Uniwersytet w Białymstoku

Wydział Matematyczno-Fizyczny

Instytut Matematyki

Andrzej Drobnikowski

Baza danych informacji o poprawkach systemu Solaris

Praca dyplomowa napisana
pod kierunkiem
dr. Mariusz Żynel

Białystok 2005

Składam serdeczne podziękowania

Andrzej Drobnikowski

Spis treści

Wstęp

Jednym z podstawowych zadań administratora systemu komputerowego jest instalowanie poprawek (ang. patch) do systemu operacyjnego i oprogramowania użytkowego. Zadanie to jest o tyle skomplikowane, że administrator dysponuje z jednej strony listą wszystkich poprawek wydanych przez producenta systemu, z drugiej listą poprawek naniesionych na system i na ich podstawie musi wyznaczyć różnicę, czyli poprawki brakujące lub poprawki o wyższym numerze wersji. Do administratora należy poza tym ocena przydatności poprawek. Nie warto bowiem nanosić poprawki na program, który nie jest w ogóle zainstalowany. Kolejną trudnością stojącą przed administratorem jest to, że poprawki muszą być nanoszone w odpowiedniej kolejności. Poza tym niektóre łatki wymagają wcześniejszego zainstalowania innych łatek, od których zależą. Jedna poprawka może mieć kilka kolejnych wersji (ang. release level) i administrator musi zadbać o zainstalowanie nowszej wersji do poprawki wcześniej naniesionej. Część poprawek wymaga, aby komputer został zrestartowany po ich naniesieniu.

Widać zatem, że pozornie proste zadanie wymaga ogromnych nakładów pracy. Sytuacja jest stosunkowo prosta, gdy administrator zarządza jednym lub dwoma komputerami. Wszystko dodatkowo komplikuje się, gdy liczba komputerów rośnie i osiąga 50 albo 500.

Celem naszego projektu jest pomoc administratorowi w uporaniu się z tym trudnym probleme jakim niewątpliwie jest zarządzanie patcha'mi. Projekt powstawał w ramach dwóch prac dyplomowych: mojej i pani Doroty Borowskiej pt: *Interfejs WWW do bazy danych informacji o poprawkach systemu Solaris*. Moim zadaniem w tym projekcie było zaprojektowanie bazy danych oraz napisanie sktyptów w Perlu do importowania i aktualizacji bazy na podstawie danych z poszczególnych komputerów i od producenta systemu operacyjnego. Pomysł takiej aplikacji nie jest nowy. Od roku Bruce Riddle adninistrator systemu w amerykańskiej korporacji prodokującej systemy scalone pracuje nad podobnym projektem. On jest pomysłodawcą projektu, a inspiracja była bardzo prozaiczna, a mianowicie około 500 komputerów z systemem Solaris których administratorem jest Bruce Riddle. W dużej mierze korzystaliśmy z doświadczeń Bruce'a, ale aplikacja i skrypty zostały napisane całkowicie od początku. Generalnie idea i funkcjonalność jest taka sama, ale zmieniony został sposób realizacji poszczególnych zadań.

Efektem współpracy z panią Dorotą Borowską jest w pełni funkcjonalna aplikacja internetowa dostępna pod adresem:

`http://patchdb.solaris-x86.net`.

Rozdział 1

Patch'e w systemie Solaris

Programy komputerowe oraz same systemy operacyjne są coraz bardziej skomplikowane i bardziej rozbudowane. Pomimo coraz lepszych technologi i coraz wiekszej wiedzy nie da się uniknąć błędów podczas tworzenia oprogramowania. Do naprawy tych błędów służą tak zwane patch'e. Patch'e wzieły się stąd, że przy większych aplikacjach, lub całych systemach wymiana całości oprogramowania jest kosztowna, a w wypadku serwerów produkcyjnych pracujących 24 godzinę na dobę nie jest wręcz możliwe zbyt częste przeinstalowywanie całego systemu.

Patch (z ang. łata) to określenie programu, który likwiduje usterki lub błędy powstałe w czasie pisania programu, początkowo przeoczone przez autorów. W wyniku uwag pierwszych użytkowników nowego oprogramowania, autorzy programów opracowują stosowne poprawki - łaty, które doinstalowywane do wadliwego oprogramowania usuwają wcześniejsze jego wady. Są to najczęściej niewielkiej wielkości programiki lub samodzielne pliki które można zazwyczaj znaleźć na stronach WWW producentów oprogramowania.

1.1 Podstawowe polecenia

Zanim prztstąpimy do pracy z patch'ami potrzebna nam będzie znajomość niktórych poleceń w systemie Solaris. Do najważniejszych poleceń do obsługi zadań dotyczących patch'y należą:

- patchadd
- patchrm
- showrev
- pkginfo

Polecenia te służą między innymi do instalowania i odinstalowywania patchy oraz do uzyskiwania wielu przydatnych informacji potrzebnych w procesie zarządzania patch'ami. Dzięki znajomośći tych poleceń nasza praca z patch'ami będzie wydajniejsza i zmniejszymy ryzyko popełnienia błędów operacyjnych.

1.1.1 patchadd

`Patchadd`, jak sama nazwa wskazuje służy do dodawania pakietów patch'y do systemu Solaris. To narzędzie instalacyjne dostępne jest w systemach Solaris od wersji 2 do 10. W Solaris 1 narzędzie to nie może być używane. Aby korzystać z `patchadd` trzeba być zalogowanym jako root. Składnia wywołania `patchadd`:

```
patchadd [-d] [-u] [-B backout_dir] [-C net_install_image|
          -R client_root_path| -S service] patch
patchadd [-d] [-u] [-B backout_dir] [-C net_install_image|
          -R client_root_path| -S service] -M patch_dir|
          patch_id...| patch_dir patch_list
patchadd [-C net_install_image| -R client_root_path| -S service]
          -p
```

Są trzy formy wywołania programu `patchadd`:

- Pierwsza forma patchadd instaluje jedną łatę na system, klienta, serwis lub na obraz do instalacji przez sieć.
- Druga forma patchadd instaluje więcej niż jedną łatę na system, klienta, serwis lub na obraz do instalacji przez sieć.
- Trzecia forma patchadd wyświetla zainstalowane łaty na klienta, serwis lub na obraz do instalacji przez sieć.

Opcje z jakimi możemy wywołać `patchadd`:

`-d` Nie tworzy kopii zapasowej plików, które mają być łatane. Łata nie może być usunięta.

`-p` Wyświetla listę łat obecnie stosowanych.

`-u` Instaluje patch'e bezwarunkowo, wyłącza sprawdzanie poprawności plików.

`-B backout_dir` Zapisanie kopi łatanych plików do kartoteki innej niż baza danych pakietów. Standardowa baza danych o zainstalowanym opragramowaniu na Solaris znajduje się w /var/sadm. Wyszczególnić `backout_dir` jako całkowita nazwa ścieżki.

`-C net_install_image` Za pomocą tej opcji możemy wgrać poprawki na obraz do instalacji przez sieć.

`-M patch_dir patch_id...| patch_dir patch_list` Instaluje do systemu więcej niż jednego patch'a. Patch'e możemy instalować na dwa sposoby:

- Podajemy dokładną ścieżkę dostępu do katalogu w którym znajdują się patch'e oraz numery patch'y które chcemy zainstalować.
- Podajemy ścieżkę dostępu do listy patch'y które chcemy zainstalować.

`-R client_root_path` Tutaj określa się ścieżkę do plików klienta bezdyskowego.

`-S service` Określa wesje systemu na kliencie bezdyskowym.

1.1.2 patchrm

Program `patchrm` usuwa pakiety patch'y i przywraca uprzednio zapisane pliki w systemie Solaris. Podobnie jak `patchadd` również `patchrm` działa w systemach Solaris w wersjach od 2 do 10. Aby korzystać z `patchrm` trzeba być zalogowanym jako root. Składnia wywołania `patchrm`:

```
patchrm [-f] [-B backout_dir] [-C net_install_image
        | -R client_root_path | -S service] patch_id
```

Opcje z jakimi możemy wywołać `patchrm` to:

`-f` Wymusza usunięcie patch'a bez względu na to czy był on przykryty innym patch'em czy nie.

`-B backout_dir` Ścieżka do kopi plików sprzed wgrania patch'a.

`-C net_install_image` Za pomocą tej opcji możemy usunąć poprawki z obrazu do instalacji przez sieć.

`-R client_root_path` Tutaj określa się ścieżkę do plików klienta bezdyskowego.

`-S service` Określa wesje systemu na kliencie bezdyskowym.

1.1.3 showrev

Program `showrev` wyświetla podstawowe informacje o systemie: nazwę hosta, numer wersji systemu, platformę, wersję jądra systemu itp. Z dodatkowymi opcjami możemy uzyskać jeszcze więcej informacji. Składnia wywołania `showrev`:

```
showrev [-a] [-p | -p -R root_path] [-w] [- c command]
        [-s hostname]
```

Opcje z jakimi możemy wywołać `showrev`:

`-a` Wyświetla wszystkie dostępnych informacje.

`-c command` Wyświetla iformacje o konkretnym programie, który jest wyszczególniony jako `command`.

`-p` Wyświetla informacje o patch'ach zainstalowanych w systemie.

`-R root_path` Ta opcja może być użyta w odniesieniu do klientów bezdyskowych.

`-s hostname` Uruchania `showrev` na podanym komputerze, który jest wyszczególniony jako `hostname`.

`-w` Wypisywana jest wersja systemu okienkowego.

1.1.4 pkginfo

Program `pkginfo` wyświetla liste informacji o pakietach zainstalowanych w systemie Solaris. Bez pocji, `pkginfo` wyświetla listę głównych kategorii, egzemplarz pakietu, i nazwy wszystkich całkowicie i częściowo zainstalowanych pakietów. Wyświetla jedną linię dla każdego wybranego pakietu. Składnia wywołania `pkginfo`:

```
pkginfo [-q | -x  |  -l]   [-p  |  -i]   [-r]  [-a arch]
        [-v version] [-c category...] [pkginst...]
pkginfo [-d device] [-R root_path] [-q | -x | -l]  [-a arch]
     [-v version] [-c category...] [pkginst...]
```

Są trzy formy wywołania programu `pkginfo`:

- Pierwsza forma `pkginfo` wyświetla informację o pakietach programów, które są zainstalowane w systemie.
- Druga forma `pkginfo` wyświetla informację o pakietach programów, które są zainstalowane na podanym urządzeniu lub w podanej kartotece.

Opcje z jakimi możemy wywołać `pkginfo`:

`-a arch` Wyświetla pakiety o wyszczególnionej architekturze.

`-c category` Wyświetla pakiety, które pasują do kategorii. Kategorie są definiowane parametrem kategorii w pliku. Jeżeli podano więcej niż jedną kategorie wystarczy, że pakiet pasuje tylko do jednej kategorii z listy.

`-d device` Opisuje narzędzie, na którym umieszczone jest oprogramowanie. Narzędzie może być pełną nazwą ścieżki lub też identyfikatorem taśmy, dyskietki, dysku.

`-i` Wyświetla informację tylko całkowicie zainstalowanych pakietach.

`-l` Precyzuje długi format, który zawiera wszystkie dostępne informacje o wyróżnionych pakietach.

`-p` Wyświetla wyłącznie informację o częściowo zainstalowanych pakietach.

`-q` Nie wyświetla żadnych informacji. Używany przez programy do sprawdzania czy pakiet był instalowany, czy nie.

`-r` Wyświetla katalog bazowy dla pakietów przenaszalnych.

`-R root_path` Informacje o pakietach pobierane będą z podanej ścieżki, wygodne jeśli mamy klientów bezdyskowych.

`-v version` Precyzuje wersję pakietu jako `version`. Każdy zainstalowany pakiet posiada określoną wersję, którą możemy sprawdzić tym poleceniem.

`-x` Wyświetla listę najważniejszych informacji o pakietach. Lista zawiera skróty pakietu, nazwę pakietu, architekturę pakietu(jeżeli dostępna) i wersję pakietu (jeżeli dostępna).

1.2 Plik patchdiag.xref

Listę wszystkich patch'y dostępnych na systemy Solaris możemy znalesć w pliku *patchdiag.xref*. Plik ten jest aktualizowany przez Sun codziennie, oprócz dni wolnych od pracy. Znajduje się on na stonie internetowej Sun'a pod adresem *http://patches.sun.com/reports/patchdiag.xref*. Tak wygląda początkowy fragment pliku *patchdiag.xref*:

```
## PATCHDIAG TOOL CROSS-REFERENCE FILE AS OF Jun/24/05 ##
##
## Please note that certain patches which are listed in
## Sun's Quick Reference Section or other patch reference
## files are not publicly available, but instead are
## available only to customers of Sun Microsystems who
## have purchased an appropriate support services contract.
## For more information about Sun support services contracts
## visit www.sun.com/service
100287|05|Oct/31/91| | | |  |Unbundled|||PC-NFS 3.5c: Jumbo patch
(updated PRT.COM to v3.5c)
100323|05|Feb/11/92| | | |  |Unbundled|||PC-NFS Advanced Telnet:
bug fixes, National Character Set support
100386|01|Sep/20/91| | | |  |Unbundled|||PC-NFS Programmer's Toolkit/2.0:
Runtime modules
100393|01|Sep/02/94| | |O|  |Unbundled|||OBSOLETED by 100394
100807|03|May/02/94| | |O| B|Unbundled|||OBSOLETED by WITHDRAWN
101176|06|Mar/31/95| |S| |  |Unbundled|sparc;|SUNWhsm:2.0;|
Online Backup 2.0: Update patch
101235|01|May/02/95| |S|O|  |2.3|||OBSOLETED by 101739
101331|08|Sep/03/97|R|S| |Y |2.3|sparc;|SUNWarc:11.5.0,REV=2.0.18;
```

```
SUNWcsu:11.5.0,REV=2.0.18; |SunOS 5.3: fixes for package utilities
101327|08|Oct/12/94|R|S| |  |2.3|sparc;|SUNWcsu:11.5.0,REV=2.0.18;|
SunOS 5.3: security and miscellaneous tar fixes
101494|04|Oct/16/98|R|S| |  |2.3|sparc;|SUNWcsu:11.5.0,REV=2.0.18;|
SunOS 5.3: rdist patch
101512|42|Aug/07/97|R|S|O| B|2.3|||OBSOLETED by WITHDRAWN
101533|07|Oct/02/97| | | |Y |2.3|sparc;|SUNWowMIT:3.3.18,REV=0.93.09.07;|
OpenWindows 3.3: xterm patch
```

Jak widać plik ten ma budowę systemową, dane w nim są pogrupowane wierszami. Niektóre wiersze w pliku z powodu braku miejsca zostały złamane. Wiersze na początku pliku zaczynające się od znaku (#) są to ogólne informacje o pliku. Każdy wiersz oprócz tych zaczynających się znakiem (#) dotycz jednego konkretnego patch'a. Informacje o danym patch'u są rozdzielone znakiem (|). Poszczególne kolumny mają następujące znaczenie:

1. Numer indentyfikacyjny danego patch'a.
2. Numer wersji danego patch'a. Jeden patch może występować w różnych wersjach. Im numer jest wyższy tym wersja patch'a jest nowsza.
3. Data umieszczenia informacji o patch'u w pliku *patchdiag.xref*.
4. R czyli recomended. Oznacza to, że dany patch jest rekomendowany przez Sun'a do zainstalowania go na naszym komputerze.
5. S czyli seciury. Znaczy to ,że patch należy do grupy tzw. seciury patches, czyli jest to patch łatający błędy w programach i systemie dotyczące bezpieczeństwa naszego komputera.
6. O czyli obsoleted. Oznacza to, że ten patch został zastąpiony przez inny patch, który łata ten sam błąd. Nie zachodzi więc potrzeba instalowania tego patch'a na naszym komputerze. Możemy zainstalować patch który go zastąpiono. Jego numer znajduje się w ostatniej rubryce.
7. B czyli bad patch. Jest to patch w którym zostały wykryte jakieś błędy i został on wycofany. Y czyli yers 2000. Patch łatający błędy związane z rokiem 2000. Obecnie nie ma to już takiego wielkiego znaczenia, ale było to potrzebne w czasach gdy obawiano się błędu kommputerów związanego z rokiem 2000.
8. Numer wersji systemu solaris.
9. Architektur komputera na którym instalowany ma być dany patch. Jeżeli chodzi o Solaris to mamy do wyboru *sparc* lub *i386*.
10. Lista pakietów których dany patch dotyczy.
11. Krótki opis danego patch'a.

1.3 Programy do zarządzania patch'ami

W Internecie możemy znaleść wiele aplikacji służących do zarządzania patch'ami. Może wię nasunąć się pytanie dlaczego my tworzymy swój własny program, skoro możemy skorzystać z gotowych narzędzi. O tuż nasz program różni się w pewnych kwestiach od tych dostępnych w internecie. Większość tych programów używa technologi Java, lub są to pojedyńcze skrypty, które trzeba uruchamiać na danym komputerze za każdym razem gdy chcemy uzyskać jakieś informacje dotyczące patch'y. My natomias tworzymy centralne miejsce dla administratora systemu, który może przez przeglądarke internetową zarządzać patch'ami na swoim komputerze. Nie musi przy tym korzystać z technologi Java, która może zawierać wiele pluskiew i tym samym może ucierpiec bezpiczenstwo komputera, nie każdy też ma zainstolowaną Jave. Używając naszego programu nie trzeba mieć również bezpośredniego dostępu do swego komputera, żeby np sprawdzić jakie patch'e mamy na nim zainstalowane a jakich nie. Nie też potrzeby instalowania naszego programu w komputerze.

W poniższych podrozdziałach przedstawie trzy programy które, służą do obsługi patch'ów, a są to:

- PatchPro Expert
- Patch Manager
- Patch Check Advanced

1.3.1 PatchPro Expert

PatchPro Expert jest najnowszym programem firmy Sun, który służy do zarządzania patch'ami w systemach Solaris. Jak twierdzi producent jest to inowacjyjne narzędzie rozwiązujące problemy administratorów systemów Solaris związane z patch'ami. PatchPro Expert używa technologi Java. Tak jak jego poprzednik PatchPro Interactive ułatwia administratorom lokalizacje i instalacje patch'y. Jednak w odróżnieniu od PatchPro Innteractive, PatchPro Expert inteligentnie analizuje cały system po to by określić które patch'e należy zainstalować. Nie ma też potrzeby ręcznego konfigurowania systemu. Po przeskanowaniu system, PatchPro Expert tworzy listę patch'y do zainstalowania w systemie. Patch'e na tej liście ułożone są w odpowiedniej kolejności do instalacji. Jak zapewnia producent program rekomenduje do instalacji tylko te patch'e które są nam naprawde potrzebne. Nie ma więc mowy ,że będziemy instalować patch'e które naprawiają błędy w pakietach których nie mamy zainstalowanych na naszym komputerze. Sposób działania programu jest bardzo prosty.

1. Pierwszym krokiem jaki należy wykonać jest wizyta na stronie PatchPro Expert.

2. Po zapoznaniu się z krótkim opisem programu, uruchamiamy go.

3. Po uzyskaniu pozwolenia Java zacznie skanować nasz system. Może to zająć kilka minut, w zależności od szybkości komputera i złożoności naszego systemu.

4. Po podaniu informacji o które pyta nas program zostanie utworzona lista potrzebnych nam patch'y.

5. Gdy mamy listę potrzebnych patch'y klikamy przycisk *download*, a program automatycznie ściąga nam patch'e z listy do wybranego przez nas katalogu.

6. Po ściągnięciu patch'y rozpakowujemy je i instalujemy w odpowiedniej kolejności postępując zgodnie z instrukcjami zawartymi w dołączonym pliku *README*.

1.3.2 Patch Manager

Kolejnym programem służącym do zarządzania patch'ami dostępnym na stronach Sun'a jest Patch Manager. Jest to narzędzie, które ma zapewnić użytkownikom systemu operacyjnego Solaris możliwość poszukiwania - zarówno lokalnie, jak i zdalnie nieszczelności, dla których opracowano już łaty programowe, i sprowadzenie oraz zainstalowanie ich automatycznie. Oprogramowanie ma zapewnić użytkownikom lepszą i łatwiejszą metodę określania, które łatki mają stosować w swoich systemach. Patch Manager jest napisaną w Javie aplikacją desktopową, której działanie polega na porównywaniu konfiguracji systemu Solaris z lokalną kopią bazy wiedzy Suna, zawierającą, oprócz wielu innych, informacje o znanych nieszczelnościach w systemie. Lokalna kopia bazy wiedzy może być uaktualniana na życzenie użytkownika lub automatycznie, po zainicjowaniu skanowania. Sun uaktualnia kopie bazy wiedzy codziennie. Z chwilą otrzymania listy niezbędnych łatek użytkownik ma możliwość sprowadzenia ich i zastosowania w sposób właściwy - automatycznie lub ręcznie, a także może określić czas, kiedy ma to być wykonane. Łatki są opatrzone podpisem cyfrowym, co zapobiega nieautoryzowanym instalacjom.

1.3.3 Patch Check Advanced

Nie tylko Sun tworzy programy do zarządzania patchami. W Internecie możemy znaleść mnóstwo programów których autorami są najczęściej zwykli użytkownicy systemu Solaris, programy te są to pojedyńcze skrypty perlowe. Jednym z takich programów jest Patch Check Advanced. Jest on oparty na podobnych programie firmy Sun - Patch Check. A o to kilka cech Patch Check Advanced, które według autora zasługują na uwage:

- Łatwo zrozumiały format raportów o zainstalowanych i odinstalowanych patch'ach.
- Pokazuje status (recommended lub security) dla zainstalowanych patch'y.
- Możemy ściągać plik *patchdiag.xref* ręcznie lub też program może to zrobić automatycznie.
- Program analizuje prerekwizyty dla poszczególnych patch'y i instaluje patche w odpowiedniej kolejności.

Aby korzystać z Patch Check Advanced musimy mieć zainstalowanego Perla, jeżeli chcemy korzystać ze wszystkich funkcji programu musimy zainstalować także Wget (program do pobierania plików z internetu). Gdy mamy już zainstalowane potrzebne programy możemy ściągnąć skrypt z internetu i uruchomić go na naszym komputerze z odpowiednimi opcjami. A o to dostępne opcje w programie:

`-i` Pokazuje listę wszystkich bierząco zainstalowanych patch'y.

`-u` Pokazuje listę wszystkich patch'y, które nie zostały zainstalowane w ostatnim przeglądzie.

`-f` Pokazuje listę wszystkich patch'y, które nie są związane z żadnymi zainstalowanymi pakietami oprogramowania (unbundled)

`-r` Ogranicza listę niezainstalowanych patch'y do tych które są oznaczone jako *recommeended* i *security*.

`-a` Pokazuje wszystkie zainstalowane i niezainstalowane patch'e. Kombinacja opcji `-i` i `-u`.

`-x` Program ściąga z Internetu aktualny plik *patchdiag.xref*.

`-d` Program ściąga wszystkie patch'e, które są wymienione w bierzącej książce adresowej.

`-p` Zaaplikuje wszystkie patch'e (po pobraniu z Internetu), które są na liście. To jest jedyna opcja przy której wymaga się bycia root'em.

`-n` Zaaplikuj tylko te poprawki patches, które nie wymagają po zainstalowaniu zrestartowania komputera. Używa się tej opcji w połączeniu z opcją -p.

`-R <ID>` Pokazuje plik README danego patch'a. `ID` oznacza numer patch'a.

`-D <ID>` Program ściąga z internetu pojedyńczy patch. `ID` oznacz numer patch'a którego chcemy ściągnąć.

Rozdział 2

Baza danych poprawek

Najważniejszą integralną częścią naszej aplikacji jest baza danych. Zdecydowaliśmy używać serwera bazy danych *MySQL*. Dlaczego wybraliśmy akurat tą bazę danych? Złożyło się na to kilka powodów:

- Jest to baza dostępna na wiele platform między innymi na: Solaris, Linux, Windows.
- Latwo można ją uruchomić na prawie każdym dostępnym obecnie komputerze.
- Jest szybka i nie zawodna, sprawdzona w wielu poważnych zastosowaniach.
- Jest bezpłatna.

2.1 Opis bazy danych patchdb

Bazę danych z której będzie korzystała nasza aplikacja nazwaliśmy *patchdb*. To własnie w tej bazie danych w poszczególnych tabelach będą przechowywane między innymi dane użytkowników korzystających z naszego programu, informacje dotyczą komputerów poszczególnych użytkowników, informacje o poszczególnych patch'ach oraz wiele innych danych potrzebnych dla prawidłowego działania naszego programu. Z tą bazą związane będą również skrypty napisane w PHP i Perlu. Oto tabele znajdujące się w bazie danych *patchdb*:

```
+-------------------+
| Tables_in_patchdb |
+-------------------+
| hosts             |
| pkginfo           |
| showrev           |
| users             |
| xref              |
| xrefarch          |
| xrefcon           |
| xrefhistory       |
| xrefpkg           |
| xrefreq           |
+-------------------+
```

Tabela 2.1: Tabele w bazie danych patchdb

Opisy poszczególnych tabel znajdujących się w tej bazie danych zostaną przedstawione w kolejnych podrozdziałach.

2.1.1 Tabela *users*

Ta tabela nie jest konieczna do prawidłowego działania naszego programu, ale tylko wtedy gdy program obsługuje tylko jednego użytkownika. My natomiast chcemy żeby, program obsługiwał wielu użytkowników. W tym celu musimy zapewnić poszczególnym użytkownikom bezpieczeństwo przechowywania udostępnionych przez nich danych. Do tego właśnie będzie służyć nam tabela *users*. To z niej będą pobierane dane o użytkowniku w czasie jego autoryzacji i autentykacji. O to struktura tej tabeli i opis kolumn:

```
+-----------+--------------+------+-----+---------+----------------+
| Field     | Type         | Null | Key | Default | Extra          |
+-----------+--------------+------+-----+---------+----------------+
| id        | int(11)      |      | PRI | NULL    | auto_increment |
| email     | varchar(64)  |      | UNI |         |                |
| password  | varchar(16)  | YES  |     | NULL    |                |
| firstname | varchar(32)  | YES  |     | NULL    |                |
| lastname  | varchar(32)  | YES  | MUL | NULL    |                |
| active    | enum('Y','N') | YES  |     | Y       |                |
| lastlogin | datetime     | YES  |     | NULL    |                |
| remoteip  | varchar(16)  | YES  |     | NULL    |                |
| note      | tinytext     | YES  |     | NULL    |                |
| updated   | timestamp(14) | YES  |     | NULL    |                |
| created   | timestamp(14) | YES  |     | NULL    |                |
+-----------+--------------+------+-----+---------+----------------+
```

Tabela 2.2: Stuktura tabeli users.

`id` - Numer identyfikujący danego użytkownika. Jest on przypisywany automatycznie przez bazę danych każdemu nowemu, wpisywanemu do tabeli użytkownikowi.

`email` - Adres email użytkownika. Używany jest jako nazwa użytkownika podczas logowania.

`password` - Hasło użytkownika. Hasło to jest przechowywane w tej tabeli w postaci zaszyfrowanej. Szyfrowane ono jest za pomocą standardowej w UNIX funfcji *crypt()*. Uniemorzliwia to podejrzenie hasła przez osoby nieporządane. W przypadku gdy użytkownik zapomniał hasła, jedynym wyjściem jest administracyjne nadanie nowego hasła, ponieważ funkcja *crypt()* jest nie odwracalna.

`firstname` - Imię użytkownika.

`lastname` - Nazwisko użytkownika.

`active` - Pole określające czy konto danego użytkownika jest aktywne czy nie. Przy pomocy tego pola administrator aplikaji może zablokować dostęp użytkownika do czasu wyjaśnienia niejasności bez konieczności usuwania konta.

`lastlogin` - Data ostatniego logowania się przez użytkownika.

`remoteip` - Adres ip komputera z którego było ostatnie logowanie.

`note` - Krótka notatka o danym użytkowniku zamieszczana przez administratora.

`updated` - Data ostatniej modyfikacji.

`created` - Data utworzenia danego rekordu.

2.1.2 Tabela *hosts*

W tabeli tej znajdują się dane dotyczące poszczególnych komputerów korzystających z naszej aplikacji. Są tu między innymi informacje dotyczące architektury danego komputera, wersji systemu Solaris jaka jest zainstalowana na danym komputerz, czy też domeny do której przypisany jest komputer. Tabela ta jest powiązana z tabelą *users* poprzez kolumnę *user*. Struktura tej tabeli wygląda następująco:

Field	Type	Null	Key	Default	Extra
id	int(11)		PRI	NULL	auto_increment
hostname	varchar(32)		MUL		
domain	varchar(64)				
osarch	varchar(64)		MUL		
user	int(11)		MUL	0	
synopsis	tinytext	YES		NULL	
updated	timestamp(14)	YES		NULL	
created	timestamp(14)	YES		NULL	

Tabela 2.3: Stuktura tabeli users.

`id` - Numer identyfikujący rekordy w tej tabeli. Jest on przypisywany automatycznie przez bazę danych każdemu nowemu, wpisywanemu do tabeli komputerowi.

`hostname` - Nazwa hosta.

`domain` - Domena do której przypisany jest komputer danego użytkownika.

`osver` - Numer wersji systemu Solaris zainstalowanego na danym komputerze. Numer ten jest pobierany w wyniku działania polecenia *uname -r*.

`arch` - Architektura danego komputera. W przypadku solarisa może to być *sparc* lub też *i386*.

`user` - Numer identyfikujący właściciela danego komputera. Numer ten pobierany jest z tabeli *users* z kolumny *id*.

`synopsis` - Krótka notatka o danym komputerze zamieszczana przez użytkownika .

`updated` - Data ostatniej modyfikacji rekordu.

`created` - Data utworzenia danego rekordu.

Trójka pól: *hostname*, *domain* i *user* musi być unikalna w tej tabeli. Oznacza to ,że nazwy komputerów w ustalonej domenie danego użytkownika nie mogą się powtórzyć. Możliwe jest natomiast aby

2.1.3 Tabela *showrev*

Tabela *showrev* zawiera dane o patch'ach zainstalowanych na danym komputerze. Jest to bardzo ważna informacja potrzebna do tego by stwierdzić jakie patch'e należy zaktualizować a jakie wgrać jako nowe. Raport na ten temat kożysta z łączonych zapytań do tabel *xref* i *showrev*.

Field	Type	Null	Key	Default	Extra
host	int(11)		PRI	0	
patchid	int(6) unsigned zerofill		PRI	000000	
patchrev	int(2) unsigned zerofill		PRI	00	
created	date	YES		NULL	

Tabela 2.4: Stuktura tabeli showrev.

`host -` Numer identyfikujący komputer. Odpowiada on polu *id* w tabeli *hosts*.

`patchid -` Numer indentyfikujący dany patch.

`patchrev -` Numer wersji patch'a.

`created -` Data wstawienia danego rekordu do tabeli.

2.1.4 Tabela *xref*

Tabela ta nie jest powiązana z żadną inną tabelą znajdującą się w naszej bazie danych. Zawiera ona dane o poszczególnych patch'ach, które są wczytywane do tej tabeli za pomocą skryptu perlowego *insert-xref.pl*(patrz rozdział 3.2.3) z pliku *patchdiag.xref*. Ta tabela jest wiernym odzwierciedleniem najnowszego pliku *patchdiag.xref*. Struktura tej tabeli wygląda następująco:

Field	Type	Null	Key	Default	Extra
patchid	int(6) unsigned zerofill		PRI	000000	
patchrev	int(2) unsigned zerofill		PRI	00	
reldate	date	YES		NULL	
rflag	enum('Y','N')			N	
sflag	enum('Y','N')			N	
oflag	enum('Y','N')			N	
bflag	enum('Y','N')			N	
yflag	enum('Y','N')			N	
osarch	varchar(64)	YES	MUL	NULL	
synopsis	text	YES		NULL	

Tabela 2.5: Stuktura tabeli xref.

`patchid -` Numer identyfikujący patch'a.

`patchrev -` Numer wersji patch'a.

`reldate -` Data dodania patch'a do pliku *patchdiag.xref*.

`rflag` - (Recommendid). Kolumna mówiąca czy dany patch jest rekomendowany przez firmę Sun do zainstalowania, czy też nie.

`sflag` - (Security). To pole mówi czy dany patch łata błędy w programach i systemie dotyczące bezpieczeństwa komputera.

`oflag` - (Obsoleted). Kolumna mówiąca nam, czy dany patch został zastąpiony przez innego patch'a który łata ten sam błąd.

`bflag` - (Bad patch). W ten sposób oznacza się patche wadliwe, które zostały wycofane z dystrybucji.

`yflag` - (Yers 2000). Pole mówiące nam, czy dany patch poprawia błędy związane z rokie 2000, czy też nie.

`osarchs` - Architektura danego komputera i numer wersji zainstalowanego systemu Solaris.

`synopsis` - Krótki opis danego patch'a.

2.1.5 Tabela *xrefhistory*

Struktura tej tabeli jest identyczna jak tabeli *xref* (patrz rozdział 2.1.4). Rzeczą która odróżnia obie tabele jest spoób zapisywania danych do obu tabel przez skrypt *insert-xref.pl* (patrz rozdział 3.2.3). Przed każdorazowym wpisaniem nowy danych do tabeli *xref* jest ona najpierw czyszczona, a dopiero potem wpisywane są dane z aktualnego pliku *patchdiag.xref*. A więc w tabeli *xref* znajdują się tylko dane z najnowszego pliku *patchdiag.xref*. Natomiast przy wpisywaniu danych do tabeli *xrefhistory*, poprzednie dane nie są z niej usuwane. Dodawane są natomiast nowe dane o patch'ach których nie było w pliku *patchdiag.xref* przy ostatnim zapisie. Dzięki takiemu zabiegowi w tabeli tej mamy dane o patch'ach które kiedykolwie pojawiły się w którymś z plików *patchdiag.xref*.

2.1.6 Tabela *xrefpkg*

Tabela ta zawiera listę pakietów oprogramowania, które jest modifikowane przez danego patch'a. Dane te są pobierane z pliku *patchdiag.xref*. Na podstawie zebranych tutaj danych będziemy mogli stwierdzić czy jest sens instalowanie danego patch'a na konkretnym komputerze. Nie warto instalować patch'a który łata błędy w programie którego nie mamy wogle zainstalowanego. Struktura tej tabeli wygląda następująco:

Field	Type	Null	Key	Default	Extra
patchid	int(6) unsigned zerofill		PRI	000000	
patchrev	int(2) unsigned zerofill		PRI	00	
pkg	varchar(32)		PRI		
version	varchar(64)		PRI		

Tabela 2.6: Stuktura tabeli xrefpkg.

`patchid` - Numer identyfikujący patch'a.

`patchrev` - Wersja danego patch'a

`pkg` - Nazwa pakietu który dotyczy danego patcha.

`version` - Wersja danego pakietu.

2.1.7 Tabela *xrefarch*

Tabela ta zawiera listę platform na których dostępny jest dany patch. To właśnie na podstawie tutaj zgromadzonych informacji możemy określić czy dany patch może być zainstalowany na konkretnym komputerze. Dane do tej tabeli pobierane są z pliku *patchdiag.xref*. Struktura tej tabeli wygląda następująco:

Field	Type	Null	Key	Default	Extra
patchid	int(6) unsigned zerofill		PRI	000000	
patchrev	int(2) unsigned zerofill		PRI	00	
arch	varchar(32)		PRI		

Tabela 2.7: Stuktura tabeli xrefarch.

`patchid` - Numer identyfikujący patch'a.

`patchrev` - Numer wersji danego patch'a.

`arch` - Architektura której dotyczy dany patch.

2.1.8 Tabela *xrefreq*

Niektóre patch'e wymagają aby na systemie zainstalowane były pewne inne patch'e. Są to tak zwane prerekwizyty. Bez zainstalowania prerekwizytów nie ma sensu instalowania patch'a. Informacje o prerekwizytach pobierane są

z pliku *patchdiag.xref*. Dane o prerekwizytach pozwolą ustawić instalowane patch'e w odpowiedniej kolejności.

Field	Type	Null	Key	Default	Extra
patchid	int(6) unsigned zerofill		PRI	000000	
patchrev	int(2) unsigned zerofill		PRI	00	
id	int(6) unsigned zerofill		PRI	000000	
rev	int(2) unsigned zerofill		PRI	00	

Tabela 2.8: Stuktura tabeli xrefreq.

`patchid` - Numer identyfikujący patch'a.

`patchrev` - Numer wersji patch'a.

`id` - Numer identyfikujący patch'a, który jest prerekwizytem.

`rev` - Numer wersji patch'a, który jest prerekwizytem.

2.1.9 Tabela *pkginfo*

W tej tabeli zawarte są informacje o pakietach oprogramowania zainstalowanego na danym komputerze. W połączeniu z tabelą *xrefpkg* będzie można określić które patch'e powiny być instalowane na danym komputerze. Dane do tej tabeli wstawia użytkownik w momencie gdy dodaje do bazy nowy komputer lub aktualizuje stary. Jako źródło tych danych służy wynik polecenia pkginfo -l, natomiast programem importującym dane jest skrypt perlowy *insert-pkginfo.pl* (patrz rozdział 3.2.6).

Field	Type	Null	Key	Default	Extra
host	int(11)			0	
pkg	varchar(32)				
version	varchar(128)	YES		NULL	

Tabela 2.9: Stuktura tabeli pkginfo.

`host` - Numer identyfikujący komputer. Odpowiada on polu *id* w tabeli *hosts*

`pkg` - Nazwa pakietu zainstalowanego na danym komputerze.

`version` - Wersja danego pakietu.

2.1.10 Tabela *xrefcon*

Wśród patch'y zdarza się tak że, niktóre z nich nie mogą być instalowane jednocześnie na danym komputerze. Takie patch'e nazywamy konfliktowymi. W tej tabeli zawarte są dane o patch'ach konfliktowych z danym patch'em. Nie będziemy proponować instalacji nowego patch'a jeśli jest on w konflikcie z jednym z zainstalowanych. Lista patch'y konfliktowych z danym jest zawarta w pliku *patchdiag.xref*.

```
+----------+-------------------------+------+-----+---------+-------+
| Field    | Type                    | Null | Key | Default | Extra |
+----------+-------------------------+------+-----+---------+-------+
| patchid  | int(6) unsigned zerofill |      | PRI | 000000  |       |
| patchrev | int(2) unsigned zerofill |      | PRI | 00      |       |
| id       | int(6) unsigned zerofill |      | PRI | 000000  |       |
| rev      | int(2) unsigned zerofill |      | PRI | 00      |       |
+----------+-------------------------+------+-----+---------+-------+
```

Tabela 2.10: Stuktura tabeli xrefcon.

`patchid -` Numer identyfikujący patch'a.

`patchrev -` Numer wersji patch'a.

`id -` Numer identyfikujący patch'a konfliktowego

`rev -` Numer wersji patch'a konfliktowego

Rozdział 3

Skrypty w Perlu

Tworzenie aplikacji wykorzystujących bazy danych jest bez wątpienia jednym z najczęstszych zastosowań Perla. Współpracuje on doskonale z wieloma różnymi formatami baz danych. Wspiera również programistę na dwóch płaszczyznach:

- W przypadku mniejszych programów Perl udostępnia moduł *DBM* (ang. DateBase Manager, czyli Zarządca bazy danych). Skrót ten stanowi ogólne określenie rodziny wielu prostych formatów baz danych, które można spotkać w powszechnym użyciu, najczęściej w systemach typu UNIX. Perl wspomaga stosowanie formatów *DBM* za pomocą tablicy asocjacyjnej, dlatego też zawartość bazy danych przedstawiona jest w postaci zwykłej zmiennej tego typu.

- W przypadku bardziej zaawansowanych programów Perl udostępnia moduł *DBI*(ang. DateBase Interface, czyli Interfejs bazy danych). Jest on ogólnym interfejsem baz danych, używanym do komunikowania się z ich serwerami, stosującymi w celu udostępniania danych język *SQL* (ang. Structured Query Language, czyli Strukturalny język zapytań). Aby wykorzystać ten moduł, konieczne jest również zainstalowanie odpowiedniego modułu sterownika *DBD* (ang. DataBase Driver) przeznaczonego dla bazy danych, z którą zamierzamy się połączyć. Istnieje wiele sterowników ,dostępnych dla wielu popularnych baz danych, włączając w to *MySQL*, *mSQL*, *Oracle* czy *Sybase*.

Moduł *DBI* umożliwia zastosowanie wyższego poziomu abstrakcji - programista pisze kod bez zwracania zbyt dużej uwagi na to, jaka baza znajduje się za interfejsem. Przy użyciu interfejsu *DBI* możliwe jest zastosowanie we wszystkich programach tego samego podejscia do problemu, niezależnie od tego, z jakiej bazy danych będą one korzystać.W naszej aplikacji również jest zastosowany interfejs *DBI* do łączenia się z bazą danych *MySQL*. Dlatego też w dalszej części pracy krótko opiszę ten interfejs.

3.1 Opis interfejsu DBI

Zanim przejdziemy do korzystania z interfejsu *DBI* będziemy musieli go najpierw zainstalować. Musimy zainstalować też sterownik do bazy danych z której będziemy korzystać. W naszym przypadku będzie to sterownik do bazy danych *MySQL*, czyli *DBD::mysql*. Interfejs *DBI* jest takim samym modułem, jak wszystkie inne w Perlu, więc instalujemy go jak pozostałe moduły. Możemy to zrobić na trzy sposoby:

- Jeśli posiadamy zainstalowany program ActivePerl, a w związku z tym również program *PPM (ang. Perl Package Manager)*, użycie go jest najprawdopodobniej najprostszym wyjściem. W tym przypadku zainstalowanie interfejsu *DBI* ogranicza się do uruchomienia w wierszu poleceń programu *PPM* oraz wydania polecenia:

  ```
  >install DBI
  ```

  Reszta procesu instalacji przebiega w sposób automatyczny.

- Możemy również zainstalować moduł *DBI* przy użyciu kodu źródłowego. Najnowsza wersja kodu źródłowego dostępna jes zawsze pod adresem *http://www.symbolstone.org/technology/perl/DBI*. Pobieramy z archiwum plik zawierający kod źródłowy, a następnie rozpakowujemy go następującym poleceniem:

  ```
  >gzip -dc nazwa_pliku.tar.gz | tar -xvf
  ```

  W celu skompilowania kodu źródłowego i zainstalowania modułu należy teraz wykonać następujące cztery polecenia:

  ```
  >perl Makefile.pl
  >make
  >make test
  >make install
  ```

- Ostatnią możliwością jest wykorzystanie modułu eksportującego *CPAN* i zainstalowanie *DBI* bezpośrednio z tego archiwum. W takim wypadku, w wierszu poleceń należy wykonać dwa proste kroki:

  ```
  >perl -MCPAN -e -shell
  cpan>install DBI
  ```

  Po zainstalowaniu kończymy pracę za pomocą polecenia *quit*.

Teraz gdy posiadamy już zainstalowany interfejs *DBI*, musimy jedynie zainstalować sterownik do bazy danych *MySQL*, czyli *DBD::MySQL*. Jest on dostępny podobnie jak inne sterowniki do baz danych pod adresem: *http://www.symbolstone.org/technology/perl/DBI*. Możemy go zainstalować podobnie jak *DBI* korzystając z:

- programu *PPM*.
- kodu źródłowego.
- archiwum *CPAN*.

Gdy mamy już zainstalowane wszystkie potrzebne nam moduły, możemy przystąpić do pracy z bazą danych. Wykorzystanie baz danych przy użyciu DBI składa się z trzech podstawowych kroków:

- Nawiązanie połączenia za pomocą *DBI->connect()*.
- Wymiana danych z bazą przy użyciu zapytań w języku *SQL*.
- Rozłączenie z bazą danych za pomocą *DBI->disconnect()*.

3.1.1 Nawiązanie połączenia z bazą danych

Oczywiście, zanim rozpoczniemy korzystanie z bazy danych, musimy najpierw uzyskać z nią połączenie. Perl musi otrzymać informacje o tym, którego sterownika używać w tym celu, do której bazy danych się podłączyć, kto chce jej używać oraz wiele innych. Wszystko to można mu przekazać za pomocą funkcji *DBI->connect()*. Składnia tej funkcji wygląda następująco:

```
$dbh=DBI->connect('dbi:datasource','user','password');
```

Jako wynik otrzymujemy uchwyt do bazy danych, dzięki któremu możliwe jest odwoływanie się do danych. W zamian wymaga się podania trzech podstawowych informacji:

- Nazwy sterownika bazy danych *DBD*, wraz z nazwą bazy danych do której chcemy się podłączyć. Podawana jest ona w postaci jednego parametru *dbi:sterownik:nazwaźrodładanych*.
- Nazwa użytkownika używającego serwera bazy danych.
- Hasło identyfikujące użytkownika.

Aby na przykład uzyskać połączenie do bazy *MySQL* o nazwie *test*, z poziomu użytkownika *anonim* oraz wykorzystując hasło *abc123*, moglibyśmy napisać:

```
$dbh=DBI->connect('dbi:mysql:test','anonim','abc123');
```

3.1.2 Wymiana danych z bazą

Obecnie prawie wszystkie systemy baz danych wspierają zapytania skonstruowane w *SQL*. Tego rodzaju zapytania, mogą zostać podzielone na dwie różne grupy:

- Zapytania, które nie zwracają wyniku, na przykład tworzenie nowej tabelki czy dodawanie do tabelki nowych rekordów.
- Zapytania zwracające wynik, na przykład wyszukanie interesujących nas rekordów za pomocą zapytania *select*.

Interfejs *DBI* umożliwia zastosowanie podczas przekazywania zapytań do bazy danych planu, składającego się z czterech kroków:

- Przygotowanie uchwytu do zapytania *SQL* za pomocą metody *prepare*.
- Wykonanie zapytania w bazi danych za pomocą metody *execute*.
- Pobranie wyników za pomocą metod *fetch*.
- Kończymy wykonanie zapytania przy użyciu metody *finish*, informując tym samym bazę danych, że skończyliśmy.

Pierwszym krokiem, czynionym zaraz po nawiązaniu połączenia z bazą danych jest utworzeni instrukcji w *SQL*, przy użyciu metody prepare. Składnia tej metody wygląda następująco:

```
$dbh->prepare("zapytanie w SQL" );
```

Teraz, kiedy posiadamy uchwyt do instrukcji, możemy ją wykonać. Wykonujemy ją za pomocą metody *execute*. Składnia tej metody wygląda nastepująco:

```
$dbh->execute();
```

Spowoduje to wysłanie zapytania do bazy danych, przy użyciu pośredniczącego modułu sterownika. Następnie możemy odczytać wyniki zapytania. Istnieje kilka sposobów na pobranie wyników, trzeba do nich zaliczyć rodzinę metod *fetch*:

- Najprostszą metodą odczytywania informacji z interfejsu DBI jest pobranie od razu jednego wiersza, przy użyciu metody *fetchrow_array*. Zwraca ona żądane w wierszu kolumny jako tablice. Kolejność umieszczania w niej elementów odpowiada kolejności, w jakiej zostały one zażądane w oryginalnej instrukcji select. W celu pobrania rekordów z zapytania, a następnie ich wyświetlenia używamy pętli, na przykład:

```
while ( @row= $sth->fetchrow_array) {
       print "@row \n";
       }
```

- Możemy również używać metody fetchrow_arrayref. Zamiast tworzyć całkiem nową tablicę i zwracać ją za każdym wywołaniem, przekazuje ona odwołanie do wewnętrznej tablicy, która jest używana ponownie dla każdego następnego wiersza. Powtórne używanie tej tablicy oznacza, że Perl może oszczędzić czas i pamięć poprzez uniknięcie tworzenia za każdym razem nowej. Różnica w pętli *while* używanej do wyświetlenia wyników jest stosunkowo niewielka:

```
while (my $row_ref= $sth->fetchrow_arrayref) {
       print "@{$row_ref}\n";
       }
```

- Kolejną z metod jest *fetchrow_hashref*. Zwraca ona wiersze w postaci tablicy asocjacyjnej (zamiast zwykłej tablicy). Zawierać ona będzie nazwy kolumn jako klucze oraz ich zawartość jako wartości tablicy np:

```
foreach (my $href=$sth->fetchrow_hashref) {
  foreach (key %{$href}) {
         print "$_ => $href->{$_}\n";
   }
}
```

- Ostatnią z przedstawianych metod jest *fetchall_arrayref*. Pobiera ona wyniki zapytania za jednym razem, w postaci jednej wielkiej tablicy. Główną wadą tej metody jest to, że tablica utworzona przez *fetchall_arrayref* może zająć w przypadku dużej liczby rekordów olbrzynią ilość pamięci. A oto składnia tej metody:

```
my @results=$sth->fetchall_arrayref();
```

Kiedy tylko wszystkie wyniki zostaną pobrane, następuje wywołanie metody *finish*, w celu poinformowania bazy danych o skończeniu pracy. Składnia tej metody jest bardzo prosta:

```
$dbh->finish();
```

Dla wszystkich instrukcji *SQL*, które nie zwracają wartości (np: *create, insert, update, delete, drop)* interfejs *DBI* udostępnia metodę o nazwie *do*, która za jednym zamachem przygotowuje i wykonuje instrukcje. A o to składnia tej metody:

```
$dbh->do("zapytanie SQL");
```

3.1.3 Rozłączenie z bazą danych

Jak już zobaczyliśmy w pewnym zarysie, istnieje wiele różnych problemów wymagających rozważenia podczas tworzenia połączenia z bazą danych. Do zakończenia tego połączenia istnieje jeden prosty sposób. Trzeba po prostu użyć metody *disconnect* wraz z uchwytem do bazy danych:

```
$dbh->disconnect();
```

Możemy również nakazać *DBI* zakończenie wszystkich aktywnych połączeń, za pomocą metody *disconnect_all*:

```
DBI->disconnect_all();
```

3.1.4 Przykład

Skoro wiemy już w jaki sposób połączyć się, rozłączyć się oraz wymieniać danę z bazą danych, to przyszedł czas na prosty przykład pokazujący zastosowanie interfejsu *DBI*. Do przykładu potrzebna nam będzie tabela wraz z danymi w bazie danych *MySQL*. Niech tabela nazywa się *lista* i znajduje się w bazie o nazwie *test*. Przykładowa tabela z danymi może wyglądać tak:

```
+----+-------+------------+------------------+
| id | imie  | nazwisko   | email            |
+----+-------+------------+------------------+
|  1 | Jan   | Kowalski   | kowalski@wp.pl   |
|  2 | Marek | Nowak      | nowak@onet.pl    |
|  3 | Paweł | Malinowski | malinowski@wp.pl |
+----+-------+------------+------------------+
```

W przykładzie chcemy aby Perl wypisał nam na ekran wszystkie dane znajdujące się w tabeli *lista*.Z bazą danych *test* łączymy się z poziomu użytkownika *anonim*, który posiada hasło *abc123*. O to nasz przykładowy skrypt:

```
#!\usr\bin\perl

use strict;
use DBI;
my ($dbh, $sth);

$bdh=DBI->connect('dbi:mysql:test','anonim','abc123');

$sth=dbh->prepare("SELECT * from lista");

   $sth->execute();
      while (my @row= $sth->fetchrow_array) {
        print "@row\n";
      }
   $sth->finish();
```

```
        print"wszystko zrobione\n";

$dbh->disconnect();
```

Utworzyliśmy prosty przykład, wykorzystujący bazę danych *test*. Przyjrzyjmy się temu, co otrzymaliśmy. O to wynik działania naszego przykładowego skryptu perlowego:

```
>perl przyklad.pl
1 Jan Kowalski kowalski@wp.pl
2 Marek Nowak nowak@onet.pl
3 Paweł Malinowski malinowski@wp.pl
wszystko zrobione
```

Jak widać zostały wypisane wszystkie dane znajdujące się w tabeli *lista*, czyli zostało zrobione dokładnie to co zamierzaliśmy.

3.2 Opis skryptów

Integralną częścią naszej aplikacji są skrypty napisane w Perlu służące do importowania i aktualizacji bazy danych na podstawie danych z poszczególnych komputerów i od producenta systemu operacyjnego. W poniszych podrozdziałach zajmiemy się opisem skryptów wykorzystywanych przez naszą aplikacje. Pełen kod źródłowy tych skryptów znajduje się w rozdziale *Dodatek A*.

3.2.1 Moduł PatchDB

Wszystkie skrypty z wyjątkiem jednego intesywnie korzystają z bazy MySQL. Po tym jak napisaliśmy ich pierwsze wersje okazało się, że są fragmenty kodu często powtarzające się w różnych skryptach. Aby uniknąć powtarzania a jednocześnie ułatwić diagnostyke, napisaliśmy zestaw funkcji i umieściliśmy je we wspólnym module, który nazwaliśmy *PatchDB.pm*. Korzystają z niego wszystkie nasze skrypty perlowe.

Funkcje z modułu PatchDB

Każdy ze skryptów może być uruchomiony z opcją `-v` lub `-d` co powoduje włączenie odpowiednio trybu *verbose* lub *debug*. Zostały one wprowadzone po to by móc śledzić wykonanie skryptów i łatwiej dignozować błędy.

```
sub verbose {
    my ($msg) = @_;
    if ($verbose || $debug) {
        print $msg;
```

```
    }
}
```

Funkcja `verbose` ma za zadanie wypisać tekst podany jako argument o ile jesteśmy w trybie *verbose* lub *debug*.

```
sub debug {
    my ($msg) = @_;
    if ($debug) {
        print "$msg\n";
    }
}
```

Funkcja `debug` wypisuje tekst z argumentu jeśli jesteśmy w trybie *debug*.

W sytacji kiedy stwierdzamy wystąpienie nieprawidłowości i kontynacja wykonania programu nie ma sensu wołamy funkcję `error`.

```
sub error {
    my ($msg) = @_;
    print "ERROR: $self: $msg\n";
    exit 1;
}
```

Ta funkcja wypisuje na ekran podany komunikat o błędzie, poprzedzając go nazwą wykonywanego programu, przechowywaną w zmiennej `$self`. Po wypisaniu komunikatu program jest przerywany poprzez `exit`.

```
sub sql_connect {
  $dbh=DBI->connect("DBI:mysql:host=localhost:database=patchdb;"
                    ,"patchdb", "abc123");
  $dbh->do("/*!40101 set names latin1 */");
}
```

Funkcja `sql_connect` wołana jest przed rozpoczęcie wymiany danych z bazą MySQL. Nawiązywane jest połączenie z bazą i w przypadku MySQL nowszego niż 4.01.01 ustawiany jest domyślny zestaw znaków na latin1. Ustalenie zestawu znaku ma znaczenie przy sortowaniu i porównanich alfanumerycznych w bazie. Wydzielenie tego fragmentu kodu upraszcza kwestię zmiany hasła lub hosta bez konieczności modyfikacji wszystkich skryptów.

```
sub sql_disconnect {
    $dbh->disconnect();
}
```

Po zakończeniu pracy z bazą wołamy `sql_disconnect`. Być może wiele się nie oszczędza na wydzieleniu tego fragmentu kodu. Robimy to jednak dlatego, że zmienna `$dbh` jest zmienną prywatną w naszym module.

```
sub sql_exec {
    my ($query) = @_;
    debug($query);
my $result = $dbh->do($query);
    if (! $result) {
        sql_disconnect();
        error("MySQL: $Mysql::db_errstr\nquery:\n$query");
    }
return $result;
}
```

Powyższa funkcja ma za zadanie wykonać zapytanie SQL podane jako argument. Robimy to przy pomocy metody *do()*. Ta funkcja stosowana jest do zapytań SQL nie zwracających wartości. W trybie *debug* wypisane zostanie na ekran wykonywane zapytanie. Jeśli w trakcie realizacji zapytania w bazie wystąpi błąd, to następuje rozłączenie z bazą, wypisywany jest komunikat o błędzie i program jest przerywany.

```
sub sql_select {
    my ($query) = @_;
    debug($query);
    my $record = $dbh->selectrow_hashref($query);
    if ($Mysql::db_errstr ne "") {
        sql_disconnect();
        error("MySQL: $Mysql::db_errstr\nquery:\n$query");
    }
    return $record;
}
```

Funkcja `sql_select` służy do pobrania pierwszego wiersza odpowiedzi na przekazane zapytanie SQL. Obsługa błędów jest taka jak w `sql_exec`.

```
sub sql_query {
    my ($query, $subroutine) = @_;
    my ($sth, $record);
    debug($query);
    if (! ($sth = $dbh->prepare($query))) {
        error("MySQL: $Mysql::db_errstr\nquery:\n$query");
    }
    $sth->execute();

    while ($record = $sth->fetchrow_hashref) {
        &$subroutine($record);
    }
}
```

Funkcja `sql_query` pobiera dwa argumenty: zapytanie SQL typu select oraz wskaźnik do procedury, która ma być wykonana na poszczególnych wierszach

wyniku tego zapytania. Procedura ta powinna pobierać jeden argument będący tablicą asocjacyjną zawierającą pojedynczy wiersz wyniku z bazy. Obsługa błędów jest taka jak w `sql_exec`.

3.2.2 Odczytywanie daty z pliku patchdiag.xref

Tak jak wiemy plik *patchdiag.xref* jest codzienie aktualizowany przez firmę Sun, z pominięciem dni wolnych od pracy. Z punktu widzenia naszej aplikacji bardzo ważne jest posiadanie najaktualniejszej wersji pliku. Aby w sposób niezawodny i pewny stwierdzić, że pobrany z internetu plik *patchdiag.xref* jest aktualny należy odczytać zawartą w nim datę. Zadanie to wykonujemy za pomocą napisanego przez nas skryptu *gedate-xref.pl*. Sposób wywołania skryptu jest następujący:

```
> gedate-xref nazwa_pliku
```

W miejsce `nazwa_pliku` podajemy ścieżkę do pliku *patchdiag.xref*. Jako wynik na ekranie powinna zostać wypisana data w standardzie *ISO yyyy-mm-dd* (zgodnym ze standardem *MySQL*). Jeżeli w trakcie działania wystąpi błąd na ekranie zostanie wypisana odpowiednia informacja. Opiszemy teraz dokładnie sposób implementacji.

Implementacja

Zaczynamy od otworzenia pliku do czytania:

```
open(XREF, $xreffile) ||
       PatchDB::error("Cannot open file: $xreffile");
```

`XREF` to deskryptor otwartego pliku. Za jego pomocą w dalszej części programu odwołujemy się do naszego pliku. W zmiennej `$xreffile` przechowywana jest ścieżka do pliku *patchdiag.xref*.
Plik czytany jest w pętli wiersz po wierszu:

```
while (<XREF>) {
```

Z przeczytanego wiersza usuwany jest znak końca lini:

```
chomp;
```

Budowa pliku *patchdiag.xref* jest systematyczna i data którą chcemy odczytać jest umieszczona w pierwszj linijce po słowach `AS OF` zatem wystarczy przeczytać ciąg znaków różnych od spacji występujących po ty sformułowniu.wykonywane jest to za pomocą następującego wyrażenia reguralnego:

```
/^#.*AS OF ([^ ]+) /;
$reldate = $1
```

W zmiennej `$reldate` zapamiętywany jest ciąg znków odpowiadający dacie. Jeśli ten ciąg znaków jest pusty to oznacza, że przeczytany wiersz nie pasuje do naszego wyrażenia reguralnego i nie zawiera daty. Oznacza to błąd:

```
if ($reldate eq "") {
     error("release date scanning error");
}
```

Jeśli nie było błedu kończymy pętle:

```
last;
}
```

Ponieważ data odczytana z pliku jest w formacie niezgodnym z ISO należy ją przekonwertować:

```
($year, $month, $day) = Decode_Date_US($reldate);
```

Po odczytaniu składników daty wypisujemy je w formacie *yyyy-mm-dd*:

```
printf("%.4d-%.2d-%.2d\n",  $year, $month, $day);
```

Na koniec zamykamy otwarty wcześniej plik:

```
close(XREF);
```

3.2.3 Wstawianie danych z patchdiag.xref

Jak to już wcześniej zostało powiedziane plik *patchdiag.xref* zawiera niezwykle cenne dane dla naszej aplikacji. Ważne jest regularne wstawianie jego zawartości do bazy. Zadanie to wykonuje skrypt *insert-xref.pl*.

Sposób wywołania skryptu jest następujący:

```
> insert-xref [ -v | -d ] nazwa_pliku
```

W miejsce `nazwa_pliku` podajemy ścieżkę do pliku *patchdiag.xref*. Opcjami `-v`, `-d` włączamy tryb *verbose* lub *debug*. Wynikiem działania skryptu jest aktualizacja następujących tabel: *xref*, *xrefhistory*, *xrefarch*, *xrefcon*, *xrefreq*, *xrefwdrawn*, *xrefpkg*

Implementacja

Zaczynamy od połączenia się z naszą bazą danych *patchdb*:

```
PatchDB::sql_connect();
```

Czyścimy zawartość tabel *xref* i *xrefwdrawn*. Do tego celu używamy polecenia *truncate*, które usuwa tabele i zakłada ją na nowo. Jest to operacja szybsza niż kasowanie wierszy przez *delete*.

```
PatchDB::sql_exec("truncate xref");
PatchDB::sql_exec("truncate xrefwdrawn");
```

Przy wpisywaniu danych do tabel będziemy korzystali z tranzakcji. Dzieki wykorzystaniu tranzakcji przyśpieszymy działanie naszego programu:

```
PatchDB::sql_exec("start transaction");
```

Otwieramy plik do czytania. W przypadku jakiś komplikacji zgłaszany jest błąd za pomocą znanej nam już funkcji *PatchDB::error()* i następuje przerwanie działania programu:

```
open(XREF,$xreffile) ||
          PatchDB::error("Cannot open file: $xreffile");
```

`XREF` to deskryptor otwartego pliku. Za jego pomocą w dalszej części programu odwołujemy się do naszego pliku. W zmiennej `$xreffile` przechowywana jest ścieżka do pliku *patchdiag.xref*.
Plik czytany jest w pętli wiersz po wierszu:

```
while (<XREF>) {
```

Z przeczytanego wiersza usuwany jest znak końca linii:

```
chomp;
```

Pomijamy wiersze z komentarzami zaczynające się od znaku (#)

```
next if (/^#/);
```

Jeżeli gdzikolwie w przeczytanym wierszu występuje słowo WITHDRAWN, to zmienna `$withdrawn` ustawiana jest na jeden:

```
my $withdrawn = 1 if (/WITHDRAWN/);
```

Dane z przeczytanego wierzsza dzielimy ze względu na znak (|) i zapisujemy w odpowiednich zmiennych:

```
my ($patchid, $patchrev, $reldate, $rflag, $sflag, $oflag,
    $byflag, $osarch, $archs, $pkgs, $synopsis) = split(/\|/);
```

W trybie *debug* wypisywany jest cały przeczytany wiersz:

```
PatchDB::debug($_);
```

Z ciągów znaków zapisanych w zmiennej `$pkgs` i `$synopsis` pozbywamy się znków (") i (′), które powodowałyby błędy przy wpisywaniu do tabel danych:

```
$pkgs =~ s/(["'])/\\\1/g;
$synopsis =~ s/(["'])/\\\1/g;
```

Ponieważ data zapisana w zmiennej `$reldate` jest w formacie nie zgodnym z ISO należy ją przekonwertować:

```
my ($year, $month, $day) = Decode_Date_US($reldate);
```

Ponieważ te same dane, w tym samym formacie, będą zapisywane w trzech różnych tabelach, więc zapamiętujemy je w zmiennej `$values`, jako fragment zapytania SQL. Kolejność pól zdeterminowana jest strukturą tabeli *xref* i w tej właśnie kolejności wstawiane są odpowiednie wartości.

```
my $values = sprintf("values(%d, %d, '%s', '%s', '%s', '%s',
                          '%s', '%s', '%s', '%s')",
     $patchid, $patchrev, "$year-$month-$day",
         $rflag eq "R"? "Y" : "N",
         $sflag eq "S"? "Y" : "N",
         $oflag eq "O"? "Y" : "N",
         $bflag =~ m/B/? "Y" : "N",
         $yflag =~ m/Y/? "Y" : "N",
             $osarch, $synopsis);
```

Jeżeli przeczytane inforacje dotyczą wycofanego patch'a (zmienna `$withdrawn` ustawiona na 1), to wypełniamy tabelę *xrefwdrawn* i przechodzimy do następnego wiersza pliku. W przeciwnym razie wypełniana jest tabela *xref*.

```
if ($withdrawn) {
     PatchDB::sql_exec("insert into xrefwdrawn $values");
     next;
 } else {
     PatchDB::sql_exec("insert into xref $values");
 }
```

Informacje o patch'u wstawiane są również do tabeli *xrefhistory*. Ponieważ w tej tabeli dane o patcha'ch są kumulowane używamy opcji *ignore* tak aby próba wstawienia danych które już są w tej tabeli nie powodowała przerwania z błędem. `PatchDB::sql_exec` zwraca wartość 1, gdy wstawiono wiersz do tabeli. Oznacza to, że wstawiany patch jest nowy, w związku z tym, należy odpowiednio uzupełnić tabele *xrefarch*, *xrefcon*, *xrefreq*, *xrefpkg*.

```
if (PatchDB::sql_exec
          ("insert ignore into xrefhistory $values") == 1) {
```

Poniższa pętla dopisuje dane do tabel *xrefreq* i *xrefarch*. W kolumnie którą nazwaliśmy `$archs` znajduje się lista wymaganych patch'y oraz lista platform na których patch może być instalowany. W kolumnie tej znakiem (;) rozdzielono od siebie poszczególne wymagane patch'e oraz listę platform. Rozbijamy `$archs` ze względu na (;). Otrzymujemy w ten sposób liste napisów. Kolejno sprawdzamy te napisy czy są numerami patch'a, jeśli tak to uzupełniami tabelę *xrefreq*, jeśli nie, to napis jest listą platform rozdzielonych (,), którą trzeba dopisać do tabeli *xrefarch*.

```
foreach my $item (split(/;/, $archs)) {
  $item =~ /([0-9]{6})-([0-9]{2})/;
```

```
  my $id = $1;
  my $rev = $2;
  if ($id && $rev) {
     PatchDB::sql_exec("insert ignore into xrefreq values
                         ($patchid, $patchrev, $id, $rev)");
   } else {
      foreach my $arch (split(/,/, $item)) {
        PatchDB::sql_exec("insert ignore into xrefarch values
                              ($patchid, $patchrev, '$arch')");
        }
    }
}
```

Kolumna `$pkgs` jest listą pakietów i patch'y konfliktowych. Elementy na tej liście rozdzielone są od siebie znakiem (;). Czytamy je kolejno w pętli i sprawdzamy czy dany element jest numerem patch'a. Jeśli tak, to takiego patch'a dopisujemy do tabeli *xrefcon*. W przeciwnym razie mamy doczynienia z pakietem. W tej sytacji informacje o pakiecie dzielimy na znaku (:), otrzymujemy identyfikator i wersję pakietu. Dane te dopisujemy do tabeli *xrefpkg*.

```
foreach my $item (split(/;/,$pkgs)) {
    $item =~ /([0-9]{6})-([0-9]{2})/;
    my $id = $1;
    my $rev = $2;
    if ($id && $rev) {
        PatchDB::sql_exec("insert ignore into xrefcon values
                             ($patchid, $patchrev, $id, $rev)");
     } else {
        my ($pkg, $version) = split(/:/, $item);
        if ($pkg && $version && $pkg =~ /^[A-Z]/) {
            PatchDB::sql_exec("insert into xrefpkg values
            (     $patchid, $patchrev, '$pkg', '$version')");
            }
        }
    }
}
```

Zamykamy otwarty wcześniej plik:

```
close(XREF);
```

Kończymy tranzakcję:

```
PatchDB::sql_exec("commit");
```

Na koniec działania naszego skryptu następuje rozłączenie z bazą danych:

```
PatchDB::sql_disconnect();
```

3.2.4 Pobieranie z Internetu pliku patchdiag.xref

Wyjątkowo ten skrypt jest skryptem shelowym. Jego zadaniem jest pobranie z Internetu pliku *patchdiag.xref* i zawołanie wcześniej opisanego skryptu *insert-xref.pl*. Nie pobiera on żadnych argumentów. Dostosowany jest do automatycznego uruchamiania codziennie. Dodatkowo jest możliwość kolekcjonowania pobranych plików. Starsze pliki mogą być nam potrzebne gdybyśmy chcieli odbudować bazę danych o patch'ach.

Implementacja

Zaczynamy od ustawienia zmiennej w której będzie przechowywana ścieżka do pliku tymczasowego, w któртm zapisany zostanie plik pobierany z Internetu. Wykorzystamy tutaj program *mktemp*, który generuje unikalną nazwę dla pliku tymczasowego. Plik ten będzi umieszczony w katalogu /tmp.

```
TMPFILE=‘mktemp -p /tmp patchdiag.xref.XXXXXX‘
```

W zmienej `XREFDIR` umieszcza się ścieżkę do katalogu gdzie będą kolekcjonowane pliki *patchdiag.xref*. Jeśli nie chcemy zbierać tych plików nie ustawiamy tej zmienej.

```
XREFDIR=/export/home/patchdb/xref
```

Ustawiamy zmienną środowiskową `PATH`, tak aby mieć dostęp do wołanych w tym skrypcie programów, łącznie z *insert-xref.pl*. Dlatego na końcu dołączamy ścieżkę, z której uruchamiany jest bierzący skrypt.

```
PATH=/usr/bin:/usr/sfw/bin:/opt/sfw/bin:
                           /opt/cfw/bin/:‘dirname $0‘
export PATH
```

Jeśli istnieje plik tymczasowy to za pomocą programu *wget* zapisujemy w nim plik *patchdiag.xref* pobrany z Internetu.

```
if [ -f $TMPFILE ]
then
   wget -q -c -nd -t 0 -O $TMPFILE
    "http://sunsolve.sun.com/pub-cgi/pdownload.pl?
                                        target=patchdiag.xref"
```

W przypadku gdy gromadzimy pliki *patchdiag.xref* to z pobranego pliku odczytywana jest data jego wydania przy pomocy *getdate-xref.pl*, następnie jeśli pliku o tej dacie jeszcze nie mamy, to jest on kopiowany do wskazanego miejsca. W nazwie zapisywanego pliku umieszczamy datę jego wydania. Program *insert-xref.pl* wołany jest gdy pobrany plik jest nowy lub gdy nie zbieramy plików *patchdiag.xref*.

```
if [ -d $XREFDIR ]
 then
     RELDATE=`getdate-xref.pl $TMPFILE | sed s/-//g`
     XREFFILE=${XREFDIR}/patchdiag.xref.$RELDATE
     if [ ! -f $XREFFILE ]
     then
         cp $TMPFILE ${XREFDIR}/patchdiag.xref.$RELDATE
         insert-xref.pl $TMPFILE
     fi
 else
     insert-xref.pl $TMPFILE
 fi
```

Na koniec usuwamy plik tymczasowy.

```
rm -f $TMPFILE
```

Jeśli nie udało się utworzyć pliku tymczasowego to zgłaszany jest błąd.

```
else
 echo "ERROR: unable to create temporary file: $TMPFILE"
fi
```

3.2.5 Import danych o zainstalowanych patch'ach

O tym jakie patch'e są zainstalowane w systemie Solaris możemy się dowiedzieć za pomocą programu *showrev*. Poniżej przedstawiamy fragment wyniku działania *showrev -p*:

```
Patch: 116531-01 Obsoletes:  Requires:  Incompatibles:  Packages:SUNWcar
Patch: 114324-05 Obsoletes:  Requires:  Incompatibles:  Packages: SUNWos86r
Patch: 115335-01 Obsoletes:  Requires:  Incompatibles:  Packages: SUNWos86r
Patch: 115875-01 Obsoletes:  Requires:  Incompatibles:  Packages: SUNWos86r
Patch: 115318-01 Obsoletes:  Requires:  Incompatibles:  Packages: SUNWtoo
Patch: 114737-01 Obsoletes:  Requires:  Incompatibles:  Packages: SUNWnisu
```

Tak uzyskane dane importujemy do bazy danych za pomocą programu *insert-showrev.pl*. Skrypt ten woła się w następujący sposób:

```
> insert-showrev [ -v | -d ] nazwa_pliku komputer
```

W miejsce `nazwa_pliku` podajemy ścieżkę do pliku zawierającego wynik *showrev -p*, natomiast w miejsce `komputer` wstawiamy identyfikator komputera z tabeli *hosts*. Opcjami `-v`, `-d` włączamy tryb *verbose* lub *debug*. Efektem działania tego skryptu jest wypełnienie tabeli *showrev*.

Implementacja

Zaczynamy od połączenia się z bazą danych:

```
PatchDB::sql_connect();
```

Zaczynamy tranzakcję:

```
PatchDB::sql_exec("start transaction");
```

Otwieramy plik do czytania. W razie problemów kończymy działanie programu za pomocą funkcji *error*.

```
open(SFILE, $sfile) ||
     PatchDB::error("Cannot open file: $sfile");
```

Plik czytamy w pętli wiersz po wierszu:

```
while (<SFILE>) {
```

Usuwamy znak końca linii:

```
chomp;
```

Sprawdzamy czy na początku wczytanego wiersza jest napis "Patch:", po którym następuje ciąg sześciu cyfr, znak (-) i dwie cyfry. Jeśli tak, to mamy identyfikator patch'a i jego wersję.

```
m/^Patch: ([0-9]{6})-([0-9]{2}) /;
my $patchid = $1;
my $patchrev = $2;
```

Jeśli identyfikator i wersja patch'a są określone to dopisujemy je do tabeli *showrev*, podając identyfikator komputera i bierzącą datę, w przeciwnym razie zgłaszamy błąd.

```
if ($patchid && $patchrev) {
     PatchDB::sql_exec("insert ignore into showrev values
               ($hostid, $patchid, $patchrev, '$today')");
 } else {
     PatchDB::error("Scan error");
 }
```

Zamykamy otwarty wcześniej plik:

```
close(SFILE);
```

Kończymy tranzakcję:

```
PatchDB::sql_exec("commit");
```

Rozłączamy się z bazą danych:

```
PatchDB::sql_disconnect();
```

3.2.6 Import danych o zainstalowanych pakietach

W systemi Solaris oprogramowanie instalowane jest w postaci pakietów. Każdy taki pakiet to zestaw plików (programów, bibliotek, konfiguracji) do wgrania do systemu plików komputera, wraz ze skryptami uruchamianymi przed i po instalacji jeśli to konieczne. System Solaris pamięta wszystkie zainstalowane pakiety, co więcej, przechowywana jest w nim lista wszystkich zainstalowanych plików z informacją do jakiego pakietu należą. Do zarządzania pakietami służy cały zestaw specjalnych programów. Przy pomocy jednego z nich możemy uzyskać pełną informację na temat zainstalowanego oprogramowania. Ten program to *pkginfo*. Poniżej przedstawiamy fragment wyniku zawołania *pkginfo -l*:

```
   PKGINST:  AMImega
      NAME:  MEGA Family SCSI Host Bus Adapter
  CATEGORY:  system
      ARCH:  i386
   VERSION:  1.1.0,REV=2002.02.23.14.36
   BASEDIR:  /
    VENDOR:  American Megatrends Inc.
      DESC:  MEGA Family SCSI Host Bus Adapter
    PSTAMP:  burn20020223143624
  INSTDATE:  Aug 18 2003 20:16
   HOTLINE:  Please contact your local service provider
    STATUS:  completely installed
     FILES:       11 installed pathnames
                   8 shared pathnames
                   7 directories
                   1 executables
                 113 blocks used (approx)

   PKGINST:  CADP160
      NAME:  Adaptec Ultra160 SCSI Host Adapter Driver
  CATEGORY:  system
      ARCH:  i386
   VERSION:  1.21,REV=2002.02.23.14.36
   BASEDIR:  /
    VENDOR:  Adaptec
      DESC:  Adaptec Ultra160 SCSI Host Adapter Driver
    PSTAMP:  burn20020223143624
  INSTDATE:  Aug 18 2003 20:16
   HOTLINE:  Please contact your local service provider
    STATUS:  completely installed
     FILES:       12 installed pathnames
                   9 shared pathnames
                   8 directories
                   3 executables
                 528 blocks used (approx)
```

Jak widać informacje o danym pakiecie to pewien zestaw zmiennych z przypisanymi wartościami.

Do importowania danych o zainstalowanym oprogramowaniu służy program *insert-pkginfo.pl*. Uruchamiamy go w poniższy sposób:

```
> insert-pkginfo [ -v | -d ] nazwa_pliku komputer
```

W miejsce `nazwa_pliku` podajemy ścieżkę do pliku zawierającego wynik *pkginfo -l*, natomiast w miejsce `komputer` wstawiamy identyfikator komputera z tabeli *hosts*. Opcjami `-v`, `-d` włączamy tryb *verbose* lub *debug*. Rezultatem działania tego skryptu jest wypełnienie tabeli *pkginfo*.

Implementacja

Zaczynamy od połączenia się z bazą danych:

```
PatchDB::sql_connect();
```

Rozpoczynamy tranzakcję:

```
PatchDB::sql_exec("start transaction");
```

Otwieramy plik do czytania. W razie problemów kończymy działanie programu za pomocą funkcji *error*.

```
open(PFILE, $pfile) ||
     PatchDB::error("Cannot open file: $pfile");
```

Otwarty wcześniej plik czytamy w pętli wiersz po wierszu:

```
while (<PFILE>) {
```

Usuwamy znak końca linii:

```
 chomp;
```

Jeśli w przeczytanym wierszu występuje nazwa pola pisana wielkimi literami, oddzielona znakiem (:) od wartości, to wartość tę zapamiętujemy w tablicy asocjacyjnej `$pkginfo` pod indeksem będącym nazwą pola.

```
if (/[ ]+([A-Z]+):  (.*)/) {
     $pkginfo{$1} = $2;
```

W przeciwnym razie jesteśmy na końcu opisu pakietu. Sprawdzamy czy zostało wypełnione pole `PKGINST` zawierające identyfikator pakietu. Jeśli tak to mamy komplet danych do wstawienia do bazy. Poleceniem `sql_exec` wstawiamy rekord zawierający identyfikator komputera, nazwę pakietu i jego wersję do tabeli *pkginfo*.

```
else {
     if ($pkginfo{PKGINST}) {
         PatchDB::sql_exec("insert into pkginfo values
         ($hostid, '$pkginfo{PKGINST}', '$pkginfo{VERSION}')");
         undef %pkginfo;
     }
 }
```

Zamykamy otwarty wcześniej plik:

```
close(PFILE);
```

Kończymy tranzakcję:

```
PatchDB::sql_exec("commit");
```

Na koniec rozłączamy się z bazą danych:

```
PatchDB::sql_disconnect();
```

Dodatek A

Skrypty

A.1 Moduł PatchDB.pm

```
#
# Synopsis: Solaris Patch Database Perl Module, provides basic
# functionality for other Perl scripts
# Last modified: Jun 14, 2005
#
# Copyright (c) 2004-2005 Bruce Riddle
# Copyright (c) 2005 Andrzej Drobnikowski & Mariusz Zynel
#
# This software is FREE.  You can use and/or redistribute it for any
# purpose in either, modified, or unmodified form, provided that the
# above copyright notice  and this  permission  are  included  in all
# copies or  substantial portions of this software.
#
# THIS  SOFTWARE IS PROVIDED AS IS AND COME WITH NO WARRANTY OF ANY
# KIND, EITHER EXPRESSED OR IMPLIED. IN NO EVENT WILL THE COPYRIGHT
# HOLDER BE  LIABLE FOR ANY DAMAGES RESULTING FROM THE  USE OF THIS
# SOFTWARE.

package PatchDB;

use POSIX;
use DBI;

my $self; ($self = $0) =~ s/.*\///;

my $dbh;

my $verbose = 0;
my $debug = 0;

sub set_verbose {
    ($verbose) = @_;
}
```

```
sub verbose {
    my ($msg) = @_;
    if ($verbose || $debug) {
        print $msg;
    }
}

sub set_debug {
    ($debug) = @_;
}

sub debug {
    my ($msg) = @_;
    if ($debug) {
        print "$msg\n";
    }
}

sub error {
    my ($msg) = @_;
    print "ERROR: $self: $msg\n";
    exit 1;
}


sub sql_connect {
    $dbh = DBI->connect("DBI:mysql:host=localhost:database=patchdb;",
                        "patchdb", "abc123");
    $dbh->do("/*!40101 set names latin1 */");
}

sub sql_disconnect {
    $dbh->disconnect();
}

sub sql_exec {
    my ($query) = @_;
    debug($query);
my $result = $dbh->do($query);
    if (! $result) {
        sql_disconnect();
        error("MySQL: $Mysql::db_errstr\nquery:\n$query");
    }
return $result;
}

sub sql_select {
    my ($query) = @_;
    debug($query);
    my $record = $dbh->selectrow_hashref($query);
    if ($Mysql::db_errstr ne "") {
        sql_disconnect();
```

```
        error("MySQL: $Mysql::db_errstr\nquery:\n$query");
    }
    return $record;
}

sub sql_query {
    my ($query, $subroutine) = @_;
    my ($sth, $record);
    debug($query);
    if (! ($sth = $dbh->prepare($query))) {
        error("MySQL: $Mysql::db_errstr\nquery:\n$query");
    }
    $sth->execute();

    while ($record = $sth->fetchrow_hashref) {
        &$subroutine($record);
    }
}

1;
```

A.2 Skrypt gedate-xref.pl

```
#!/opt/cfw/bin/perl
#
# Synopsis: Read release date from patchdiag.xref file
#
# Last modified: Jun 14, 2005
#
# Copyright (c) 2004-2005 Bruce Riddle
# Copyright (c) 2005 Andrzej Drobnikowski & Mariusz Zynel
#
# This software is FREE.  You can use and/or redistribute it for any
# purpose in either, modified, or unmodified form, provided that the
# above copyright notice  and this  permission  are  included in all
# copies or  substantial  portions of this software.
#
# THIS SOFTWARE IS PROVIDED  AS IS AND COME WITH NO WARRANTY  OF ANY
# KIND, EITHER  EXPRESSED OR IMPLIED. IN NO EVENT WILL THE COPYRIGHT
# HOLDER  BE LIABLE  FOR ANY DAMAGES  RESULTING FROM THE USE OF THIS
# SOFTWARE.

use strict;
use FindBin;
use lib $FindBin::Bin;

use PatchDB;

use Date::Calc qw(Decode_Date_US);

my ($self, $xreffile);
my ($reldate, $year, $month, $day);

sub usage {
    print "usage: $PatchDB::self filename\n";
    exit 1;
}

if ($#ARGV != 0) {
    usage();
}

$xreffile = @ARGV[0];

open(XREF, $xreffile) || PatchDB::error("Cannot open file: $xreffile");

while (<XREF>) {
    chomp;
    /^#.*AS OF ([^ ]+) /;
    $reldate = $1;
    if ($reldate eq "") {
        PatchDB::error("Release date scanning error");
    }
```

```
    last;
}

($year, $month, $day) = Decode_Date_US($reldate);

printf("%.4d-%.2d-%.2d\n",  $year, $month, $day);

close(XREF);
```

A.3 Skrypt insert-xref.pl

```
#!/opt/cfw/bin/perl
#
# Synopsis: Insert patchdiag.xref into database
#
# Last modified: Jun 16, 2005
#
# Copyright (c) 2004-2005 Bruce Riddle
# Copyright (c) 2005 Andrzej Drobnikowski & Mariusz Zynel
#
# This software is FREE.  You can use and/or redistribute it for any
# purpose in either, modified, or unmodified form, provided that the
# above copyright notice  and this  permission  are included  in all
# copies or  substantial portions of this software.
#
# THIS  SOFTWARE IS  PROVIDED AS IS AND COME WITH NO WARRANTY OF ANY
# KIND, EITHER EXPRESSED OR IMPLIED. IN NO EVENT WILL THE  COPYRIGHT
# HOLDER BE  LIABLE FOR ANY DAMAGES  RESULTING FROM THE USE  OF THIS
# SOFTWARE.

use strict;
use FindBin;
use lib $FindBin::Bin;

use PatchDB;

use Getopt::Std;
use Date::Calc qw(Decode_Date_US);

our ($opt_v, $opt_d);

sub usage {
    print "usage: $PatchDB::self [ -v | -d ] filename\n";
    exit 1;
}

if (! getopts('vd') || ($opt_v && $opt_d)) {
    usage();
}

if ($opt_v) {
    PatchDB::set_verbose(1);
}

if ($opt_d) {
    PatchDB::set_debug(1);
}

if ($#ARGV != 0) {
    usage();
}
```

```
my $xreffile = @ARGV[0];

PatchDB::sql_connect();

PatchDB::sql_exec("truncate xref");
PatchDB::sql_exec("truncate xrefwdrawn");
PatchDB::sql_exec("start transaction");

open(XREF, $xreffile) ||
     PatchDB::error("Cannot open file: $xreffile");

while (<XREF>) {
    chomp;
    next if (/^#/);

    my $withdrawn = 1 if (/WITHDRAWN/);

    my ($patchid, $patchrev, $reldate, $rflag, $sflag, $oflag,
        $byflag, $osarch, $archs, $pkgs, $synopsis) = split(/\|/);

    PatchDB::debug($_);

    $pkgs =~ s/(["'])/\\\1/g;
    $synopsis =~ s/(["'])/\\\1/g;

    my ($year, $month, $day) = Decode_Date_US($reldate);

    my $values = sprintf("values(%d, %d, '%s', '%s', '%s', '%s', '%s',
                                  '%s', '%s', '%s')",
        $patchid, $patchrev, "$year-$month-$day",
            $rflag eq "R" ? "Y" : "N",
            $sflag eq "S" ? "Y" : "N",
            $oflag eq "O" ? "Y" : "N",
            $byflag =~ m/B/ ? "Y" : "N",
            $byflag =~ m/Y/ ? "Y" : "N",
                $osarch, $synopsis);

    if ($withdrawn) {
        PatchDB::sql_exec("insert into xrefwdrawn $values");
        next;
    } else {
        PatchDB::sql_exec("insert into xref $values");
    }

    if (PatchDB::sql_exec
                ("insert ignore into xrefhistory $values") == 1) {
        foreach my $item (split(/;/, $archs)) {
            $item =~ /([0-9]{6})-([0-9]{2})/;
            my $id = $1;
            my $rev = $2;
            if ($id && $rev) {
```

```
                PatchDB::sql_exec("insert ignore into xrefreq values
                                ($patchid, $patchrev, $id, $rev)");
            } else {
                foreach my $arch (split(/,/, $item)) {
                    PatchDB::sql_exec("insert ignore into xrefarch values
                                    ($patchid, $patchrev, '$arch')");
                }
            }
        }
        foreach my $item (split(/;/,$pkgs)) {
            $item =~ /([0-9]{6})-([0-9]{2})/;
            my $id = $1;
            my $rev = $2;
            if ($id && $rev) {
                PatchDB::sql_exec("insert ignore into xrefcon values
                                ($patchid, $patchrev, $id, $rev)");
            } else {
                my ($pkg, $version) = split(/:/, $item);
                if ($pkg && $version && $pkg =~ /^[A-Z]/) {
                    PatchDB::sql_exec("insert into xrefpkg values
                        ($patchid, $patchrev, '$pkg', '$version')");
                }
            }
        }
    }
}

close(XREF);

PatchDB::sql_exec("commit");

PatchDB::sql_disconnect();
```

A.4 Skrypt update-xref.pl

```
#!/bin/sh
#
# Synopsis:  Get patchdiag.xref from SunSolve, add to a local repository
# and insert its contents into database.
#
# Last modified: May 24, 2005
#
# Copyright (c) 2004-2005 Bruce Riddle
# Copyright (c) 2005 Andrzej Drobnikowski & Mariusz Zynel
#
# This software is FREE.  You can use and/or redistribute it for any
# purpose in either, modified, or unmodified form, provided that the
# above copyright notice  and this  permission  are  included in all
# copies or  substantial portions of this software.
#
# THIS  SOFTWARE IS  PROVIDED  AS IS AND COME WITH NO WARRANTY OF ANY
# KIND, EITHER  EXPRESSED  OR IMPLIED. IN NO EVENT WILL THE COPYRIGHT
# HOLDER  BE LIABLE  FOR ANY DAMAGES  RESULTING  FROM THE USE OF THIS
# SOFTWARE.


TMPFILE=‘mktemp -p /tmp patchdiag.xref.XXXXXX‘
XREFDIR=/export/home/patchdb/xref

PATH=/usr/bin:/usr/sfw/bin:/opt/sfw/bin:/opt/cfw/bin/:‘dirname $0‘
export PATH

if [ -f $TMPFILE ]
then
    # Download patchdiag.xref from SunSolve

 wget -q -c -nd -t 0 -O $TMPFILE
  "http://sunsolve.sun.com/pub-cgi/pdownload.pl?target=patchdiag.xref"

    # Add the file to the local respository if such exists. Do that
    # before inserting into database to not loose the file in case
    # insertion fails.

    if [ -d $XREFDIR ]
    then
        RELDATE=‘getdate-xref.pl $TMPFILE | sed s/-//g‘
        XREFFILE=${XREFDIR}/patchdiag.xref.$RELDATE
        if [ ! -f $XREFFILE ]
        then
            cp $TMPFILE ${XREFDIR}/patchdiag.xref.$RELDATE
            insert-xref.pl $TMPFILE
        fi
    else
        insert-xref.pl $TMPFILE
    fi
```

```
    # Cleanup temporary files

    rm -f $TMPFILE
else
    echo "ERROR: unable to create temporary file: $TMPFILE"
fi
```

A.5 Skrytp insert-showrev.pl

```
#!/opt/cfw/bin/perl
#
# Synopsis: Insert showrev  file into database
#
# Last modified: Jun 14, 2005
#
# Copyright (c) 2004-2005 Bruce Riddle
# Copyright (c) 2005 Andrzej Drobnikowski & Mariusz Zynel
#
# This software is FREE.  You can use and/or redistribute it for any
# purpose in either, modified, or unmodified form, provided that the
# above copyright notice  and this  permission  are  included in all
# copies or  substantial portions of this software.
#
# THIS  SOFTWARE IS  PROVIDED AS IS AND COME WITH NO WARRANTY OF ANY
# KIND, EITHER  EXPRESSED OR IMPLIED. IN NO EVENT WILL THE COPYRIGHT
# HOLDER BE LIABLE  FOR ANY DAMAGES  RESULTING FROM THE USE  OF THIS
# SOFTWARE.

use strict;
use FindBin;
use lib $FindBin::Bin;

use PatchDB;

use Getopt::Std;
use Date::Calc qw(Today);

our ($opt_v, $opt_d);

sub usage {
    print "usage: $PatchDB::self [ -v | -d ] filename hostid\n";
    exit 1;
}

if (! getopts('vd') || ($opt_v && $opt_d)) {
    usage();
}

if ($opt_v) {
    PatchDB::set_verbose(1);
}

if ($opt_d) {
    PatchDB::set_debug(1);
}

if ($#ARGV != 1) {
    usage();
}
```

```
my $sfile = @ARGV[0];
my $hostid = @ARGV[1];

my ($year, $month, $day) = Today();
my $today = sprintf("%d-%.2d-%.2d", $year, $month, $day);

PatchDB::sql_connect();


PatchDB::sql_exec("start transaction");

open(SFILE, $sfile) || PatchDB::error("Cannot open file: $sfile");

while (<SFILE>) {
    chomp;

    m/^Patch: ([0-9]{6})-([0-9]{2}) /;

    my $patchid = $1;
    my $patchrev = $2;

    if ($patchid && $patchrev) {
        PatchDB::sql_exec("insert ignore into showrev values($hostid,
                                        $patchid,$patchrev, '$today')");
    } else {
        PatchDB::error("Scan error");
    }
}

close(SFILE);

PatchDB::sql_exec("commit");

PatchDB::sql_disconnect();
```

A.6 Skrypt insert-pkginfo.pl

```
#!/opt/cfw/bin/perl
#
# Synopsis: Insert pkginfo file into database
#
# Last modified: Jun 14, 2005
#
# Copyright (c) 2004-2005 Bruce Riddle
# Copyright (c) 2005 Andrzej Drobnikowski & Mariusz Zynel
#
# This software is FREE.  You can use and/or redistribute it for any
# purpose in either, modified, or unmodified form, provided that the
# above copyright notice  and this  permission  are  included in all
# copies or  substantial portions of this software.
#
# THIS  SOFTWARE IS  PROVIDED AS IS AND COME WITH NO WARRANTY OF ANY
# KIND, EITHER EXPRESSED  OR IMPLIED. IN NO EVENT WILL THE COPYRIGHT
# HOLDER BE  LIABLE FOR ANY  DAMAGES RESULTING  FROM THE USE OF THIS
# SOFTWARE.

use strict;
use FindBin;
use lib $FindBin::Bin;

use PatchDB;

use Getopt::Std;

my %pkginfo;

our ($opt_v, $opt_d);

sub usage {
    print "usage: $PatchDB::self [ -v | -d ] filename hostid\n";
    exit 1;
}

if (! getopts('vd') || ($opt_v && $opt_d)) {
    usage();
}

if ($opt_v) {
    PatchDB::set_verbose(1);
}

if ($opt_d) {
    PatchDB::set_debug(1);
}

if ($#ARGV != 1) {
    usage();
```

```
}

my $pfile = @ARGV[0];
my $hostid = @ARGV[1];

PatchDB::sql_connect();

PatchDB::sql_exec("start transaction");

open(PFILE, $pfile) || PatchDB::error("Cannot open file: $pfile");

while (<PFILE>) {
    chomp;
    if (/[ ]+([A-Z]+):  (.*)/) {
        $pkginfo{$1} = $2;
    } else {
        if ($pkginfo{PKGINST}) {
            PatchDB::sql_exec("insert into pkginfo values($hostid,
                        '$pkginfo{PKGINST}', '$pkginfo{VERSION}')");
            undef %pkginfo;
        }
    }
}

close(PFILE);

PatchDB::sql_exec("commit");

PatchDB::sql_disconnect();
```

Bibliografia

[1] Simon Cozens, *Perl od podstaw*, Helion, 2003.

[2] Dokumentacja systemu Solaris
http://docs.sun.com

[3] Dokumentacja techniczna MySQL
http://www.mysql.com

[4] Dokumentacja i opisy Perl
http://www.perl.org

[5] Sunsolve, Centrum informacji o poprawkach dla Solaris
http://sunsolve.sun.com