Работа с локальными файлами сервера. Облачное хранилище файлов.

Лабораторная работа № В-3.

ЦЕЛЬ РАБОТЫ

Ознакомление с основными принципами и функциями при работе с локальными файлами на Веб-сервере, включая организацию совместного доступа. Понимание возможных путей реализации механизма аутентификации программными методами *PHP*.

Безопасность и сохранность данных – одни из важнейших требований к современным информационным системам, в том числе и построенным в Интернете. Основным механизмом для ее обеспечения – это подтверждение личности пользователя с помощью пароля. При этом проводится комплекс из трех процедур, которые обычно неотделимы друг от друга – процесс доступа к информации или функционалу сопровождается тремя этапами: идентификация пользователя, его аутентификация и авторизация для проверки прав доступа к информации.

Идентификация

Определение объекта по его характерным свойствам. Например, человек по лицу идентифицирует своих знакомых; по номеру паспорта и ИНН бухгалтерия идентифицирует работника предприятия; инспектор ГИБДД по номеру автомобиля идентифицирует транспортное средство. В компьютерных системах чаще всего для идентификации используется имя пользователя (логин). Но наличие характерных особенностей не позволяет гарантировать подлинность объекта – вполне возможно, что под маской прячется злоумышленник.

Аутентификация

Подтверждение подлинности идентифицируемого объекта. Наиболее распространенным механизмом аутентификации в информационных системах является пароль. Знание секретного слова, которое должно быть известно только системе и пользователю подтверждает и удостоверяет личность последнего.

Авторизация

Проверка права определенного пользователя использовать тот или иной ресурс. Например, администратор сайта получает доступ к полной функциональности административной панели, редактор новостей – только к функциям работы с новостной лентой; системный администратор – доступ к базе данных и т.д. Таким образом, все части информационной системы, включая данные и ее функции, имеют свои права доступа для разных групп пользователей.

Локальные файлы

Сегодня использование локальных файлов на Веб-сервере применяется очень редко. Наиболее часто они применяются при загрузке файлов на сервер с локального компьютера и их последующей обработки: чтения, переноса информации в базу данных и т.д.

Тем не менее до сих пор возможны ситуации, когда хранение информации именно с помощью файловой системы оправдано. Например, для хранения информации о сессии *PHP* использует специальные файлы определенной структуры, при этом в *PHP* есть возможность переопределить функции их обработки, оптимизировав таким образом решение конкретных задач.

Также можно выделить реализацию создания независимых копий баз данных, результатов обработки и т.п., информацию в которых важно сохранить даже при сбое базы данных. Помимо этого аналогично файлам обрабатываются и потоки данных – очень важный объект PHP, речь о котором пойдет в последующих лабораторных работах.

РЕЗУЛЬТАТ РАБОТЫ

Размещенные на Веб-сервере и доступные по протоколу *HTTP* документы:

- *index.php* – главная страница сайта, использующаяся для аутентификации пользователя;
- *tree.php* – модуль отображения доступных файлов;
- *viewer.php* – модуль отображения содержащейся в файле информации.

Веб-сервис в целом должен позволять загружать файлы на сервер и отображать их информацию в браузере. Необходимо предусмотреть механизм защиты файлов от несанкционированного доступа со стороны других пользователей сервера. Задание должно быть выполнено без использования баз данных.

ДОПОЛНИТЕЛЬНЫЕ ТРАБОВАНИЯ К РАБОТЕ

Модуль *index.php* должен удовлетворять следующим требованиям.

1. При первом открытии страницы в браузере отображается форма для ввода логина и пароля.
2. При отправке формы производится проверка наличия соответствующих данных в специальном файле *users.csv* – если их нет (пользователь не аутентифицирован) – выводится соответствующая надпись и повторно отображается форма. Если данные файле присутствуют (аутентификация пройдена) – запускается второй модуль.
3. Обновление страницы (возврат назад средствами браузера) после успешной аутентификация не должно приводить к потере аутентификации или выводом браузером каких-либо предупреждений (например, о необходимости повторно отправить данные формы).
4. В случае успешной аутентификации, вне зависимости от отображаемой информации, в правом верхнем углу должна размещаться ссылка "Выход" при переходе по которой аутентификация снимается. В этом случае повторно загружается форма ввода логина и пароля.

Модуль *tree.php* должен удовлетворять следующим требованиям.

1. При попытке выполнения без аутентификации выводится сообщение о прекращении работы и дальнейшее выполнение программы приостанавливается.
2. При открытии страницы в браузере отображается содержание текущего каталога (дерево файлов).
3. Каждый элемент дерева выводится в отдельном блоке (тег `<div>`), которые за счет отступов слева формируют легко читаемую структуру дерева каталогов.
4. Элементы типа "каталог" и "файл" должны отличаться друг от друга внешне.
5. Любой элемент "каталог" должен содержать все содержащиеся в нем элементы.
6. Элементы "файл" должны представлять собой ссылки, передающие третьему документу в качестве *GET*-параметра его полное имя. Ссылка должна открываться в другом окне или вкладке браузера.
7. Внизу дерева выводится форма, позволяющая загружать файлы с локального компьютера на сайт. Каталог, в который будет загружен файл указывается пользователем в специальном поле. Если каталог не существует – он должен быть создан. Если указан существующий каталог, но не выбран файл для загрузки – каталог вместе со всеми файлами должен быть удален.
8. Попытки удаления системных файлов и каталогов должны блокироваться.
9. Информация о принадлежности успешно загруженных файлов сохраняется в специальный файл *users.csv*. Имя файла на сервере генерируются как последовательность чисел от 1 с шагом 1; расширение загруженного файла сохраняется. В каждом каталоге последовательность чисел начинается заново.

Модуль *viewer.php* при передаче ему в качестве параметра имени файла должен удовлетворять следующим требованиям.

1. если пользователь не аутентифицирован – выводится ссылка: "Необходима аутентификация!" ведущая на главную страницу сайта, дальнейшее выполнение программы прекращается;
2. если в специальном файле *users.csv* присутствует информация об принадлежности файла не аутентифицированному в настоящее время пользователю – выводится надпись: "Нет прав доступа!";

3. при попытке отображения данных из специального файла *users.csv* выводится надпись: "Секретная информация!";
4. если параметр не передан – выводится надпись: "Имя файла не указано!";
5. если параметр передан, но файл не найден – выводится надпись: "Файл не найден!";
6. если параметр передан и файл существует – выводится содержимое этого файла в браузере, включая *HTML*-теги (если файл содержит тег `<br>`, то в браузере должен быть выведен текст "
", а не осуществлен перевод строки);
7. работа с файлами не должна приводить к потенциальным конфликтам доступа между различными пользователями сайта.
8. Если информации о файле в *users.csv* нет – выводится сообщение об этом.

РЕКОМЕНДАЦИИ К СТРУКТУРЕ ПРОГРАММЫ

АУТЕНТИФИКАЦИЯ

Отобразить дерево файлов и вывести из них данные очень просто, немного сложнее построить механизм аутентификации. Поэтому начнем разработку программы именно с него, тем более что соответствующий модуль идет первым по порядку. Смысл механизма достаточно прост – необходимо ввести имя пользователя и пароль, и проверить его: существует такой пользователь с таким паролем или нет. При обновлении страницы сайта или загрузке других страниц необходимость повторно вводить имя и пароль отсутствует – это было бы весьма неудобно, поэтому состояние аутентификации должно сохраняться на все время работы с сайтом. Из предыдущих лабораторных работ известно, что сохранять такую информацию можно либо с помощью *cookie*, либо с помощью механизма сессий.

Cookie очень удобный механизм, но все данные хранятся в браузере, а значит доступны любому пользователю локального компьютера – это не безопасно. Поэтому для аутентификации в лабораторной работе будут использоваться сессии – в этом случае снимается еще один недостаток *cookie*: ограничение на размер хранимых данных. Выбрав механизм аутентификации, составим естественно-языковое описание алгоритма его реализации.

1. Если пользователь не аутентифицирован, но в параметрах переданы его имя и пароль – осуществляем проверку существования пользователя. Если она прошла – сохраняем информацию о пользователе в сессии.
2. Если в сессии отсутствует информация об аутентифицированном пользователе – выводим форму для ввода логина и пароля.
3. Если в сессии есть информация об аутентифицированном пользователе – обрабатываем соответствующие параметры, выводим информацию и т.д.

Последовательное выполнения пунктов алгоритма обеспечивает нормальную работу механизма аутентификации. Действительно, при первой загрузке страницы никакой информации о пользователе в сессии не хранится и никакие параметры не переданы. Поэтому первый и третий пункты алгоритма не выполняются – выводится форма для ввода логина и пароля. Далее мы вводим эти данные и опять выполняем алгоритм.

В этом случае в параметрах указаны имя и пароль, но аутентификации пока нет. Поэтому выполняется первый пункт, который может привести к двум вариантам событий: если проверка прошла успешно (пользователь аутентифицирован) – данные о нем сохраняются в сессии. В таком случае п.2 не выполняется, но выполняется п.3 – выводится доступная пользователю информация или же предлагаются соответствующие сервисы. Обратите внимание – при таком подходе признаком аутентификации пользователя является наличие информации о нем в сессии! При провале проверки (пользователь с таким именем не существует или не верно задан его пароль) – информация в сессии не будет сохранена, поэтому далее будет выполнен только п.2 – форма будет выведена повторно. На языке *PHP* рассмотренный алгоритм можно записать следующим образом.

Листинг В-3. 1

```
<?php
	session_start();	// подключаем механизм сессий
	//////////////////////////////////////////////////////////////////
	// если аутентификации нет, но переданы данные для ее проведения
	//////////////////////////////////////////////////////////////////
	if( !isset($_SESSION['user']) && is_set($_POST['login']) )
	{
		... // попытка аутентификации
		if( /* результат аутентификации успешен */ )
			$_SESSION['user'] = $user; // сохраняем данные о пользователе
	}
	//////////////////////////////////////////////////////////////////
	// если аутентификации все еще нет
	//////////////////////////////////////////////////////////////////
	if( !isset($_SESSION['user']) )
	{
		// выводим форму для аутентификации
		echo '<form name="auth" method="post" action="">
			<input type="text" name="login">
			<input type="password" name="password">
			<input type="submit" value="Войти">
			</form>';
	}
	else
	//////////////////////////////////////////////////////////////////
	// если аутентификация успешно произведена
	//////////////////////////////////////////////////////////////////
	{
		// выводится информация для аутентифицированного пользователя
		echo '<p>Добро пожаловать, '.$_SESSION['user']['name'].'!</p>';
	}
?>
```

Действительно, первый модуль разделен на три условных блока. Если пользователь не аутентифицирован (отсутствует признак этого, а именно наличия в сессии информации о нем, которая в данной реализации хранится в элементе массива `$_SESSION` с ключом `'user'`), но переданы параметры для его аутентификации (п. 1 алгоритма) – производится соответствующая проверка данных и, если она успешна, сохранение полученных при ее выполнении данных о пользователе (например, его ФИО) в элементе массива.

Если пользователь не аутентифицирован и данные не были переданы, либо же аутентификация не удалась – выводится форма для ввода и передачи этих данных на обработку. Иначе (если аутентификация удалась или была проведена при предыдущих выполнениях программы) – выводится информация для пользователя: в данном примере это его имя.

Доработаем предложенный "скелет" программы до требуемой в лабораторной работе функциональности, а именно:

- в случае успешной аутентификации организуем вывод содержимого локального каталога;
- реализуем проверку данных пользователя для его аутентификации (с помощью файла users.csv);
- предотвратим вывод сообщений браузера о повторной загрузке данных.

Первый пункт на данном этапе не должен создать никаких затруднений: ведь исходя из задания, модуль *tree.php* должен делать именно это. Поэтому сразу после вывода приветствия просто подключим его конструкцией `include`, а программирование самой функции отложим на будущее, до момента реализации модуля.

Проверку пользователя и его пароля очень легко реализовать с помощью баз данных, но в задании необходимо использовать файл с именами и паролями. В принципе, это неправильно – но задание есть задание, его необходимо выполнять. Поэтому, с помощью *Excel* создадим специальный файл *users.csv* каждая строка которого будет содержать информацию об одном пользователе: логин, пароль (подробнее о файлах .csv – см. справочную информацию к данной работе) и т.д. Тогда открыв этот файл, считав из него данные и найдя среди строк полученные из формы логин и пароль,

можно быть уверенным, что пользователь найден, т.е. аутентификация пройдена. Остается "расшифровать" эту строку и сохранить все данные о пользователе в сессии.

Листинг В-3. 2

```
// попытка аутентификации
$f=fopen('users.csv', 'rt');      // открываем текстовый файл для чтения
if( $f )                          // если файл успешно открыт
{
       while( !feof($f) )                 // пока не найден конец файла
       {
              $str = fgets($f);           // читаем текущую строку
              $test_user = explode(';', $str);          // разбиваем строку в массив
              if(trim($test_user[0])==$_POST['login']) // если первый элемент совпадает
              {                                         // с переданным именем
                                                          // если второй совпадает
                     if(trim($test_user[1])==$_POST['password'])
                     {                                    // с переданным паролем
                            $user = $test_user;           // найден аутентифицированный
                     }                                    // пользователь
                     break;               // заканчиваем цикл досрочно
              }
       }
       fclose($f);                        // закрываем файл
}
if( is_set($user) )                       // результат аутентификации успешен
       $_SESSION['user'] = $user;         // сохраняем данные о пользователе
```

В *PHP*, как и в большинстве языков программирования, для чтения данных из файлов или записи в них, файлы необходимо предварительно "открыть". Именно это мы делаем в первой строке программы: указываем имя файла, с которым собираемся работать и тип операций с данными – в данном случае чтение (`r` – от слова *read*) текстовой информации (`t` – от *text*). Функция `fopen()` возвращает так называем дескриптор файла, т.е. его код по которому *PHP* понимает к какому именно файлу в дальнейшем будет происходить обращение. Если функция вернула `false` – значит файл не существует или же он не может быть открыт. Выполнение этого условия проверяется в следующей строке – если файла нет, то и аутентификация невозможна, т.к. нам неизвестно ни об одном пользователе.

Если же файл успешно открыт, то в цикле можно последовательно прочесть все его строки, это происходит следующим образом. В начале работы данные не считаны, а файловый указатель (номер считываемого при следующей операции байта) установлен на начало. Поэтому, если файл не пустой, функция `feof()` возвратит `false` – она делает это всегда, когда файловый указатель не достигнул конца файла. Т.к. в операторе цикла это условие стоит под операцией отрицания – цикл будет продолжать свои итерации до тех пор, пока не достигнут конец файла, т.е. все данные не прочитаны.

Для чтения данных в примере используется функция `fgets()`, которая читает данные от текущего положения файлового указателя и до конца строки. После этого файловый указатель смещается на следующую после окончания строки позицию. Таким образом в цикле читается из файла первая строка, а файловый указатель передвигается на первый байт после ее окончания. На следующей итерации все проверяется: если указатель достиг конца файла – цикла прекращается, если же нет – аналогично читается следующая строка.

Итак – организован цикл, последовательно обрабатывающий все строки файла *users.csv*. Т.к. каждой строке соответствует описание только одного пользователя, то все что нам остается сделать – это поверить совпадение переданного логина текущему пользователю. Обратите внимание – т.к. в строки полученные из файла содержат в конце пробельные символы, то лучше использовать функцию `trim()` для их удаления – это помимо всего удалит и ошибочно введенные при заполнении файла пробелы в начале или конце строки. Если логин совпал – то кандидат на аутентификацию найден и за этим следует проверка пароля. Но для описанного процесса необходимо предварительно "дешифровать" строку файла *csv*, "вынув" из нее необходимые

данные. Это можно делать с помощью уже известной функции `explode()` – первые два элемента полученного массива и являются логином и паролем текущего пользователя.

Теперь требуемая проверка легко производится условным оператором: сравнивается первый элемент массива с полученным из формы логином пользователя. Если они совпадают – дальнейшие итерации можно прекратить и перейти к сравнению второго элемента массива и переданного пароля. Если пароли совпали – аутентификация произошла, и информация о пользователе сохраняется в специальную переменную `$user`. Если же пароль передан не верно – то переменная так и не будет создана, а дальнейший перебор строк файла – лишняя трата времени.

После окончания цикла необходимо закрыть файл (он будет закрыт и автоматически после завершения программы, но такой подход более правильный). Затем организуется проверка существования переменной `$user`: если в результате выполнения данного кода она появилась – пользователь был аутентифицирован, и информация о нем сохраняется в сессии, если же ее нет – то информации о нем в сессии не появляется. Т.е. признак автооризации полностью реализован предложенным способом.

Разобранный пример абсолютно корректно будет работать, но код несколько "шероховат". Давайте отшлифуем его – ведь чем более красива ваша программа, тем проще с ней работать и вносить необходимые изменения. Заодно немного доработаем код, проверив возможные приводящие к потенциальным ошибкам выполнения программы ситуации.

Листинг В-3. 3

```
// попытка аутентификации
if( $f=fopen('users.csv', 'rt') )              // если файл успешно открыт
{
       while( !feof($f) )                 // пока не найден конец файла
       {
              // разбиваем текущую строку файла в массив
              $test_user = explode(';', fgets($f) );
              if(trim($test_user[0])==$_POST['login'] )      // если найден логин
              {
                     if( is_set($test_user[1]) &&     // если пароли совпали
                         trim($test_user[1])==$_POST['password'] )
                            $_SESSION['user'] = $test_user;  // сохраняем в сессию
                     break; // проверка закончена – следующих проверять нет смысла
              }
       }
       fclose($f);                        // закрываем файл
}
```

В примере открытие файла встроено прямо в условный оператор для проверки его существования: если его не удется открыть, значит будем считать, что он не существует. Строку для передачи в `explode()` получаем непосредственно вызвав функцию `fgets()`, без сохранения результата ее работы в промежуточной переменной `$str`. Для более стабильной работы перед проверкой совпадения переданного пароля и текущего производится проверка существования в полученном массиве элемента с индексом "`1`" – он будет отсутствовать если в строке файла по каким-то причинам нет разделителей элементов. И наконец, нет необходимости вводить промежуточную переменную `$user` и сохранять в нее данные пользователя – можно сразу создать соответствующий элемент массива сессии.

Представим себе работу скрипта при удачной аутентификации: *PHP*-программа получила параметры, обработала их, сохранила данные о пользователе в сессию, вывела необходимую пользователю информацию. Все хорошо, но что произойдет при попытке браузера обновить страницу (если нажать клавишу *F5*)? Браузер понимает, что обновляемая страница была получена как результат обработки каких-то данных и честно предупреждает об этом пользователя, предлагая утвердить повторную их отправку на сервер. Вполне логично для браузера, но очень неудобно для человека, особенно если учесть, что подобная же ситуация возникает и при переходе с помощью кнопок браузера "Назад" и "Вперед". Для того чтобы избежать этого удобно использовать редирект.

Листинг В-3. 4

```
<?php
	session_start();	// подключаем механизм сессий
	////////////////////////////////////////////////////////////////////
	// если аутентификации нет, но переданы данные для ее проведения
	////////////////////////////////////////////////////////////////////
	if( !isset($_SESSION['user']) &&
	    is_set($_POST['login']) && is_set($_POST['password']) &&
	    $f=fopen('users.csv', 'rt')) // попытка аутентификации
	{
		while( !feof($f) )		// пока не найден конец файла
		{
			// разбиваем текущую строку файла в массив
			$test_user = explode(';', fgets($f) );
			if( trim($test_user[0])==$_POST['login'] )	// если найден логин
			{
				if( is_set($test_user[1]) && // если пароли совпали
				    trim($test_user[1])==$_POST['password'] ) // сохраняем
					$_SESSION['user'] = $test_user;	// в сессию
				header('Location: /');	// редиректна главную
				exit();		// дальнейшая работа скрипта излишняя
			}
		}
		fclose($f);  // закрываем файл
	}
	////////////////////////////////////////////////////////////////////
	// если аутентификации все еще нет
	////////////////////////////////////////////////////////////////////
	if( !isset($_SESSION['user']) )
	{
		// выводим форму для аутентификации
		echo '<form name="auth" method="post" action="">
			<input type="text" name="login">
			<input type="password" name="password">
			<input type="submit" value="Войти">
			</form>';
	}
	else
	////////////////////////////////////////////////////////////////////
	// если аутентификация успешно произведена
	////////////////////////////////////////////////////////////////////
	{
		// выводится информация для аутентифицированного пользователя
		echo '<p>Добро пожаловать, '.$_SESSION['user']['name'].'!</p>';
		include 'tree.php';		// выводим содержимое дерева файлов
	}
?>
```

При объедении примеров кода была добавлена проверка существования в передаваемых параметрах не только имени пользователя, но и пароля. В принципе, эта проверка излишне, т.к. ситуация передачи логина без пароля невозможна, но для "универсальности" кода ее можно производить. Принудительное завершение цикла заменено выводом заголовка для перенаправления (редиректа) на главную страницу сайта. Причем для уменьшения трафика и увеличения скорости работы дальнейшая работа *PHP* и вывод каких-либо данных в браузер прекращается, а страница очень быстро и незаметно для пользователя перезагружается. Кроме того, при успешной аутентификации осуществляется подключения модуля отображения дерева каталогов и вывода формы загрузки файла (*tree.php*).

Получившийся вариант кода уже вполне работоспособен, но все же обладает некоторыми недостатками. В первую очередь это невозможность зайти под другим логином: для этого необходима кнопка "Выход". Кроме того, при попытке неудачной аутентификации следует выводить соответствующую надпись: без этого у пользователя будут возникать обоснованные сомнения в работоспособности программы. Ну и, наконец, для удобства следует автоматически заполнять поле `login` переданным в программу значение – если пароль задан с опечаткой, то и

вводить заново следует только его; ведь ввод логина в такой ситуации – дополнительное неудобство для пользователя.

Листинг В-3. 5

```
<?php
	session_start();	// подключаем механизм сессий
	////////////////////////////////////////////////////////////////////
	// обрабатываем выход
	////////////////////////////////////////////////////////////////////
	if( is_set($_GET['logout']) )		// если был переход по ссылке Выход
	{
		unset( $_SESSION['user'] );	// удаляем информацию о пользователе
		header('Location: /');		// переадресация на главную страницу
		exit();				// дальнейшая работа скрипта излишняя
	}
	////////////////////////////////////////////////////////////////////
	// если аутентификации нет, но переданы данные для ее проведения
	////////////////////////////////////////////////////////////////////
	if( !isset($_SESSION['user']) && is_set($_POST['login']) &&
	    is_set($_POST['password']) && $f=fopen('users.csv', 'rt'))
	{
		while( !feof($f) )		// пока не найден конец файла
		{
			// разбиваем текущую строку файла в массив
			$test_user = explode(';', fgets($f) );
			if( trim($test_user[0])==$_POST['login'] )	// если найден логин
			{
				if( is_set($test_user[1]) && // если пароли совпали
				    trim($test_user[1])==$_POST['password'] ) // сохраняем
				{
					$_SESSION['user'] = $test_user;	// в сессию
					header('Location: /');	// редирект на главную
					exit();		// дальнейшая работа скрипта излишняя
				}
				else			// если пароль не совпал
					break;		// прекращаем итерации
			}
		}
		echo '</div>Неверный логин или пароль!</div>';
		fclose($f);	// закрываем файл
	}
	////////////////////////////////////////////////////////////////////
	// если аутентификации все еще нет
	////////////////////////////////////////////////////////////////////
	if( !isset($_SESSION['user']) )
	{
		// выводим форму для аутентификации
		echo '<form name="auth" method="post" action="">
			<input type="text" name="login"';
		//если логин уже вводился ранее и был передан в программу
		if( is_set($_POST['login']) )
			 echo ' value="'.$_POST['login'].'"';	// заполняем значение поля
		echo '><input type="password" name="password">
			<input type="submit" value="Войти">
			</form>';
	}
	else
	////////////////////////////////////////////////////////////////////
	// если аутентификация успешно произведена
	////////////////////////////////////////////////////////////////////
	{
		echo '<a href="/?logout=">Выход</a>';	// выводится ссылка для выхода
		// выводится информация для аутентифицированного пользователя
		echo '<p>Добро пожаловать, '.$_SESSION['user']['name'].'!</p>';
		include 'tree.php';		// выводим содержимое дерева файлов
	}
?>
```

В первую очередь обратите внимание на обработку выхода из аутентификации. Для этого ссылка на главную страницу с *GET*-параметром `logout` выводится над персональной информацией пользователя при подтвержденной аутентификации. В принципе, можно выбрать любое имя параметра и передавать его как в *GET*- так и в *POST*-параметрах – необходимо только правильно написать обработчик. В данном примере код обработки выхода выполняется в самом начале программы: если в массиве `$_GET` найден элемент с ключом `'logout'` – значит был совершен переход по соответствующей ссылке: информация о пользователе из сессии удаляется, после чего происходит переадресация на главную страницу (по тем же соображениям, что и в случае успешной аутентификации).

Вывод сообщения об неудачной аутентификации осуществляется при завершении цикла без ее подтверждения. Обратите внимание: если пользователь найден в текущих, но его пароль указан неверно – переадресация на главную страницу не даст нам возможность вывести требуемое сообщение. Поэтому в примере переадресация осуществляется только в случае совпадения и логина, и пароля; если пароли разные – цикл просто принудительно прекращает свою работу. Соответственно, если в какой-либо итерации цикла аутентификация не произошла – это значит, что или пользователь не найден (ошибка логина), или его пароль указан неверно. В обоих случаях необходимо вывести предупреждающее сообщение, что и было реализовано.

Наконец, при формировании *HTML*-кода формы проверяется наличие в переданных параметрах логина. Если такой параметр был передан – то он передается в соответственном поле как его начальное значение.

Самостоятельно сформируйте *css*-файл для красивого и удобного отображения элементов модуля. Доработайте представленный код таким образом, чтобы при открытии сайта даже после перезагрузки компьютера поля формы `login` и `password` были заполнены последними введенными при отправке формы значениями (используйте *cookies*). Обратите внимание – элементы массива с данными о пользователя в некоторых случаях могут быть дополнительно заключены в кавычки – обработайте эту ситуацию.

Содержимое каталога и загрузка файлов

Следующий модуль используется для отображения существующих на сервере файлов и загрузки на него новых. Фактически это стандартный обработчик данных: он отображает какие-то данные и предоставляет функции для их изменения. Ясно, что обработка должна производится до отображения (если данные сначала отобразить, а уже потом изменить – то пользователь увидит старый вариант данных, а новая информация будет доступна ему только после перезагрузки страницы). Поэтому структура модуля будет иметь следующий вид.

Листинг В-3. 6

```
<?php
	if(  /* необходимо загрузить файл (изменить данные) */  )
	{
		... // загружаем файл (изменяем данные)
	}

	... // отображаем содержимое текущего каталога (отображаем данные)

	... // выводим форму для загрузки файлов (инструменты обработки данных)
?>
```

Начнем описание работы модуля со второй его части – это удобно т.к. без отображения текущего состояния данных очень сложно проверить корректность их обработки. Для построения дерева каталогов используются функции `opendir()`, `closedir()`, `readdir()` и некоторые другие функции манипулирования файлами. Т.к. необходимо отобразить все элементы каталога (и файлы, и подкаталоги, для которых так же должны отображаться все их подкаталоги), то воспользуемся рекурсивным методом. Для этого построим пользовательскую функцию `outdirInfo()`, выводящую информацию о содержимом указанного каталога.

Листинг B-3. 7

```
function outdirInfo( $name, $path )
{
	echo '<div>';                    // начало блока с содержимым каталога
	echo 'Каталог '.$name.'<br>';    // выводим имя каталога

	$dir = opendir( $path );         // открываем каталог
	// перебираем элементы каталога пока они не закончатся
	while( ($file=readdir($dir) ) !== false )
	{
		if( is_dir($d) )                        // если элемент каталог
			echo 'Подкаталог '.$d.'<br>';   // выводим его имя
		else
		if( is_file($d) )                       // если элемент файл
			echo 'Файл '.$d.'<br>';         // выводим его имя
	}
	closedir($dir);                  // закрываем каталог

	echo '</div>';                   // конец блока с содержимым каталога
}
```

В качестве параметров в функцию передается имя каталога, содержимое которого мы хотим вывести, и отдельно полный путь к нему (включая и само имя, например: `outdirInfo( 'mysite', 'htdocs/www/mysite')`). В функции мы предварительно выводим тег `<div>` для оформления с помощью *CSS* соответствующих отступов и другого оформления содержимого каталога, затем выводим его имя (переданный в функцию первый параметр `$name`). Для получения содержимого каталога его необходимо "открыть" и последовательно прочитать все элементы – здесь прослеживается полная аналогия с открытием текстового файла и чтением его строк, только используются не функции `fopen()` и `fgets()`, а `opendir()` и `readdir()`. Перед выводом с помощью функций `is_dir()` и `is_file()` определяется тип элемента каталога (соответственно подкаталог это или файл) и выводится вместе с названием этого элемента. В конце функции каталог необходимо закрыть и вывести закрывающую часть тега `</div>`.

Как результат, функция в отдельном блоке выводит имя каталога и на отдельных строках все его элементы: каталоги и подкаталоги. Теперь, согласно требованию лабораторной работы, доработаем ее так, чтобы выводилась также и информация о подкаталогах.

Листинг B-3. 8

```
function outdirInfo( $name, $path )
{
	echo '<div>Каталог '.$name.'<br>';      // выводим имя каталога

	$dir = opendir( $path );         // открываем каталог
	// перебираем элементы каталога пока они не закончатся
	while( ($file=readdir($dir) ) !== false )
	{
		if( is_dir($d) )                        // если элемент каталог
			outdirInfo( $d, $path.'/'.$d );  // выводим его содержимое
		else
		if( is_file($d) )                       // если элемент файл
			echo makeLink($d, $path);       // выводим его имя
	}
	closedir($dir);                  // закрываем каталог

	echo '</div>';                   // конец блока с содержимым каталога
}
```

Теперь функция не просто выводит имя подкаталога, а запускает саму себя для вывода содержимого. При этом достаточно легко скорректировать передаваемые в нее параметры: имя каталога – это имя найденного элемента, а полный путь к нему – предыдущий полный путь плюс имя. Таким образом внутри блока (тега `<div>`) выводятся имена файлов в качестве строк и другие блоки с содержимым подкаталогов, которые также содержат строки и блоки сколь угодно большой

глубины вложенности. Самостоятельно доработайте функцию так, чтобы ограничить эту глубину каким-либо параметром – обычно это необходимо для избежания возможных бесконечных вызовов функцией самой себя.

Следует отметить, что полученная программа может приводить к бесконечному циклу или ошибке, т.к. элементами каталога могут являться как родительский каталог (".."), так и сам каталог (".."). Модифицировать функцию так, чтобы исключить эти случаи также необходимо самостоятельно.

Был изменен и способ вывода имени файла – для этого теперь используется специальная функция `makeLink()`, которая формирует ссылку на третий модуль, открывающий переданный ему файл для чтения его содержимого в браузере. В функцию передается два параметра: имя файла и полный путь к содержащему его каталогу. В принципе, возможно и объединение обеих параметров в один: например, в полный путь к файлу. Но тогда нам придется для вывода адреса и текста ссылки вновь "разъединять" их, что несколько портит архитектуру программы.

Листинг B-3. 9

```
function makeLink( $name, $path )
{
	// формируем адрес ссылки
	$link='viewer.php?filename='.$path.'/'.$name;
	// выводим ссылку в HTML-код страницы
	echo '<a href=".$link.'">Файл '.$name.'</a>';
}
```

Функция формирует и выводит ссылку, ведущую к документу *viewer.php* и передающий в него имя открываемого файла. Чтобы дополнительно обезопасить себя от символов, которые в ссылке могут трактоваться как элементы *URL*, модифицируем функцию, выводя параметры в специальном *URL*-кодированном виде с помощью функции `UrlEncode()`.

Листинг B-3. 10

```
function makeLink( $name, $path )
{
	echo '<a href=".'viewer.php?filename='. UrlEncode($path).
		'/'.$name.'">Файл '.$name.'</a>'; // выводим ссылку в HTML-код страницы
}
```

Действительно, не всякую строку можно передать как *GET*-параметр. Т.е. если нам необходимо передать в *PHP*-программу в параметре *parameter* строку "*&f=#100*", то при формировании ссылки `href="?parameter=&f=#100"` в скрип будет передано два значения: пустая строка в параметре *parameter* и *100* в параметре *f*. Функция `UrlEncode()` позволяет избежать этого, заменяя "опасные" символы понятными браузеру их *URL*-кодами.

Самостоятельно доработайте функцию так, чтобы формируемая ссылка открывалась в новом окне (вкладке) браузера. Обратите внимание: *CSS* описание блока (`<div>`) с отступом слева автоматически придаст выводимым данным форму дерева каталогов – сделайте это, а также придайте в CSS различный внешний вид именам файлов и каталогов самостоятельно.

После подготовки соответствующих функций остается только вызвать их для формирования дерева текущего каталога. Для этого передадим в функцию `outdirInfo()` пустые строки, т.е. начнем вывод прямо от корневого каталога сайта – именно он скорее всего и будет текущим.

Листинг B-3. 11

```
echo '<div id="dir_tree">';	// выводит начало тега блока дерева каталогов

outdirInfo( '', '' );		// выводит дерево каталогов

echo '</div>';			// конец блока дерева каталогов
```

Для большей стабильности программы пустой относительный путь можно заменить абсолютным, получив имя текущего каталога с помощью функции `getcwd()`. Сделайте это самостоятельно и проанализируйте отличие работы обоих вариантов для вашей операционной системы.

Итак, после того как реализован функционал по отображению данных (вывода содержимого каталога) – следует разработать функции по их изменению, в данном случае для загрузки файлов. При этом необходим *HTML*-код, который бы позволил выбрать загружаемый на сервер файл, механизм его передачи и собственно *PHP*-код для обработки переданного файла. К счастью, *HTML* предоставляет в наше распоряжение удобный механизм загрузки файлов, а именно формы `multipart/form-data` и текстовые поля типа `file`.

Листинг В-3. 12

```
<form method="post" enctype="multipart/form-data" action="/tree.php">
	<label for="dir-name">Каталог на сервере</label>
	<input type="text" name="dir-name" id="dir-name">

	<label for="myfilename">Локальный файл</label>
	<input type="file" name="myfilename">

	<input type="submit" value="Отправить файл на сервер">
</form>
```

Отправка такой формы (обязательно методом *POST*) передаст в скрипт *tree.php* не только параметр `dir-name`, но и целиком выбранный в поле `myfilename` файл. Надо сказать, что возможность загрузки файла может быть запрещена в настройках сервера, причем не только запрещена, но и ограничена: кроме явно задаваемого максимального размера загружаемого файла, влияние оказывает и максимальное время выполнения *PHP*-скрипта, и максимальный размер *POST*-данных. Если для хранения файлов используется база данных – в ней также могут присутствовать настройки по ограничению объема данных в запросе. Самостоятельно изучите все указанные выше настройки сервера.

После удачной отправки такой формы выбранный файл будет загружен на сервер и данныее из него будут записаны во временном файле. Т.к. сразу после окончания работы скрипта последний будет автоматически удален, необходим обработчик загрузки, который бы прочитал данные из файла и сохранил их в нужный каталог под нужным именем (отправил в базу данных, вывел в браузер или обработал данные каким-либо иным образом).

Листинг В-3. 13

```
if( isset($_FILES['myfilename']) ) // были отправлены данные формы
{
	if( isset($_FILES['myfilename']['tmp_name']) )	// если файл загружен
	{
		move_uploaded_file( $_FILES['myfilename']['tmp_name'],	// копируем
		   makeName($_FILES['myfilename']['name'])); // файл с новым именем
		echo 'Файл '.$_FILES['myfilename']['name'].' загружен на сервер';
	}
}
```

Вся информация о загруженном файле хранится в суперглобальном массиве `$_FILES`, откуда она и может быть извлечена методами аналогичными обработки массива `$_POST` или `$_GET`. Причем в качестве хранилища данных о файле используется элемент этого массива с ключевым значением, совпадающим с именем поля типа `file`: в данном примере это `myfilename`. Таким образом, если необходимо загрузить несколько файлов для разных целей (например, копию различных документов) – в форме можно разместить несколько полей, а в массиве `$_FILES` появится несколько элементов.

Признаком отправки формы и попытки загрузки на сервер можно считать наличие в массиве `$_FILES` соответствующего элемента: `$_FILES['myfilename']['tmp_name']`, содержащее имя временного файла. Этот файл можно открыть и прочитать из него данные, или же просто

скопировать куда-либо целиком. При необходимости можно узнать о нем и другую полезную информацию, например, имя на локальном компьютере: `$_FILES['myfilename']['name']`.

Однако, пользователь может отправить форму, не выбрав файл – аналогично тому, как он может не заполнить поле. Тогда в массиве `$_FILES` все равно будет содержаться элементы, соответствующие полю типа `file`, но все они будут пусты, что дает возможность отследить такую ситуацию и правильно обработать ее. Например, это происходит при обновлении информации о человеке, содержащую как ФИО, так и его фотографию: если в форме были заполнены поля ФИО, но не указана фотография – сайт должен изменить только их, оставив старую фотографию. Если же фотография была загружена – она должна заменить предыдущее фото.

Согласно заданию, при отсутствии загружаемого файла должен быть удален указанный в поле `dir-name` каталог – модифицируем код для реализации этого требования.

Листинг В-3. 14

```
if( isset($_FILES['myfilename']) ) // были отправлены данные формы
{
	if( isset($_FILES['myfilename']['tmp_name']) )	// если файл загружен
	{
		if( $_FILES['myfilename']['tmp_name'] )	// если файл существует
		{
			// копируем его и выводим сообщение об успешной загрузке
			move_uploaded_file($_FILES['myfilename']['tmp_name'],
				makeName(.$_FILES['myfilename']['name']) );
			echo 'Файл '.$_FILES['myfilename']['name'].' загружен на сервер';
		}
		else
			rmdir( $_POST['dir-name'] );	// удаляем каталог
	}
}
```

Дополнительный условный оператор выполнит копирование временного файла только если он существует, в противном случае – указанный каталог будет удален. Обратите внимание: функция `rmdir()` удаляет только пустой каталог: чтобы удалить каталог с файлами необходимо предварительно удалить всё его содержимое – файлы и подкаталоги. Таким образом для удаления каталога необходимо разработать рекурсивную функцию `deleteCatalog()`, которая бы при запуске определяла содержимое каталога, проверяла возможность удаления файлов в нем, и, емли это возможно, удаляла их, после чего запускала саму себя для удаления подкаталогов – сделайте это самостоятельно и замените на нее стандартную функцию `rmdir()`.

Для завершения работы над кодом также необходимо реализовать функцию `makeName()`, которая использовалась выше для формирования имени файла на сервере. Для этого необходимо учесть два требования лабораторной работы: полное имя должно содержать каталог, в котором будет находится файл; имя файла генерируется как число от 1 с шагом 1.

Листинг В-3. 15

```
function makeName($filename)
{
	if( !file_exists($_POST['dir-name']) ) // если каталога не существует
	{
		umask(0);			// сбрасываем значение umask
		mkdir($_POST['dir-name'], 0777, true);	// создаем ее
	}
	$ext = end(explode('.', $filename));

	$n=1;		// начиная с 1 цикл пока существует файл
	while( file_exists($_POST['dir-name'].'/'.$n.'.'.$ext )	// с текущем номером
		$n++;					// - увеличиваем номер
	return ($_POST['dir-name'].'/'.$n.'.'.$ext;		// возвращаем свободное имя
}
```

Функция сначала проверяет существование каталога, в котором будет размещен файл – если его не существует, он создается с максимально возможными правами доступа (см. справочную информацию). Для получения расширения файла разобьем строку из его имени в массив, использовав в качетсве разделителя «;» и возьмем его последний элемент. Затем в цикле перебираются все числа от 1 и далее до тех пор, пока в каталоге существует файл из текущего числа и полученного расширения. Как только такого файла не будет – искомое имя будет найдено и оно возвратится как результат работы функции.

Теперь остается реализовать последний функционал модуля – сохранение информации о принадлежности файла, т.е. о том какой пользователь его загрузил. Для этого проще всего реализовать такой формат: в соответствующей строке файла *users.csv* после указания логина и пароля будет перечислены полные имена всех загруженных пользователем файлов.

Листинг В-3. 16

```
function updateFileList($filename)
{
	$info = file($filename);		// читаем все строки файла в массив
	$f=fopen($filename, 'wt');		// открываем файл для записи
	flock($f, LOCK_EX);		// блокируем файл исключительно

	foreach( $info as $k=>$user )	// для всех строк массива
	{
		$data = str_getcsv($user, ';');		// декодируем данные
		if( $data[0]== $_SESSION['user'][0] )	// если найден пользователь
			$user.=';'.$filename;		// добавляем к его файлам новый
		fputs($f, $user);			// сохраняем данные в файл
	}
	flock($f, LOCK_UN );		// разблокируем файл
	fclose($f);			// закрываем файл
}
```

Реализуем функцию `updateFileList()`, которая читает файл *users.csv* и для каждой его строки проверяет: если она описывает текущего пользователя, то в ее конец после символа «;» дописывается имя загруженного файла. После этого строка (измененная или нет) вновь сохраняется в файле – таким образом реализуется алгоритм: считать все из файла, обработать данные, сохранить их в тот же файл.

Но вполне возможна ситуация, когда во время загрузки файла программа будет проверять возможность аутентификации другого пользователя. Нетрудно видеть, что функция стирает файл *users.csv* – поэтому другой процесс при его чтении вообще не найдет в нем данных, если первый не успел их туда записать. Чтобы этого не случилось реализуется механизм блокировки с помощью функции `flock()`: заблокированный таким образом файл будет недоступен для чтения другим процессом до тех пор, пока не будет разблокирован. Самостоятельно модифицируйте код первого модуля для безопасного чтения файла *users.csv* с помощью механизма блокировки – тогда в описанной ситуации второй процесс будет ждать окончания сохранения данных первым и коллизии не случится.

Обратите внимание: при удалении файлов информация о их принадлежности сохранится в файле users.csv, что может привести к коллизиям если другой пользователь загрузит их: в этом случае окажется что файл будет загружен одновременно двумя пользователями, что невозможно. Во избежание этого при удалении файлов необходимо удалять и информацию о них – сделайте это самостоятельно.

Следует также упомянуть о существовании простого способа загрузки сразу нескольких файлов. Для этого необходимо немного изменить *HTML*-код описания поля типа `file` следующим образом: `<input type="file" name="myfilename[]" multiple>`. Тогда после запуска скрипта в элементе массива `$_FILES['myfilename']['tmp_name']` и других будет не строка, а массив со строками, что позволяет обработать загрузку любого количества файлов. Но применять такой подход разумно только если все они принадлежат одной категории: при обработке не будет возможности отличить фотографию от скана документа.

Листинг В-3. 17

```
if( isset($_FILES['myfilename']) ) // были отправлены данные формы
{
	foreach( $_FILES['myfilename']['tmp_name'] as $i=>$f )
	{
		echo 'Загрузка '.($i).' – отправлено во временный файл '.$f;
	}
}
```

Для окончания работы над модулем самостоятельно реализуйте указанный тип загрузки нескольких файлов и окончательно скомпонуйте *PHP*-скрипт согласно указанной в листинге В-3.6 структуре.

Вывод содержимого файла в браузере

Третий модуль самый простой из реализуемых в данной работе – он должен выводить содержимое файла в браузере. Для этого используются функции манипулирования данными файла.

Листинг В-3. 18

```
if( !isset($_SESSION['user']) ) { echo 'Необходима авторизация'; exit(); }
$f = fopen( $_GET['filename'], 'rt' );		// открываем файл в текстовом режиме
if( $f )					// если файл успешно открыт
{
	$content = '';				// содержимое файла пока пусто
	while( !feof($f) )			// цикл, пока не достигнут конец файла
		$content .= fgets( $f );	// читаем строку файла

	echo $content;				// выводим содержимое файла
	fclose( $f );				// закрываем файл
}
else
	echo 'Ошибка открытия файла '. $_GET['filename'];
```

Файл открывается для чтения и в случае успеха построчно копируется в переменную `$content`, которая и выводится в браузер. Самостоятельно доработайте код таким образом, чтобы:

- выводилось сообщение при отсутствии параметра filename среди переданных;
- выводилось сообщение при отсутствии на сервере запрашиваемого файла;
- выводить сообщение при обращении к файлу users.csv;
- выводилось сообщение если открываемый файл был загружен другим пользователем – в этом случае содержимое файла не должно отображаться;
- для html-файлов в браузере выводился их html-код, а не внешний вид страницы.

Справочная информация

Файл – это именованная область данных на носителе информации. Работа с файлами – важная часть любого языка программирования. Даже в такой области, как ВЕБ-разработка, файлы используются довольно часто: для загрузки большого объема информации в базу данных, ведения журналов, составление и передача различных отчетов и т.д.

Все функции для работы файлами можно разделить на две группы:

- манипулирование с данными в файлах;
- манипулирование собственно файлами.

Первая группа операций предназначена для работы с информацией в файлах (запись и чтение), тогда как вторая группа – для работы с файлами целиком (копирование, переименование, удаление файлов и т.д.). Рассмотрим каждую группу подробнее.

Манипулирование данными файлов

PHP предоставляет ряд функций для доступа к данным в файлах. Но прежде чем рассматривать сами функции, необходимо понять разницу между двумя режимами доступа: бинарным и текстовым.

Текстовый режим предполагает работу со строками, т.е. основными функциями являются чтение и запись строки. При этом в разных ОС разные признаки конца строки: в *Linux* это символ «\n», в *Windows* – два символа «\r\n». Интерпретатор *PHP* транслирует (преобразовывает) эти признаки для файлов, созданных в других ОС, в поддерживаемый текущей ОС формат. Т.е. при чтении файла в текстовом режиме в ОС *Linux* из него автоматически будут удалены все символы «\r».

Если же они важны, т.е. если файл на самом деле не текстовый (но в нем все равно есть байты, которые совпадают с кодом символа конца строки), то прочитанные данные будут критически отличаться от хранящихся в файле, что недопустимо для их дальнейшей обработки. Поэтому для цифровых файлов используется бинарный режим, предполагающий работу с информацией неопределенного типа. При использовании этого режима записанная и прочитанная информация измеряется байтами, чтение и запись происходит без изменения данных.

Открытие файла	`$f=fopen($filename, $mode);` Открывает файл `$filename` в режиме `$mode`: прежде чем прочесть или записать данные в файл его необходимо открыть – операционная система считает в память его заголовки, подготовится к работе с данными файла. Функция может открывать как локальные файлы, так и файлы на удаленных серверах (используя *URL*). Имя локального файла `$filename` может быть относительным (от текущего каталога, обычно совпадает с каталогом открывающего файл *PHP*-скрипта) или абсолютным. В относительном имени возможно указывать каталоги «.» (текущий каталог) и «..» (родительский каталог). Например, «*../files/file.txt*» – в каталоге, содержащим текущий каталог, необходимо открыть каталог «*files*» и уже в нем найти файл file.txt. Режим доступа к данным файлам (режим открытия файла) `$mode` определяется следующей строкой: • `r` – режим только для чтения, указатель помещается в начало файла. • `r+` – режим для чтения и записи, указатель помещается в начало файла. • `w` – режим только для записи, указатель помещается в начало файла. Если файл существует – все данные из него удаляются (длина файла обрезается до 0). Если файл не существует – создается новый файл. • `w+` – режим для чтения и записи, указатель помещается в начало файла. Если файл существует – все данные из него удаляются (длина файла обрезается до 0). Если файл не существует – создается новый файл. • `a` – режим только для записи, указатель помещается в конец файла. Если файл не существует – создается новый файл. • `a+` – режим только для чтения и записи, указатель помещается в конец файла. Если файл не существует – создается новый файл. Упоминающийся в описании режимов указатель (файловый указатель) – номер байта данных (смещение от начала файла) с которого будет производиться следующая операции чтения или записи в файл. По умолчанию все файлы открываются в бинарном режиме, для открытия в текстовом режиме к параметру необходимо добавить флаг `t` (например, режим доступа `rt` – только для чтения, текстовый режим). Следует также учитывать и саму возможность открытия файла в требуемом режиме: если открывается локальный файл, то это определяется правами доступа, как к самому файлу, так и к содержащей его папке. Если указывается *URL* – то допустимость режима определяется используемым протоколом: например, файл «*http://mysite.ru*» не может быть открыт для записи. Кроме того, директива (чаще всего настройка в файле *php.ini*) *allow_url_fopen* может запрещать работу с объектами *URL* как с обычными файлами. В этом случае для работы с удаленными файлами следует использовать альтернативные способы. Кроме указанных выше, в функции есть ряд дополнительных параметров, расширяющих ее возможности, но лежащих за пределами данного описания. Функция возвращает

<table>
<tr><td></td><td>идентификатор ресурса (дескриптор файла, указатель файла) для дальнейшей работы других функций, или <code>false</code> в случае какой-либо ошибки.</td></tr>
<tr><td>Закрытие файла</td><td><code>fclose($f);</code>
Закрывает файл с дескриптором <code>$f</code>, соответственно до этого файл должен быть открыт функцией <code>fopen()</code>. Возвращает <code>true</code>, в случае успешного закрытия файла, <code>false</code> – в случае ошибки.</td></tr>
<tr><td>Определение размера файла</td><td><code>filesize($filename)</code>
Функция возвращает для файла с именем в параметре <code>$filename</code> его размер в байтах.</td></tr>
<tr><td>Чтение из файла данных</td><td><code>$str=fread($f, $length);</code>
Функция читает из открытого в поддерживающем чтение данных режиме файла с идентификатором <code>$f</code> блок данных размером <code>$length</code> байт (или менее, если при чтении данных достигнут конец файла). При успешном чтении <code>$length</code> байт файловый указатель смещается на это же количество байт ближе к концу файла, т.е. при следующем обращении к функции будет прочитан новый блок данных. Функция возвращает прочитанные данные в виде строки. Может использоваться совместно с <code>filesize()</code> для чтения всего файла разом.</td></tr>
<tr><td>Запись данных в файл</td><td><code>fwrite($f, $data);</code>
Функция записывает в открытый в предусматривающем запись режиме файл с идентификатором <code>$f</code> блок данных из строки <code>$data</code>. Строка используется как наиболее удобный тип данных в PHP и может содержать любые данные, в том числе и не текстовые. Если в функцию передан необязательный параметр <code>$length</code>, то он определит максимальное количество записанных в файл байт данных. Т.е. запись остановится после <code>$length</code> записанных байт или при записи всех данных <code>$data</code>, смотря, что про изойдет первым. После записи файловый указатель смещается ближе к концу файла на записанное количество байт. Возвращает количество записанных байт или <code>false</code> в случае ошибки.</td></tr>
<tr><td>Чтение строки из текстового файла</td><td><code>$str = fgets($f);</code>
Читает из открытого в текстовом режиме файла одну строку, включая символ окончания строки (если он есть), и возвращает ее как результат. Если указан необязательный параметр <code>$length</code>, то строка будет содержать не более <code>$length-1</code> символов. Сдвигает файловый указатель на прочитанное количество символов ближе к концу файла – обычно (если параметр <code>$length</code> не указан) это начало следующей строки. Для ОС Linux результат последовательного вызова нескольких функции fgets() можно представить следующим образом.
<table>
<tr><th>Содержимое файла ($f)</th><th>Вызов функции</th><th>Результат</th><th>Длина результата</th></tr>
<tr><td rowspan="5">1234\n
5678\n
90AB\n
CDEF</td><td>fgets($f)</td><td>1234\n</td><td>5</td></tr>
<tr><td>fgets($f, 4)</td><td>567</td><td>3</td></tr>
<tr><td>fgets($f, 4)</td><td>8</td><td>1</td></tr>
<tr><td>fgets($f, 100)</td><td>90AB\n</td><td>5</td></tr>
<tr><td>fgets($f, 100)</td><td>CDEF</td><td>4</td></tr>
</table>
Файл из четырех строк («1234», «5678», «90AB», «CDEF») считывается четырьмя вызовами функции <code>fgets()</code>. Первый вызов (без указания длины строки) возвращает первую строку вместе с символом «\n». Второй вызов начинает чтение со второй строки и возвращает 3 символа, т.к. в функцию передано ограничение длины читаемой строки. Третий вызов начинает чтение не с третьей строки, а с символа «8», т.к. именно на него был перемещен файловый указатель. Четвертый вызов, аналогично первому, возвращает строку из 5 символов. Пятый – только строку из 4-х символов, т.к. последняя строка в файле не заканчивается символом «\n».</td></tr>
<tr><td>Перемещение файлового указателя</td><td><code>fseek($f, $offset);</code>
Смещает файловый указатель открытого файла на <code>$offset</code> байтов. Точка отсчета определяется необязательным параметром <code>$whence</code>:
• <code>SEEK_SET</code> – смещение указано от начала файла.
• <code>SEEK_CUR</code> – смещение указано от текущего положения указателя.
• <code>SEEK_END</code> – смещение указано от конца файла.</td></tr>
</table>

	По умолчанию (если параметр `$whence` не указан) смещение происходит от начала файла. Функция возвращает «0» в случае успешной смены позиции указателя, «-1» – в случае возникновения ошибки. Например, если открыть текстовый файл и перед чтением строки функцией `fgets()` вызвать данную функцию `fseek($f, 2)`, то строка будет прочитана не с начала, а с третьего символа (первый плюс два).
Определение конца файла	`feof($f);` Функция проверяет текущее положение файлового указателя. Возвращает `true` если достигнут конец файла и `false` в противном случае.
Получение массива строк текстового файла	`$array = file($filename);` Функция читает файл с указанным именем (предварительно открывать файл не надо, необходимо просто указать его имя) и разбивает его на массив строк. Строки включают в себя символ окончания строки («\n» или «\r\n»), причем трансляции символов не происходит – окончание строки будет отмечено именно так, как записано в файле при его создании.
Чтение всего файла сразу	`$str = file_get_contents($filename);` Читает указанный файл с именем `$filename` в одну строку. Предварительно открывать файл не надо – следует просто указать имя файла.

Блокировка и совместный доступ к файлам

Основной целью любой программы на *PHP* является подготовка *HTML*-кода страниц сайта, доступных множеству посетителей одновременно. Поэтому следует учитывать возможность ситуации, при которой к одному и тому же файлу будут одновременно обращаться два процесса. Если оба процесса читают содержимое файла, то ничего страшного. Но если один из них записывает, а другой читает, или же оба записывают – конфликты неизбежны.

Для их разрешения *PHP* предоставляет механизм блокировки файлов с помощью функции `flock($f, $operation)`. Для открытого файла с дескриптором `$f` устанавливается режим блокировки `$operation`, определяемый одним из следующих значений.

- `LOCK_SH` – разделяемая блокировка. Если процесс «А» блокирует файл с помощью такой блокировки, то другие процессы могут открывать его для чтения и читать данные. Однако попытка открыть файл для записи будет блокирована – процесс «В» перед началом записи данных будет ожидать снятия блокировки процессом «А».
- `LOCK_EX` – исключительная блокировка. Никакой другой процесс, кроме установившего блокировку, не может ни читать, ни записывать данные в файл до тех пор, пока блокировка не будет снята. Другие процессы при обращении к файлу приостановят свою работу до тех пор, пока файл не будет разблокирован. Такая блокировка устанавливается перед обновлением (записью) информации в файле.
- `LOCK_UN` – снятие блокировки с файла. Файл считается заблокированным до тех пор, пока блокировка не снята. Если Вы завершили работу с файлом (завершили запись данных) – немедленно снимайте блокировку, чтобы другие процессы могли продолжить свою работу.

Блокировка заставляет другие процессы ожидать его разблокирования. Иногда полезно не ждать этого, а продолжить работу: вывести сообщение об ошибке или обратиться к другому файлу. Для этого к значению параметра `$operation` следует прибавить константу `LOCK_NB` – она запрещает процессу ожидать разблокировки. В этом случае попытка обращения к заблокированному файлу приведет не к ожиданию, а к возврату ошибки. Необязательный третий параметр принимает значение `true` если файл недоступен для блокировки (заблокирован другим процессом). Фактически он дублирует возвращаемый функцией результат.

Листинг В-3. 19

```
$f = fopen('filename.txt', 'a+');        // открываем файл для записи
if( $f )                                 // если файл успешно открыт
{
       flock( $f, LOCK_EX );             // блокируем файл исключительно
```

```
                                        // если файл используется другим сценарием –
                                        // ожидание разблокировки файла
        fwrite( $f, $data );            // запись в файл данных
        flock( $f, LOCK_UN );           // разблокировка файла
        fclose($f);                     // закрытие файла
}
```

В примере производится безопасная запись данных в файл. Перед началом записи осуществляется попытка исключительной блокировки файла. Если файл используется другими скриптами (или этим же скриптом, но в другом процессе) – интерпретатор ждет разблокировки файла. Запись производится только тогда, когда файл заблокирован текущим процессом.

Листинг В-3. 20

```
$f = fopen('filename.txt', 'r');        открываем файл для чтения
if( $f )                                // если файл успешно открыт
{
        flock( $f, LOCK_SH );           // блокируем файл разделяемо
        $data = fgets( $f );            // чтение из файла данных
        flock( $f, LOCK_UN );           // разблокировка файла
        fclose($f);                     // закрытие файла
}
```

Безопасное чтение из файла осуществляется аналогичным образом. При попытке разделяемой блокировки программа приостановит свое выполнение, пока существует процесс с исключительной блокировкой файла (пока в этот файл другой ппроцесс записывает какие-либо данные).

Листинг В-3. 21

```
function writeFileNow( $base, $data )
{
        $f = fopen( $base, 'w');                        // открываем файл для записи
        while( !flock($f, LOCK_EX+LOCK_NB) )            // цикл пока файл заблокирован
        {
                fclose( $f );               // закрываем файл
                $base .= 'A';               // переименовываем его
                $f = fopen($base, 'w');     // открываем заново
        }
        fwrite( $f, $data );                // запись в файл данных
        flock( $f, LOCK_UN );               // разблокировка файла
        fclose($f);                         // закрытие файла
        return $base;                       // возвращаем имя файла
}
```

Рассмотрим пример блокировки файла без ожидания. Пусть функция `writeFileNow()` записывает данные в файл: в начале файл открывается, затем в цикле проверяется возможность блокировки файла. Если файл нельзя заблокировать – он закрывается, формируется новое имя файла, файл снова открывается. На следующей итерации попытка блокировки продолжается. В конце концов находится незаблокированный файл, в который и сохраняются данные, его имя возвращается как результат работы функции. Таким образом блокировка файлов эффективно решает задачу совместного их использования в многопользовательских системах.

Использование CSV-файлов

Формат хранения данных *CSV* (*Comma Separated Values* — значения, разделённые запятыми) – удобных способ передачи табличных данных между различными программами. Он представляет собой текстовый файл, данные которого организованны в таблицу следующим образом: строки таблицы – это строки в тексте; столбцы (ячейки) таблицы – это данные в строке, разделенные символом «;» или «;». Формат очень удобен тем, что с ним корректно работает *Excel*: таблицу можно преобразовать в такой формат при сохранении. Стоит отметить, что иногда (особенно если в ячейке таблицы присутствует текст с кавычками) – *Excel* при конвертировании «обернет» содержимое ячейки в дополнительные кавычки.

Таблица с данными

Москва	128	Все "ОК"
Санкт-Петербург	2345	В июле
Казань	12742	

CSV-файл с данными

Москва;128;"Все "ОК""

Санкт-Петербург;2345;В июле

Казань;12742;

В *PHP CSV*-файлы можно обрабатывать как обычные текстовые файлы: открывать, читать и сохранять в них данные – главное придерживаться указанного формата и структуры. Однако есть и дополнительные функции, позволяющие эффективнее работать с такими файлами.

Чтение и разбор сроки из CSV-файла	`fgetcsv($f, $length, $delimiter);` Функция работает аналогично `fgets()`, но возвращает строку, не просто как она есть, а уже разобрав ее в массив, используя разделитель `$delimiter`. Второй параметр `$length` указывает максимальную длину считываемой и разбираемой строки. Второй и третий параметр не обязательны: по умолчанию равны `0` (строка без ограничений, но скорость работы несколько медленнее) и `","`.
Запись данных из массива в файл	`fputcsv($f, $array, $delimiter);` Для записи данных из массива `$array` в CSV-файл также можно использовать специальную функцию. В переменной `$delimiter` указывается разделитель ячеек таблицы. Функция имеет и другие необязательные параметры.
Разбор строки в массив	`str_getcsv($str, $delimiter);` Преобразует строку из разделенных символом `$delimiter` элементов в массив. Аналог функции `explode()`.

Манипулирование файлами и каталогами

Функции манипулирования работают собственно с файлами и каталогами. Они позволяют удалять, переименовывать, копировать файлы и т.д.

Перенос загруженного по *http* файла на сервер	`move_uploaded_file( $src, $dst )` Проверяет был ли файл `$src` загружен по протоколу *http* и, если да – переносит его с переименованием в `$dst` (полное имя файла). При попытке переноса системного или любого другого файла, кроме загруженных по *http*, функция работать не будет. Если файл с именем `$dst` уже существует – он будет заменен. Возвращает `true` при удачном завершении, `false` – в случае каких-либо ошибок.
Проверка был ли файл загружен на сервер по *http*	`is_uploaded_file($filename)` Проверяет был ли указанный файл `$filename` (имя файла уже на сервере) загружен по *http* – применяется для обеспечения безопасности и недопущении доступа злоумышленников к системным файлам.
Копирование файла	`copy($src, $dst)` Копирует файл с именем `$src` в файл с новым именем `$dst`. Если файл `$dst` уже существует – он будет перезаписан заново. В случае успешного копирования возвращает `true`, в случае ошибки – `false`.
Проверка существования файла	`file_exists($filename)` Проверяет существование файла или каталога с указанным именем `$filename`. Если файл или каталог существует – возвращает `true`, если нет – `false`.
Переименование файла	`rename($oldname, $newname)` Переименовывает файл с именем `$oldname` (должен существовать) в файл с именем `$newname` (не должен существовать). Возвращается `true`, если переименование удалось, `false` – если нет.
Удаление файла	`unlink($filename)`

	Удаляет файл с именем `$filename`. Если удалить файл оказалось невозможно – возвращает `false`.
Создание нового каталога	`mkdir($path, $perms)` Создает пустой каталог именем `$path` и указанными правами доступа `$perms` (см. следующий подраздел). При создании нового каталога, во-первых, следует обращать внимание на существование каталога с таким же именем, во-вторых, у процесса с *PHP*-скриптом должны быть права записи в тот каталог, где он создается. Третий, необязательный параметр определяет: создавать ли указанные в `$path` несуществующие подкаталоги (`true`), или же запрещать (`false` – по умолчанию). Если создать каталог невозможно – функция вернет `false`.
Удаление каталога	`rmdir($path)` Удаляет пустой каталог с именем `$path`. Если каталог содержит файлы или подкаталоги, либо же каталог не может быть удален по другой причине – функция вернет `false`.
Определение текущего каталога	`getcwd()` Возвращает абсолютное имя текущего каталога. Часто применяется для построения абсолютных имен к файлам при модульном программировании.
Изменение текущего каталога	`chdir($path)` Заменяет текущий каталог на указанный в `$path`. Если смена невозможна – возвращает `false`. Если смена произошла успешно – возвращает `true`.
Проверка файла	`is_file($filename)` Функция возвращает `true`, если указанный `$filename` является обычным файлом, `false` – в противном случае.
Проверка каталога	`is_dir($path)` Функция возвращает `true`, если указанный `$path` является обычным каталогом, `false` – в противном случае. Именем каталога может быть не только абсолютный путь, но и относительный, с использованием имен «.» (текущий каталог) и «..» (родительский каталог).
Доступ к элементам каталога (к файлам и подкаталогам)	`opendir($path)` Открывает каталог `$path` для чтения его структуры и возвращает его дескриптор. Если каталог не существует или не может быть открыт по каким-либо иным причинам – функция вернет `false`.
Окончание работы с каталогом	`closedir($handler)` Закрывает предварительно открытый функцией `opendir()` каталог с дескриптором `$handler`. Функция ничего не возвращает.
Чтение элемента каталога	`readdir($handler)` Функция возвращает следующий по порядку (зависит от ОС) элемент открытого функцией `opendir()` каталога с дескриптором `$handler`. Если следующего элемента нет – возвращает `false`.

ПРАВА ДОСТУПА К ФАЙЛАМ

Операционная система для каждого файла и каталога всегда определяет права доступа – т.е. тот набор операций, которые могут совершать с ним те или иные пользователи. Эти права часто не дают манипулировать файлами или данными так, как это необходимо. Поэтому следует понимать не только как их прочитать и интерпретировать, но и как изменить.

0	7	5	4
Признак шестнадцатеричного числа.	**Права владельца файла**. 1+2+4=7, т.е.	**Права участников группы владельца сайта**. 5=4+1, т.е. права	**Права обычных пользователей**. 4 – только на чтение файла.

	права на чтение, запись и исполнение.	только на чтение и исполнение.	

Чаще всего права доступа определяются в виде восьмеричного числа (начинается на ноль) в котором поразрядно определяются права для владельца файла, для группы владельца файла и для всех остальных пользователей. Сумма чисел 4 (доступ на чтение), 2 (на запись) и 1 (на выполнение) определяют искомый доступ.

Кроме того, при работе с файлами на их права накладывается текущее значение маски *umask*: маска *umask* (определяется в настройках *Linux*) по умолчанию равна 0002 для пользователя и 0022 для пользователя *root*. Маска "отбирает" указываемые права поразрядно: 7 – все права, 2 – на запись, 0 – оставляет как есть. Поэтому, чтобы, например, создать каталог с правами 0777 необходимо предварительно сбросить маску 0002 в 0000, иначе созданный каталог будет иметь "меньше прав" – всего 0755. Для изменения прав доступа к файлу или каталогу применяются следующие функции (в зависимости от ОС результат работы функций может отличаться).

Изменение маски прав *umask*	`umask($mask)` Изменяет маску прав доступа umsk на новое значение `$mask` и возвращает предыдущее значение. Если параметр `$mask` не указан – просто возвращает значение маски.
Изменение прав доступа к файлу или каталогу	`chmod($filename, $perms)` Изменяет права доступа к файлу или каталогу `$filename` на `$perms`. Возвращает `true`, если права изменены, `false` – если нет.
Определение прав доступа к файлу	`fileperms ($filename)` Возвращает восьмеричное число – права доступа к файлу `$filename`.

Для улучшения внешнего вида и упрощения программы можно не определять и "декодировать" права доступа, а использовать следующие функции, автоматически учитывающие статус текущего процесса *PHP* для проверяемого файла.

Проверка возможности читать из файла	`is_readable($filename)` В соответствии с правами доступа для процесса с программой *PHP* определяет возможность открытия файла с именем `$filename` для чтения. Возвращает `true`, если файл может быть прочитан, `false` – если нет.
проверка возможности записи в файл	`is_writeable($filename)` В соответствии с правами доступа для процесса с программой *PHP* определяет возможность открытия файла с именем `$filename` для записи. Возвращает `true`, если файл может быть записан, `false` – если нет.
Проверка возможности запустить файл на выполнение	`is_executable($filename)` В соответствии с правами доступа для процесса с программой *PHP* определяет возможность выполнения файла с именем `$filename`. Возвращает `true`, если файл является исполняемой программой и имеет соответствующие права доступа, `false` – если нет.

Использование URL-кодирования

При использовании ссылок с *GET*-параметрами может возникнуть ситуация, когда какое-либо значение может быть интерпретировано как часть *URL*. Например, если параметр содержит строку с символом «#», то все что будет размещено правее него будет не будет воспринято как его значение. Более того, все параметры правее символа будут интерпретированы как имя якоря, их значения не будут переданы для обработки. Похоже дело обстоит и с другими значимыми символами: «@», «&» и др.

Например, передача параметров *p1=”123#45”* и *p2=”x&y=z”* в адресе ссылки без специальных мер приведет к переходу по URL `?p1=123#45&p2=x&y=z`. Если бы мы не знали заранее о передаваемых параметрах, легко можно было бы предположить, что их три: `p1`, `p2` и `y`. Более того, символ # означает переход на соответствующий якорь, а значит реально в программу придет только значение только одного параметра, да и то не до конца: *p1=”123”*.

Для предотвращения этого разумно при формировании адреса ссылки кодировать значение параметров, заменяя все символы их значений на *URL*-коды. Тогда можно совершенно безопасно передать любую информацию в *GET*-параметре ссылки, будучи уверенным что информация не исказится.

Кодировать все символы строки в URL-коды	`$param = urlencode($str);` Возвращает закодированную для безопасной передачи в браузер строку.
Раскодировать строку из URL-последовательности	`$str = urlencode($param);` Раскодирует строку. При передаче закодированных значений параметров методом GET раскодировать их самостоятельно в PHP-программе нет необходимости – в массиве $_GET они окажутся уже декодированными!

КОНТРОЛЬНЫЕ ВОПРОСЫ К ЛАБОРАТОРНОЙ РАБОТЕ

Для успешной защиты работы помимо соответствующего требованиям результата необходимо уверенно отвечать на нижеперечисленные и другие вопросы.

1. Назовите отличия идентификации, аутентификации и авторизации?
2. Назовите группы функций для работы с файлами и их отличие друг от друга.
3. Сформулируйте отличие бинарного и текстового режима работы с файлами.
4. Назовите режимы доступа к фалам в PHP.
5. Укажите отличие функций fread() и fgets().
6. Определите файловый указатель, расскажите о перемещающей его функции.
7. Расскажите известные Вам способы чтение файла в строку и массив.
8. Расскажите, когда и как следует проводить проверку существование файла.
9. Сформулируйте отличие при удалении файла и каталога.
10. Расскажите о создание нового каталога и правах доступа к нему.
11. Расскажите об использовании маски прав доступа umask.
12. Расскажите о способе получения элементов каталога. Назовите элементы каталога.
13. Сформулируйте режимы блокировки файла и способы их реализации.
14. Как изменится естественно-языковое описание алгоритма аутентификации после добавления в него возможности выхода (снятия состояния аутентификации пользователя)?
15. Почему в листинге В-3.2 проверка совпадения логина и пароля переданным параметрам разнесены в разные условные операторы?
16. Почему в листинге В-3.2 следует использовать прекращение цикла break? Что изменится в проверке аутентификации, если его убрать?
17. Возможна ли в листинге В-3.3 ситуация, когда при проверке совпадения имени пользователя произойдет ошибка из-за того, что массив $test_user не содержит элемента с индексом '0'?
18. Как изменится результат работы программы, если в листинге В-3.4 заменить конструкцию include на require, include_once, require_once?
19. Как для пользователя изменится работа сайта, если из обработчика выхода в листинге убрать переадресацию и принудительное завершение программы?
20. Как изменится работа программы, если в листинге В-3.6 перенести обработку загрузки файлов в конец программы?
21. В чем отличие кода листинга В-3.7 и В-3.8 кроме приведенных в описании лабораторной работы?
22. Где и каким образом определяются настройки сервера, влияющие на возможность загрузки файлов. Перечислите их.
23. Возможна ли загрузка файла через форму с методом GET?
24. Как организована структура CSV-файлов?

25. Какие специальные функции PHP используются для работы с CSV-файлами? Можно ли обойтись без них? Если да – то как?
26. Необходимо ли в листинге В-3.18 для добавления проверки принадлежности открываемого файла аутентифицированному пользователю обращаться к файлу users.csv? Если да – нужно ли при этом реализовывать механизм блокировки файла? Если использовать блокировку не нужно – то почему, если да – то какого типа блокировку использовать?
27. Какие изменения в какие листинги необходимо внести для добавления к информации о пользователе в файле user.csv его ФИО?