Systemy pakietów w środowisku UNIX

Tomasz Luchowski <zuntum@netbsd.org>

21 maja 2003

1 Wstęp

Systemy pakietów zostały stworzone w celu ułatwienia instalalacji dodatkowych aplikacji - umożliwiają zarówno łatwą kompilację programu ze źródeł jak i użycie prekompilowanego pakietu binarnego. Powodują one, że zainstalowanie aplikacji jest niezwykle proste i sprowadza się praktycznie tylko do wydania jednej komendy.

1.1 Historia

Pierwszy system pakietów w BSD pojawił się we FreeBSD w sierpniu 1994 roku - znany jest pod nazwą *ports* [FreeBSD-ports]. Początkowo miał on spore braki - nie obsługiwał zależności ani nie ściągał automatycznie plików źródłowych, jednak prace szybko postępowały - już w styczniu następnego roku dostępnych było ponad 100 przygotowanych i gotowych do zainstalowania aplikacji. [Moolenaar]

System pakietów używany w OpenBSD (także zwany *ports*) [OpenBSD-ports] powstał w czerwcu 1996 roku - wywodzi się on z FreeBSD. Początkowo planowano zintegrować wszystkie dokonane modyfikacje z powrotem do FreeBSD, lecz wraz z upływem czasu oraz coraz bardziej pogłębiającymi się różnicami stawało się to coraz mniej prawdopodobne. OpenBSD modyfikowało swój system pakietów niezależnie od FreeBSD, w tym czasie w obu systemach pojawiły się warte uwagi ulepszenia. Obecnie nie widać już planów połączenia z systemem ports FreeBSD.

NetBSD rozpoczęło prace nad swoim systemem pakietów (zwanym *pkgsrc*) w sierpniu 1997 - on także wywodzi się z systemu *ports* FreeBSD. Wybrano jednak inną nazwę, ponieważ w terminologii NetBSD, *port* oznacza konkretną architekturę (mówi się o portach na różne platformy). NetBSD także dodało wiele własnych ulepszeń, które niestety nie zostały zintegrowane do innych systemów pakietów.

1.2 Podziały...

Stworzenie przez każdy projekt własnego systemu pakietów miało niewątpliwe zalety - każdy projekt mógł zaimplementować ważne dla siebie funkcje oraz dokonywać dowolnych zmian, nie musząc wcześniej uzyskiwać akceptacji oraz prowadzić długich i kontrowersyjnych dyskusji; dzięki temu prace postępowały szybko i sprawnie. Jest jednak oczywiste, że pogłębiająca się coraz bardziej separacja była do uniknięcia - nie było technicznych powodów aby nie synchronizować zmian, bądź wrócić do jednego systemu pakietów, zaspokajającego potrzeby

każdego z projektów. Niestety po pewnym czasie poszczególne systemy pakietów stały się zupełnie niezależne; mimo że czasami następuje pewna wymiana kodu, obecnie mają już inną funkcjonalność i nie wpływają wzajemnie na siebie.

1.3 Wiele systemów pakietów - wiele wad

Mimo że utrzymywanie własnego systemu pakietów niewątpliwie daje dużą wygodę oraz możliwości, ma także ogromną wadę - konieczność duplikowania tej samej pracy. Żeby zdać sobie sprawę z wielkości tego problemu wystarczy przytoczyć orientacyjną liczbę pakietów w poszczególnych systemach: FreeBSD - około 8500, NetBSD - około 3700, OpenBSD - około 1800.

W momencie gdy wydana zostaje nowa wersja programu, odpowiadający mu pakiet musi zostać uaktualniony osobno w trzech różnych kolekcjach; zmiany dokonywane są przez różne osoby, nie w każdym projekcie może znaleźć się do tego ochotnik. Także w przypadku wykrycia ważnego błędu w bezpieczeństwie odpowiednie pakiety także muszą być poprawiane osobno, natomiast właśnie w tym przypadku szybkość działania ma ogromne znaczenie (nawet jeżeli użytkownik próbowałby sam poprawić dany pakiet używając dostępnej łaty (ang. *patch*) zanim jeszcze oficjalny pakiet zostanie uaktualniony, nie zawsze jest to bezproblemowe, gdyż łata ta może konfliktować z ewentualnymi innymi zmianami jakie mogły być dokonane w danym systemie pakietów).

Ponieważ każdy system pakietów rozwijany jest osobno, niestety użyteczne nowinki i ulepszenia także pozostają lokalne i najczęściej nie są implementowane w innych systemach.

Oczywiście oprócz systemów pakietów BSD istnieje także wiele innych, często ciekawych rozwiązań (jak choćby system *portage* używany w Gentoo Linux [Gentoo]).

1.4 Alternatywne rozwiązania oraz próby ujednolicenia

Rozwiązaniem wielu z tych problemów byłoby opracowanie i powszechne stosowanie we wszystkich systemach operacyjnych jednego, wspólnego systemu pakietów. Pomimo różnic w budowie i funkcjonalności poszczególnych systemów operacyjnych nie jest to zadanie niewykonalne - nawet korzystając z tak podstawowych narzędzi jak pliki `Makefile` można zbudować niezwykle elastyczny system pakietów.

Używanie ujednoliconego systemu pakietów pozwoliłoby skupić się na naprawdę ważnych rzeczach, zamiast marnować czas przynajmniej jednej osoby z każdego projektu za każdym razem, kiedy trzeba dodać lub uaktualnić jakiś pakiet. Przy wykorzystaniu pracy nawet wielokrotnie mniejszej liczby osób niż obecnie, możnaby było osiągnąć znacznie lepsze efekty i tym samym podnieść jakość oprogramowania. W momencie wprowadzenia wygodnego ulepszenia do systemu pakietów mogłyby z niego skorzystać wszystkie osoby, a nie tylko te mające szczęście używać tego systemu, w którym zaimplementowana została dana nowość.

Powstało kilka różnych systemów pakietów, które nie są związane z żadnym systemem operacyjnym, lecz chcą być alternatywą, z której można korzystać na dowolnym z systemów - aby wszycy mogli używać właśnie tego systemu pakietów. Żaden z tych projektów nie zdobył większej popularności, lecz warto wspomnieć tutaj przynajmniej o A-A-P [A-A-P], który ma jedną niewątpliwą zaletę - możliwość współistnienia z dotychczas używanymi pakietami. Ma on umożliwiać integrację z obecnie używanym przez użytkownika systemem pakietów (nawet zależności mogą być akceptowane z poprzedniego systemu) i pozwalać na stopniową migrację. Projekt ten jest jednak obecnie we wczesnej fazie rozwoju.

1.5 pkgsrc - przenośny system pakietów

Mimo że *pkgsrc* jest rozwijane przez deweloperów NetBSD oraz jest oficjalnym system pakietów tego systemu, może być z powodzeniem używane także na innych systemach operacyjnych. Obecnie wspierane platformy to *NetBSD, Linux, Solaris, Darwin (MacOS X), FreeBSD, OpenBSD, IRIX*.

1.6 Terminologia

W dalszej części tekstu będę używać terminu *distfiles* - oznacza on wszystkie pliki, które są potrzebne w celu zbudowania danego pakietu - najczęściej jest to skompresowane archiwum ze źródłami oraz ewentualnie dodatkowe dane, np. oddzielna dokumentacja. *prefix* jest to natomiast główny katalog, w którym instalowane prez nas pakiety będą umieszczały swoje pliki (np. `/usr/pkg` bądź `/usr/local`).

2 Wprowadzenie do pkgsrc

2.1 Dlaczego akurat pkgsrc?

- Łatwe budowanie pakietów ze źródeł; po dokonaniu tego możemy jedną komendą sami utworzyć pakiet binarny.
- Pakiety instalowane są w spójnym drzewie katalogów.
- Jeden plik konfiguracyjny, w którym można umieścić wszystkie istotne ustawienia, np. akceptowalne licencje, serwery z których będą ściągane pliki, dodatkowe flagi dla kompilatora oraz wiele innych.
- Dostępne na licencji BSD, dzięki czemu każdy może modyfikować oraz dostosować do swoich potrzeb dowolną część systemu pakietów oraz zrobić z niej dowolny użytek.
- *buildlink* - technologia gwarantująca, że skompilowane pakiety będą zawsze wyglądać identycznie, niezależnie od tego, jakie pakiety mamy zainstalowane w systemie w czasie dokonywania kompilacji.
- Spójna obsługa plików konfiguracyjnych aplikacji - możemy ustalić, w jakiej lokalizacji chcemy je przechowywać; instalowane pakiety automatycznie będą honorować te ustawienia.
- *just-in-time su(1)* - pakiety mogą być kompilowane z konta zwykłego użytkownika, jedynie na czas instalacji potrzebne są dodatkowe przywileje.

2.2 Binaria czy źródła?

pkgsrc jest systemem pakietów umożliwiającym stosowanie prekompilowanych pakietów (binarnych, gotowych do natychmiastowego zainstalowania) bądź też samodzielną kompilację ze źródeł. Należy pamiętać, że pakiety zainstalowane ze źródeł jak i z binariów są rejestrowane w systemie w identyczny sposób - dzięki temu całkowicie dozwolone oraz bezproblemowe jest używanie obu metod jednocześnie. Identycznie wygląda także późniejsze kasowanie pakietów oraz uzyskiwanie informacji o nich - niezależnie od sposobu w jaki zostały zainstalowane.

Użycie pakietu binarnego jest bardzo proste oraz pozwala na szybkie rozpoczęcie korzystania z programu nawet na bardzo wolnym sprzęcie, na którym kompilacja zajęłaby dużo czasu. Możemy ściągnąć pakiet na dysk oraz zainstalować przy użyciu komendy `pkg_add`, bądź też zainstalować bezpośrednio z internetu - zamiast nazwy pliku z pakietem możemy podać miejsce (*URL*) pod którym jest on dostępny. Pakiety binarne nie są jednak dostępne dla wszystkich platform sprzętowo-programowych - oficjalne pakiety budowane oraz udostępniane są tylko dla NetBSD na niektórych architekturach. Zawsze jednak możemy taki pakiet stworzyć sami - gdy już mamy skompilowany dany pakiet ze źródeł, utworzenie binarnego pakietu sprowadza się do wydania komendy `make package`. Po utworzeniu takiego pakietu binarnego możemy go swodobnie używać, np. zainstalować na innej maszynie (o tej samej architekturze i systemie operacyjnym).

Kompilacja ze źródeł daje dużo większe możliwości. Możemy dostosować wszystkie opcje dostępne jeszcze na etapie kompilacji, możemy także skompilować pakiet natychmiast po jego umieszczeniu/zmodyfikowaniu w repozytorium systemu pakietów - nie musimy czekać na udostępnienie pakietu binarnego. Używanie źródeł to także oszczędność - zarówno łącza internetowego jak i czasu. Często autorzy aplikacji, w których wykryto błędy w bezpieczeństwie publikują łaty na kod źródłowy - w takim przypadku możemy po prostu przekompilować aplikację korzystając z już posiadanych źródeł, zamiast ściągać od nowa cały (i zazwyczaj duży) pakiet binarny. Przydatne jest to także przy korzystaniu z kilku systemów operacyjnych - zamiast ściągać binaria dla każdego z tych systemów, wystarczy jeden raz ściągnąć źródło, przy użyciu którego będziemy mogli skompilować pakiet na każdym systemie. Odzwierciedla to obecną sytuację, kiedy czas procesora jest relatywnie tani a moc łatwo może być zwiększona, natomiast łącza internetowe są drogie oraz przeciążone.

Kompilacja ze źródeł może być konieczna w jeszcze jednym przypadku - gdy licencja programu nie pozwala na rozpowszechnianie binariów.

3 Przygotowania

3.1 Gdzie utrzymywane jest pkgsrc

pkgsrc jest utrzymywane w repozytorium CVS projektu NetBSD. Głównym serwerem CVS jest *cvs.netbsd.org*, jednak dla użytkowników końcowych udostępniono serwer *anoncvs.netbsd.org*. Jest on regularnie (gdy wszystko działa zgodnie z planem, po dokonaniu każdej zmiany) synchronizowany z głównym serwerem CVS, zawiera więc zawsze najnowsze wersje plików.

3.2 Używanie CVS

Przykładowe ustawienia umożliwiające dostęp do repozytorium CVS Projektu wyglądają następująco:

```
CVSROOT=anoncvs@anoncvs.netbsd.org:/cvsroot
CVS_RSH=ssh
```

Warto utworzyć w swoim katalogu domowym plik o nazwie `.cvsrc` - możemy w nim umieścić opcje, które zawsze będą używane przy wywołaniu `cvs`.

```
checkout -P
update -d -P
```

```
diff -u
rdiff -u
cvs -z3 -q
```

Szczególnie istotne są opcje "`-P`" oraz "`-d`". Pierwsza z nich powoduje usunięcie pustych katalogów, druga natomiast pozwala na tworzenie w razie potrzeby nowych katalogów. Powinny one być zawsze używane.

W przykładach użycia cvs w dalszej części tekstu zakładam, że zmienne CVSROOT oraz CVS_RSH są ustawione oraz używamy opcji "`-P`" przy "`cvs checkout`" oraz opcji "`-d -P`" przy "`cvs update`". [1]

3.3 Jak pobrać pkgsrc

Istnieje wiele różnych metod pobierania pkgsrc, jednak najłatwiej jest skorzystać z FTP:

```
ftp ftp://ftp.netbsd.org/pub/NetBSD-current/tar_files/pkgsrc.tar.gz
gzip -c -d pkgsrc.tar.gz \
  | (cd /usr; tar -xpf -)
```

bądź CVS:

```
cd /usr
cvs checkout pkgsrc
```

Pamiętajmy, że korzystając z CVS zawsze otrzymamy najnowsze wersje plików, podczas gdy archiwum dostępne na serwerze FTP jest uaktualnianie jedynie co pewien czas.

Nawet jeżeli nie pobraliśmy pkgsrc poprzez CVS (tylko np. rozpakowaliśmy archiwum), nie będzie potem problemów z jego użyciem do uaktualniania pkgsrc - w pliku `.tar.gz` znajdują się bowiem wszystkie niezbędne katalogi używane wewnętrznie przez cvs.

3.4 bootstrap-pkgsrc

Jeżeli chcemy używać pkgsrc na systemie innym niż NetBSD, należy zainstalować pakiet *bootstrap-pkgsrc* (w przypadku NetBSD nie trzeba tego robić, wszystkie niezbędne programy znajdują się już w *base systemie* - możemy pominąć ten punkt). Pakiet ten zawiera wszystkie niezbędne do korzystania z pkgsrc narzędzia. Znajduje się w nim *pkg_install* (zawierające m.in. polecenia `pkg_add`, `pkg_delete`) oraz `bmake`, `digest`, `ftp`, `mtree`, `pax`. Dzięki dostarczeniu wszystkich tych programów wymagania stawiane systemowi, na którym chcemy używać pkgsrc są minimalne - niezwykle łatwe jest dodawanie wsparcia dla kolejnych platform. *bootstrap-pkgsrc* zastąpiło rozwijany poprzednio projekt *Zoularis*.

Dla niektórych platform dostępna [pkgsrc.org] jest prekompilowana wersja *bootstrap-pkgsrc* - wtedy wystarczy tylko rozpakować archiwum do głównego katalogu, np.:

```
gzip -c -d bootstrap-pkgsrc-SunOS-5.9-sparc-20030411.tar.gz \
  | (cd /; tar -xpf -)
```

[1] Oczywiście nie ma potrzeby podawania tych opcji ręcznie - wystarczy umieszczenie ich w pliku `~/.cvsrc`

Jeżeli natomiast chcemy samodzielnie skompilować ten pakiet, bądź też binaria nie są dostępne dla naszej platformy, wystarczy tylko ściągnąć źródła *bootstrap-pkgsrc*:

```
cvs checkout othersrc/bootstrap-pkgsrc
```

Oraz skompilować i zainstalować:

```
cd othersrc/bootstrap-pkgsrc
./bootstrap
```

Jeżeli nie zmienimy domyślnych wartości (w jaki sposób można to zrobić dowiemy się uruchamiając skrypt `bootstrap` z parametrem `--help`), *prefix* zostanie ustawiony na `/usr/pkg`, nastomiast baza danych informacji o pakietach (*pkg_db*) znajdować się będzie w `/var/db/pkg`.

UWAGA: Jeżeli korzystamy z *bootstrap-pkgsrc*, we wszystkich przykładach powinniśmy zastąpić wywołanie "`make`" komendą "`bmake`".

4 pkgsrc w praktyce

4.1 Pakiety binarne

Instalowanie gotowych pakietów binarnych jest niezwykle proste - służy do tego polecenie `pkg_add`. Możemy ściągnąć plik z pakietem na dysk, bądź też instalować pakiety bezpośrednio ze zdalnego serwera:

```
pkg_add mutt-1.4.1.tgz
pkg_add ftp://ftp.netbsd.org/pub/NetBSD/packages/1.6.1/i386/All/screen-3.9.13.tgz
```

Aby przy instalacji pakietów poprzez sieć nie trzeba było za każdym razem podawać pełnego adresu URL, możemy ustawić zmienną środowiskową PKG_PATH, która będzie wskazywać na zdalną lokalizację.

4.2 Kompilacja ze źródeł

Kompilacja pakietu ze źródeł jest niezwykle prosta:

```
zuntum@zunpc:/usr/pkgsrc/misc/screen> make
```

Co się wtedy dzieje?

- Dokonywane jest sprawdzenie, czy wersja pakietu, którą chcemy skompilować, nie jest oznaczona jako posiadająca błędy w bezpieczeństwie (jest to opisane dokładniej w dalszej częsci).
- Ściągnięcie odpowiednich distfiles'ów, jeżeli nie zostały one odnalezione.
- Porównanie sum obliczonych na podstawie ściągniętych plików z sumami zapisanymi w pliku `distinfo`.

- Rozpakowanie źródeł.
- Zaaplikowanie łat.
- Utworzenie odpowiednich linków symbolicznych dla mechanizmu *buildlink* (o tym później).
- Uruchomienie skryptu `configure` pakietu.
- Właściwa kompilacja programu - w większości przypadków jest to uruchomienie komendy `make` w odpowiednim środowisku.

Do zainstalowania pakietu służy polecenie `make install`, jeżeli natomiast chcemy zainstalować pakiet oraz utworzyć z niego pakiet binarny, powinniśmy użyć `make package`. Możemy teraz wyczyścić katalog, w którym dokonywana była kompilacja - `make clean`. Aby wyczyścić katalogi tymczasowe pakietów, które zostały zbudowane jako zależności, używamy `make clean-depends`.

4.3 Wyszukiwanie pakietów

Wygodny interfejs do wyszukiwania pakietów znajduje się pod adresem http://www.pkgsrc.org/ - możemy tam podać część nazwy, słowa kluczowe połączone warunkiem logicznym, bądź też wyrażenie regularne (perl regex).

Jeżeli nie chcemy korzystać z wyszukiwania przez www, możemy po prostu wyszukać katalogi zawierające pewnien wzorzec w nazwie:

```
zuntum@zunpc:/usr/pkgsrc> echo */*screen*
misc/screen sysutils/screentest x11/xscreensaver x11/xscreensaver-gnome
```

W większości przypadków nie mamy problemu z ustaleniem, w którym z wyświetlonych katalogów znajduje się interesujący nas pakiet, lecz w razie wątpliwości możemy poprzeglądać krótkie opisy:

```
zuntum@zunpc:/usr/pkgsrc> grep COMMENT */*screen*/Makefile
misc/screen/Makefile:COMMENT= Multi-screen window manager
sysutils/screentest/Makefile:COMMENT= CRT screen testing utility using GTK+
x11/xscreensaver-gnome/Makefile:COMMENT= Screen saver and locker for the X window system
(with GNOME support)
x11/xscreensaver/Makefile:COMMENT= Screen saver and locker for the X window system
```

bądź też dokładniejsze opisy pakietów, znajdujące się w plikach `DESCR` w poszczególnych katalogach.

Możemy także przeglądać listę wszystkich dostępnych w pkgsrc pakietów poprzez WWW. [pkgsrc-index]

4.4 Uzyskiwanie informacji o pakietach

Do uzyskania różnego typu informacji o zainstalowanych pakietach można użyć komendy `pkg_info(1)` - wywołanie jej bez parametrów wyświetli listę pakietów w naszym systemie, np.:

```
zuntum@zenith:~> pkg_info
digest-20021220     Message digest wrapper utility
mutt-1.4.1          text-based MIME mail client with PGP support
screen-3.9.13       Multi-screen window manager
[...]
```

Można także skorzystać z faktu, że wszystkie informacje o zarejestrowanych pakietach znajdują się w katalogu *pkg_db* i po prostu wyświetlić jego zawartość*:*

```
zuntum@zenith:~> ls -F /var/db/pkg
audit-packages-1.14/           mysql-server-3.23.49nb3/       standalone-tcsh-6.12.00nb1/
bozohttpd-20030313/            nmap-3.20/                     t1lib-1.3.1/
dialog-0.6znb1/                p0f-1.8.2nb1/                  teTeX2-2.0.2/
digest-20021220/               perl-5.6.1nb7/                 teTeX2-bin-2.0.2nb1/
epic4-1.1.6/                   pkgchk-1.34/                   teTeX2-share-2.0.2/
gmake-3.80nb1/                 pkgdb.byfile.db                teTeX2-sharesrc-2.0.2/
ipcalc-0.35/                   pkgdiff-0.108/                 tex2rtf-164/
libiconv-1.8nb1/               png-1.2.5nb2/                  texi2html-1.64nb1/
libtool-base-1.4.20010614nb14/ readline-4.3/                  trafshow-3.1nb1/
libwww-5.4.0/                  rsync-2.5.6nb1/                vim-6.1.0/
lynx-2.8.4.1.4/                samba-2.2.8a/                  vim-share-6.1.0/
mutt-1.4.1/                    screen-3.9.13/                 wget-1.8.2/
mysql-client-3.23.49nb3/       skill-4.1nb2/                  wol-0.6.0/
zuntum@zenith:~>
```

Jak widać, oprócz katalogów z poszczególnymi pakietami znajduje się w nim także plik `pkgdb.byfile.db` - jest on wykorzystywany przez polecenie `pkg_admin(1)`.

4.4.1 pkg_info(1)

Polecenie pkg_info wywołane bez parametrów wyświetla zainstalowane w naszym systemie pakiety, jednak posiada ono znacznie większe możliwości. Wszystkie dostępne opcje są oczywiście opisane w manualu, lecz przytoczę tutaj kilka najciekawszych:

- Wyświetlenie listy plików należących do pakietu

```
zuntum@zenith:~> pkg_info -L rsync
Information for rsync-2.5.6nb1:
Files:
/usr/pkg/bin/rsync
/usr/pkg/man/man1/rsync.1
/usr/pkg/man/man5/rsyncd.conf.5
/usr/pkg/share/doc/rsync/README
/usr/pkg/share/doc/rsync/tech_report.tex
```

- Wyświetlenie ilości miejsca na dysku jakie zajmuje pakiet

```
zuntum@zenith:~> pkg_info -s rsync
Size of this package in bytes: 271213
```

4.5 Usuwanie pakietów

Odinstalowanie pakietu jest bardzo proste - wystarczy tylko podać nazwę pakietu jako argument polecenia `pkg_delete`.

4.6 /etc/mk.conf

Jest to plik konfiguracyjny, w którym możemy ustawić zmienne mające wpływ na pkgsrc - domyślnie nie istnieje, lecz możemy go utworzyć i umieścić tam swoje ustawienia. Warto także przejrzeć zawartość pliku `pkgsrc/mk/bsd.pkg.defaults.mk`, który zawiera wartości domyślne używane przez pkgsrc. Kilka zmiennych których znaczenie warto znać:

- **LOCALBASE** (domyślnie `/usr/pkg`)

  Prefix (katalog), w którym będą umieszczane pliki instalowane przez pakiety. Nie powinniśmy zmieniać tego ustawienia, jeżeli już instalowaliśmy jakieś pakiety przy użyciu pkgsrc.

- **DISTDIR** (domyślnie `/usr/pkgsrc/distfiles`)

  Katalog, w którym będą umieszczane distfiles'y (ściągane źródła oraz inne pliki). Warto ustawić ten katalog np. na `/usr/distfiles` - drzewo pkgsrc jest zawsze łatwe do odtworzenia a tym samym może być w razie potrzeby bez przeszkód usunięte - warto jednak zachować distfiles'y, żeby uniknąć ich ponownego ściągania. Umieszczenie ich poza drzewem pkgsrc znacznie utrudnia ich przypadkowe skasowanie.

- **PACKAGES** (domyślnie foo `/usr/pkgsrc/packages`)

  Miejsce, w którym będą umieszczane tworzone przez nas pakiety binarne. Analogicznie jak z DISTDIR, warto ustawić ten katalog na lokalizację poza drzewem pkgsrc.

- **PKG_SUFX** (domyślnie `.tgz`)

  Format, w którym będą tworzone pakiety binarne (`.tgz` bądź `.tbz`).

- **FETCH_CMD** (domyślnie `/usr/bin/ftp`)

  Program używany do pobierania distfiles'ów z sieci.

- **MASTER_SITE_****

  Możemy tutaj ustawić optymalne serwery, z których pobierane będą distfiles'y, np.:

```
MASTER_SITE_BACKUP=     ftp://sunsite.icm.edu.pl/pub/NetBSD/packages/distfiles/
MASTER_SITE_OVERRIDE=   ftp://sunsite.icm.edu.pl/pub/NetBSD/packages/distfiles/
MASTER_SITE_SUNSITE=    ftp://sunsite.icm.edu.pl/pub/
```

- **WRKOBJDIR**

  Lokalizacja katalogu tymczasowego używanego przy budowaniu pakietów (domyślnie będzie to katalog `work` w katalogu z pakietem, np. `/usr/pkgsrc/misc/screen/work`).

- **PKG_SYSCONFBASE** (domyślnie `/usr/pkg/etc`)

  Główny katalog, w którym będą znajdować się pliki konfiguracyjne pakietów; ewentualnie będą w nim specyficzne dla danego pakietu katalogi, w których dopiero będą umieszczane właściwe pliki konfiguracyjne.

- **DEPENDS_TARGET** (domyślnie `reinstall`)

  Określa akcję, jaka zostanie podjęta na pakietach automatycznie instalowanych jako zależności - warto ją ustawić na `package`, wtedy zawsze tworzone będą pakiety binarne z wszystkich pakietów, które zostały automatycznie zainstalowane.

5 pkgsrc a bezpieczeństwo

5.1 pkg-vulnerabilities

Deweloperzy pkgsrc oraz *Security-Officer NetBSD* [NetBSD-SO] utrzymują (oraz niestety często muszą aktualizować) bazę, w której umieszczane są informacje o lukach w bezpieczeństwie (ang. *vulnerabilities*) poszczególnych wersji pakietów, wraz z wyszczególnieniem, o jakiego typu błąd chodzi. Jest to plik tekstowy [pkg-vuln], w którym poszczególne kolumny oddzielone są tabulatorami.

Aby w pełni docenić zalety tego typu bazy danych, warto zainstalować pakiet `security/audit-packages`. Znajduje się w nim skrypt, który porównuje wersje zainstalowanych pakietów z bazą danych oraz informuje, jeżeli któryś z zainstalowanych w naszym systemie pakietów figuruje na czarnej liście oraz ma ważny błąd w bezpieczeństwie. Jednym z możliwych rozwiązań jest uruchamianie tego skryptu np. z `/etc/security.local`, wtedy ewentualne ostrzeżenie otrzymamy (najczęściej pocztą) razem z innymi raportami systemowymi.

Powinniśmy regularnie uaktualniać tę bazę poprzez uruchamianie skryptu `download-vulnerability-list`. Można to robić np. przy użyciu *cron'a*:

```
0 3 * * * /usr/pkg/sbin/download-vulnerability-list > /dev/null 2>&1
```

Baza będzie także przeszukiwana w momencie próby instalacji pakietu - jeżeli wersja pakietu w naszym drzewie pkgsrc zostanie uznana za niebezpieczną, zostaniemy o tym poinformowani, a kompilacja pakietu zostanie przerwana:[2]

```
zuntum@zunpc:/usr/pkgsrc/net/samba> make
===> Checking for vulnerabilities in samba-2.0.7.1.3
*** WARNING - local-symlink-race vulnerability in samba-2.0.7.1.3 - see
http://www.securityfocus.com/templates/archive.pike?list=1&mid=177370 for more information ***
*** WARNING - local-root-shell vulnerability in samba-2.0.7.1.3 - see http://www.samba.org/samba/whatsnew/
```

[2] Specjalnie cofnąłem tutaj wersję pakietu do tej z 2001 roku, żeby dokładniej zademonstrować działanie przedstawianego mechanizmu.

```
macroexploit.html for more information ***
*** WARNING - remote-code-execution vulnerability in samba-2.0.7.1.3 - see http://us1.samba.org/samba/
whatsnew/samba-2.2.8.html for more information ***
*** WARNING - remote-root-access vulnerability in samba-2.0.7.1.3 - see http://lists.samba.org/pipermail/
samba-announce/2003-April/000065.html for more information ***
or define ALLOW_VULNERABLE_PACKAGES if this package is absolutely essential
*** Error code 1

Stop.
make: stopped in /usr/cvs/pkgsrc/net/samba
*** Error code 1
```

Jeżeli natomiast baza ta nie zostanie odnaleziona, dostaniemy komunikat ostrzegawczy:

```
zuntum@zunpc:/usr/pkgsrc/net/samba> make
===> *** No /usr/distfiles/vulnerabilities file found - skipping vulnerability checks ***
```

5.2 Łatanie dziur w systemie

pkgsrc może być przydatne do łatania dziur w bezpieczeństwie aplikacji wchodzących w skład systemu operacyjnego - *base systemu* bądź *dystrybucji*. Jeżeli wersja programu dostępna w systemie zawiera błąd, natomiast w pkgsrc znajduje się już poprawiona wersja, możemy zainstalować dany pakiet z pkgsrc oraz wyłączyć wersję znajdującą się w systemie (w przypadku *daemonów*). Jeżeli chcemy uniemożliwić używanie systemowej wersji normalnej aplikacji, a nie chcemy jej kasować, wystarczy ustawić prawa dostępu do pliku wykonywalnego programu na `000` - w momencie próby uruchomienia programu shell nie będzie próbował wykonywać tego pliku, tylko będzie dalej przeszukiwać zmienną *path* - takie uaktualnienie będzie więc zupełnie niezauważalne i niekłopotliwe dla użytkownków, np.:

```
cd /usr/pkgsrc/devel/cvs
make; make install
chmod 0 /usr/bin/cvs
```

Sposób ten jest niezwykle wygodny także w przypadku, gdy z pewnych powodów nie chcemy bądź nie możemy uaktualnić całego systemu operacyjnego, a zależy nam na nowszej wersji tylko jednego z komponentów.

6 Budowa pkgsrc

Kod odpowiedzialny za całą logikę systemu pakietów znajduje się w podkatalogu `mk` drzewa pkgsrc:

```
zuntum@zunpc:~> ls -F /usr/pkgsrc/mk
CVS/                SunOS.pkg.dist         bulk/               ghostscript.mk
Darwin.pkg.dist     SunOS.x11.dist         defs.Darwin.mk      gnu-config/
Darwin.x11.dist     autoconf.mk            defs.FreeBSD.mk     install/
FreeBSD.pkg.dist    automake.mk            defs.IRIX.mk        java-vm.mk
IRIX.pkg.dist       bsd.pkg.defaults.mk    defs.Linux.mk       motif.buildlink2.mk
Linux.pkg.dist      bsd.pkg.install.mk     defs.NetBSD.mk      ossaudio.buildlink2.mk
Linux.x11.dist      bsd.pkg.mk             defs.OpenBSD.mk     pthread.buildlink2.mk
NetBSD.pkg.dist     bsd.pkg.obsolete.mk    defs.SunOS.mk       scripts/
NetBSD.x11.dist     bsd.pkg.subdir.mk      emacs.mk            texinfo.mk
OpenBSD.pkg.dist    bsd.prefs.mk           endian.mk           x11.buildlink2.mk
OpenBSD.x11.dist    buildlink2/            fonts.mk            xaw.buildlink2.mk
```

Większość kodu pkgsrc znajduje się w pliku `bsd.pkg.mk`. Plik ten jest bardzo ogólny - nie ma jeszcze rozróżnienia co do stosowanego systemu operacyjnego. Wartości zależne od systemu są deklarowane dopiero w plikach `defs.${OPSYS}.mk` - wybierany jest właściwy z nich. Powoduje to, że dodawanie wsparcia dla innych systemów operacyjnych jest dużo łatwiejsze. Dzięki przeniesieniu plików `.mk` do podkatalogu drzewa pkgsrc oraz stosowaniu relatywnych ścieżek możliwe stało się używanie jednocześnie więcej niż jednego drzewa pkgsrc - nie występują przy tym konflikty. W systemach ports FreeBSD/OpenBSD pliki te znajdują się natomiast w `/usr/share/mk`.

Do korzystania z pkgsrc niezbędny jest także pakiet *pkg_install* - zawiera on narzędzia takie jak `pkg_add(1)`, `pkg_admin(1)`, `pkg_create(1)`, `pkg_delete(1)` oraz `pkg_info(1)`. Znajduje się on w katalogu `src/usr.sbin/pkg_install` w drzewie źródeł NetBSD, jest także utrzymywany w pakiecie bootstrap-pkgsrc.

6.1 Struktura katalogu pakietu

Zawartość przykładowego katalogu z pakietem wygląda następująco:

```
zuntum@zunpc:~> ls -F /usr/pkgsrc/audio/xmms
CVS/           MESSAGE        PLIST          distinfo       patches/
DESCR          Makefile       buildlink2.mk  files/
```

Opis poszczególnych elementów:

- `Makefile`

  Są tutaj zapisane wszystkie informacje o tym, co należy zrobić, aby poprawnie skompilować i zainstalować pakiet - informacje o lokalizacji distfiles'ów, numerze wersji, kategorii do której należy pakiet, osobie utrzymującej pakiet, stronie domowej programu oraz wiele innych.

- `PLIST`

  Lista plików oraz katalogów instalowanych przez pakiet - dzięki niej możliwe będzie późniejsze czyste odinstalowanie pakietu oraz wszystkich jego plików.

- `distinfo`

  Sumy kontrolne distfiles'ów, używane do sprawdzania integralności plików.

- `DESCR`

  Dokładniejszy opis pakietu (jednolinijkowy opis znajduje się w polu COMMENT w pliku `Makefile`).

- `buildlink2.mk`

  Zawiera informacje o bibliotekach i plikach nagłówkowych które dostarcza dany pakiet oraz które będą dostępne dla pakietów, które będą dołączać ten plik.

- `patches/`

  Znajdują się tutaj łatki które zostaną zaaplikowane przed właściwą kompilacją pakietu.

- `files/`

  Wszystkie pozostałe pliki potrzebne do zbudowania pakietu.

6.2 buildlink

buildlink kontroluje, jakie biblioteki oraz pliki nagłówkowe mogą zostać odnalezione przez pakiet w czasie konfiguracji i kompilacji. Dzięki temu proces budowania programu będzie zawsze przebiegał tak samo, niezależnie od tego, jakie inne pakiety mamy zainstalowane w momencie jego kompilacji. Pozwala to np. uniknąć problemów z pakietami, które starają się używać autodetekcji i wkompilowują opcjonalne składniki. Bardzo ułatwia także tworzenie pakietów dla nowych programów - najczęściej wystarczy tylko dołączyć (*include*) odpowiedni plik `buildlink2.mk` pakietu, aby biblioteki oraz pliki nagłówkowe zostały poprawnie odnalezione.

buildlink powoduje:

1. Utworzonie linków symbolicznych do bibliotek i plików nagłówkowych w tymczasowym katalogu BUILDLINK_DIR.
2. Użycie zamiast standardowego kompilatora specjalnego skryptu, który powoduje, że biblioteki oraz pliki nagłówkowe są poszukiwane w katalogu BUILDLINK_DIR zamiast w standardowych lokalizacjach.

Tworzenie plików `buildlink2.mk`, które będą potem wykorzystywane przez inne pakiety, jest dosyć proste - wystarczy posłużyc się narzędziem `pkgtools/createbuildlink` - skrypt wygeneruje nam szkielet pliku, który należy przejrzeć oraz ewentualnie dostosować. *buildlink2* jest ulepszeniem stosowanej kiedyś pierwszej wersji buildlink.

6.3 just-in-time su(1)

Kompilowanie pakietów nie wymaga uprawnień administratora systemu - możemy robić to z konta zwykłego użytkownika. Oczywiście użytkownik ten powinien (da się to także zrobić w inny sposób) posiadać prawo zapisu do katalogu pkgsrc oraz do katalogu DISTDIR. Wtedy możemy normalnie kompilować pakiet - dodatkowe przywileje potrzebne są jedynie na czas instalacji - możemy wtedy podać hasło root'a, bądź też użyć narzędzia typu *sudo* aby nie wprowadzać hasła z klawiatury.

6.4 Samodzielne tworzenie pakietów dla pkgsrc

Tworzenie pakietów dla pkgsrc jest stosunkowo dobrze udokumentowane oraz najczęściej niezbyt trudne (większe problemy sprawiają jedynie niedbale napisane aplikacje, bądź ogromne pakiety pisane tylko z myślą o Linuxie).

W dokumentacji pkgsrc [Packages.txt] znajdziemy dokładny opis jak stworzyć swój własny pakiet. Na początek warto użyć programu *url2pkg* (`pkgtools/url2pkg`) - jako argument podajemy URL źródeł programu, natomiast skrypt ten automatycznie wygeneruje szkielet pliku `Makefile` oraz postara się automatycznie dostosować niektóre opcje. Jest to dobry sposób na start, gdyż oszczędza nam bezmyślnego wypełniania podstawowych pól.

6.5 pkgsrc-wip

pkgsrc work in progress [pkgsrc-wip] jest utrzymywanym na SourceForge [SourceForge] projektem mającym na celu umożliwienie aktywnej pracy nad pkgsrc każdej chętnej osobie - każdy zainteresowany może otrzymać konto z dostępem do CVS oraz umieszczać tam swoje pakiety. Gdy będą one już w pełni gotowe (dopuszczalne jest importowanie do *pkgsrc-wip* nawet "pół-działającego" pakietu) oraz gdy zostaną przejrzane, są przenoszone z *pkgsrc-wip* do oficjalnej kolekcji pkgsrc (tego już dokonuje osoba z prawem zapisu do repozytorium CVS Projektu).

Zanim powstało *pkgsrc-wip*, zalecanym sposobem na "zgłaszanie" własnego pakietu do kolekcji pkgsrc było wysłanie *PR (Problem Report)* [NetBSD-PR] z kategorią *pkg*; obecnie jednak sugerowane jest raczej użycie *pkgsrc-wip*, gdyż jest to znacznie wygodniejsze oraz umożliwia szybkie przeniesienie pakietu do oficjalnego pkgsrc przy stosunkowo niewielkim nakładzie pracy.

Jest także dostępny wygodny interfejs do przeglądania PR dotyczących kategorii *pkg.* [Gnats-pkg]

6.6 Różne gałęzie pkgsrc, różne gałęzie NetBSD

pkgsrc podlega nieustannym zmianom - pakiety są uaktualniane, dodawane, dokonywane są także inne modyfikacje. Przed wydaniem każdej nowej wersji NetBSD pkgsrc jest zamrażane - dopuszczalne są w tym czasie tylko zmiany, które naprawiają nie kompilujące się pakiety, bądź też zmiany ważne ze względów bezpieczeństwa. "Zamrożenie" kończy się dopiero po znacznym zmniejszenie liczby otwartych PR dotyczących kategorii *pkg* oraz osiągnięciu takiego stanu, aby wszystkie pakiety (w sytuacjach beznadziejnych robi się wyjątki) budowały się poprawnie przynajmniej na platformie *i386*.

W tym momencie tworzona jest nowa gałąź w CVS - np. pkgsrc przygotowane dla NetBSD 1.6.1 możemy uzyskać korzystając z opcji "`-rnetbsd-1-6-1`" przy dokonywaniu "`cvs checkout`" bądź "`cvs update`".

Gałęzie pkgsrc dostarczają zestaw przetestowanych pakietów i bardzo rzadko podlegają zmianom - w już utworzonej gałęzi będą dokonywane jedynie zmiany, które mają wpływ na bezpieczeństwo bądź naprawiające kompilację pakietów, które nie były w stanie się zbudować (np. dodane później poprawki specyficzne dla danej platformy).

Używanie galęzi pkgsrc jest jednak bardzo konserwatywnym podejściem. Jeżeli zależy nam na aktualnych wersjach oprogramowania warto korzystać z *pkgsrc-current*, tzn. wersji bez żadnej gałęzi. Mimo dokonywanych w niej zmian jej używanie jest praktycznie zawsze bezproblemowe.

7 Uaktualnianie

7.1 Sprawdzanie wersji zainstalowanych pakietów

Aby sprawdzić, czy zainstalowane w systemie pakiety są aktualne, można posłużyć się np. narzędziem *pkg_chk* (`pkgtools/pkgchk`) - oczywiście aby to sprawdzanie miało sens, nasze drzewo pkgsrc musi być aktualne:

```
zuntum@zunpc:~> pkg_chk -i
rsync-2.5.6: version mismatch - rsync-2.5.5
unrar-3.2.1: version mismatch - unrar-3.1.3
```

7.2 Aktualizacja drzewa pkgsrc

Co pewien czas (zwłaszcza przed budowaniem dużej ilości pakietów) warto uaktualnić swoje drzewo pkgsrc. Dzięki temu zawsze będziemy mieć najświeższe wersje pakietów oraz wszystkie poprawki.

Gdy już mamy na dysku pkgsrc, najłatwiejszym sposobem będzie użycie CVS:

```
cd /usr/pkgsrc
cvs update
```

Jeżeli nie korzystamy z *pkgsrc-current*, tylko z jednej z gałęzi, należy użyć opcji -r przy `cvs update`, np. `-rnetbsd-1-6-1`

7.2.1 Selektywna aktualizacja

Przy zachowaniu ostrożności możliwe jest także uaktualnianie jedynie części drzewa pkgsrc. Ponieważ utrzymywane jest ono w CVS[3], możemy bez problemu manipulować na pojedynczych katalogach bądź plikach. Należy jednak zawsze mieć aktualny katalog `mk` - pakiety mogą wymagać wprowadzonych do niego modyfikacji, żeby poprawie się zbudować. Może to być dodana funkcjonalność lub też poprawka jakiegoś błędu.

Następną pułapką są pliki `buildlink2.mk`. Podczas projektowania systemu *buildlink* zastanawiano się nad najlepszą dla nich lokalizacją - pierwszą opcją było umieszczanie tych plików w katalogach z pakietem, drugą natomiast umieszczenie ich wszystkich w podkatalogu `mk`. Żadne z tych rozwiązań nie jest doskonałe oraz niesie pewne ryzyko przy selektywnych aktualizacjach, jednak zdecydowano się umieszczać te pliki w katalogach z pakietem. W przeciwnym przypadku pliki `buildlink2.mk` mogłyby nawet pochodzić z innych wersji pakietów niż ta w naszym drzewie pkgsrc, co byłoby znacznie bardziej ryzykowne.

Nie sposób podać prostego sposobu na "bezpieczne" aktualizowanie jedynie części pkgsrc - jednak jeżeli zaktualizujemy dodatkowo katalog `mk` oraz nasze drzewo nie jest jeszcze zbyt stare (chodzi o szansę na to, że nie nastąpiły duże zmiany), nie powinniśmy doświadczyć żadnych problemów. Szczególnie beztrosko możemy wykorzystywać ten sposób do aktualizacji pakietów które nie mają zbyt dużo zależności i tym samym nie korzystają też z innych pakietów (głównie z ich plików `buildlink2.mk`). Zawsze jednak najbezpieczniejszym wyjściem jest aktualizacja całego drzewa pkgsrc.

7.3 Uaktualnianie pakietów przy użyciu źródeł

Aby uaktualnić pakiet korzystając ze źródeł, wystarczy wystarczy wejść do katalogu z danym pakietem oraz wydać polecenie `make update` - wszystkie pakiety, które wymagają danego pakietu zostaną odinstalowane, sam pakiet zostanie odinstalowany, skompilowana oraz zainstalowana zostanie jego nowa wersja oraz zostaną skompilowane i zainstalowane wszystkie pakiety, które musiały zostać usunięte, aby być przekompilowane z nowszą wersją tego pakietu.

Jeżeli chcemy uaktualnić wszystkie pakiety w systemie, możemy posłużyć się jednym z kilku dostępnych narzędzi - np. programem `pkg_chk` z opcją "`-u`". Jednak osobiście nie stosuję tego sposobu, gdyż wolę mieć pełną kontrolę nad tym procesem, chociażby kolejnością deinstalacji oraz aktualizacji pakietów.

[3] Aktualizowanie na poziomie pojedynczych plików w drzewie *ports* (FreeBSD/OpenBSD), zazwyczaj uaktualnianym przy pomocy CVSup, nie jest tak prosta.

7.4 Uaktualnianie pakietów przy użyciu binariów

Uaktualniania przy użyciu pakierów binarnych również jest proste. Najpierw kasujemy dany pakiet (jeżeli ma jakieś zależności, także powinniśmy je odinstalować); następnie możemy już zainstalować jego nowszą wersję - jeżeli potrzebne są jakieś brakujące zależności, zostaną dodane automatycznie (jeżeli są dostępne w tej samej lokalizacji co uaktualniany pakiet).

7.5 Przydatne narzędzia

Warto poznać zastosowanie kilku przydatnych programów dostępnych w kategorii `pkgtools`:

- **distfetch**. Umożliwia ściągnięcie distfiles'ów danego pakietu oraz wszystkich wymaganych zależności. Tak przygotowany zestaw plików możemy np. nagrać na CD oraz użyć do kompilacji pakietu w miejscu, gdzie nie mamy podłączenia do sieci.
- **dfdisk**. Umożliwia ściąganie distfiles'ów z wielu lokalizacji (obecnie obsługiwane są CD-ROM oraz sieć). Tworzona jest baza, w której dysk CD jest kojarzony z zestawem distfiles'ów. Kiedy dany plik jest potrzebny oraz zostanie odnaleziony w tej bazie, użytkownik zostanie poproszony o umieszczenie właściwej płyty CD; w przeciwnym wypadku plik zostanie ściągnięty z sieci.
- **cdpack**. Pozwala na łatwe stworzenie płyt CD z zestawem pakietów binarnych (program jest m.in. wykorzystywany do tworzenia oficjalnej kolekcji płyt z zestawem pakietów).

8 Plany na przyszłość

- pelne wsparcie dla innych OSow (bootstrap-pkgsrc, pkgsrc/doc/STATUS)
- integracja pkgviews
- bulk builds na platformach różnych od NetBSD

9 Kontakt

Została utworzona specjalna lista pocztowa poświęcona dyskusji na temat pkgsrc [tech-pkg]. Możemy tam uzyskać pomoc zarówno odnośnie istniejących pakietów jak i przy próbie samodzielnego tworzenia pakietów.

Literatura

[FreeBSD-ports] FreeBSD ports
http://www.freebsd.org/ports/index.html

[OpenBSD-ports] OpenBSD Ports and Packages collection
http://www.openbsd.org/ports.html

[Moolenaar] All For One Port, One Port For All
Bram Moolenaar, prezentacja na BSDCon Europe 2002
http://eurobsdcon.org/papers/moolenaar.pdf

[Gentoo] Gentoo Linux
http://www.gentoo.org/

[A-A-P] A-A-P project
http://www.a-a-p.org/

[NetBSD-SO] NetBSD Security-Officer
http://www.netbsd.org/People/groups/#securityofficer

[pkg-vuln] Lista znanych luk w bezpieczeństwie pakietów pkgsrc
ftp://ftp.netbsd.org/pub/NetBSD/packages/distfiles/vulnerabilities

[pkgsrc.org] pkgsrc: The NetBSD Packages Collection
http://www.pkgsrc.org

[pkgsrc-index] Spis pakietów dostępnych w pkgsrc
ftp://ftp.netbsd.org/pub/NetBSD/packages/pkgsrc/README.html

[Packages.txt] Dokumentacja systemu pakietów NetBSD
ftp://ftp.netbsd.org/pub/NetBSD/packages/pkgsrc/Packages.txt

[pkgsrc-wip] pkgsrc-wip (work in progress)
http://pkgsrc-wip.sourceforge.net/

[SourceForge] SourceForge.net
http://www.sourceforge.net/

[Crooks] Package Views - a more flexible infrastructure for third-party software
Alistair Crooks, prezentacja na BSDCon Europe 2002
http://eurobsdcon.org/papers/crooks.pdf

[NetBSD-PR] NetBSD Problem Reports
http://www.netbsd.org/Misc/send-pr.html

[Gnats-pkg] Summary of Problem Reports for Category „pkg”
http://www.netbsd.org/Gnats/category/pkg.html

[tech-pkg] Lista pocztowa tech-pkg
http://www.netbsd.org/MailingLists/#tech-pkg