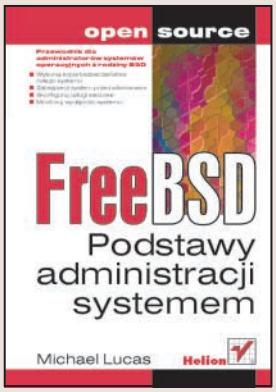
FreeBSD. Podstawy administracji systemem



Autor: Michael Lucas
Tłumaczenie: Marek Pętlicki, Grzegorz Werner
ISBN: 83-7361-604-7
Tytuł oryginału: Absolute BSD
Format: B5, stron: 616

Przewodnik dla administratorów systemów operacyjnych z rodziny BSD

- Wykonaj kopie bezpieczeństwa całego systemu
- Zabezpiecz system przed włamaniami
- Skonfiguruj usługi sieciowe
- Monitoruj wydajność systemu

FreeBSD to drugi obok Linuksa najpopularniejszy system operacyjny dostępny na zasadach open-source. Powstał w oparciu o system BSD, nad którym prace rozpoczęły się w 1976 roku. Zwolennicy systemu zabrali się do pracy nad kolejnymi jego wersjami i wywiązali się z tego zadania wręcz doskonale. FreeBSD stał się systemem wykorzystywanym przez wiele firm z różnych branż. Systemy z rodziny BSD ustanawiają kolejne rekordy wydajności i bezawaryjności pracy, powodując, że skrót BSD stał się synonimem bezpieczeństwa i wysokiej niezawodności w świecie systemów operacyjnych.

Książka „FreeBSD. Podstawy administracji systemem" to podręcznik dla tych, którzy chcą poznać i opanować możliwości tego systemu operacyjnego i sprawnie nim administrować. Autor – współtwórca systemu FreeBSD – dzieli się swoją ogromną wiedzą dotyczącą systemów z rodziny BSD, przedstawiając zasady instalacji, pracy i administrowania nimi.

- Instalacja systemu
- Wyszukiwanie informacji w systemie pomocy
- Tworzenie kopii bezpieczeństwa systemu
- Konfigurowanie jądra pod kątem konkretnych potrzeb
- Aktualizowanie systemu
- Mechanizmy zabezpieczeń
- Instalacja i zarządzanie oprogramowaniem
- Technologia DNS
- Poczta elektroniczna, WWW i usługi sieciowe
- System plików, dyski i macierze dyskowe
- Monitorowanie pracy systemu i kontrola jego wydajności
- Wykorzystywanie systemu FreeBSD w komputerze osobistym

Wydawnictwo Helion
ul. Chopina 6
44-100 Gliwice
tel. (32)230-98-63
e-mail: helion@helion.pl

Spis treści

Rozdział 4.

Zabawy z jądrem

Pierwszym krokiem do optymalizacji systemu FreeBSD jest skonfigurowanie jądra. Osoby niedoświadczone w systemach Unix mogą być nieco zmieszane brzmieniem nazwy: jądro systemu. W końcu jest to tajemna i ciemna strona systemu, gdzie zwykły śmiertelnik nie ma dostępu. W niektórych wersjach systemów Unix, takich jak Solaris, manipulacje w jądrze systemu nie są dostępne użytkownikom końcowym. W systemach Unix z rodziny open source optymalizacja jądra jest najlepszym sposobem zwiększenia wydajności systemu. W innych systemach operacyjnych zapewne również jest to prawda, lecz ich producenci po prostu nie udostępniają takiej możliwości.

Jądro systemu FreeBSD można w szerokim zakresie dostosowywać do własnych potrzeb, nawet podczas pracy. Można prawie dowolnie modyfikować ustawienia jądra mające wpływ na wydajność systemu. Omówimy interfejs opcji *sysctl* oraz ich wykorzystanie do zmiany parametrów pracującego jądra.

Niektórych elementów jądra nie można jednak zmienić podczas jego pracy, inne z nich wymagają bardzo skomplikowanej konfiguracji. Warto również postarać się jak najbardziej ograniczyć rozmiar jądra, usuwając z niego zbędne elementy. Najlepszy sposób dokonania tego polega na skompilowaniu własnej wersji jądra.

Na końcu rozdziału omówimy ładowalne moduły jądra. Są to podsystemy jądra, które można ładować i usuwać z jądra w zależności od potrzeb.

Czym jest jądro?

Istnieje wiele różnych definicji jądra. Wiele z nich trudno zrozumieć. Ja posłużę się prostą, choć może niekompletną definicją. Definicja ta jest wystarczająca dla większości przypadków i jest zrozumiała: *jądro to interfejs pomiędzy sprzętem a oprogramowaniem*.

Jądro pozwala zapisywać dane na dysk i w zasobach sieciowych. Obsługuje procesor oraz operacje na pamięci. Tłumaczy dane dźwiękowe na strumień zer i jedynek, zrozumiałych dla karty dźwiękowej, wskazuje monitorowi, gdzie ma ustawić małe kolorowe kropki. Jądro udostępnia swoje interfejsy programom, które wymagają dostępu do sprzętu.

Zadanie jądra jest jasno zdefiniowane (przynajmniej w naszym uproszczonym ujęciu), lecz wcale nie jest łatwe w realizacji. Różne programy oczekują, że jądro udostępni im rozmaite interfejsy do sprzętu, natomiast różne podzespoły sprzętowe udostępnią swoje zasoby w niejednolity sposób. Jądro musi sobie radzić z tymi wszystkimi szczegółami. Na przykład jądro obsługuje pamięć i może zdarzyć się, że program zażąda alokacji pamięci w sposób nieobsługiwany przez jądro. W takim przypadku program nie będzie działał poprawnie; nie jest to sytuacja teoretyczna, ponieważ programy obsługują pamięć na bardzo różne sposoby. Jeśli natomiast jądro nie potrafi skomunikować się z kartą sieciową, nie będzie można skorzystać z niej w systemie. Sposób, w jaki jądro diagnozuje sprzęt w procesie rozruchowym systemu ma bezpośredni wpływ na działanie tego sprzętu i czasem niezbędna jest ingerencja administratora. Na przykład niektóre karty sieciowe po wysłaniu odpowiedniego sygnału przedstawiają się jądru, natomiast inne w wyniku takiego sygnału potrafią się zawiesić.

Jądro jest tak naprawdę plikiem na dysku o nazwie */kernel*. Moduły jądra, czyli jego podsystemy, które mogą być ładowane i usuwane na żądanie, są zapisane w katalogu */modules*[1]. W czasach popularności sprzętu podłączanego do komputera podczas jego pracy, jak karty PC Card oraz urządzenia USB, dynamicznie ładowane moduły jądra są szczególnie przydatne. Moduły mogą również realizować dodatkowe funkcje, które są rzadko używane i dlatego nie warto uwzględniać ich w standardowej konfiguracji jądra. Wszystkie pliki w systemie, oprócz */kernel* oraz */modules* są niezależne od jądra i jako takie są określane mianem programów użytkownika. Choć są niezależne od jądra, wykorzystują funkcje przez nie udostępniane.

W nowo zainstalowanym systemie można znaleźć plik */kernel.GENERIC*. Jest to standardowo instalowane jądro systemu. W systemach, które działają od dłuższego czasu, można znaleźć sporą liczbę innych jąder. Niektóre z nich są starymi, nieużywanymi jądrami, inne stanowią pozostałość nieudanych eksperymentów. Zespół twórców systemu FreeBSD starał się maksymalnie uprościć procedurę tworzenia i instalacji jądra. Sprawdźmy to.

Konfiguracja jądra

W systemie FreeBSD dostępne są dwa programy do konfiguracji istniejącego jądra: `sysctl`(8) oraz program rozruchowy (ang. *boot loader*).

Mechanizm sysctl

Program `sysctl` pozwala podglądać wartości parametrów jądra, jak również w niektórych przypadkach modyfikować je. Aby nieco zagmatwać sprawę, parametry te są często nazywane opcjami sysctl. Mechanizm sysctl jest bardzo skuteczną funkcją, ponieważ czasami pozwala rozwiązać problemy z wydajnością bez konieczności ponownej kompilacji jądra lub konfiguracji aplikacji. Niestety, te potężne możliwości, błędnie wykorzystane, mogą spowodować pogorszenie wydajności, a nawet problemy z działaniem.

[1] W wersji 5. systemu jądro razem ze swoimi modułami umieszczane jest w katalogu */boot/kernel*

Mechanizm sysctl jest obsługiwany przez program `sysctl`(8). W wielu miejscach tej książki będę zwracał uwagę na użyteczne opcje sysctl, za pomocą których można wpłynąć na zachowanie jądra. Aby jednak skutecznie z nich korzystać, trzeba dowiedzieć się o nich czegoś więcej.

Zanim zaczniemy manipulować opcjami sysctl, zapoznajmy się z ich listą. Opcje sysctl, dostępne w systemie, można sprawdzić za pomocą następującego polecenia:

```
# sysctl -A > sysctl.out
```

Po uruchomieniu powyższego plecenia w pliku *sysctl.out* znajdziemy setki opcji sysctl wraz z ich wartościami. W większości przypadków nazwy tych opcji niewiele mówią początkującemu użytkownikowi, lecz część z nich wydaje się zrozumiała:

```
kern.hostname:  bigbox.blackhelicopters.org
```

Ta konkretna opcja nosi nazwę `kern.hostname` i ma wartość „bigbox.blackhelicopters.org". System, na którym uruchamiałem to polecenie, nosi nazwę właśnie *bigbox.blackhelicopters.org*. Z nazwy opcji można wywnioskować, ze zawiera ona nazwę systemu, jaka wykorzystywana jest przez jądro.

Niektóre opcje sysctl są trudniejsze do rozgryzienia:

```
p1003_1b.memory_protection: 0
```

Jako użytkownik nie mam pojęcia, co oznacza ta wartość. Gdybym jednak miał problemy z jakimś programem, mogę zwrócić się o pomoc do obsługi technicznej programu lub zadać pytanie na liście dyskusyjnej. Jeśli osoba udzielająca pomocy poprosi mnie o to, mogę bez przeszkód odczytać i przekazać mu tę wartość lub zmienić ją w zależności od potrzeb.

Opcje sysctl są zorganizowane w formacie drzewa noszącego nazwę Management Information Base lub MIB. Drzewo to posiada kilka kategorii, jak *net*, *vm* oraz *kern*. Drzewo Management Information Base jest wykorzystywane również w innych zadaniach administracji systemem, o czym przekonamy się w dalszej części książki. Każda z tych kategorii jest podzielona na podkategorie, na przykład kategoria *net* jest podzielona na podkategorie: *IP*, *ICMP*, *TCP*, i *UDP*. Istnieją różne typy drzew MIB — w rozdziale 19. poznamy na przykład drzewo MIB protokołu SNMP. W niniejszym rozdziale skupimy się jednak na drzewie MIB mechanizmu sysctl.

Zapoznaliśmy się już z opcją sysctl `kern.hostname`. Przyglądając się bliżej dostępnym opcjom sysctl, zauważymy, że opcji zawierających przedrostek `kern` jest więcej. Są to ogólne wartości dotyczące jądra. Znajdziemy również sporą liczbę opcji zawierających przedrostek `kern.ipc`, na przykład:

```
kern.ipc.maxsockbuf: 262144
kern.ipc.sockbuf_waste_factor: 8
kern.ipc.somaxconn: 128
...
```

Są to opcje mechanizmu IPC[2] jądra. Drzewo opcji sysctl posiada zatem kilka poziomów rozgałęzień.

[2] IPC jest mechanizmem komunikacji międzyprocesowej (ang. *interprocess communication*). Różne programy wykorzystują ten mechanizm. Jeśli program wymaga modyfikacji tych opcji jądra, informacje na ten temat znajdziemy w jego dokumentacji.

Na końcu każdej z gałęzi znajdują się indywidualne opcje, jak `net.inet.raw.recvspace`. Każda opcja opisuje cechę jądra, taką jak bufor czy ustawienie. Zmieniając wartości, mamy wpływ na działanie jądra. Na przykład niektóre opcje sysctl określają parametry buforów pamięci biorących udział w operacjach sieciowych. Gdy wydajność sieci jest niska, można zwiększyć ilość pamięci wykorzystywanej przez ten podsystem jądra. W tabeli 4.1. przedstawiamy wybrane główne gałęzie drzewa MIB opcji sysctl.

Tabela 4.1. *Wybrane gałęzie drzewa MIB opcji sysctl*

Opcja sysctl	Funkcja
`kern`	podstawowe funkcje jądra
`vm`	pamięć wirtualna
`vfs`	systemy plików
`net`	sieć
`debug`	informacje diagnostyczne
`hw`	informacje o sprzęcie
`machdep`	zmienne zależne od platformy (np. Alpha, i386)
`user`	parametry interfejsów użytkownika
`p1003_1b`	parametry opcji POSIX[3]

Każda wartość sysctl może być typu znakowego (*string*), całkowitego (*integer*), logicznego (*binary*) lub specjalnego, zawierającego kod maszynowy zrozumiały dla specjalizowanych programów (*opaque*).

Opcje sysctl nie są dobrze udokumentowane i nie ma pojedynczego dokumentu, w którym wszystkie byłyby omówione. Każda z nich jest omawiana w podręcznikach systemowych funkcji, których bezpośrednio dotyczą. Na przykład opcja sysctl `kern.securelevel` (o której wspominamy szerzej w rozdziale 7.) jest omówiona w podręczniku systemowych programu `init`(8). Wiele opcji sysctl nie ma żadnej dokumentacji. W dodatku A przedstawiam listę najczęściej stosowanych opcji sysctl oraz omawiam ich zastosowanie.

Na szczęście niektóre z tych opcji mają oczywiste znaczenie. Na przykład zaraz na początku listy znajdziemy opcję następującą:

```
kern.bootfile: /kernel
```

Jest ona ważna dla osób, które testują różne jądra, co omówimy szerzej w dalszej części tego rozdziału. Podczas żmudnej procedury usuwania problemów z jądrem zdarza się często, że zapominamy, jakie jądro wykorzystujemy w danej chwili. Nieraz zdarzało mi się szukać przyczyny problemu i jej nie odnaleźć, po czym okazywało się, że mam uruchomioną inną wersję jądra niż ta, w której był problem.

[3] POSIX jest międzynarodowym standardem systemów Unix definiującym funkcje programów i jądra. FreeBSD w większości jest zgodny ze standardem POSIX.

Aby obejrzeć poddrzewo sysctl (na przykład ke9rn), wywołujemy następujące polecenie:

```
# sysctl kern
kern.ostype: FreeBSD
kern.osrelease: 5.0-CURRENT
kern.osrevision: 199506
...
```

Powyżej prezentujemy jedynie kilka wierszy początkowych dosyć pokaźnej listy opcji. Osoby zainteresowane funkcjami sysctl powinny przejrzeć pełną listę i zorientować się w tym, co jest dostępne. Aby odczytać wartość konkretnej opcji, należy podać pełną ścieżkę w drzewie MIB:

```
# sysctl kern.securelevel
kern.securelevel: -1
#
```

W tym przypadku opcja `kern.securelevel` ma wartość `-1`. W rozdziale 7. omówimy dokładnie znaczenie tej opcji.

Modyfikowanie wartości opcji sysctl

Niektóre wartości sysctl są tylko do odczytu. Tego typu wartości znajdują się między innymi w gałęziach MIB hw (sprzęt) oraz machdep (uzależnione od platformy).

```
hw.machine: i386
```

Zmiana tego typu wartości w najlepszym wypadku spowodowałaby zawieszenie systemu, więc system FreeBSD zabezpiecza użytkownika przed tego rodzaju niebezpieczeństwem, blokując możliwość dokonania zmiany. Dzięki temu próba zmiany tej wartości niczego nie popsuje, jedynie pojawi się komunikat, że operacja się nie udała.

Weźmy pod uwagę następującą opcję:

```
vfs.usermount: 0
```

Powyższa opcja, określająca możliwość montowania zasobów przez użytkowników, może być zmieniana. Domyślnie ma wartość 0, co oznacza, że jest wyłączona. Aby ją włączyć wykorzystujemy opcję `-w` [4] polecenia `sysctl`:

```
# sysctl -w vfs.usermount=1
vfs.usermount: 0 -> 1
#
```

Program powoduje wypisanie komunikatu sygnalizującego dokonanie zmiany, informującego o poprzedniej i bieżącej wartości. Zmiana wartości opcji sysctl jest aż tak prostym zadaniem.

[4] W nowszych wersjach FreeBSD opcja `-w` jest zbyteczna — wystarczy podać przypisanie nowej wartości.

Ustawianie opcji sysctl podczas uruchamiania systemu

Wartości opcji sysctl wpisujemy do pliku */etc/sysctl.conf*. Po prostu wpisujemy nazwę opcji oraz pożądaną wartość. Na przykład w celu umożliwienia użytkownikom montowania systemów plików, do pliku */etc/sysctl.conf* dopisujemy następujący wiersz:

```
vfs.usermount=1
```

Konfiguracja jądra za pomocą pliku loader.conf

Niektóre opcje jądra muszą być ustawione przed rozpoczęciem procesu uruchamiania systemu. Na przykład podczas detekcji i konfiguracji urządzeń IDE należy podjąć decyzję o tym, czy ma być aktywna opcja buforowania zapisu (ang. *write caching*). Trzeba to ustalić w początkowej fazie uruchamiania systemu i nie można później tego zmienić. Podobnie w przypadku nowej karty sieciowej może być konieczne załadowanie jej sterowników przed rozpoczęciem procedury rozruchowej. W tych przypadkach pomocny jest mechanizm ładujący jądro.

Mechanizm ten (ang. *loader*) spełnia wiele funkcji — odnajduje dysk twardy, na którym zapisany jest plik jądra, ładuje je do pamięci, inicjuje proces startowy systemu i przekazuje jądru jego parametry pracy. Najważniejszymi informacjami, jakie program ładujący przekazuje jądru, są opcje sysctl, które muszą być ustawione podczas uruchamiania systemu.

Najczęściej stosowany sposób konfiguracji jądra polega na zmodyfikowaniu pliku konfiguracyjnego programu ładującego. Inny sposób polega na wpisaniu opcji w wierszu poleceń programu ładującego. Drugi sposób sprawdza się raczej dla jednorazowych zmian, jednak, biorąc pod uwagę dłuższy okres czasu, lepiej jest wpisać odpowiednie parametry w pliku */boot/loader.conf*.

Program ładujący wykorzystuje dwa pliki konfiguracyjne: */boot/loader.conf* oraz */boot/defaults/loader.conf*. Na razie zajmiemy się tylko plikiem */boot/loader.conf*. Wpisy w pliku */boot/defaults/loader.conf* stanowią domyślne ustawienia systemu i każdy wpis w pliku */boot/loader.conf* przesłania te domyślne wartości.

Plik *loader.conf* zawiera dwa rodzaje informacji: wiadomości niezbędne do załadowania jądra oraz dane pomocnicze dla sterowników urządzeń, które tak naprawdę są opcjami sysctl, ustawianymi w trakcie rozruchu systemu.

W pliku */boot/defaults/loader.conf* znajduje się duża liczba opcji, które mogą okazać się pomocne w różnych sytuacjach, takich jak uruchomienie jądra z pliku innego, niż plik domyślny, lub włączenie diagnostycznego (tzn. powodującego wypisanie dużej ilości komunikatów) trybu uruchomienia systemu. Oto fragment pliku */boot/defaults/loader.conf*:

```
kernel="/kernel"
kernel_options=""
```

```
userconfig_script_load="NO"
userconfig_script_name="/boot/kernel.conf"
userconfig_script_type="userconfig_script"
...
```

Aby zmienić ustawienia domyślne, należy wprowadzić zmienione wersje odpowiednich opcji w pliku */boot/loader.conf*.

W pierwszym wierszu powyższego przykładu znajduje się nazwa pliku jądra (którą poznaliśmy przy okazji omawiania jednej z opcji sysctl). Załóżmy, że pracujemy zdalnie i chcielibyśmy ponownie uruchomić system z innym jądrem, lecz nie chcemy nadpisywać standardowego jądra znajdującego się w pliku */kernel*. Zmieniając nazwę pliku jądra możemy przy następnym uruchomieniu zmusić system do załadowania innego jądra. Ta opcja sysctl może być oczywiście ustawiona wyłącznie podczas rozruchu systemu.

Przeanalizujmy dwa inne przykłady: przekazywania opcji pomocniczych sterownikom urządzeń oraz uaktywniania opcji automatycznego ładowania modułów jądra.

Przekazywanie opcji sterownikom urządzeń

Za pomocą pliku *loader.conf* można przekazywać urządzeniom opcje pomocnicze. Są to właściwie sugestie (ang. *hints*) dla sterowników, ponieważ nie wszystkie sterowniki danego typu obsługują wszystkie opcje. Informacje na ten temat można znaleźć w podręczniku systemowym danego sterownika.

Jak wspomniałem wcześniej, sterownik dysku IDE informacje na temat aktywacji trybu buforowanego zapisu musi otrzymać we wczesnej fazie rozruchu systemu. Szczegóły na ten temat można znaleźć w podręczniku sterownika `ata`(4) oraz w rozdziale 16. Aby włączyć opcję buforowania zapisu, należy ustawić opcję `hw.ata.wc` na wartość `1`. W tym celu w pliku */boot/loader.conf* wpisujemy następujący wiersz:

```
hw.ata.wc="1"
```

Powyższy wpis bardzo przypomina zapis opcji sysctl. Po uruchomieniu systemu sprawdźmy wartość odpowiedniej opcji sysctl, aby upewnić się, że mamy tu do czynienia z tą samą opcją:

```
# sysctl hw.ata.wc
hw.ata.wc: 1
#
```

Jak widać, wszystko się zgadza.

Podczas pracy systemu nie ma możliwości zmiany tej opcji. Można to sprawdzić bez obawy o popsucie czegokolwiek. Niektóre opcje sysctl, przeznaczone tylko do odczytu, można modyfikować podczas rozruchu systemu. Oczywiście, nie da się w ten sposób zmienić procesora Pentium na Alphę, ale w niektórych przypadkach opcje sysctl dają bardzo ciekawe możliwości.

Automatyczne ładowanie modułów jądra

Moduły jądra są elementami, które w razie potrzeby można ładować i usuwać z jądra, gdy już nie są potrzebne. Pozwala to zaoszczędzić nieco pamięci i stanowi znaczne usprawnienie elastyczności systemu.

Automatyczne ładowanie modułu podczas rozruchu systemu jest prostym zadaniem. W domyślnym pliku *loader.conf* znajdziemy sporo przykładów. W skrócie: bierzemy nazwę modułu, rozszerzenie *.ko* zastępujemy tekstem `_load="YES"` i wpisujemy ją do pliku *loader.conf*. Na przykład, aby załadować automatycznie moduł */module/procfs.ko* do pliku *loader.conf* wpisujemy następujący wiersz:

```
procfs_load="YES"
```

Najtrudniej jest określić moduły, które warto załadować. Mniejszy problem jest ze sterownikami urządzeń: gdy do systemu dodajemy kontroler SCSI lub kartę sieciową można dodać odpowiedni moduł, zamiast od nowa kompilować jądro. Decyzja o załadowaniu modułu jądra w celu rozwiązania określonego problemu z reguły jest podejmowana w wyniku przeczytanej dokumentacji lub za czyjąś poradą. Ponadto wiedza na temat modułów przychodzi wraz z doświadczeniem, dzięki zapoznaniu się z odpowiednią literaturą i zdobyciu informacji na temat tego, jakie moduły będą potrzebne do realizacji konkretnych celów. W dalszej części rozdziału przekażę kilka zaleceń dotyczących określonych modułów jądra.

Ręczna konfiguracja programu ładującego

W przypadku częstego uruchamiania systemu, na przykład podczas wprowadzania testów modułów i opcji sysctl, nie warto nanosić za każdym razem zmian w pliku */boot/loader.conf*. Zamiast tego opcje programu ładującego można wprowadzać przed uruchomieniem systemu. Gdy testy zostaną zakończone, optymalne parametry można zmienić w pliku */boot/loader.conf*.

Jak wspominałem wcześniej podczas uruchomienia systemu FreeBSD pojawia się dziesięciosekundowe odliczanie. Jeśli w tym czasie zostanie naciśnięty jakikolwiek klawisz, pojawi (oprócz *Enter*) się wiersz poleceń programu ładującego, w którym można wprowadzić dodatkowe opcje rozruchowe. W wierszu poleceń programu ładującego wyświetlony zostanie następujący wiersz zachęty:

```
ok
```

Program ładujący nie jest Uniksem — to prosty interpreter poleceń napisany w języku Forth[5]. Niektóre polecenia programu ładującego przypominają co prawda polecenia systemu Unix, lecz uczyniono tak wyłącznie dla ułatwienia korzystania z tego narzędzia.

[5] Język Forth jest jednym z nielicznych interpreterów, które zmieszczą się w rekordzie rozruchowym dysku twardego. Podobny program w języku C zająłby więcej miejsca. Od czasu do czasu pojawia się ochotnik obiecujący napisanie programu ładującego w innym języku programowania, jak C, BASIC itp. Po krótkim okresie osoba taka przestaje jednak zdradzać oznaki istnienia.

Polecenia programu ładującego

Po wpisaniu znaku zapytania i naciśnięciu klawisza *Enter* w wierszu poleceń programu ładującego zostanie wyświetlona krótka informacja pomocy. Omówmy najczęściej stosowane polecenia programu ładującego.

ls

Polecenie `ls` powoduje wypisanie nazw plików, podobnie jak w systemie Unix. Domyślnie wypisywana jest zawartość katalogu głównego; w poleceniu tym można również podać pełną ścieżkę katalogu, którego zawartość chcemy poznać.

unload

Polecenie `unload` powoduje wyczyszczenie pamięci z jądra oraz jego modułów zdefiniowanych w pliku *loader.conf*.

load

Oznacza załadowanie pliku do pamięci. Polecenie to służy do załadowania modułu, a nawet samego jądra. Aby załadować jądro, należy najpierw wyczyścić pamięć z poprzedniej zawartości (poleceniem `unload`).

Załadowanie modułu sterownika karty Intel EtherExpress odbywa się w następujący sposób:

```
ok load /modules/if_fxp.ko
ok
```

set

Polecenie `set` pozwala ustawić wartość opcji sysctl. Na przykład buforowanie zapisu na dyskach IDE włączymy w następujący sposób:

```
ok set hw.ata.wc=1
ok
```

Ładowanie i usuwanie modułów w trybie wielu użytkowników

Niektóre moduły jądra nie muszą być ładowane podczas rozruchu systemu i można je ładować i usuwać podczas jego normalnej pracy. Przyjrzyjmy się temu, w jaki sposób odczytać moduły załadowane w systemie i w jaki sposób ładować je i usuwać z pamięci.

Odczyt listy załadowanych modułów

Po uruchomieniu systemu listę załadowanych modułów można odczytać za pomocą polecenia `kldstat`(8):

```
# kldstat
Id Refs Address    Size     Name
 1    5 0xc0100000 2d505c   kernel
 2    1 0xc0c6c000 13000    linux.ko
#
```

W tym systemie (na komputerze klasy laptop) załadowane są w tej chwili dwa moduły: jądro (*kernel*) oraz moduł zgodności z systemem Linux (*linux.ko*, więcej szczegółów w rozdziale 11.). Każdy z tych modułów posiada podmoduły, które można odczytać za pomocą polecenia `kldstat -v` — w takim przypadku należy oczekiwać kilkuset wierszy wyniku.

Ładowanie i usuwanie modułów

Moduł można załadować do pamięci za pomocą polecenia `kldload`(8), natomiast do usuwania modułów służy polecenie `kldunload`(8). Na przykład za pomocą następującego polecenia zostanie załadowany konsolowy wygaszacz ekranu *warp*:

```
# kldload /modules/warp_saver.ko
#
```

Gdy moduł już nie jest potrzebny, można go usunąć:

```
# kldunload warp_saver.ko
#
```

Gdyby wszystkie niezbędne funkcje wkompilować w jądro, przybrałoby ono bardzo duże rozmiary. Dzięki temu, że unikniemy wkompilowania w nie wszystkich modułów, jądro jest stosunkowo niewielkie i przez to wydajniejsze, natomiast niezbędne funkcje można uzyskać poprzez załadowanie modułów do pamięci jądra.

Dostosowanie jądra do potrzeb

Prędzej czy później może się okazać, że za pomocą modułów oraz opcji sysctl nie da się dostosować jądra do potrzeb w takim zakresie, w jakim byłoby to niezbędne. Czasem jedynym rozwiązaniem staje się samodzielne skonfigurowanie jądra. Nie należy jednak obawiać się tego; procedura konfiguracji jądra jest prosta i logiczna. Omówmy ją krok po kroku.

Jądro dostarczone z systemem określa się jako *GENERIC*. Jądro to zostało zaprojektowane w uniwersalny sposób, tak aby mogło współpracować z jak największą ilością sprzętu. Nie oznacza to jednak, że w każdym przypadku jądro takie będzie działać poprawnie, a tym bardziej optymalnie. Jądro *GENERIC* będzie pracować z systemem

486 i nowszymi, lecz nowsze procesory posiadają zaawansowane funkcje i optymalizacje, które pozwalają na wydajniejszą pracę. Jądro *GENERIC* nie wykorzystuje tych funkcji i służy raczej jako rozwiązanie kompromisowe.

Dzięki dostosowaniu jądra możemy poprawić wydajność systemu, jak również możemy uwzględnić w jądrze dodatkowe funkcje oraz obsługę nowego sprzętu, nie uwzględnionego w jądrze podstawowym.

Przygotowanie

Przed rozpoczęciem budowy nowego jądra należy zdobyć jego kod źródłowy. Czytelnicy, którzy zastosowali się do porad z rozdziału 1. posiadają już wszystko w swoim komputerze. W przeciwnym razie muszą ponownie uruchomić instalator i zainstalować źródła jądra lub, korzystając z rozdziału 6., skonfigurować podsystem CVSup.

Osoby, które nie potrafią stwierdzić, czy w systemie zainstalowane zostały kody źródłowe jądra, powinny to sprawdzić w katalogu */usr/src/sys*. Jeśli katalog ten istnieje i znajduje się w nim spora liczba plików i katalogów, oznacza to, że kod źródłowy jądra został zainstalowany.

Przed rozpoczęciem procesu budowy jądra należy sporządzić listę posiadanego sprzętu. Czasem zadanie to bywa niełatwe, ponieważ nazwa producenta i marka podzespołu niewiele mówią o możliwościach tego elementu i jego rzeczywistej konstrukcji. Wiele firm produkowało karty sieciowe kompatybilne z kartą NE2000, czasem nawet karty marki 3Com posiadały układ oznakowany jako ne2000[6]. Ponadto rozmaite firmy, na przykład Linksys, często firmują własną marką urządzenia, które skonstruowane są na bazie zupełnie różnych wzajemnie układów. Na wszystkich pudełkach występuje napis „Linksys", a w każdym może być produkt zbudowany z układów scalonych innych producentów (niejednokrotnie konfigurację urządzenia można „w ciemno" oszacować na podstawie daty wypuszczenia go na rynek).

Na szczęście sprzęt PCI posiada zaawansowane mechanizmy identyfikacji, a FreeBSD dobrze radzi sobie z tego typu urządzeniami, w większości przypadków poprawnie rozpoznając je podczas rozruchu systemu. Jeśli wykorzystujemy starszy system zawierający urządzenia na magistrali ISA konieczne może się okazać uzyskanie dokładnych informacji na temat typu i wersji urządzenia, obsługiwanych przerwań IRQ oraz portów wejścia-wyjścia (I/O).

Najprostszym sposobem identyfikacji posiadanego sprzętu jest sprawdzenie zawartości pliku */var/run/dmesg.boot*, który zawiera bufor komunikatów jądra, czyli sprawdzenie tych wszystkich informacji, które pojawiają się na ekranie podczas uruchamiania się systemu (patrz rozdział 3.). Osoby, które nie znają zawartości pliku *dmesg.boot*, powinny koniecznie się z nim zapoznać w tej chwili. Niejeden użytkownik będzie zaskoczony znajdującą się tam tak dużą ilością wiadomości na temat posiadanego sprzętu.

[6] Umieszczanie takiej etykiety na głównym układzie karty często bywa ponad siły producenta. W końcu znacznie uprościłoby to życie użytkowników, zatem nie warto się tym przejmować.

W wierszach pliku *dmesg.boot*, zawierających informacje na temat sprzętu, pierwszym elementem jest nazwa urządzenia. Każdy podzespół sprzętowy jest identyfikowany w systemie za pomocą własnego pliku urządzenia; jego nazwa składa się z kilku liter, po których następuje liczba, na przykład *npx0*. Litery stanowią nazwę sterownika (*npx*), natomiast liczba określa egzemplarz urządzenia w systemie, rozpoczynając od zera. Jednemu urządzeniu może odpowiadać kilka wierszy w pliku *dmesg.boot*. Gdy w systemie zainstalowanych jest więcej urządzeń tego samego typu, będą miały kolejne numery.

Kopia zapasowa jądra

Nieprawidłowe jądro spowoduje, że system nie da się uruchomić. Z tego powodu bezwzględnie należy zachować kopię działającego jądra. Podczas instalacji jądra system zachowuje jedną jego kopię, lecz bardzo łatwo jest nadpisać starą i sprawną wersję jądra wersją nową i niesprawdzoną.

Co może się stać w przypadku utraty starego jądra? Załóżmy, że kompilowaliśmy nowe jądro, którego nie zdążyliśmy sprawdzić, bo zapomnieliśmy włączyć do niego sterownik karty sieciowej. Musimy ponownie skompilować jądro. Poprzednio skompilowane (niesprawdzone) jądro zostanie podczas instalacji skopiowane na pierwotną kopię jądra (poprawnego), a nowe jądro zostanie ustawione jako domyślne. W ten sposób będziemy posiadać tylko dwa jądra, obydwa o niepewnej jakości. To proste przeoczenie, polegające na utracie działającego jądra, może nie tylko nie pozwolić teraz na uruchomienie systemu, ale również poskutkować większymi problemami.

Utartą nazwą sprawdzonego jądra jest */kernel.good*. Przed rozpoczęciem zabaw z kompilacją jądra skopiujmy więc starą i sprawdzoną jego wersję:

```
# mkdir modules.good
# cp kernel kernel.good
# cp -R modules/* modules.good/
#
```

Nawet posiadanie większej liczby jąder niczym nie grozi. Wiele znanych mi osób umieszcza kopię jądra w katalogu, którego nazwa zawiera datę i w ten sposób kolekcjonuje pełną historię jąder. Jedyną konsekwencją istnienia takiej kolekcji może być zapełnienie partycji głównej sytemu.

Edycja plików jądra

Po wykonaniu kopii zapasowej działającego jądra można rozpocząć konfigurację nowego. Na początek zaglądamy do katalogu */usr/src/sys/i386/conf*, gdzie znajdziemy kilka plików. Najważniejsze dla nas są pliki *GENERIC* oraz *LINT*. Plik *GENERIC* jest plikiem konfiguracyjnym standardowego jądra dostarczanego z instalatorem FreeBSD. Plik *LINT* zawiera wszystkie dostępne opcje jądra wraz z opisami.

Żadnego z plików występujących w kastalogu */usr/src/sys/i386/conf* nie należy edytować bezpośrednio. Zamiast tego należy skopiować plik *GENERIC* pod inną nazwą

i zmiany wprowadzać w kopii, nie w oryginale. Plikowi należy nadać nazwę zgodną z nazwą maszyny (jest to najczęściej stosowana konwencja). Na przykład gdy posiadamy system o nazwie „webserver", plikowi konfiguracyjnemu jego jądra nadamy nazwę:

```
# cp GENERIC WEBSERVER
```

Nowy plik konfiguracyjny otwieramy w dowolnym edytorze. Poniżej przedstawiony jest fragment pliku *GENERIC*, obejmujący konfigurację urządzeń IDE (ATAPI):

```
# urządzenia ATA oraz ATAPI
device          ata0    at isa? port IO_WD1 irq 14
device          ata1    at isa?   port IO_WD2 irq 15
device          ata
device          atadisk           # dyski twarde ATA
device          atapicd           # napędy CDROM ATAPI
device          atapifd           # stacje dyskietek ATAPI
device          atapist           # napędy taśmowe ATAPI
options         ATA_STATIC_ID     # statyczna numeracja urządzeń
```

W każdym wierszu pliku konfiguracyjnego znajduje się informacja konfiguracyjna lub komentarz opisujący daną opcję. Jako znak komentarza służy znak #. Treść od tego znaku do końca wiersza jest ignorowana przez mechanizmy konfigurujące jądro. Komentarze służą wyłącznie do zwiększenia czytelności pliku.

Komentarze mogą rozpoczynać się od dowolnego miejsca w wierszu, często występują po definicji opcji, informując o jej znaczeniu. W wierszach definiujących urządzenia na początku występuje słowo kluczowe `device`. W naszym przykładzie mamy wpisy dla dysków twardych IDE, napędów CD-ROM IDE, napędów dyskietek IDE oraz napędów taśmowych IDE. Znajdziemy również wpisy dla magistrali IDE płyty głównej oraz dla każdego ze złączy IDE.

Wpisy oznaczone słowem kluczowym `options` dotyczą opcji sterowników. Na przykład opcja `ATA_STATIC_ID` powoduje włączenie statycznej numeracji urządzeń. Znaczenie tej opcji poznamy w rozdziale 16. Poznamy również kilka słów kluczowych specjalnego przeznaczenia, jak `pseudo-device` oraz `cpu`. Opcje te dotyczą najczęściej opcji sterowników lub sprzętu.

Jądro *GENERIC* zostało zaprojektowane w taki sposób, żeby mogło działać w jak największej ilości konfiguracji sprzętowych; znajdują się w nim opcje sterowników takich urządzeń, jak karty sieciowe, dyski, kontrolery itp. Na początku konfiguracji własnego jądra należy wyłączyć zbędne elementy, co spowoduje zmniejszenie i uproszczenie jądra. Oczywiście po takim odchudzeniu jądra każda zmiana w konfiguracji sprzętowej wymusza konieczność ponownej kompilacji jądra. Z tego powodu użytkownicy pasjonujący się częstymi zmianami w wykorzystywanym sprzęcie nie będą zainteresowani takim okrojeniem konfiguracji jądra. Z drugiej strony użytkownik wykorzystujący ściśle określoną konfigurację sprzętową, a ponadto często kompilujący jądro, powinien zdecydować się na zmniejszenie jego objętości.

Skopiowana konfiguracja jądra (w naszym przypadku plik *WEBSERVER*) rozpoczyna się od komentarza opisującego przeznaczenie tego pliku oraz zawierającego odwołania do oficjalnej dokumentacji systemu FreeBSD. Po pominięciu komentarzy natrafimy na następujący blok ustawień:

```
machine         i386
cpu             I486_CPU
cpu             I586_CPU
cpu             I686_CPU
ident           GENERIC
maxusers        0
```

machine

Słowo kluczowe `machine` określa architekturę systemu. Nie ma sensu zmieniać tej opcji, chyba, że w systemie x86 budujemy jądro dla systemów Alpha.

cpu

Opcja `cpu` określa funkcje obsługiwane przez procesor. Jest to ważna opcja, ponieważ współczesne procesory znacznie różnią się od siebie w zakresie obsługiwanych funkcji (weźmy pod uwagę choćby procesory Pentium i Pentium MMX).

W tej opcji należy wpisać oznaczenie posiadanego procesora. Gdy nie za bardzo wiemy, co tam wpisać, można posłużyć się zawartością pliku *dmesg.boot*. Plik *dmesg.boot* na moim laptopie zawiera następującą treść:

```
CPU: Pentium III/Pentium III Xeon/Celeron (497.56-MHz 686-class CPU)
  Origin = "GenuineIntel"  Id = 0x681  Stepping = 1

Features=0x383f9ff<FPU,VME,DE,PSE,TSC,MSR,PAE,MCE,CX8,SEP,MTRR,PGE,MCA,CMOV,PAT,PSE
36,MMX,FXSR,SSE>
```

Dla nas w tym momencie najważniejszy jest fragment *686-class CPU* w pierwszym wierszu. Oznacza on, że mogę usunąć wiersze z opcjami `I486_CPU` oraz `I586_CPU`, dzięki czemu budowane przeze mnie jądro będzie szybsze i mniejsze. W wyniku tych modyfikacji jądro będzie zoptymalizowane pod kątem procesorów klasy 686 i zostanie pominięty wolniejszy, uniwersalny kod.

ident

Opcja `ident` określa nazwę jądra, która z reguły jest taka sama, jak nazwa serwera. Gdy budujemy jedno jądro przeznaczone dla większej liczby maszyn, warto nadać mu nazwę najlepiej oddającą ich przeznaczenie, jak na przykład WEBSERVER.

maxusers

Wartość `maxusers` służy do obliczenia różnych wewnętrznych parametrów jądra, jak rozmiary tablic wykorzystywanych w obsłudze połączeń sieciowych oraz liczby jednocześnie otwartych plików.

Począwszy od FreeBSD 4.5 jądro sprawdza ilość dostępnych zasobów i na tej podstawie ustawia wartość opcji `maxusers` w sposób optymalny dla większości zastosowań. Wartość `0` opcji `maxusers` spowoduje wykorzystanie tych domyślnych parametrów. Takie ustawienie skutkuje zadowalającą konfiguracją w przypadku większości systemów. Wartość tę można również ustawić ręcznie.

W systemie FreeBSD 4.4 i wcześniejszych istnieje konieczność ręcznego ustawienia wartości `maxusers`. Zwykle na swoim laptopie z systemem X Window wykorzystuję wartość `maxusers` równą `16`. Jest to dobra wartość podczas pracy na laptopie, ponieważ niezależnie od prac, jakie wykonuję, jestem jedynym użytkownikiem komputera. Z tego powodu nie mam praktycznie możliwości otworzyć większej liczby plików i połączeń sieciowych niż pozwala opcja `maxusers 16`. Na obciążonym serwerze internetowym wartość opcji `maxusers` ustawiam z reguły na `256`. Taka wartość sprawia, że serwer jest w stanie otworzyć tysiące połączeń sieciowych i kontrolować tysiące jednocześnie otwartych plików.

Gdy wartość opcji `maxusers` jest zbyt niska, system nie będzie w stanie obsłużyć plików i połączeń sieciowych. Jądro wychwyci tę sytuację i zacznie wypisywać komunikaty. Na konsoli i w pliku */var/log/messages* pojawią się komunikaty o konieczności zwiększenia wartości opcji `maxusers`.

Nie należy jednak zwiększać wartości tej opcji powyżej 256, chyba że serwer obsługuje miliony jednocześnie otwartych plików lub wiele szerokopasmowych łączy internetowych.

Opcje podstawowe

Po opcji `maxusers` w pliku konfiguracyjnym jądra następuje seria podstawowych opcji jądra. Opcje te definiują szczegóły obsługi TCP/IP, FFS oraz systemu plików. W tej sekcji występują również mniej powszechnie stosowane opcje, z których część można usunąć. Nie będziemy omawiać wszystkich możliwych opcji, lecz omówimy działanie najczęściej wykorzystanych. Szczególny nacisk kładę na opcje typowe dla serwera internetowego.

Przeanalizujmy następujące opcje:

```
options         MATH_EMULATE
```

Niektóre starsze procesory (jak 386, 486SX) nie posiadały koprocesora arytmetycznego. Gdy tego typu procesor jest zamontowany w wykorzystywanym systemie, można pozostawić opcję `MATH_EMULATE`, co spowoduje wkompilowanie w jądro mechanizmów emulacji koprocesora. Wszystkie procesory produkowane od wielu lat posiadają jednak wbudowane koprocesory arytmetyczne i opcja ta w większości przypadków jest zbędna.

```
options         INET
```

Opcja `INET` powoduje umieszczenie w jądrze mechanizmów sieciowych, jak protokołu TCP/IP. Jest to opcja praktycznie niezbędna.

```
options         INET6                   # protokoły komunikacyjne IPv6
```

Opcja niezbędna dla użytkowników IPv6, pozostali powinni ją usunąć.

```
options         FFS
```

Opcja włączająca obsługę systemu plików FFS (Fast Filesystem). Jest to domyślny system plików systemu FreeBSD. Opcję należy pozostawić bez zmian.

```
options         SOFTUPDATES
```

Mechanizm miękkich uaktualnień (ang. *softupdates*) służy zapewnieniu integralności dysku w systemie plików FFS (więcej informacji na temat tego mechanizmu można znaleźć w rozdziale 13.). Opcję `softupdates` należy pozostawić, chyba że świadomie rezygnujemy z jej wykorzystania.

```
options         MD_ROOT
```

Opcję tę stosuje się w stacjach bezdyskowych wykorzystujących MFS. W pozostałych zastosowaniach można ją usunąć.

```
options         NFS
options         NFS_ROOT
```

Powyższe dwie opcje włączają obsługę Network File System. Opcja `NFS_ROOT` daje możliwość uruchomienia sytemu z katalogiem głównym zaimportowanym z serwera NFS. Jest to rzadko spotykana opcja w serwerze internetowym. Opcje te można usunąć, jeśli NFS nie będzie wykorzystany.

```
options         MSDOSFS
```

Opcja `MSDOSFS` włącza obsługę dysków twardych i dyskietek sformatowanych na potrzeby systemu MS DOS, Jest ona przydatna dla użytkowników planujących wykorzystanie dyskietek oraz dysków twardych z systemem plików FAT. Można ją również włączyć do systemu, ładując moduł `msdos.ko`.

```
options         CD9660
```

Opcja włączająca do jądra obsługę standardowych systemów plików dysków CD-ROM. Podobnie jak w przypadku systemu plików MS-DOS obsługę systemu plików ISO-9660 można włączyć, ładując do pamięci moduł `cd9660.ko`.

```
options         PROCFS
options         COMPAT_43
```

Po usunięciu dwóch powyższych wierszy system przestanie działać. Wiele programów jest uzależnionych od funkcji BSD 4.3. Opcja `COMPAT_43` powoduje, że jądro będzie współpracować z BSD 4.3. Od funkcji `PROCFS` uzależniona jest natomiast funkcja monitoringu procesów.

```
options         SCSI_DELAY=15000
```

Opcja `SCSI_DELAY` definiuje czas (w milisekundach) oczekiwania systemu przed wyszukaniem urządzeń SCSI po wykryciu kontrolera SCSI. Opcja ta ma na celu umożliwienie rozpędzenia się napędów. Jeśli w systemie nie są wykorzystywane napędy SCSI, opcje te można usunąć. W przypadku nowszych urządzeń SCSI wartość opcji można zmniejszyć do 5000 (5 sekund) lub mniejszej.

```
options         UCONSOLE
```

Niektóre programy pozwalają użytkownikom podglądać konsole systemową z poziomu terminalu X. Opcja `UCONSOLE` włącza w jądrze obsługę tej funkcji. Gdy w systemie nie jest zainstalowany system X Window, opcję można usunąć.

```
options         USERCONFIG
options         VISUAL_USERCONFIG
```

Opcje pozwalające włączać i wyłączać urządzenia przed załadowaniem jądra. Opcje te z reguły są opcjonalne, lecz warto je pozostawić, ponieważ nie szkodzą, a w niektórych przypadkach mogą okazać się potrzebne.

```
options         KTRACE
```

Opcja KTRACE włącza śledzenie na poziomie jądra. Warto zostawić tę opcję, chyba że ktoś dokładnie wie, co robi, usuwając ją z konfiguracji.

```
options         SYSVSHM
options         SYSVMSG
options         SYSVSEM
```

Opcje uaktywniające komunikację międzyprocesową w stylu System V. Wiele aplikacji jest uzależnionych od tej opcji. Powyższe ustawienia mogą być również załadowane w postaci modułów.

```
options         P1003_1B
options         _KPOSIX_PRIORITY_SCHEDULING
```

Opcje powodujące aktywność funkcji POSIX. Wiele programów zależy od włączenia tych opcji w jądrze.

Wiele procesorów

Gdy system posiada więcej niż jeden procesor, należy koniecznie włączyć następujące opcje konfiguracji jądra:

```
options         SMP                     # obsługa wielu procesorów
options         APIC_IO                 # obsługa funkcji symetrycznego
wejścia-wyjścia APIC (Symmetric APIC I/O)
```

Opcja SMP wskazuje jądru, aby wykorzystywało specjalny kod przeznaczony dla systemów wieloprocesorowych. Opcja APIC_IO włącza funkcje obsługi wejścia-wyjścia w systemach wieloprocesorowych.

Gdy planujemy skompilować jądra wieloprocesorowe, należy usunąć opcje I386_CPU oraz I486_CPU. FreeBSD obsługuje wiele procesorów tylko w tych systemach, które przestrzegają specyfikacji SMP firmy Intel. Specyfikacja ta nie przewiduje wykorzystania procesorów 386 oraz 486.

W przypadku gdy w systemie wykorzystywany jest pojedynczy procesor, należy zablokować te opcje.

Urządzenia

Po opcjach ogólnych w pliku konfiguracyjnym jądra występują wpisy dotyczące urządzeń pogrupowanych w sposób intuicyjny według typów.

Magistrale

Na początku występują wpisy dotyczące różnego typu magistral, jak `device pci` czy też `device isa`. Wpisy te należy pozostawić, chyba że dany typ magistrali nie występuje w systemie. Może się zdarzyć, że nowoczesny system posiada magistralę ISA, choć mogłoby się wydawać inaczej. Na przykład mój nowoczesny laptop posiada magistralę ISA wykorzystywaną wewnętrznie w systemie. W większości przypadków można bez obaw usunąć wpis dotyczący magistrali EISA, która nie jest spotykana w nowszych komputerach.

Interfejsy

W kolejnej sekcji występują interfejsy IDE/ATAPI. Nawet gdy w danej chwili nie planujemy wykorzystywać dysków twardych tego typu, a płyta główna wyposażona jest w złącza IDE, warto w konfiguracji jądra pozostawić te wpisy. Można natomiast usunąć wpisy dotyczące typów urządzeń IDE, z których nie planujemy korzystać.

Dalej następują wpisy dotyczące kontrolerów i kart SCSI oraz wykorzystywanych funkcji SCSI. W tej grupie znajdują się również wpisy dotyczące urządzeń Zip na złączu równoległym oraz USB. Jeśli nie planujemy stosować tych urządzeń, można usunąć całą sekcję. Jeśli SCSI będzie używane, warto usunąć wpisy dotyczące niewykorzystanych urządzeń.

```
# kontrolery SCSI
device          ahb                     # rodzina EISA AHA1742
device          ahc                     # urządzenia AHA2940 oraz wbudowane
                                          w płytę główną AIC7xxx
...
```

Po sekcji SCSI znajdziemy urządzenia ogólnego przeznaczenia, jak klawiatury, monitory, port PS/2 itp. Należy je oczywiście pozostawić.

W dalszej części pliku znajdują się wpisy kart sieciowych. Jest to dosyć długa lista, podobna do sekcji SCSI i IDE. Jeśli przez dłuższy czas nie planujemy wymiany karty sieciowej, warto usunąć wpisy dotyczące niepotrzebnych kart. Można w każdym razie usunąć wpisy odnoszące się do kart stosowanych przy złączach ISA, jeśli system takich złączy nie posiada.

Pseudourządzenia

W pobliżu końca pliku konfiguracyjnego jądra *GENERIC* znajdziemy listę pseudourządzeń. Jak może sugerować nazwa, urządzenia te nie występują w postaci sprzętowej, lecz ich funkcje są realizowane przez program. Jednym z takich urządzeń jest urządzenie pseudoterminalu wykorzystywanego podczas zdalnych sesji (za pomocą usług telnet lub SSH, patrz rozdział 13.). Urządzenie pseudoterminalu traktuje zdalne połączenie w podobny sposób, w jaki obsługiwany jest monitor i klawiatura przyłączone lokalnie do systemu. Jądro traktuje mechanizmy tego typu podobnie, jak traktuje urządzenia fizyczne, przylgnął więc do nich termin pseudourządzenia. Przykładem takiego urządzenia może być:

```
pseudo-device       loop
```

Jest to urządzenie pętli zwrotnej (ang. *loopback*) `lo0`. To urządzenie sieciowe jest „przyłączone" zawsze do systemu, w którym się znajduje. Po usunięciu tego urządzenia wiele programów przestanie działać. Choć taki eksperyment może być bardzo pouczający, na produkcyjnym systemie jednak nie jest wskazany.

```
pseudo-device      ether
```

To pseudourządzenie obsługuje niezależne od sprzętu mechanizmy protokołu Ethernet. Należy je pozostawić.

```
pseudo-device      sl
```

Pseudourządzenie `sl` obsługuje protokół SLIP (*Serial Line Internet Protocol*). Jest to starszy protokół, wykorzystywany w połączeniach modemowych, wyparty przez PPP (*Point-to-Point Protocol*). Prawdopodobnie nie jest potrzebne, chyba że jest wymagane przez dostawcę usług internetowych.

```
pseudo-device      ppp        1
```

Pseudourządzenie `ppp` implementuje w jądrze obsługę protokołu PPP. Jest to rozwiązanie, które przestało być preferowane na korzyść implementacji PPP w przestrzeni użytkownika (*userland*). W większości przypadków nie jest potrzebne.

Liczba na końcu wiersza określa liczbę pseudourządzeń PPP, które będą utworzone przez jądro.

```
pseudo-device      tun
```

Pseudourządzenie `tun` obsługuje pseudourządzenie logicznego tunelu komunikacyjnego. Mechanizm ten jest wykorzystywany przez programy w celu przekazywania danych bezpośrednio do jądra. To pseudourządzenie wykorzystywane jest między innymi przez mechanizmy PPP działające w przestrzeni użytkownika, wspomniane wyżej.

```
pseudo-device      pty
```

Pseudoterminal wykorzystywany do zdalnego logowania się do systemu za pomocą usług SSH itp. Należy pozostawić.

```
pseudo-device      md
```

Pseudourządzenie `md` obsługuje tzw. dyski w pamięci. Użytkownicy nie planujący wykorzystania tej funkcji systemu mogą usunąć ten wpis. W większości serwerów internetowych dyski w pamięci są po prostu marnowaniem pamięci. Jednak niektóre specjalizowane serwery (jak anonimowe serwery CVS) wykorzystują ten mechanizm.

```
pseudo-device      gif
pseudo-device      faith
pseudo-device      bpf
```

Pseudourządzenie `bpf` (Berkley Packet Filter) to filtr, który służy do kontrolowania pakietów wchodzących do systemu i do obsługi mechanizmu DHCP. Gdy wspomniane mechanizmy nie są wykorzystywane, można bez przeszkód usunąć te opcje.

Urządzenia USB

Po bloku pseudourządzeń znajdują się wpisy dotyczące urządzeń USB. Wpisy te dotyczą modułów ładowanych dynamicznie w razie potrzeby za pomocą mechanizmu `kldload`. Większość serwerów internetowych nie wykorzystuje urządzeń USB, w takich przypadkach można wyłączyć ich obsługę w jądrze.

Kompilacja jądra

W poprzednich punktach omówiliśmy podstawową konfigurację jądra. Przed wprowadzeniem bardziej zaawansowanych zmian warto na próbę skompilować jądro, aby sprawdzić, czy na tym etapie kompilacja przebiegnie poprawnie.

> W tym podrozdziale opisane są zmiany w jądrze przeprowadzane bez uaktualniania systemu. W przypadku instalowania nowej wersji na starej należy posłużyć się nieco inną procedurą, opisaną w rozdziale 6.

Po wprowadzeniu zmian w opcjach jądra należy je skompilować. W tym celu wykonujemy polecenie `config`(8), które sprawdzi poprawność składni zmodyfikowanego pliku konfiguracyjnego i wygeneruje niezbędne pliki konfiguracyjne. Uruchamiamy polecenie `config` z wykorzystaniem pliku konfiguracyjnego `MYKERNEL`:

```
# config MYKERNEL
Kernel build directory is ../../compile/MYKERNEL
Don't forget to do a ''make depend''
#
```

Za pomocą polecenia `config` nie sprawdzimy, czy konfiguracja jądra została przeprowadzona poprawnie, otrzymamy jedynie szereg komunikatów sygnalizujących istnienie błędów konfiguracyjnych. Stanie się tak na przykład w przypadku zastosowania nieistniejącej opcji jądra. Za pomocą programu `config` otrzymamy również informację o tym, że należy uruchomić polecenie `make depend`. Na razie nie omawialiśmy tego zagadnienia, należy jednak pamiętać, że pominięcie tego etapu stanowi najczęściej popełniany błąd podczas samodzielnej kompilacji jądra systemu.

Niektóre komunikaty wypisywane przez program `config` są bardzo czytelne. Na przykład można usunąć opcje obsługi systemu plików UFS, lecz pozostawić konfigurację uruchamiania systemu z systemu plików UFS. Wiedząc, że druga z opcji wymaga zastosowania pierwszej można z łatwością odgadnąć przyczynę błędu. Inne komunikaty mogą być jednak mniej zrozumiałe i czasem konieczna jest ich dogłębna analiza, jak to opisałem w rozdziale 2.

Gdy polecenie `config` zostanie wykonane bez błędów, na ekranie pojawi się katalog zawierający elementy konfiguracyjne jądra. W powyższym przykładzie jest to *../../compile/MYKERNEL*. Należy przejść do tego katalogu i wykonać następujące polecenie:

```
# make depend && make all install
```

Polecenie to składa się z dwóch części. Pierwsza z nich, polecenie `make depend`, powoduje sprawdzenie, czy w systemie znajdują się wszystkie elementy niezbędne do konfiguracji jądra i modułów, a następnie połączenie ich w jedną całość. Drugie z poleceń, `make all install`, służy do skompilowania jądra z wykorzystaniem kodu źródłowego oraz zależności zbudowanych w poprzednim poleceniu.

Ten etap, w zależności od mocy procesora, może trwać dość długo. Na procesorze 486 25 MHz trwa kilka godzin, natomiast na dwuprocesorowym Pentium 1 GHz trwa kilka minut. Na ekranie będą się przesuwać niezrozumiałe komunikaty kompilatora. Na końcu nastąpi faza instalacji, w której stare jądro zostanie zapisane w pliku */kernel.old* a nowe jądro — w pliku */kernel*.

Po zakończeniu procesu kompilacji należy ponownie uruchomić komputer i uważnie obserwować komunikaty startowe. Na początku komunikatów pojawi się informacja na temat katalogu, w którym kompilowane było jądro. W poniższym przykładzie informacja ta została wyróżnione pogrubieniem:

```
Copyright (c) 1992-2001 The FreeBSD Project.
Copyright (c) 1979, 1980, 1983, 1986, 1988, 1989, 1991, 1992, 1993, 1994
        The Regents of the University of California. All rights reserved.
FreeBSD 5.0-CURRENT #0:  Sun May 20 16:49:05 EDT 2001
   mwlucas@turtledawn.blackhelicopters.org:/usr/src/sys/compile/MYKERNEL.
...
```

Gdy pojawi się powyższy komunikat, mamy pewność, że kompilacja powiodła się. Czas wypróbować nowe jądro.

Problemy z kompilacją jądra

Gdy kompilacja jądra się nie uda, należy przede wszystkim sprawdzić ostatni wiersz komunikatów. Na podstawie komunikatów można często odgadnąć przyczynę problemu, lecz czasem jest to niemożliwe i wtedy należy zwrócić się o pomoc do osób, które znają kod jądra lepiej niż własną kieszeń.

Poniżej przedstawiamy przykład błędu kompilacji jądra:

```
===> sys/modules/xl
cc -O -pipe  -D_KERNEL -Wall -Wredundant-decls -Wnested-externs -Wstrict-prototy
pes  -Wmissing-prototypes -Wpointer-arith -Winline -Wcast-qual  -fformat-extensi
ons -ansi -DKLD_MODULE -nostdinc -I-  -I. -Ig -I@/../include  -mpreferred-stack-
boundary=2 -c /usr/src/sys/modules/xl/../../pci/if_xl.c
❶ /usr/src/sys/modules/xl/../../pci/if_xl.c:155: syntax error before`<'
cpp: output pipe has been closed
*** Error code 1

Stop in /usr/src/sys/modules/xl.
*** Error code 1

Stop in /usr/src/sys/modules.
*** Error code 1

Stop in /usr/src/sys.
*** Error code 1
```

```
Stop in /usr/src.
*** Error code 1

Stop in /usr/src.
*** Error code 1
```

Na początku komunikatu mamy informację, że problem wystąpił wówczas, gdy kompilator znajdował się w katalogu *sys/modules/xl* i kompilował znajdujący się tam moduł jądra. Widzimy wykonywane polecenie; jest zapisane w kilku kolejnych wierszach i rozpoczyna się od `cc -0`. Polecenie jest tak naprawdę zapisane w jednym wierszu i po prostu zajmuje kilka wierszy na ekranie. Polecenie kończy się przed wierszem oznaczonym znakiem ❶.

Potem następuje wiersz (❶), w którym znajduje się informacja o błędzie (`syntax error before'<'`) oraz wiersz, w którym wystąpił błąd. Jest to błąd, który spowodował zatrzymanie kompilacji. W dalszej części listingu widzimy serię błędów. Moduł jądra nie może być skompilowany, co zatrzymuje kompilację innych modułów (zależnych). Sytuacja ta uniemożliwia również ukończenie kompilacji jądra i w konsekwencji sprowadza się do porażki całego przedsięwzięcia.

Na szczęście zanim cokolwiek zostanie zainstalowane konieczne jest pomyślne zakończenie kompilacji wszystkich bez wyjątku elementów jądra. Dzięki temu nie ma obawy o uszkodzenie systemu za pomocą instalacji niekompletnego środowiska jądra. W przypadku niepomyślnej kompilacji istnieje możliwość poprawienia środowiska i ponownej kompilacji.

Wiemy, w którym momencie kompilacji pojawił się błąd (znamy całe polecenie, którego realizacja zakończyła się niepowodzeniem). Wiemy też, co to za błąd i w którym miejscu w pliku wystąpił (błąd składni przed znakiem < w pliku */usr/src/sys/modules/xl/../../pci/if_xl.c* w wierszu l55). Błędy, które są sygnalizowane w dalszej części komunikatu nie mają znaczenia, ponieważ wynikają wprost z tego pierwszego.

Nie należy martwić się, jeśli komunikaty błędów okażą się niezrozumiałe. Większość użytkowników ma takie wrażenie. Należy po prostu zastosować się do porad z rozdziału 2. i poszukać pomocy w internecie. Na początek należy sprawdzić archiwa listy dyskusyjnej FreeBSD-questions. W pole tekstowe wyszukiwarki wklejamy komunikat błędu z nazwą pliku (`if_xl.c:l55: syntax error before'<'`) i sprawdzamy, czy podobny problem mieli inni użytkownicy. Gdy nie uda się znaleźć żadnych podpowiedzi, warto spróbować wyszukać następny wiersz komunikatu (`cpp: output pipe has been closed`).

Jeśli nie uda się znaleźć pomocy w archiwach listy dyskusyjnej, można wysłać pytanie na adres *FreeBSD-questions@FreeBSD.org*. W liście należy zawrzeć następujące informacje:

- końcowy fragment komunikatów z błędnej kompilacji;
- informacje o wersji wykorzystywanego systemu FreeBSD;
- zawartość pliku */var/run/dmesg.boot*;
- wynik działania polecenia `uname -a`;
- plik konfiguracyjny kompilowanego jądra.

Być może problem ten łatwo rozwiązać i po uzyskaniu tych informacji znajdzie się ktoś, kto podsunie jakieś sugestie rozwiązania. Błędy tego typu występują z reguły w wyniku niewłaściwej konfiguracji jądra.

Uruchamianie systemu z nowym jądrem

Co należy zrobić, gdy nowe jądro nie do końca działa poprawnie? Na przykład zapomnieliśmy skonfigurować kompilację pseudourządzenia *ppp* i nie możemy połączyć się z internetem za pomocą modemu. Nie należy wpadać w popłoch, ponieważ nic nie jest stracone. Podczas uruchamiania systemu należy przerwać proces w sposób opisany w podrozdziale „Ręczna konfiguracja programu ładującego". Gdy pojawi się wiersz poleceń programu ładującego, należy usunąć z pamięci stare jądro:

```
ok unload
ok
```

W tym momencie program ładujący powinien znajdować się w katalogu głównym systemu plików. Możemy przypomnieć nazwy posiadanych jąder, wywołując polecenie `ls`, które spowoduje wypisanie zawartości katalogu /.

Następnie wybieramy jądro, którym chcemy uruchomić system. Należy pamiętać, aby załadować również wszystkie niezbędne moduły.

```
ok load /kernel.good
ok load /modules/if_fxp.ko
ok boot
```

Po wykonaniu tych poleceń powinien uruchomić się system z wybranym jądrem.

Jeśli nie wykonaliśmy kopii poprawnego jądra i wszystkie posiadane jądra są błędne, nadal nie ma powodu do niepokoju. Instalator systemu FreeBSD umieszcza standardowe jądro w pliku */kernel.GENERIC*. To jądro powinno dać możliwość uruchomienia systemu z wierszem poleceń lub przynajmniej w trybie jednego użytkownika.

Dodawanie funkcji do jądra

Jeśli wszystko odbyło się poprawnie, w tym momencie powinniśmy mieć do dyspozycji jądro z minimalną funkcjonalnością. Nadszedł czas, aby je dostosować do własnych potrzeb.

LINT

Pełną listę opcji jądra i sterowników wraz z ich dokumentacją można znaleźć w pliku */usr/src/sys/i386/conf/LINT*.

Jeśli w standardowym jądrze nie jest obsługiwane jakieś urządzenie, należy zajrzeć do pliku *LINT*. Niektóre z obsługiwanych opcji są nieco toporne, lecz użytkownicy nietypowego sprzętu z pewnością je docenią. Na przykład system FreeBSD pozwala uaktywnić specjalne funkcje procesora IBM BlueLightning, dzięki czemu posiadacze sprzętu tego typu mogą wykorzystać go w szerszym zakresie.

Typowy fragment pliku *LINT* wygląda następująco:

```
# Opcja CPU_PPRO2CELERON włącza obsługę cache drugiego poziomu w procesorach
Celeron
# Mendocino. Opcja ta jest użyteczna dla użytkowników przejściówek z gniazda Socket
8
# do gniazda Socket 370 ponieważ większość BIOS-ów dla Pentium Pro nie uaktywnia
# pamięci cache drugiego poziomu w procesorach Celeron Mendocino.

options         CPU_PPRO2CELERON
```

Dzięki tej opcji możemy uaktywnić pewną opcję sprzętową w niestandardowej konfiguracji (wykorzystanie przejściówki pozwalającej zainstalować procesor nowszego typu w płycie głównej starszego typu). FreeBSD pracuje również ze starszym sprzętem, więc zastosowanie mogą znaleźć także takie nietypowe opcje. Ta konfiguracja nie jest jednak na tyle powszechnie stosowana, aby usankcjonować włączenie opcji `CPU_PPRO2CELERON` w standardowym jądrze. Nie zaszkodzi jednak wiedzieć, gdzie należy szukać tej opcji, gdyby okazała się potrzebna. Warto przejrzeć plik *LINT* choćby po to, by nabrać orientacji w dostępnych opcjach.

Dlaczego jednak nie są włączone wszystkie opcje dostępne w pliku *LINT*? Wiele z opcji zapisanych w tym pliku koliduje z innymi. Na przykład opcja `CPU_PPRO2CELERON` informuje jądro, że w systemie wykorzystywana jest płyta główna do procesorów Pentium Pro z procesorem Celeron. Opcja `CPU_RSTK_EN` włącza obsługę stosu powrotu w procesorach Cyrix 5x86. Nie ma jednak procesorów Celeron produkowanych przez firmę Cyrix, a nawet gdyby istniały, nie wykorzystywałyby wspomnianej wyżej przejściówki do płyt do procesorów Pentium Pro.

Wykorzystanie opcji do poprawiania błędów

Niektóre z opcji można wykorzystać do eliminacji błędów. Jeden z moich znajomych posiada kilka serwerów WWW zainstalowanych na starych komputerach PC. Gdy jeden z nich był zmuszony do obsługi kilkuset wywołań na sekundę, zaczął generować następujące komunikaty o błędach:

```
Jun    9 16:23:17 ralph/kernel: pmap_collect: collecting pv entries -
suggest increasing PMAP_SHPGPERPROC
```

Administrator zignorował te komunikaty i wkrótce system załamał się. Zostałem poproszony o pomoc. Na podstawie sugestii z komunikatu przestudiowałem dokładnie plik *LINT* i znalazłem następujący fragment:

```
# Liczba pozycji PV na proces. Zwiększenie tej wartości może pomóc w przypadku błędów
# wynikających z intensywnego wykorzystania pamięci współdzielonej. Ustawienie zbyt
# dużej wartości w połączeniu ze dużym rozmiarem pamięci fizycznej w systemie może
```

```
# wywołać błędy podczas rozruchu systemu spowodowane wyczerpaniem
# pamięci wirtualnej.
#
# Zwiększenie tej wartości powinno iść w parze z modyfikacją opcji sysctl
# "vm.v_free_min", "vm.v_free_reserved" oraz "vm.v_free_target".
#
# poniższa wartość jest większa o 1 od wartości domyślnej.

options         PMAP_SHPGPERPROC=201
```

Po przeczytaniu tych informacji zdecydowaliśmy się na rozwiązanie. Zapisaliśmy kopię jądra w pliku */boot/kernel.pmap-crash*. Nie było to, co prawda, poprawne jądro, lecz warto zachować kopię działającego, choć niedoskonałego jądra na wypadek, gdyby nowe jądro miało okazać się gorsze. Następnie zwiększyliśmy wartość opcji `PMAP_SHPGPERPROC` do 400 i zwiększyliśmy ilość pamięci systemowej do 192 MB. Komputer, na którym było konfigurowane jądro, staruszek z 64 MB pamięci RAM, obsługiwał kilkaset wywołań stron WWW na sekundę. Po skonfigurowaniu i kompilacji jądra komunikaty o błędach ustały zupełnie.

Bez możliwości dostosowania szczegółowych ustawień jądra nie byłoby innego wyjścia, jak tylko kupić nowy serwer. Oczywiście w tym konkretnym przypadku mieliśmy do czynienia ze sprzętem z bardzo niskiej półki. Jeśli jednak można było przedłużyć życie tego sprzętu za pomocą prostej modyfikacji konfiguracji, dlaczego z tego nie skorzystać? Gdyby jednak ktoś czuł silną potrzebę wydawania pieniędzy, może po prostu przesyłać je do mnie.

Poprawianie wydajności jądra

A może poprawić wydajność?

Najpoważniejszym wąskim gardłem systemu są bufory sieciowe mbuf. W rozdziale 5. omówimy więcej szczegółów na ich temat; przyjmijmy teraz, że są to fragmenty pamięci wykorzystywane do obsługi połączeń sieciowych. Każde połączenie może wykorzystywać większą ilość buforów mbuf.

Liczba buforów mbuf zmienia się wskutek modyfikacji opcji jądra `maxusers`, omówionej wcześniej. W przypadku serwera produkcyjnego, dla którego spodziewamy się sporego obciążenia, należy dodatkowo zmodyfikować te parametry. Dostosowanie opcji mbuf dzięki opcji `maxusers` często pomaga, lecz omówione niżej zmiany w konfiguracji są dosyć często stosowaną opcją konfiguracyjną.

Opcja `NMBCLUSTERS` kontroluje liczbę buforów mbuf wykorzystywanych przez jądro. Ta opcja nie występuje w standardowym pliku konfiguracyjnym jądra, trzeba ją tam dopisać; występuje natomiast w pliku *LINT*.

```
options         NMBCLUSTERS=1024
```

Klastry pamięci mbuf są alokowane w pamięci na stałe, nie można więc po prostu zwiększyć tej wartości do miliona, ponieważ pamięć zajęta nie będzie dostępna dla innych funkcji systemowych, jak otwieranie plików i obsługa serwera WWW.

Jeden *nmbcluster* zajmuje około 2 kB pamięci, więc przykładowe ustawienie spowoduje zaalokowanie 2 MB pamięci na mechanizmy obsługi sieci. W nowoczesnym komputerze nie jest to dużo, lecz w przypadku komputerów klasy 486, na których można znaleźć instalacje FreeBSD, jest to już poważny rozmiar pamięci. Po co jednak zmieniać tę wartość?

Aby obliczyć liczbę buforów mbuf, należy sprawdzić liczbę jednocześnie otwartych połączeń w okresach największego obciążenia. W tym celu posłużymy się poleceniem `netstat`(1). Polecenie to służy do wypisania liczby aktywnych w danej chwili operacji sieciowych, wliczając w to połączenia TCP, UDP, połączenia na interfejsie pętli zwrotnej oraz połączenia za pomocą gniazd uniksowych. Nas interesują tylko połączenia TCP i UDP, więc posłużymy się poleceniem `grep`(1), aby odrzucić zbędne informacje. Po odfiltrowaniu wynik prześlemy do polecenia `wc`(1), które zliczy liczbę wierszy wyniku, co w konsekwencji spowoduje wypisanie liczby połączeń TCP i UDP[7]. Oto przykład takich wywołań:

```
# netstat -na | grep tcp | wc -l
     427
# netstat -na | grep udp | wc -l
     377
#
```

Aby sprawdzić liczbę wykorzystywanych w danej chwili buforów mbuf, należy wykonać polecenie `netstat -m`. Program netstat omówimy nieco szerzej w rozdziale 5.

W naszym przykładzie w danej chwili w systemie było aktywnych 427 połączeń TCP oraz 377 UDP. W sumie daje to około 800 połączeń. Aby wziąć pod uwagę okresowe szczyty obciążenia, należy wynik zmierzony w okresie typowego obciążenia pomnożyć przez dwa.

Gdy wiemy już, ile połączeń przyjdzie nam obsłużyć, możemy określić przypuszczalną ilość pamięci wymaganej do ich obsługi. Każde połączenie TCP wymaga wykorzystania bufora wysyłki i bufora odbioru. Ich rozmiar (w bitach) można odczytać z opcji sysctl `net.inet.tcp.sendspace` oraz `net.inet.tcp.recvspace`:

```
# sysctl net.inet.tcp.sendspace
net.inet.tcp.sendspace: 16384
# sysctl net.inet.tcp.recvspace
net.inet.tcp.recvspace: 16384
```

[7] Na początku książki napisałem, że Unix jest czymś na kształt języka mówionego. W tym miejscu mamy doskonały tego przykład: z elementów w postaci poleceń systemowych składamy większe polecenie wykonujące określone działanie. Osoba postronna może uznać, że administratorzy systemów Unix są ponadprzeciętnie inteligentni. W większości jesteśmy jednak po prostu leniwi na sposób kreatywny.

Trudno jest pracować z bajtami, przeliczmy to więc na kilobajty: 16 384 dzielone przez 1024 daje 16, zatem każdy bufor zajmuje w systemie 16 kB. Należy jednak sprawdzić to w wykorzystywanym systemie, ponieważ na przykład pomiędzy FreeBSD w wersjach 4.3 i 4.4 nastąpiła zmiana domyślnego rozmiaru bufora. Biorąc pod uwagę 2 bufory na połączenie, otrzymujemy wynik 32 kB.

Każde połączenie UDP również wymaga zastosowania bufora. W tym przypadku nie ma większych możliwości manipulacji, dla naszych potrzeb możemy jednak założyć, że połączenia UDP wymagają podobnych ilości pamięci jak połączenia TCP.

Wiemy więc, że na każde połączenie potrzebujemy 32 kB pamięci i że w szczycie będziemy mieli do czynienia z około 800 połączeniami. 800 × 32 kB = 25 600 kB, czyli około 25 MB. Aby wziąć pod uwagę nagłe skoki obciążenia, pomnóżmy to przez dwa, co w wyniku da 50 MB.

Każdy klaster mbuf ma rozmiar 2 kB, 1024 klastry mają rozmiar 2 MB. Potrzebujemy 50 MB na bufory mbuf, więc dzielimy 50 MB przez 1024, po czym dzielimy to przez 2 i otrzymujemy 25 600 klastrów mbuf. Zatem opcję `NMBCLUSTERS` ustawiamy na wartość `25600`:

```
options         NMBCLUSTERS=25600
```

Konfigurując serwer sieciowy, warto przeznaczyć dla opcji `NMBCLUSTERS` około jedną czwartą posiadanej pamięci RAM. Na przykład w przypadku pamięci 128 MB na bufory mbuf przeznaczamy 32 MB pamięci, czyli omawianą opcję ustawiamy następująco: `NMBCLUSTERS = 16384`. Być może to za mało lub za dużo, lecz na początek stanowi dobre oszacowanie.

Współdzielenie jąder

Gdy mamy do dyspozycji większą liczbę identycznych komputerów, które planujemy używać jako serwery, nie ma potrzeby na każdym z nich konfigurować i kompilować jądra. Można wykorzystać jedno jądro skonfigurowane na potrzeby wszystkich komputerów. Jądro jest w końcu po prostu plikiem binarnym, który można skopiować.

Aby wykorzystać tę możliwość, należy skompilować jądro i porządnie je przetestować. Następnie należy skopiować plik */kernel* oraz katalog */modules* na wszystkie pozostałe serwery. Przed umieszczeniem takiego wspólnego jądra na każdym z serwerów należy oczywiście wykonać kopię oryginalnego jądra. Teraz wystarczy ponownie uruchomić komputer i gotowe.