rzadko jest zmuszony do uruchamiania programu naprawczego. NTFS gwarantuje spójność woluminu używając standardowych technik rejestrowania transakcji i odzyskiwania danych. W przypadku awarii systemu NTFS korzysta ze swojego rejestru w celu automatycznego przywrócenia spójności systemu plików.
- Kompresja woluminów, folderów i plików. Pliki, które są kompresowane w woluminie NTFS, mogą być odczytywane i zapisywane przez dowolną aplikację systemu Windows bez uprzedniej dekompresji za pomocą oddzielnego programu – dekompresja następuje automatycznie podczas otwierania pliku. Plik jest ponownie kompresowany po zamknięciu lub zapisaniu.
- Dostępność wszystkich funkcji systemu Windows 2000 związanych z systemami plików.
- Brak ograniczeń ilości wpisów w katalogu głównym.

- System Windows 2000 może formatować woluminy NTFS o rozmiarze do 2 terabajtów.
- Efektywniejsze zarządzanie przestrzenią dyskową niż w systemie FAT, dzięki użyciu klastrów o mniejszym rozmiarze (4 KB dla woluminów o rozmiarze do 2 terabajtów).
- Kopiowanie sektora rozruchowego do sektora na końcu woluminu.
- Minimalizacja ilości dostępów do dysku, wymaganych do odnalezienia pliku.
- Przydzielanie uprawnień do udziałów, folderów i plików. Określają one, jakie grupy i użytkownicy mają dostęp do tych obiektów oraz w jakim stopniu. Uprawnienia plików i folderów w systemie NTFS są stosowane dla użytkowników pracujących lokalnie na komputerze oraz korzystających z niego przez sieć. Można także określić uprawnienia udziałów, stosowane podczas pracy w sieci w kombinacji z uprawnieniami plików i folderów.
- Rodzimy dla NTFS system szyfrowania, EFS, korzystający z kluczy symetrycznych w połączeniu z technologią kluczy publicznych. Umożliwia to zabezpieczenie zawartości plików przed niepowołanym dostępem.
- Punkty specjalne, umożliwiające stosowanie nowych technologii, takich jak punkty instalacji woluminów.
- Przydziały dyskowe, które mogą być używane do ograniczania ilości przestrzeni dyskowej wykorzystywanej przez użytkowników.
- Dziennik zmian, umożliwiający śledzenie zmian dokonywanych w systemie plików.
- Śledzenie łączy, zapewniające integralność skrótów oraz łączy OLE.
- Obsługa plików rozrzedzonych, dzięki której bardzo duże pliki mogą być zapisywane przy wykorzystaniu jedynie niewielkiej przestrzeni dyskowej.

Wady NTFS
Mimo że NTFS jest zalecany dla większości użytkowników Windows 2000, to w niektórych sytuacjach nie jest odpowiednim systemem plików. Do wad systemu NTFS zaliczają się:
- Woluminy NTFS nie są dostępne z poziomu systemów MS-DOS, Windows 95 i Windows 98. Zaawansowane funkcje NTFS wprowadzone w systemie Windows 2000 nie są dostępne w systemie Windows NT.
- W przypadku bardzo małych woluminów, zawierających w większości małe pliki, zarządzanie NTFS może spowodować niewielki spadek wydajności w porównaniu do systemu FAT.

Narzędzia systemu plików w Windows 2000
- Cacls - wyświetla i modyfikuje listy kontroli dostępu NTFS
- Cipher - wyświetla lub zmienia ustawienia szyfrowania plików lub folderów
- Compact - kompresuje i dekompresuje pliki i foldery NTFS
- Compress - kompresuje pliki lub foldery
- Convert - konwertuje wolumin FAT do NTFS
- DirUse - skanuje folder i informuje o wykorzystaniu miejsca na dysku
- Efsinfo - wyświetla informacje o zaszyfrowanych plikach i folderach
- Expand - dekompresuje skompresowane pliki

Mountvol - wyświetla, tworzy i usuwa punkty instalacji woluminów

VI. SYSTEM PLIKÓW EXT3FS

Trzeci Rozszerzony System Plików wywodzi się bezpośrednio z Ext2. Został stworzony przez Stephen'a Tweedie z firmy RedHat. Wersja alfa pojawiła się w 1999 roku. Podstawową różnicą w stosunku do poprzednika jest mechanizm księgowania (journaling), który wprowadzono w celu skrócenia czasu _wstawania_ systemu po awarii. Dodatkowo, wprowadzono na stałe kilka usprawnień, które były wcześniej dostępne jako łaty dla systemu Ext2.

Cechy systemu Ext3
Spośród dodatkowych cech w stosunku do poprzedniej wersji warto wymienić następujące możliwości systemu Ext3:
- Journaling - mechanizm księgowania zwiększa bezpieczeństwo systemu (niepodzielność operacji), ale przede wszystkim skraca do minimum czas sprawdzania systemu plików po awarii.
- Indeksowane katalogi - znacznie zwiększają wydajność systemu przy dużej ilości plików.
- Zapis synchroniczny - w najnowszych wersjach systemu Ext3 (jądro 2.4.19) działa ponad 10 razy szybciej od wersji z Ext2 (dotyczy zapisu do zwykłych plików).

Kompatybilność z Ext2
Celem autora systemu Ext3 było usprawnienie Ext2, a nie tworzenie zupełnie nowego systemu. Dało to

następujące korzyści:
- Możliwość montowania systemu Ext3 jako Ext2 (o ile system został poprawnie odmontowany); w chwili obecnej nie jest możliwe montowanie systemu Ext2 jako systemu Ext3, ale ma to być możliwe w przyszłości
- Możliwość korzystania z wielu sprawdzonych programów narzędziowych (m.in. fsck)
- Możliwość łatwej i bezpiecznej migracji z Ext2 (nie trzeba nawet odmontowywać systemu Ext2 podczas konwersji do Ext3) przy pomocy programu tune
- Przewidywalność, stabilność i uniknięcie wielu błędów
- Zaufanie użytkowników

Struktura fizyczna i logiczna Ext3
Nie wprowadzono wielu zmian w stosunku do Ext2. Do najistotniejszych należy wprowadzenie pól związanych z obsługą księgowania.
Należy zauważyć, że mechanizm księgowania jest niezależny od systemu Ext3.

Superblok
Oto wprowadzone pola związane z obsługą księgowania:
- s_journal_inum - numer i-węzła pliku księgującego
- s_journal_dev - urządzenie, na którym znajduje się ten plik
- s_last_orphan - początek listy z i-węzłami do usunięcia

Struktura ext3_dir_entry2
Struktura ext3_dir_entry_2 różni się od struktury ext3_dir_entry tym, że w systemie Ext3 pole name_len jest liczbą 8 bitową (więc długość nazwy pliku nie może przekroczyć 255) a pozostałe 8 bitów jest wykorzystywane do przechowywania pola file_type.
Warto zauważyć, że w systemie Ext3 istnieje również starsza wersja struktury dir_entry, a w nowszych wersjach systemu Ext2 występuje również nowa wersja tej struktury.
Dodatkowe pole umożliwia nam szybszy dostęp do informacji o typie pliku (nie musimy pobierać i-węzła). Z drugiej strony, zmniejszenie name_len do 8 bitów uniemożliwia korzystanie z bardzo długich nazw plików (co było teoretycznie możliwe w Ext2), choć nie jest to specjalna strata.

Indeksowane katalogi
W systemie Ext3 wprowadzono możliwość wydajniejszej obsługi katalogów zawierających dużo plików. W tym celu zastąpiono strukturę listową stosowaną do tej pory przez mechanizm oparty na haszowaniu i strukturach drzewiastych. Do włączenia mechanizmu indeksowania katalogów służy flaga EXT3_INDEX_FL i-węzła.Uwaga: opis dla bloków o wielkości 4096 bajtów.

Struktura indeksująca
System rezerwuje pierwsze 512 bloków (do zerowego do 511.) katalogu dla przetrzymywania struktury indeksującej. Pozostałe bloki-liście (tzn. od 512.) to zwykłe bloki z wpisami, więc mogą być obsługiwane przez standardową funkcję ext3_readdir.
Blok zerowy jest korzeniem struktury drzewiastej i przechowuje na pierwszych 8 bajtach nagłówek (długość nagłówka, typ indeksowania, wersja, głębokość w drzewie). Pozostała część korzenia jest traktowana jak każdy blok indeksujący.
Blok indeksujący jest złożony z 512 wpisów postaci: (klucz, adres), gdzie klucz jest wartością funkcji haszującej (na ostatnim bicie jest znacznik kolizji mówiący, że kolidujące klucze zostały rozbite pomiędzy dwa bloki) a adres to logiczny adres odpowiedniego bloku lub kolejnej struktury indeksującej w przypadku struktur wielopoziomowych.
W ten sposób można obsłużyć około 90000 plików w katalogu. Jeśli to nie wystarcza, można dołożyć kolejne poziomy - dla następnego otrzymujemy już około 50 milionów wpisów.

Szukanie
Przeszukiwanie katalogu przebiega w następujący sposób:
- obliczenie klucza na podstawie nazwy pliku, służy do tego funkcja int ext3fs_dirhash(const char *name, int len, struct dx_hash_info *hinfo)
- przeczytanie korzenia struktury indeksującej
- przeszukiwanie (binarne lub liniowe w starszych wersjach) indeksów, aż do najniższego poziomu
- przeczytanie liścia i postępowanie jak w wersji ze strukturą listową
- w przypadku kolizji - przeszukiwanie kolejnych bloków indeksujących

Wstawianie
Wstawianie jest bardziej złożone:
- przeszukanie, jak w poprzednim punkcie
- jeśli blok pełny, to rozbicie (następny punkt)
- wstawienie do liścia w standardowy sposób

Rozbijanie
Rozbijanie pełnego węzła (węzła w drzewie indeksującym) polega na przydzieleniu jego kluczy do dwóch bloków. Oczywiście zależy nam, aby w obu nowych blokach było około połowy kluczy z pierwotnego bloku. W tym celu sortujemy klucze, wybieramy środkowy i dokonujemy podziału.

Kolizje
W przypadku kolizji kluczy, staramy się umieścić wpisy w tym samym bloku.Jeśli nie jest to możliwe, to zaznaczamy najmniej znaczący bit w kluczu, co oznacza, że należy szukać w kolejnym bloku.

Polepszenie wydajności
Użycie indeksowanych katalogów potrafi drastycznie zwiększyć wydajność systemu, płacimy jednak 2MB za stworzenie struktury. Dlatego też, indeksowanie należy stosować tylko dla katalogów, w których przewidujemy bardzo dużą liczbę plików.

Nowy sposób indeksowania
W najnowszych, rozwojowych wersjach jądra (oraz w niektórych gałęziach wersji stabilnej) zmieniono opisaną wyżej prostą, drzewiastą strukturę indeksującą na opartą o drzewa czerwono-czarne.
Struktura fname
Struktura fname zawiera węzły drzewa r-b, używane do przechowywania wpisu o katalogu.Ciekawe pola tej struktury to:
- hash czyli klucz (podstawowy)
- minor_hash niższa wartość; umożliwia przechowywanie 64 bitowych kluczy (łącznie z hash, ale obecnie nieużywane ze względu na problemy związane ze współpracą z VFS
- rb_hash węzeł drzewa r-b
- next do łączenia w listę w przypadku kolizji
- inode i-węzeł pliku odpowiadającego wpisowi
- file_type dzięki temu od razu znamy typ pliku (bez zaglądania do i-węzła)

Księgowanie
Mechanizm księgowania był głównym powodem powstania systemu Ext3. Journaling w Ext3 różni się w kilku znaczących szczegółach w porównaniu z podejściami zaprezentowanymi w innych systemach plików.

Journaling Block Device
Mechanizm księgowania jest realizowany niezależnie od systemu plików Ext3. Do właściwego journalingu służy interfejs JBD - rozwijany niezależnie od Ext3 interfejs, umożliwiający dodanie księgowania do dowolnego systemu plików na urządzeniu blokowym. Z poziomu Ext3 obsługa księgowania ogranicza się do wywoływania odpowiednich funkcji JBD. Journal jest przechowywany w i-węźle.
.journal - jeśli system Ext3 powstał z systemu Ext2 bez odmontowywania, to journal będzie widoczny jako plik .journal w korzeniu struktury katalogów.
Interfejs JBD zachowuje w journalu całe fizyczne bloki zamiast pojedynczych bajtów. Zalety i wady takiego postępowania zostaną omówione w następnych punktach.

Plik dziennika
Dziennik dla systemu plików Ext3 może być umieszczony wewnątrz samego systemu (jako zwykły plik), lub na innej partycji.
Do dziennika zapisywane są kompletne zmienione bloki, które mają być zapisane na dysk. Rozwiązanie to zwiększa co prawda rozmiar dziennika i ilość zapisywanych danych, ale jest prostsze do zaimplementowania i wymaga mniej czasu procesora niż zapisywanie tylko zmienionych fragmentów bloków. JBD oznacza transakcję za zakończoną poprzez zapisanie w dzienniku bloku zawierającego specjalne sekwencje - korzysta przy tym z założenia, że dysk zawsze skończy zapisywanie pojedynczego sektora (używając własnego zapasu energii). Istnienie pustego pliku dziennika jest rozszerzeniem "kompatybilnym" systemu Ext2 (tzn. zostaje ono

zignorowane przez poprzednie wersje sterownika). Jednak istnienie niezakończonych transakcji w dzienniku powoduje ustawienie flagi RECOVER, co oznacza, że taki system plików nie zostanie rozpoznany przez zwykły moduł Ext2. Dzięki temu nie da się rozspójnić dziennika poprzez zamontowanie partycji bez odtworzenia informacji z niego.
Warto też zauważyć, że niczym nie grozi ponowna awaria sprzętu w trakcie aplikowania zmian z dziennika - przy ponownym uruchamianiu bloki zostaną nagrane ponownie. Dziennik jest oznaczany jako pusty dopiero po zakończeniu całego odtwarzania.
Istotną kwestią w unikaniu blokad jest przewidywanie ile bloków może zostać zmodyfikowancych w ramach jednej transakcji. Póki w pliku dziennika jest miejsce, JBD łączy równolegle zachodzące modyfikacje w jedną. Jeśli jednak kończy się miejsce, nowe transakcje zostają wstrzymane. Dlatego przy każdym rozpoczęciu transakcji, Ext3 musi określić ile maksymalnie bloków może zostać zmodyfikowanych.

Zalety przechowywania bloków
Podstawową przewagą przechowywania całych bloków nad przechowywaniem modyfikowanych bajtów jest olbrzymie uproszczenie całego systemu. Nie ma potrzeby budowania specjalnych struktur do opisywania modyfikacji, wystarczy przechowywać dane tak samo, jak czynimy to przy zapisywaniu ich na dysk.
Przechowywanie danych w blokach zmniejsza zużycie procesora, ponieważ przesyłanie i zapisywanie bloku nie wymaga przesyłania ani przetwarzania danych w pamięci w celu przygotowania ich do zapisania na dysku.

Wady przechowywania bloków
Do wad takiego postępowania należy zaliczyć duże zużycie pamięci (zapisujemy cały blok nawet przy modyfikacji pojedynczego bitu). Teoretycznie, przechowywanie dużych bloków zamiast pojedynczych bajtów powinno znacznie obniżyć wydajność systemu, ale tak nie jest.
System stara się optymalizować zużycie pamięci poprzez ść iskanie (squish) bloków - sprawdza, czy nie przechowuje już tego samego bloku w związku z poprzednią modyfikacją i zapisuje obydwa tylko raz.

Księgowanie danych
Cechą odróżniającą system Ext3 od innych systemów plików z księgowaniem jest możliwość księgowania nie tylko meta-danych, ale również danych. Oznacza to, że np. o ile reiserfs zachowa prawidłowe struktury systemowe, ale być może utraci dane z pliku, który był modyfikowany w trakcie awarii, o tyle Ext3 (w trybie journal) zarówno naprawi własne struktury, jak i odzyska nasz plik.

Tryby działania
Mechanizm księgowania ma następujące tryby działania (ustalane przy montowaniu systemu plików):
- data=writeback księgowanie tylko meta-danych, podobnie jak w pozostałych systemach plików; teoretyczne, powinno dawać najwyższą wydajność
- data=journal pełne księgowanie (meta-dane i dane); zapisuje bloki dwa razy najpierw do journalu, potem na właściwą pozycję; daje pełne bezpieczeństwo, choć (teoretycznie) kosztem niskiej wydajności; duży obszar przeznaczony na journal znacząco zwiększa wydajność
- data=ordered opcja dodana niedawno; zapisywane są tylko meta-dane, jednak, dzięki specjalnemu postępowaniu, zachowuje się podobnie do pełnego księgowania:

Mechanizm księgowania od wewnątrz
Księgowanie w Ext3 przebiega w następujący sposób:
- Modyfikowane bloki są umieszczane w buforach
- Bufory są łączone w jednostki logiczne zwane transakcjami ; transakcja działa atomowo - modyfikacje dotyczą wszystkich buforów w obrębie transakcji lub żadnego
- Transakcja przechodzi przez punkty kontrolne
- Zakończona(logicznie) transakcja może zostać zatwierdzona (commit) i zapisana do logu, oznacza to, że modyfikacje należy zapisać na dysku
- Zatwierdzona transakcja jest zapisywana (flush) na dysk; w razie awarii wszystkie dane można odzyskać, bo są już zapisane w logu
- Mechanizm księgowania odnotowuje, że transakcja została zakończona (fizycznie)

Wydajność
Teoretycznie, tryb księgowania data=journal powinien mieć najniższą wydajność, jako że dane są zapisywane na dysk dwukrotnie. Jednak w specyficznych warunkach może się okazać inaczej, co pokazał Andrew Morton w interesującym eksperymencie.Test polegał na odczycie 16 MB porcji danych z systemu plików, na który

jednocześnie zapisywane są duże ilości danych. Wyniki zebrane w czasie testów pokazały, że Ext3 wyjątkowo dobrze nadaje się do systemów, w których dane są jednocześnie zapisywane i odczytywane, a wydajność odczytu ma dla nas krytyczne znaczenie. Zbliżone wyniki uzyskamy również, gdy zapisujemy na system plików Ext3 i czytamy jednocześnie z innego systemu plików