Pliki, jako dyskowe zbiory danych, są najprostszym sposobem magazynowania i przechowywania danych. W przypadku PHP obsługa plików przypomina do złudzenia tę znaną z języka C, więc czytelnicy, którzy mieli już do czynienia z C, powinni bardzo szybko wdrożyć się w zagadnienie, które jest tematem tego rozdziału.

W rozdziale tym wyjaśniono, w jaki sposób odczytywać pliki z dysku twardego lokalnego komputera oraz z odległego serwera, jak zapisywać dane do plików, oraz przedstawiono klika aspektów bezpieczeństwa związanych z obsługą plików. Niezależnie od tego, że bezpieczeństwu serwisów WWW poświęcono osobny rozdział (rozdział 8.), problemy bezpieczeństwa często będą się przewijać podczas prezentacji różnych technik i metod programowania i tworzenia stron WWW w PHP — po prostu zagadnień bezpieczeństwa nie można traktować w sposób oderwany.

Odczyt pliku z dysku lokalnego

Odczyt danych z pliku zapisanego na dysku lokalnym komputera (w przypadku skryptów PHP komputerem lokalnym jest serwer) odbywa się według prostego algorytmu: otwarcie pliku, odczyt danych, zamknięcie pliku. Termin otwarcie pliku to, w uproszczeniu, informacja dla systemu, że będziemy korzystać z danego pliku, i związana z tym rezerwacja odpowiednich zasobów. W wyniku tej czynności otrzymamy deskryptor, czyli identyfikator lub — mówiąc jeszcze inaczej — uchwyt pliku, którego należy używać przy wykonywaniu na nim wszelkich kolejnych operacji. Do otwarcia pliku służy funkcja fopen().

Wywołanie fopen ma schematyczną postać:

fopen('nazwa_pliku', 'tryb_otwarcia')

Argument nazwa_pliku to ciąg znaków wskazujący nazwę pliku, który należy otworzyć. Może on zawierać zarówno względną, jak i bezwzględną ścieżkę dostępu. Jeśli ścieżka dostępu nie zostanie podana, plik będzie poszukiwany w katalogu bieżącym. Prawidłowe są zatem przykładowe wartości:

/usr/tmp/abc/plik1.zip

../bcd/plik2.zip

./plik3.zip

plik4.zip

Parametr tryb_otwarcia określa tryb otwarcia pliku i może przyjmować wartości przedstawione w tabeli 6.1.

Tabela 6.1. Tryby otwierania plików

Tryb

Zachowanie funkcji fopen()

r

Otwarcie pliku o nazwie podanej jako pierwszy parametr w trybie do odczytu.

r+

Otwarcie pliku w trybie do odczytu i zapisu.

w

Otwarcie pliku do zapisu. Jeśli plik o podanej nazwie nie istnieje, zostanie utworzony. Jeśli plik istnieje, jego zawartość jest

w+

Otwarcie pliku do zapisu i odczytu. Jeśli plik nie istnieje, zostanie utworzony. Jeśli plik istnieje, jego zawartość jest tracon

a

Otwarcie pliku w trybie dołączania (dane będą zapisywane na koniec pliku — zawartość pliku nie będzie utracona). Jeśli pl

a+

Otwarcie pliku w trybie dołączania i odczytu (dane będą zapisywane na koniec pliku — zawartość pliku nie będzie utracona

x

Otwarcie pliku do zapisu. Jeśli plik istnieje, funkcja fopen() wygeneruje błąd. Jeśli plik nie istnieje, zostanie utworzony.

x+

Otwarcie pliku do zapisu i odczytu. Jeśli plik istnieje, funkcja fopen() wygeneruje błąd. Jeśli plik nie istnieje, zostanie utwo

b

Znak ten po dołączeniu do jednego ze znaków wymienionych wcześniej spowoduje otwarcie pliku w trybie danych binarnyc

t

Znak ten po dołączeniu do jednego ze znaków wymienionych wcześniej spowoduje otwarcie pliku w trybie danych tekstowy

Dwa ostatnie wymienione w tabeli tryby są stosowane tylko w połączeniu z pozostałymi, np. rb, at, x+t. Określenia te mają znaczenie jedynie w przypadku systemów operacyjnych, które dokonują rozróżnienia między trybem binarnym i tekstowym (jak np. systemy Windows). W przypadku otwarcia w trybie binarnym dane są odczytywane i zapisywane w plikach bez żadnych modyfikacji. W przypadku otwarcia w trybie tekstowym przetwarzane są znaki końca wiersza. Dzieje się tak dlatego, że różne systemy w różny sposób określają koniec linii tekstu, np. przypadku Windows jest to sekwencja rn, w przypadku Uniksa sekwencja n, a w przypadku MacOS sekwencja r. W PHP5 (a dokładniej — we wszystkich wersjach, począwszy od 4.3.2.) domyślnym trybem otwarcia pliku jest tryb binarny, nie ma zatem potrzeby stosowania znaku b w parametrze tryb_otwarcia. Jest to również zalecany sposób otwierania plików.

Słowo komentarza należy się też sposobowi tworzenia plików. Otóż, jak wynika z opisów z tabeli 6.1, plik można utworzyć poprzez wywołanie funkcji fopen, wykorzystując jeden z trybów: w, w+, a, a+. W PHP nie ma więc oddzielnej funkcji do wykonywania tej czynności.

Przykład użycia funkcji fopen():

$uchwyt_pliku = fopen("plik.txt","r");

Takie wywołanie w skrypcie PHP spowoduje podjęcie próby otwarcia pliku plik.txt, znajdującego się w tym samym katalogu co skrypt, w trybie odczytu. Zmienna $uchwyt_pliku w przypadku powodzenia zawierać będzie — jak wskazuje nazwa zmiennej — uchwyt pliku (ang. handler). Skorzystamy z niego w dalszej części skryptu do odczytywania danych z pliku.

Po dokonaniu odczytu danych plik należy zamknąć — służy do tego funkcja fclose(), przyjmująca jako parametr uchwyt pliku:

fclose($uchwyt_pliku);

Zamykanie pliku nie jest czynnością obowiązkową, ponieważ PHP po zakończeniu działania skryptu spróbuje zamknąć wszystkie pliki otwarte z poziomu skryptu. Jednak pozostawianie dużej ilości otwartych plików może ujemnie wpłynąć na wydajność aplikacji. Dobrym zwyczajem jest zamykanie plików po ich użyciu, za pomocą fclose.

Z otwartym plikiem (do odczytu, zapisu lub dołączania) skojarzony jest tzw. wskaźnik pliku, który wskazuje aktualną pozycję w pliku (dane, które np. będą odczytywane przy następnej operacji odczytu). Nie mamy bezpośredniego dostępu do tego wskaźnika, możemy go modyfikować przez funkcje odczytu lub zapisu do pliku. Funkcją, za pomocą której można manipulować wskaźnikiem pliku bez operacji odczytu i zapisu, jest fseek() — umożliwia ona przesuwanie wskaźnika pliku w przód i w tył.

Na listingu 6.1 przedstawiono skrypt, który odczytuje z dysku i wyświetla w oknie przeglądarki zawartość pliku strona.html (zawartość tego pliku przedstawiona jest na listingu 6.2).

Listing 6.1. Skrypt odczytujący plik strona.html (plik_1.php)

```
<!DOCTYPE HTML PUBLIC &amp;quot;-//W3C//DTD HTML 4.01//EN&amp;quot;

  &amp;quot;http://www.w3.org/TR/html4/strict.dtd&amp;quot;>

<html lang=&amp;quot;pl&amp;quot;>

<head>

 <meta http-equiv=&amp;quot;Content-Type&amp;quot; content=&amp;quot;text/html; charset=utf-8&amp;quot;>

 <title>Wczytanie pliku z dysku lokalnego</title>

</head>

<body>

<?php

 $plik = fopen(&amp;quot;strona.html&amp;quot;, &amp;quot;r&amp;quot;);

 if ($plik === false) {
```

```
    echo &quot;Błąd otwarcia pliku&quot;;

  } else {

    while (!feof($plik)) {

      $bufor = fgets($plik);

      echo &quot;$bufor <br />&quot;;

    }

    fclose($plik);

  }

?>

</body>

</html>
```

Na początku bloku kodu PHP do zmiennej $plik wprowadzany jest rezultat próby otwarcia pliku strona.html w trybie tylko do odczytu. W następnym wierszu sprawdzamy, czy zmienna $plik zawiera wartość false (nie udało się otworzyć pliku) — jeśli tak, wyświetlany jest komunikat o błędzie otwarcia pliku. W przeciwnym razie w pętli while następuje odczyt zawartości pliku wiersz po wierszu. Należy zwrócić uwagę na specyficzną konstrukcję warunku pętli. Jako warunek pętli zastosowano wyrażenie logiczne, którego głównym elementem jest wywołanie funkcji feof() z uchwytem pliku jako argumentem. Funkcja ta zwraca wartość true w momencie, gdy wskaźnik pliku wskazuje koniec pliku. Wyrażenie zwraca wartość, która odpowiada na pytanie, czy nie osiągnięto końca pliku.

Ciało pętli składa się z dwóch wierszy: pierwszy z nich zawiera wywołanie funkcji fgets() z uchwytem pliku jako parametrem. Wartość tej funkcji zwracana jest do zmiennej $bufor. Funkcja fgets() w przypadku skryptu z listingu 6.1 odczytuje z pliku identyfikowanego przez uchwyt wiersz tekstu i zwraca go jako wartość. Odczytany wiersz wypisywany jest za pomocą instrukcji echo w oknie przeglądarki. Widzimy w tym przykładzie, że nazwę zmiennej możemy zastosować również wewnątrz napisu (pomiędzy znakami cudzysłowu). Po wierszu z pliku wyprowadzany jest znacznik
 (łamania wiersza). Pętla odczytująca dane z pliku i wypisująca je w oknie przeglądarki wykonywana jest aż do momentu osiągnięcia końca pliku.

Na koniec plik jest zamykany przez funkcję fclose().

Jeśli plik strona.html będzie miał zawartość taką, jak na listingu 6.2, i będzie umieszczony w tym samym katalogu, co odczytujący go skrypt, to efekt działania skryptu podobny będzie do tego z rysunku 6.1.

Listing 6.2. Treść przykładowego pliku strona.html

```
<?xml version=&quot;1.0&quot; encoding=&quot;utf-8&quot;?>

<!DOCTYPE html PUBLIC &quot;-//W3C//DTD XHTML 1.1//EN&quot;

 &quot;http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd&quot;>

<html xmlns=&quot;http://www.w3.org/1999/xhtml&quot; xml:lang=&quot;pl&quot;>

<head>
```

```
  <meta  http-equiv=&quot;Content-Type&quot; content=&quot;application/xhtml+xml;  charset=utf-8&quot; />

  <title>Do wczytania...</title>

</head>

<body>

<h1>Plik do wczytania</h1>

<hr />

<p>

Plik do wczytania przez skrypt PHP.

</p>

</body>

</html>
```

Rysunek 6.1. Efekt widoczny po wczytaniu pliku przez skrypt

Jak wynika z rysunku 6.1, zawartość pliku strona.html po wysłaniu jej do okna przeglądarki została częściowo zinterpretowana. Nie powinno to dziwić, w końcu plik zawiera znaczniki HTML. Jak więc wyświetlić w oknie przeglądarki zawartość pliku, aby znaczniki HTML też były widoczne (czyli rzeczywistą zawartość pliku)? Z pomocą przychodzi funkcja htmlentities() — jej zadaniem jest zamiana znaków tworzących znaczniki HTML na symbole, które przeglądarka wyświetli, ale nie zinterpretuje ich jako HTML. Funkcja ta przyjmuje jako parametr ciąg znaków zawierających znaczniki HTML, a zwraca ciąg przetworzony. Aby wyświetlić w oknie przeglądarki rzeczywistą zawartość pliku strona.html, należy np. wiersz z wywołaniem funkcji fgets() (listing 6.1) przepisać w następujący sposób:

```
$bufor = htmlentities(fgets($plik));
```

Wynik zwracany przez funkcję fgets() nie musi być zapamiętywany w osobnej zmiennej — wystarczy wynik wywołania funkcji fgets() przekazać jako parametr funkcji htmlentities(). Tak zmodyfikowany skrypt wygeneruje stronę podobną do tej z rysunku 6.2.

Rysunek 6.2. Strona uzyskana po zastosowaniu htmlentities()

Kiedy chcemy odczytać określoną liczbę bajtów lub odczytujemy dane z pliku binarnego, powinniśmy użyć funkcji fread. Jej wywołanie ma postać:

```
fread(deskryptor,  ile)
```

deskryptor to deskryptor pliku zwrócony przez funkcję fopen, a ile określa liczbę bajtów do odczytania. Funkcja

zwraca odczytany fragment pliku w postaci ciągu typu string. Należy pamiętać, że w przypadku systemów rozróżniających pliki binarne i tekstowe (jak np. Windows) plik nie powinien być otwierany w trybie tekstowym, a zatem podczas jego otwierania parametr tryb nie może zawierać modyfikatora t. Przykład zastosowania funkcji fread do odczytania zawartości pliku i wysłania jej do przeglądarki jest widoczny na listingu 6.3 — dane odczytywane są w blokach po 4096 bajtów.

Listing 6.3. Odczyt bloków danych

```
<!DOCTYPE HTML PUBLIC &quot;-//W3C//DTD HTML 4.01//EN&quot;

 &quot;http://www.w3.org/TR/html4/strict.dtd&quot;>

<html lang=&quot;pl&quot;>

<head>

 <meta http-equiv=&quot;Content-Type&quot; content=&quot;text/html; charset=utf-8&quot;>

 <title>Wczytanie pliku z dysku lokalnego</title>

</head>

<body>

<?php

if(!$fd = fopen('test.txt', 'rb')){

 echo(&quot;Nie mogę otworzyć pliku test.txt.&quot;);

}

else{

 while(!feof($fd)){

   echo fread($fd, 4096);

 }

 fclose($fd);

}

?>

</body>

</html>
```

W powyższych przykładach dane odczytywane były w porcjach. Czasem jednak przydaje się odczyt całej zawartości pliku naraz. Można to wykonać na kilka sposobów. Sposób pierwszy to wykorzystanie przedstawionej wyżej funkcji fread, której jako drugi argument zostanie przekazana wielkość pliku. Wartość tę możemy uzyskać, stosując funkcję filesize. Przykładowy skrypt odczytujący tą metodą całą zawartość pliku o nazwie test.txt został przedstawiony na listingu 6.4.

Listing 6.4. Jednoczesny odczyt całej zawartości pliku

```
<?php

if(!$fd = fopen(&quot;test.txt&quot;, 'r')){

  echo(&quot;Nie mogę otworzyć pliku  test.txt.&quot;);

}

else{

  $rozmiar = filesize(&quot;test.txt&quot;);

  echo fread($fd, $rozmiar);

  fclose($fd);

}

?>
```

Czynność taka może zostać przeprowadzona dużo prościej, jeśli skorzystamy z funkcji dedykowanych fpassthru, readfile lub file_get_contents.

Wywołanie funkcji fpassthru ma schematyczną postać:

```
fpassthru(deskryptor)
```

deskryptor to wskaźnik pliku zwrócony przez wywołanie fopen. Funkcja odczytuje zawartość pliku i wysyła ją do standardowego wyjścia — w typowych zastosowaniach do przeglądarki. Zwracaną wartością jest liczba odczytanych bajtów lub też wartość false, jeżeli odczyt nie zakończył się powodzeniem. Odczyt i wysłanie do przeglądarki za pomocą fpassthru zawartości testowego pliku można zatem przeprowadzić za pomocą kodu widocznego na listingu 6.5.

Listing 6.5. Użycie funkcji fpassthru

```
<?php

if(!$fd = fopen(&quot;test.txt&quot;, 'rb')){

  echo(&quot;Nie mogę otworzyć pliku test.txt.&quot;);

}

else{

  if(fpassthru($fd) === false){

    echo(&quot;Próba wysłania  zawartości pliku zakończyła się niepowodzeniem.&quot;);

  }

  fclose($fd);
```

```
}

?>
```

Jeszcze prostsze jest użycie funkcji readfile. Przyjmuje ona jako argument ciąg znaków określający nazwę pliku i wysyła jego zawartość do bufora wyjściowego, co w opisywanych przez nas przypadkach będzie skutkowało wysłaniem danych do przeglądarki. Zwracaną wartością jest liczba odczytanych bajtów, czyli wielkość pliku, lub też wartość false, jeśli wykonanie operacji zakończyło się niepowodzeniem. Aby zatem wysłać zawartość pliku test.txt do przeglądarki, wystarczy wykonać tylko jedną instrukcję:

```
readfile('test.txt');
```

W podobny sposób działa też file_get_contents. Jako argument przyjmuje ona nazwę pliku, zwraca natomiast jego odczytaną zawartość w postaci wartości typu string lub wartość false, jeśli odczyt się nie udał. Odczytanie zawartości pliku test.txt przy użyciu file_get_contents i wysłanie jej do przeglądarki będzie zatem wymagało użycia instrukcji:

```
echo file_get_contents('test.txt');
```

lub też dwóch instrukcji:

```
$str = file_get_contents('test.txt');

echo $str;
```

Jak widać, odczyt pliku z lokalnego dysku nie nastręcza większych problemów. Mogą pojawić się jednak ograniczenia. Duży wpływ na operacje dotyczące plików mają niewątpliwie prawa dostępu do plików i katalogów, różne w różnych systemach operacyjnych. Kolejna rzecz związana z operacjami na plikach to ograniczenia narzucone przez scharakteryzowany w rozdziale poświęconym bezpieczeństwu tryb bezpieczny oraz dyrektywy pliku konfiguracyjnego php.ini.

Odczyt pliku ze zdalnego serwera

Funkcja fopen() pozwala również na odczyt pliku ze zdalnego serwera. Jedyne, co trzeba zrobić, to przekazać tej funkcji jako pierwszy parametr adres bezwzględny pliku do odczytu. Na listingu 6.6 przedstawiono modyfikację skryptu z listingu 6.1 — skrypt odczytujący plik strona.html ze zdalnego serwera.

Listing 6.6. Skrypt odczytujący plik ze zdalnego serwera

```
<!DOCTYPE HTML PUBLIC &quot;-//W3C//DTD HTML 4.01//EN&quot;

  &quot;http://www.w3.org/TR/html4/strict.dtd&quot;>

<html lang=&quot;pl&quot;>

<head>

 <meta http-equiv=&quot;Content-Type&quot; content=&quot;text/html; charset=utf-8&quot;>

 <title>Wczytanie pliku z serwera zdalnego</title>

</head>

<body>

<?php
```

```
 $plik =  fopen(&quot;http://192.168.20.135/~rafal/strona.html&quot;, &quot;r&quot;);

 if ($plik == NULL) {

   echo &quot;Błąd otwarcia pliku&quot;;

 } else {

   while (!feof($plik)) {

     $bufor = htmlentities(fgets($plik));

     echo &quot;$bufor <br />&quot;;

   }

   fclose($plik);

 }

?>

</body>

</html>
```

Efekt działania tego skryptu jest podobny do pokazanego wcześniej na rysunku 6.2 — operacja na pliku zdalnym jest dla użytkownika niewidoczna, może on najwyżej odczuć, że strona ładuje się trochę dłużej.

Zapis do pliku

Algorytm zapisu danych do pliku różni się od algorytmu odczytu tylko innym trybem dostępu przekazanym jako parametr funkcji fopen() oraz inną funkcją wykorzystywaną do zapisu danych. Na listingu 6.7 przedstawiono skrypt, który jest modyfikacją skryptów pokazywanych wcześniej i wykonuje kopiowanie zawartości pliku strona.html do pliku strona2.html. Jak wynika z opisów trybów otwarcia pliku, jeśli plik strona2.html nie istnieje, zostanie utworzony.

Listing 6.7. Skrypt kopiujący zawartość pliku strona.html do pliku strona2.html

```
<!DOCTYPE HTML PUBLIC &quot;-//W3C//DTD HTML 4.01//EN&quot;

  &quot;http://www.w3.org/TR/html4/strict.dtd&quot;>

<html lang=&quot;pl&quot;>

<head>

 <meta http-equiv=&quot;Content-Type&quot; content=&quot;text/html; charset=utf-8&quot;>

 <title>Kopiowanie pliku</title>

</head>

<body>
```

```
<?php

  $plik_we = fopen(&quot;strona.html&quot;,  &quot;r&quot;);

  $plik_wy =  fopen(&quot;strona2.html&quot;, &quot;w&quot;);



  if ($plik_we === false) {

    echo &quot;Nie można otworzyć pliku do odczytu!&quot;;      } else {

    while (!feof($plik_we)) {

     $bufor = fgets($plik_we);

     echo htmlentities($bufor).&quot;<br  />&quot;;

     fputs($plik_wy, $bufor);

    }

    fclose($plik_we);

  }

  fclose($plik_wy);

?>

</body>

</html>
```

W skrypcie tym otwierane są dwa pliki — pierwszy z nich, strona.html, otwierany jest w trybie tylko do odczytu, drugi, strona2.html, w trybie do zapisu. Dalej skrypt przypomina poprzednie przykłady. Pierwsza różnica, jaką znajdziemy w tym skrypcie, to ostatni wiersz ciała pętli while. Wiersz ten zawiera wywołanie funkcji fputs(), która zapisuje do pliku identyfikowanego przez uchwyt $plik_wy ciąg znaków umieszczony w zmiennej $bufor. Aby nie zmieniać zawartości zmiennej $bufor ani nie wprowadzać nowej zmiennej, funkcja htmlentities() użyta jest tylko podczas wysyłania odczytanego z pliku ciągu znaków do przeglądarki. Warto jeszcze zwrócić uwagę na zamykanie plików. Plik wejściowy, identyfikowany przez uchwyt $plik_we, zamykany jest tylko wtedy, gdy rzeczywiście został otwarty i wykonano na nim operacje odczytu. Jeśli spróbujemy za pomocą funkcji fclose() zamknąć plik, który nie był otwarty, wygenerowane zostanie ostrzeżenie (w zależności od konfiguracji PHP ostrzeżenie to zostanie zignorowane, wyświetlone w oknie przeglądarki i (lub) zapisane w dzienniku PHP).

Drugą funkcją, która pozwala na zapis danych do pliku, jest file_put_contents. Upraszcza ona bardzo to zadanie, jako że jest swoistym połączeniem fopen, fwrite i fclose. Może przyjmować do 4 parametrów, dla nas jednak interesujące są trzy pierwsze. Schematyczne wywołanie jest więc następujące:

```
file_put_contents(&quot;nazwa_pliku&quot;, dane[, flagi]);
```

Oznacza ono zapisanie do pliku o nazwie wskazywanej przez argument nazwa_pliku danych wskazywanych przez argument dane. Argument dane może być ciągiem znaków bądź tablicą (może też wskazywać strumień danych — omówienie tego tematu wykracza poza ramy niniejszej publikacji). Parametr flagi może przyjmować kilka wartości, nie będziemy się tym jednak bliżej zajmować. Ważne jest natomiast, że może on przyjąć wartość

FILE_APPEND, co pozwala na modyfikację zachowania funkcji w sytuacji, kiedy plik istnieje na dysku. Otóż w takim przypadku, jeśli parametr flagi nie zostanie zastosowany, dotychczasowe dane z pliku zostaną skasowane, jeśli zaś przyjmie wartość FILE_APPEND, nowe dane zostaną dopisane na końcu pliku. Zapis przykładowego ciągu za pomocą funkcji file_put_contents może zatem zostać zrealizowany za pomocą następującego fragmentu kodu:

```
<?php

$str = &amp;quot;Przykładowa linia tekstun&amp;quot;;

if(file_put_contents(&amp;quot;./test.txt&amp;quot;,  $str, FILE_APPEND) === false){

  echo(&amp;quot;Wystąpił błąd. Zapis nie został dokonany.&amp;quot;);

}

else{

  echo(&amp;quot;Ciąg został zapisany.&amp;quot;);

}

?>
```

Przy każdym wywołaniu takiego skryptu napis zawarty w zmiennej $str będzie dopisywany na końcu pliku test.txt. W przypadku pierwszego wywołania, o ile wymieniony plik nie istnieje w katalogu, w którym znajduje się kod skryptu, plik zostanie utworzony. Gdybyśmy chcieli, aby podczas zapisu była kasowana poprzednia zawartość pliku test.txt, wywołanie file_put_contents powinno mieć postać:

```
file_put_contents(&amp;quot;./test.txt&amp;quot;,  $str);
```

Podczas tworzenia skryptów zapisujących dane do pliku trzeba jeszcze bardziej niż w przypadku operacji odczytu zwracać uwagę na prawa dostępu do plików i katalogów. Poza tym powinno się szczególną uwagę zwracać na pochodzenie danych, które mają być zapisywane do pliku (np. czy są to dane podane przez użytkownika itp.) — jak się okaże w dalszej części kursu, może to zaoszczędzić wielu stresów.

Blokowanie dostępu do pliku na czas operacji odczytu i zapisu

Jak można się już było przekonać, odczyt danych z pliku i zapis danych do pliku to operacje w miarę proste do zaprogramowania, ale zwykle jest tak, że określony skrypt jest jednocześnie uruchamiany na żądanie wielu klientów (użytkowników internetu). Pojawia się wtedy problem: jak zapobiec błędom i przekłamaniom podczas operacji na plikach? Często zdarza się, że z wykorzystaniem plików tekstowych realizowane są np. liczniki odwiedzin — jak zapobiec sytuacji, że plik licznika będzie zawierał dane niepoprawne, bo jednocześnie odczytywane i zapisywane przez wielu użytkowników?

Odpowiedź jest prosta: trzeba znaleźć sposób, żeby na czas operacji na pliku skrypt mógł przejąć plik „na własność”, a po zakończeniu operacji na pliku „oddać” ten plik. Okazuje się, że twórcy PHP zadbali, żeby taki sposób istniał: należy zastosować funkcję flock(). Funkcja ta pozwala zablokować dostęp do pliku na czas operacji odczytu lub zapisu, tak aby inne skrypty (lub instancje tego samego skryptu), zanim podejmą próbę odczytu lub zapisu do pliku, musiały poczekać, aż dany skrypt zakończy operacje na tym pliku. Po zakończeniu operacji na pliku ta sama funkcja pozwala odblokować dostęp do pliku. Na listingu 6.8 przedstawiono modyfikację skryptu z listingu 6.7 — pliki wejściowy i wyjściowy są na czas operacji plikowych blokowane, przy czym w przypadku pliku wejściowego nie jest to do końca blokada, a raczej umożliwienie odczytu pliku przez wiele skryptów jednocześnie, natomiast w przypadku pliku wyjściowego jest to blokada na wyłączność (żaden inny

skrypt w tym momencie nie będzie mógł wykonywać operacji na tym pliku). Aby uprościć skrypt, zakładamy, że nie ma problemów z otwarciem pliku strona.html.

Listing 6.8. Zastosowanie blokady plików

```
<!DOCTYPE HTML PUBLIC &quot;-//W3C//DTD HTML 4.01//EN&quot;
 &quot;http://www.w3.org/TR/html4/strict.dtd&quot;>
<html lang=&quot;pl&quot;>
<head>
 <meta http-equiv=&quot;Content-Type&quot; content=&quot;text/html; charset=utf-8&quot;>
 <title>Kopiowanie pliku z synchronizacją dostępu</title>
</head>
<body>
<?php
 $plik_we = fopen(&quot;strona.html&quot;, &quot;r&quot;);
 $plik_wy = fopen(&quot;strona2.html&quot;, &quot;w&quot;);

 if (!flock($plik_we, LOCK_SH) || !flock($plik_wy, LOCK_EX)) {
  echo &quot;Nie mozna zablokować plików&quot;;
 } else {
  while (!feof($plik_we)) {
   $bufor = fgets($plik_we);
   echo htmlentities($bufor).&quot;<br />&quot;;
   fputs($plik_wy, $bufor);
  }
  // Poniższe linie nie są konieczne:
  flock($plik_we, LOCK_UN);
  flock($plik_wy, LOCK_UN);
 }
 fclose($plik_wy);
```

```
  fclose($plik_we);

?>

</body>

</html>
```

W bloku kodu PHP na listingu 6.8 na początku otwierane są pliki wejściowy i wyjściowy i podejmowana zostaje próba nałożenia blokad na oba pliki (przy okazji skrypt sprawdza, czy były problemy z blokadami — jeśli tak, to użytkownik otrzymuje stosowny komunikat, w przeciwnym razie skrypt jest wykonywany normalnie). Należy zwrócić uwagę na drugi parametr funkcji flock() (pierwszy to uchwyt pliku) — parametr ten oznacza tryb blokady pliku. LOCK_SH oznacza „blokadę" umożliwiającą odczyt z pliku wielu instancjom skryptu jednocześnie, natomiast LOCK_EX oznacza blokadę pliku na wyłączność. Po wykonaniu w pętli operacji na plikach oba pliki są odblokowywane za pomocą tej samej funkcji flock() — w tym przypadku drugi parametr ma wartość LOCK_UN, która oznacza zwolnienie blokady. Zwolnienie blokady nie jest koniecznością, ponieważ tę samą operację może wykonać funkcja fclose(), która wywoływana jest w skrypcie lub automatycznie po zakończeniu działania skryptu. Warto jednak pamiętać, że blokowanie plików, tym bardziej w trybie na wyłączność, powinno być procesem krótkotrwałym — zaraz po zakończeniu operacji na plikach wymagających blokady powinno się ją zdejmować, co pozwoli na płynne działanie skryptu w warunkach produkcyjnych.

Podsumowanie

Mimo swej prostoty operacje na plikach wymagają ostrożności — głównie chodzi tu o względy bezpieczeństwa serwisu i serwera. W rozdziale poświęconym bezpieczeństwu stron WWW dowiesz się, dlaczego operacje na plikach i pliki z różnymi danymi powinny podlegać specjalnej kontroli. Jednak już ta garstka informacji podanych w bieżącym rozdziale pozwoli na wprowadzenie do swojego serwisu WWW prostego systemu przechowywania informacji z wykorzystaniem plików dyskowych.

Przedstawione w tym rozdziale funkcje to zaledwie ułamek możliwości PHP w temacie plików. Warto przejrzeć dokumentację PHP, szczególnie zaś jej rozdział „System plików" dostępny pod adresem http://www.php.net/manual/pl/ref.filesystem.php. Dodatkowo, aby dowiedzieć się więcej o funkcjach typu htmlentities(), warto zajrzeć do rozdziału „Funkcje łańcuchów znaków" dokumentacji PHP, dostępnego pod adresem http://www.php.net/manual/pl/ref.strings.php.