The Pennsylvania State University
The Graduate School
Capital College

An Assignment Management System

A Master's Paper in
Computer Science
by
Xiaoyu Sun

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Master of Science

December, 2002

Abstract

Traditional assignment management is becoming obsolete due to its inconvenience, inefficiency, and low accuracy. Currently, web-based management systems have been widely implemented due to the development of Web and CGI technologies. This paper introduces a new web-based Assignment Management System, which combines all useful features in other commercial systems and implements new functions that are practical in assignment management. Its powerful features and friendly user interfaces allow instructors and students to handle their assignments in a convenient, efficient, and systematical way. In addition, this system also has very good portability and extensibility, and the system security has been strongly enhanced by multiple security strategies.

Acknowledgement

I would like to have this opportunity to thank my project advisor, Dr. Pavel Naumov. His patience, encouragement and advices have greatly helped me during the implementation. I could not achieve the success without his help.

I am grateful to the other committee members: Dr. Linda Null, Dr. Thang N. Bui, and Dr. Qin Ding for reviewing my paper. I also appreciate the help from Dr. Bui, Dr. Null, and Mr. Hans Royer during my Master's study.

Finally, I feel a deep sense of gratitude of my family who has been giving me continuous support throughout my study.

Table of Contents

List of Figures

1. Introduction

Assignment management is one of the fundamental activities in education. In traditional assignment management schemes, assignments are recorded on paper, floppy disks, and emails. They have to be delivered or organized manually. This is inconvenient and inefficient, and may cause many problems due to material limits and human errors, such as disordered printouts (without page numbers), damaged floppy disks, etc. Furthermore, instructors and students usually are unaware of the ongoing academic information, which can be used to help them find out problems and improve their teaching and learning qualities. Therefore, it becomes important to implement an assignment management method, which can provide both instructors and students with quality educational services [1].

Along with the development of web and Common Gateway Interface (CGI) technologies, more and more web-based systems have been implemented. In such systems, users can handle their management in an extremely convenient way: they can access the systems from anywhere at any time; get responses immediately; make use of countless online resources and share their own with others. More importantly, they do not need to worry about the operating systems and different application software on either server or client side. All they need to have is a browser and the ability to get online. Presently, many assignment-management-related systems can be found easily on the Internet. Following are some of them:

1. The Blackboard System at Carnegie Mellon is a course management system, whose web site is http://www.cmu.edu/blackboard. With this system, instructors can add new courses and upload course materials (including announcements, course information, staff information, and assignments) online; while students can submit information to instructors, check course calendars and grades, manage homepages, and edit their profiles online. Furthermore, instructors and students can communicate with each other via discussion board or groups [2].

2. Edulink is an online home school system, which can be accessed at http://www.edu-link.com. In this system, parents and teachers can make assignments, review student's lessons, access curriculum materials, which are presented in a variety of formats from illustrated text to fully interactive 3-D experiences, and utilize teaching tools to create new assignments and view student assignment information. Students can review assignments, access curriculum materials and utilize learning tools. Similar to *the Blackboard System*, both educators and students can communicate with each other via discussion tools [3].

3. The Pennsylvania State University's Angel System is another example of a course management system, which can be accessed at http://cms.psu.edu/frameIndex.html. The main features that *Angel* provides are: a user profile page, which lists all courses that the user is either taking or teaching; a syllabus template, which helps instructors create an HTML syllabus; a course calendar, which can be used by instructors to post events and announcements; a lessons tool, which is used to build and manage course content; drop boxes, which are used by students to submit their homework assignments; and an in touch menu, which lets users compose and read course mails, and participate in online chats and discussion [4].

This paper introduces a self-developed web-based Assignment Management System (AMS). It is not designed to compete with the above commercial systems, which cover all education activities: teaching, learning, communication, assignment management, and provide some fancy features that are more appropriate for online education. Instead, this system focuses on the assignment management part. All useful features about assignment management of other systems are included in this system and new functions that are practical in the management are added. The objective is to provide instructors and students with a convenient, efficient, and secure way to handle their assignment activities.

2. System Features and Design

The system was designed based on our teaching and studying experience at Penn State Harrisburg. It can perfectly meet the needs of most instructors and students. The system applications are installed on *abacus*, an Apache web server with Linux operating system. All CGI scripts are written in Perl[1] and generate a set of friendly dynamic HTML pages according to user requests. This system also implements a Database system, which uses MySQL DBMS (Database Management System). Because of the cross-platform attributes of Perl and MySQL, this system can easily migrate to other operating systems or employ to other DBMSs with slight modifications. The main features of the system and its implementation will be discussed in detail in the following sections.

2.1 Features – From a User's Perspective

Three types of users typically handle different tasks in assignment management: administrators, instructors, and students. For an administrator, the main tasks are database cleanup and database initialization. Database initialization includes course schedule and instructor information initialization. For an instructor, the main tasks are: setting up a syllabus, which includes all the information related to grading criteria; posting assignments, which may include both electronic files and hand-outs; monitoring assignment submission status; grading; checking class academic status; and searching student information. For a student, tasks include checking posted homework assignments, uploading electronic assignment files, and checking both personal and class academic status. All users access the system via a login page, which is shown in Figure 1. Upon logging in, the system automatically executes user identification validating process. If both the user name and password are considered valid, the user enters the system. Otherwise, an error message will prompt the user to attempt the login process again.

[1] Perl stands for Practical Extension and Report Language.

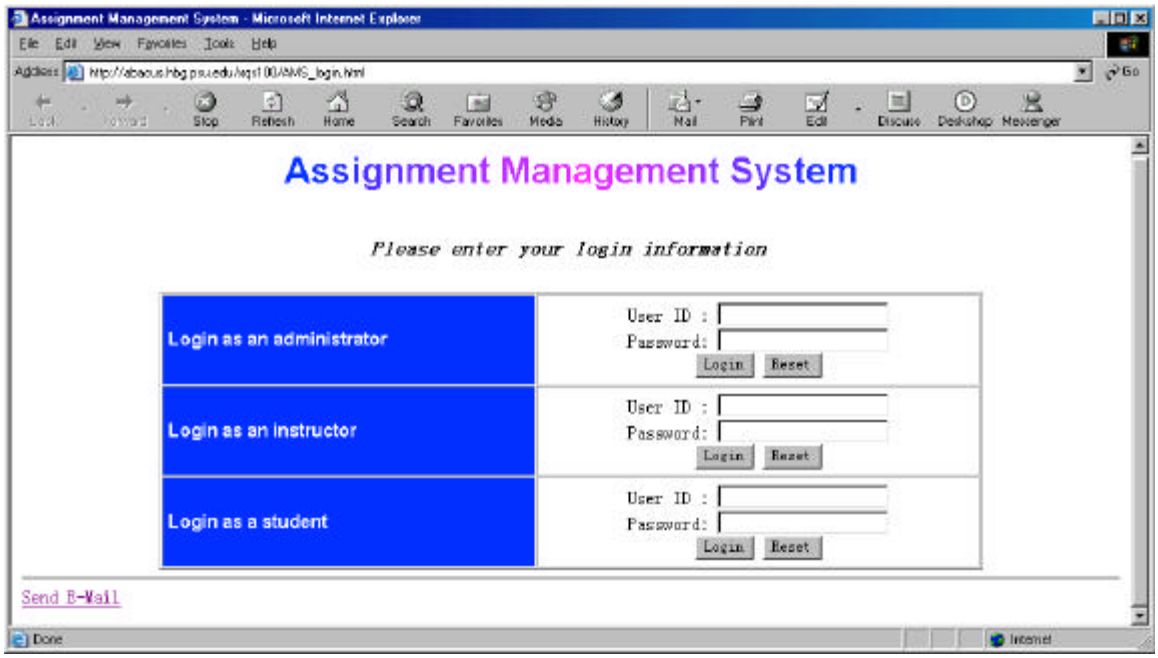

Figure 1. Login page

2.1.1 Instructor

The instructor part is the most important part of the system since instructors are usually responsible for most of the assignment management. The main function bar and log out button are displayed on all pages, so that instructors can conveniently switch between different web pages. The main function bar is composed of eight submit buttons: *Home*, *Password*, *Syllabus*, *Class List*, *Assignment*, *Submission*, *Grading*, and *Student Information* (see Figure 2). They are used to perform different management tasks and are explained following.

2.1.1.1 Home

After an instructor logs onto the system, a personal home page is displayed (Figure 2). All the personal and teaching information of this instructor is shown on that page.

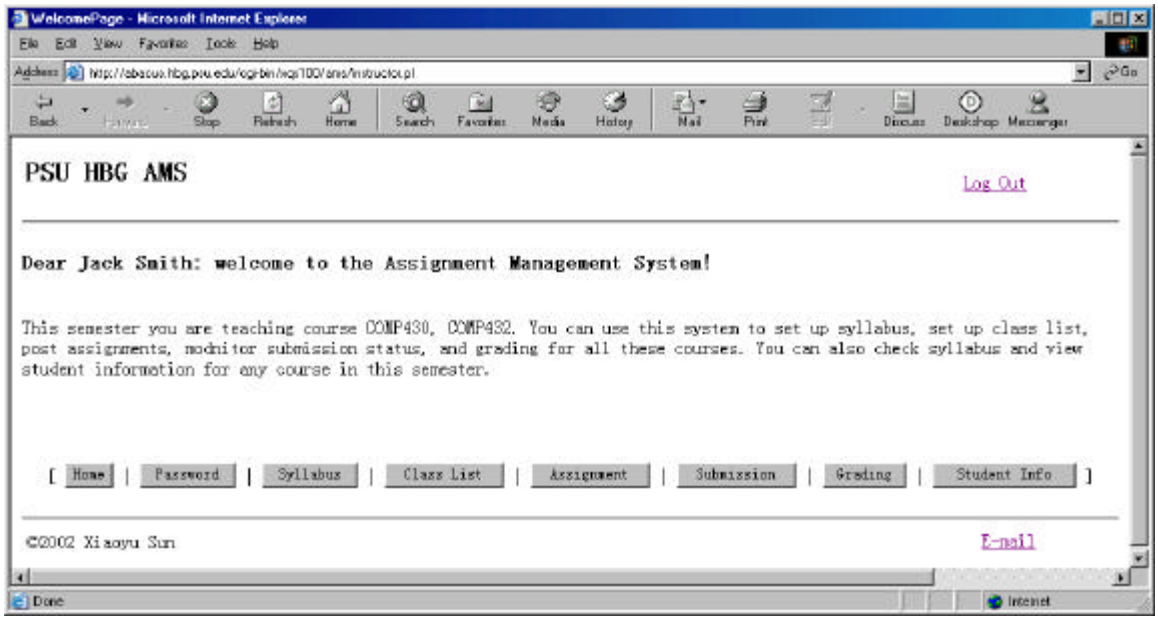

Figure 2. Example of an instructor home page

2.1.1.2 Password

In order to change his/her password, an instructor needs to input both the old and the new passwords. First, the AMS system checks if the old password is the correct one for the current user. Then it checks the new password, which needs to be typed twice, and should be eight to twenty characters long. For valid inputs, the system automatically updates the instructor's password and returns a success notice to the user. For invalid inputs, an error message will be displayed to prompt the user re-execute the process. Figure 3 shows such an example.

Figure 3. Example of a changing password page

2.1.1.3 Syllabus

The syllabus is defined here differently than in other assignment management systems. It is mainly used to maintain the information about course grading criteria. The instructor first needs to select the required course and section number from a system generated course list, which contains all courses in the current semester, then current settings are displayed in a table. If the selected course is not being taught by this instructor, only a "Back" button is displayed, which can be used to go back to the course list page. Otherwise, an additional "Modify" button will be displayed. In the modification page, the instructor inputs new values for each item and then either saves or cancels the changes. The system automatically checks the inputs and prints error messages for invalid settings. Examples of displaying and editing a syllabus are shown in Figures 4 and 5.

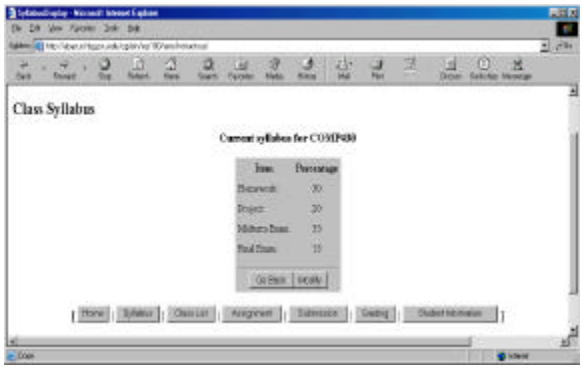

Figure 4. Example of a syllabus displaying page

Figure 5. Example of an editing syllabus page

2.1.1.4 Class List

Before a semester begins, an instructor needs to create a class list for each course he/she is going to teach. The *Class List* feature is used to perform this task. First, the instructor selects the required course from the given course list, which contains all courses the instructor is going to teach. Then the current class list is displayed with each student's student ID, name, user ID, Penn State email address, and semester status. Student Names are shown as HTML links which link to the corresponding *Penn State Directories*. An example is shown in Figure 6.

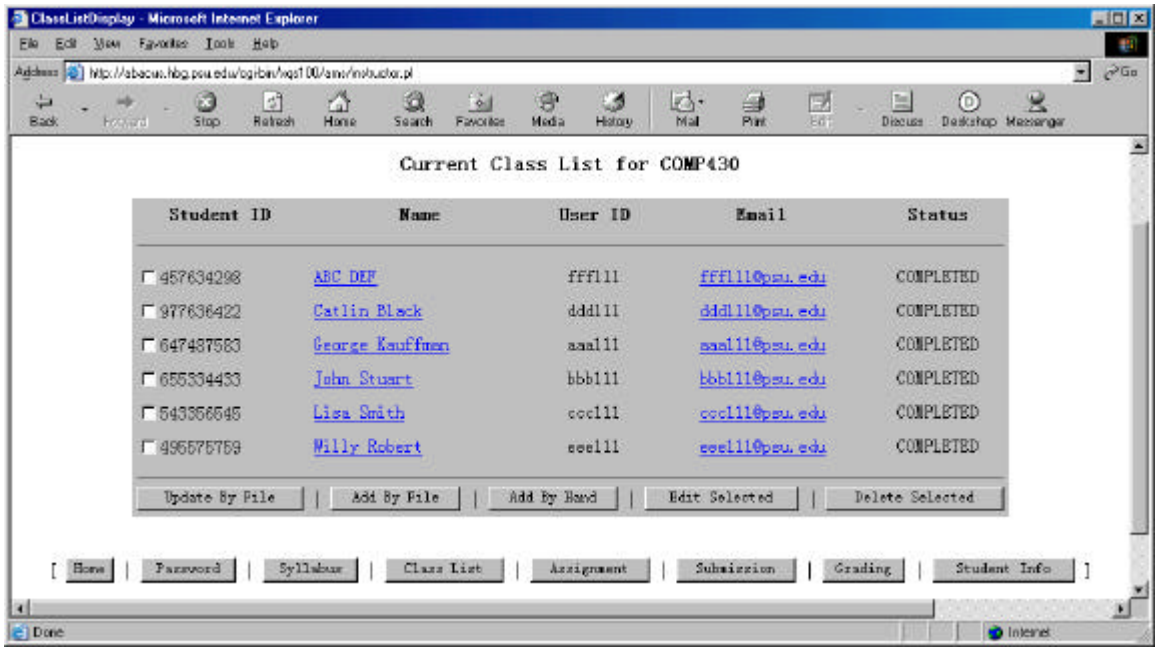

Figure 6. Example of a class list page

In this web page, an instructor can update/add class list by file, add by hand, and edit/delete current records. Since information input is time consuming and boring, we decided to make use of the available information resources to avoid such repetitive work. *The Penn State University's Elion System* provides instructors an electronic version of their class lists, which are sent to the instructors' Penn State access account email addresses. There are two different data forms available: *Spreadsheet* and *Email Distribution*. The *Spreadsheet* format contains more detailed student information and it is the one used in this AMS system. AMS automatically checks the format of the selected class list file, and if any mismatch is found, an error message is displayed and the modification process is terminated. For a valid class list file, this system also checks the information in each record. Only valid records are modified according to the file input. All invalid records are listed in an error message to help the user check errors later. This is very useful for batch processing cases.

Sometimes, the instructor may need to add a single or just a few student records. In this case, the *Add By Hand* function can be used (Figure 7). The instructor simply fills out the HTML form and chooses either add or cancel. Then the work is done. For both batch

inputs and single input, if a student account is being added for the first time, the system generates a random password as the default password of this account and saves it with other information. After successful insertion, an email will be sent automatically to the student's Penn State email address to notify the student the password of his/her AMS account.

In cases where an instructor needs to change existing student information, the *Edit Selected* function can be used to conveniently modify multiple records at the same time (Figure 8). Before any changes (insertions and modifications) are made to the database, input checking is executed to avoid any human errors, such as invalid values, duplicated user IDs, mismatched student IDs, or duplicated student IDs. The delete function is also provided for the instructor to delete student records.

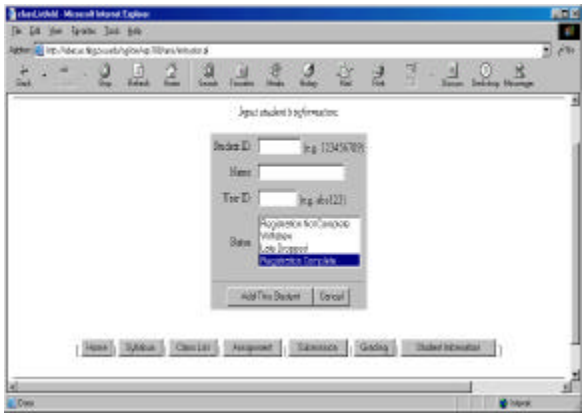

Figure 7. Example of adding class list

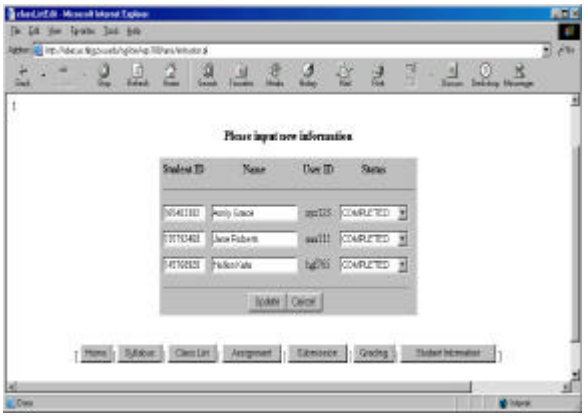

Figure 8. Example of editing class list

2.1.1.5 Assignment

The *Assignment* function takes care of assignment viewing, posting, and modifying. Once an instructor selects a course from all his/her teaching courses, all posted assignments will be displayed in the order of due time. This information includes the assignment name, the post date, the due time, the uploaded assignment files, the uploaded answer files, and the total points for this assignment. All files are shown as HTML links, so that the instructor can easily open or save files by clicking the corresponding links. Figure 9 shows an example of the assignment page.

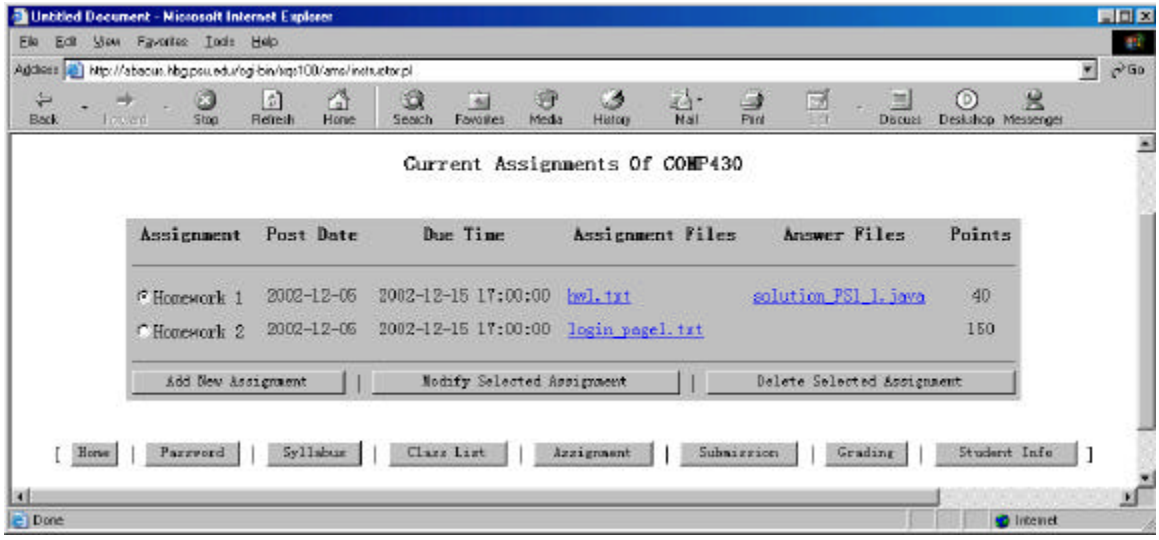

Figure 9. Example of an assignment page

To add a new assignment, an instructor needs to input all the required information. For an electronic assignment, dialogue windows are displayed repeatedly until all assignment files are uploaded. If there is no file for uploading, a new assignment record is added to the database. This covers the case of handout assignments. Error checking of date, time and file names is executed before real database modification and uploading.

In front of each item of the current assignment list, there is a radio button, which can be used to modify or delete a selected assignment. The *Modify Selected Assignment* function provides the instructor with a clear and easy-to-use way to change assignment information, which is shown in Figure 10. First, current settings are displayed. Then the instructor types in a new points value, a new due date, and a new due time if necessary. Third, for each assignment and answer file (if it exists), the instructor can simply select expected operations (keep, delete, or replace), and select a new file if replace is selected. Then, the instructor can select a new assignment and a new answer file in the following file fields, which provide users a convenient way to select needed files from local storage. Last, the instructor can decide either to save or to cancel the changes.

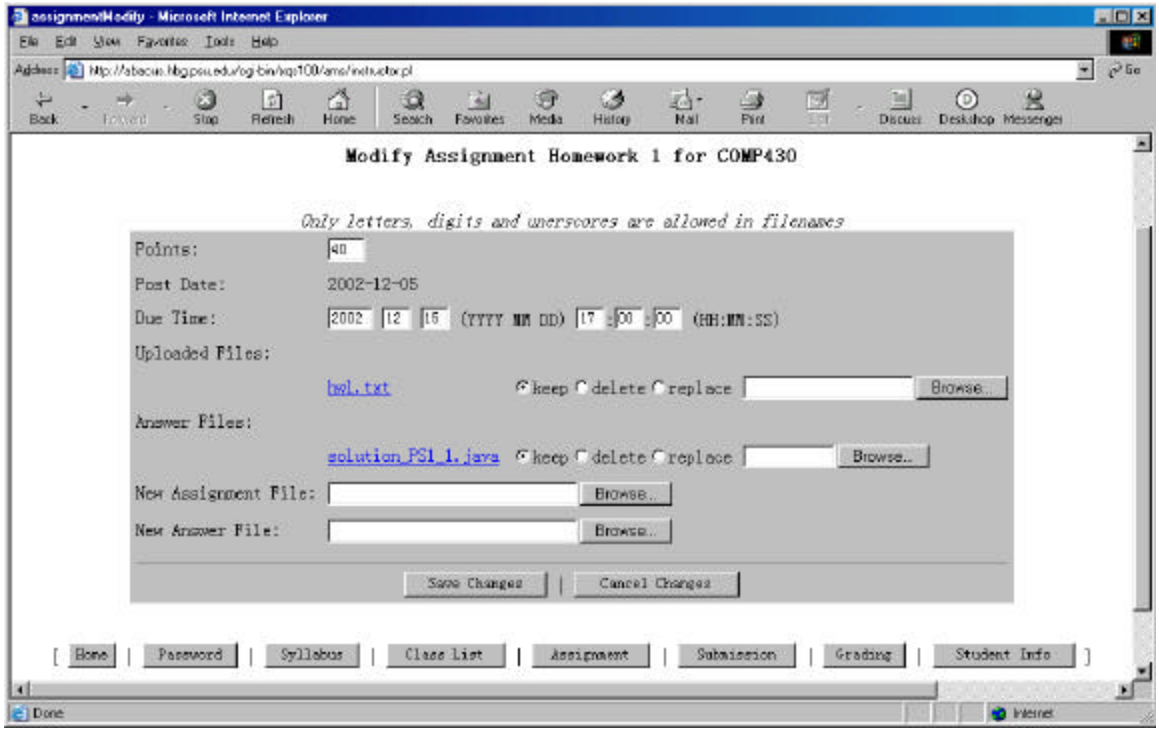

Figure 10. Example of an assignment modification page

To delete an assignment, this system will output a warning message to let the instructor be sure of his/her operation.

2.1.1.6 Submission

The *Submission* function is useful when an instructor needs to monitor or modify the submission status. After an instructor selects a course name from all courses he/she is teaching, an assignment list is displayed. The instructor needs to select an assignment name, after which its current submission status is displayed. In this page, all students names, student IDs, user IDs, submission time, statuses that indicate if the submissions are on time or late according to the due times, and all uploaded files are displayed. Files are also shown in HTML links to let the instructor easily save or open them. An example of the submission page is shown in Figure 11. When students submit their assignments online, the submission time and assignment files are recorded by the AMS system.

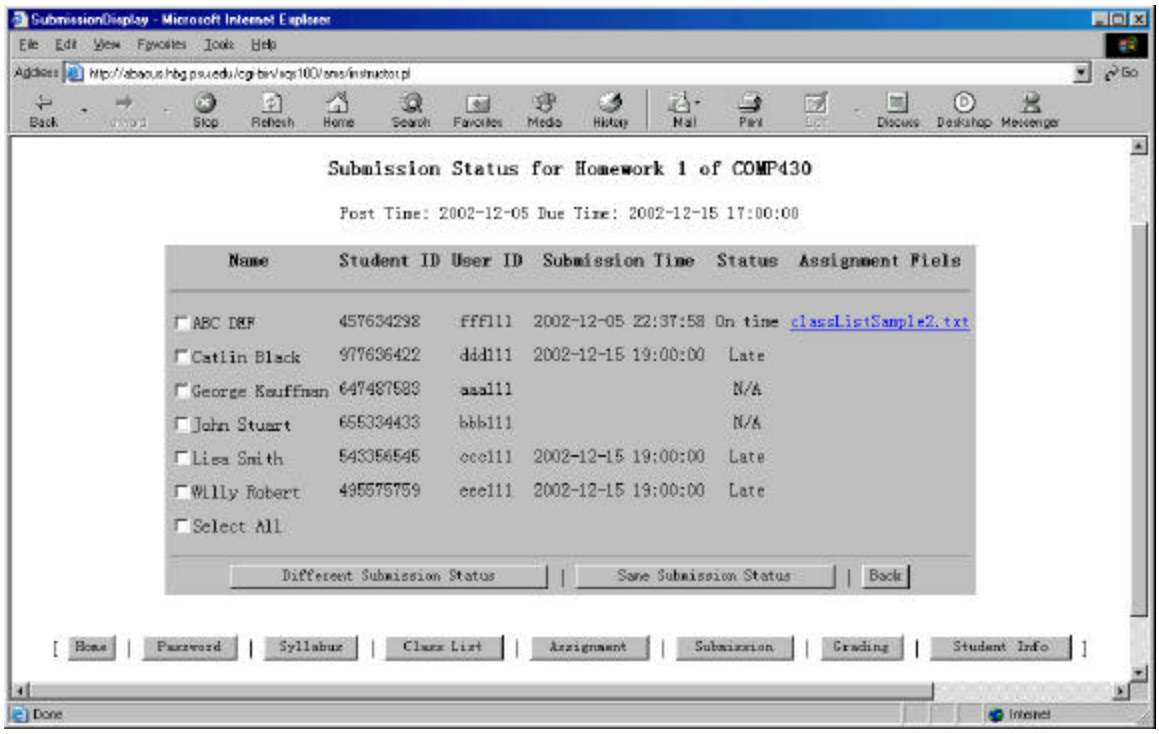

Figure 11. Example of a submission information page

Sometimes assignments are submitted in hard copies or in other formats. In this way, the AMS system does not have any knowledge of the submission, so the instructor needs to input the submission time for the students manually. The AMS system provides functions *Different Submission Status* and *Same Submission Status* for such needs. In *different submission status* page (Figure 12), the instructor modifies the submission time for each selected student, while in *same submission status* page (Figure 13), the instructor assigns a submission time to all the selected students. As shown in Figure 10, a checkbox "*Select All" i*s provided to let the instructor conveniently select all the students in the class.

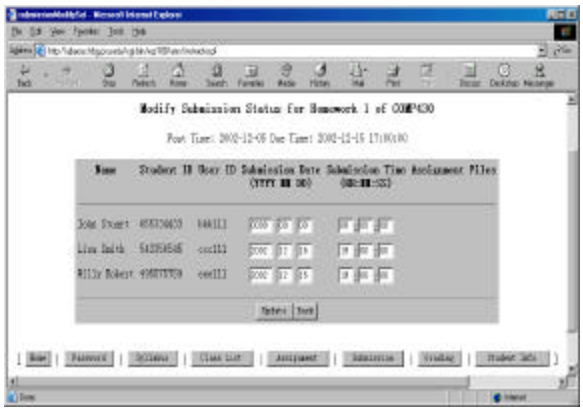

Figure 12. Example of a different submission status page

Figure 13. Example of a same submission status page

2.1.1.7 Grading

Another important function the AMS system provides instructors is related to grades. An instructor can grade assignments online, check class academic status, and even view statistic reports. To do this, the instructor needs to select the course again. This time, not only are all of his/her teaching courses listed, but one more item is added to the list: *Total Grades* (see Figure 14). By selecting any item except *Total Grades*, the instructor can check class grades of this assignment and change grades for all students. First, the due time and the total points for this assignment are displayed. All students' current grades are listed with their personal information, submission status, and assignment file links. Then the instructor can press *Change Grades* button to enter the grading page (see Figure 15). Grading work is very easy - simply input new grades for students and choose to save updates then it is done. The system automatically checks all grades, which have to be numeric type and cannot be greater than the total points of this assignment. A new grade list will be displayed to indicate the changes.

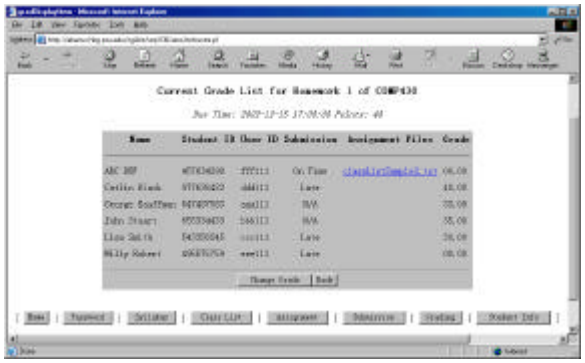

Figure 14. Example of a grade list

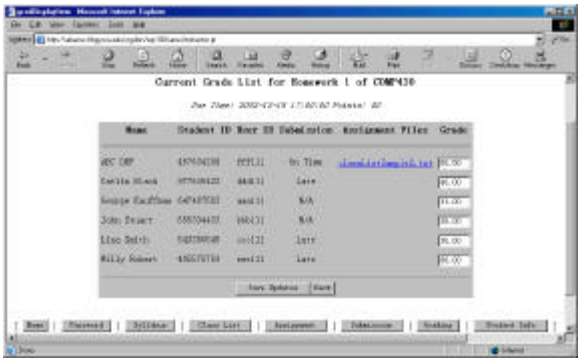

Figure 15. Example of a grading page

The *Total Grades* function is different from the others, and can be used to view class grades for not just one assignment but also all the assignments having been assigned. Displayed information includes student personal information, grades for each assignment, and final grades that are calculated according to the grading criteria. If no syllabus is set up for this course, the final grade for each student will be the sum of all his/her assignment grades. All student records are listed in the order of final grades, from the highest to the lowest. Under the list is a statistic report about the class grades. Minimum grade, average grade, standard deviation, and maximum grade for each item are displayed. Figure 16 is an example of class grades report page.

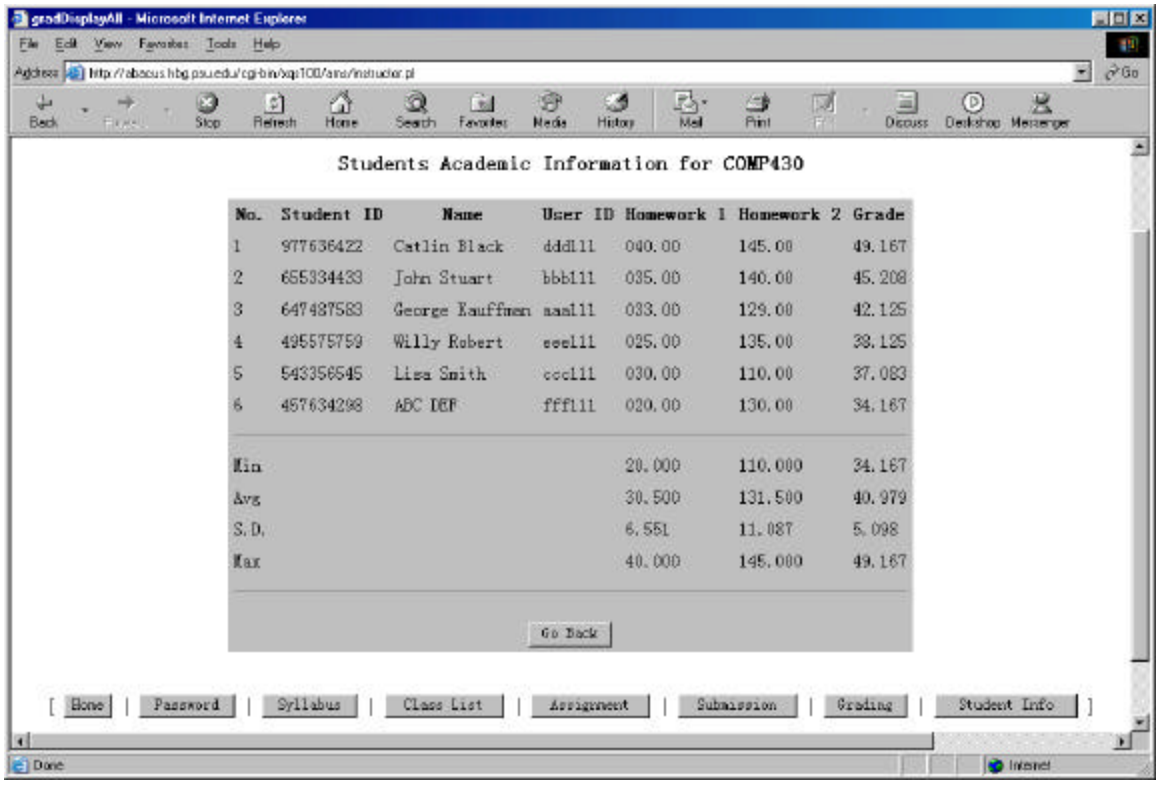

Figure 16. Example of a class grades report page

2.1.1.8 Student Information

In the student information page, an instructor can search student information in a very convenient way (see Figure 17). First, all students who are taking courses in the current semester are listed with their personal information: student ID, name, user ID, current courses, and semester status. Above the list is the searching criteria composer. The instructor can use it to generate a search index by using any part of the information or a combination of several parts. For example, the instructor can search a student record with a specific student ID, name, or user ID. Alternatively, the instructor can search student records by specifying the course or semester status. Pattern matches are supported in searching index. For example, "%" can be used to match any number of characters and "_" can used to match exactly one character. The system automatically adds "%" characters to both sides of each index item, so that the instructor can search even by partial item information. The instructor can also select the order by which all records are

displayed. The possible orders include student ID, student name, student user ID, and student semester status.

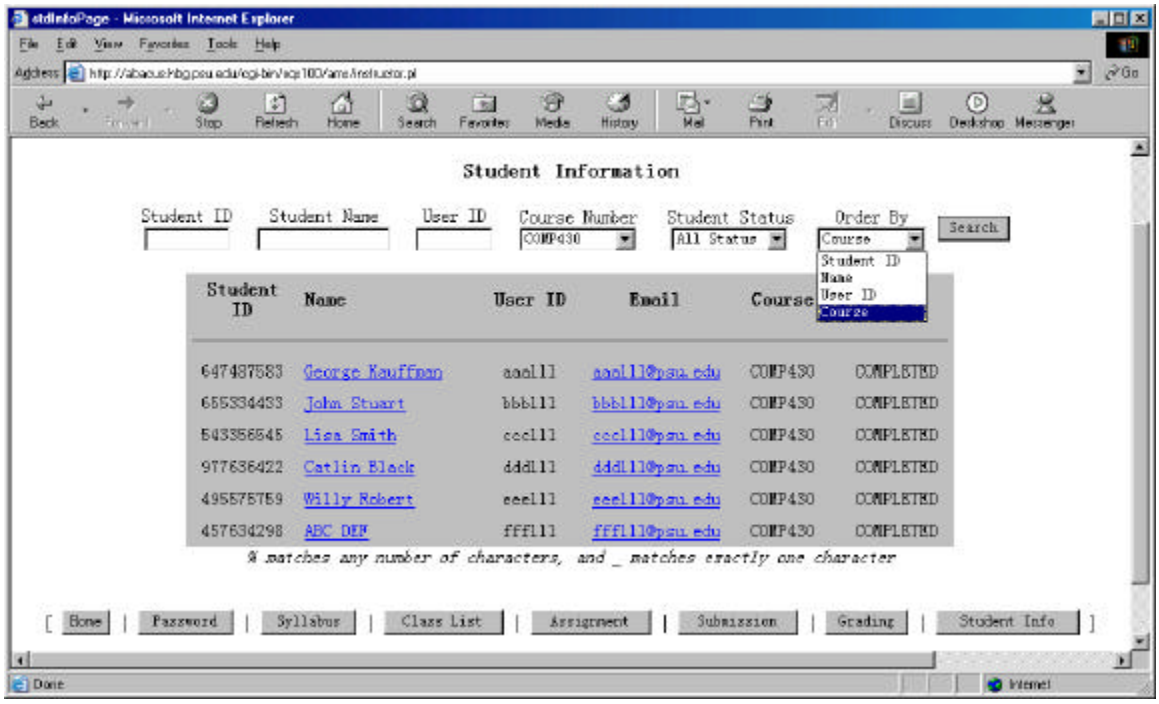

Figure 17. Example of a student information page

2.1.2 Administrator

Before each semester, the administrator needs to clean up the database, i.e., back up all assignment information of the previous semester to corresponding backup tables, and clean current tables. The administrator also needs to update course schedules and instructor information for the coming semester. The administrator home page is shown in Figure 18.

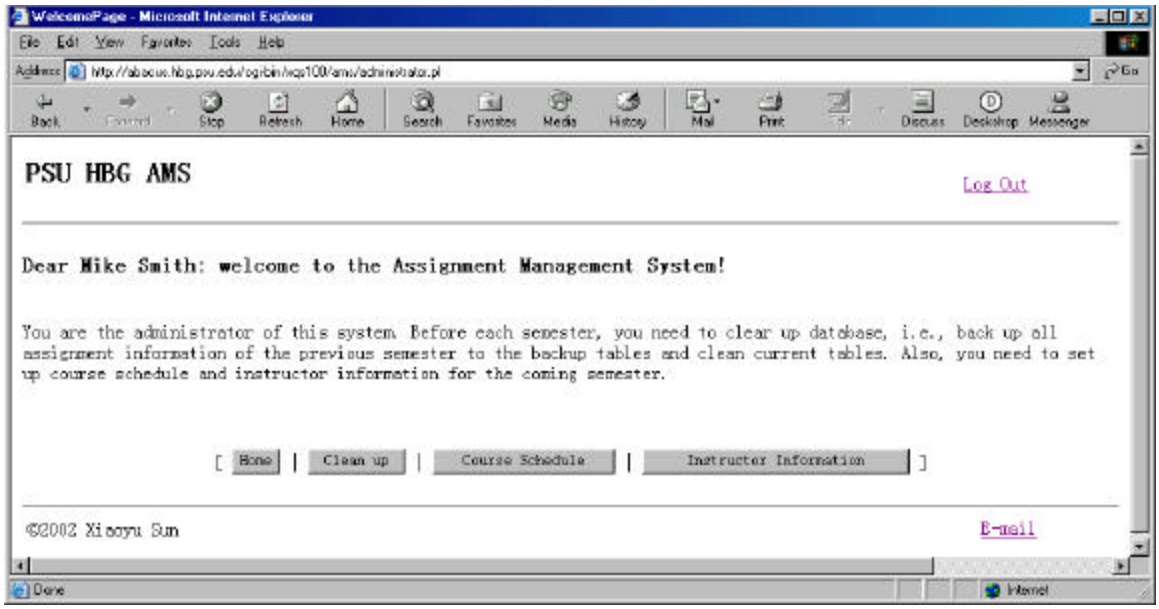

Figure 18. Example of an administrator home page

To clean up a database, the administrator needs to input the values of the year and semester for which the data will be saved. After that, system checks the inputs and prompts out a confirmation page if the inputs are valid, otherwise, an error message will be displayed. Then the system checks if the data for the selected semester has already existed in the backup tables. The system will execute a cleanup process directly if no existence is found. Otherwise, a notice about this and three alternatives will be provided to the user: *Back up all tables*, *Back up undone tables only*, and *Quit*. According to the user's choice, the corresponding operation will be executed. The cleanup process includes two parts: backup and cleanup. First, all data of the ended semester are saved to the backup tables and then deleted from the current tables. Then, all data in the current tables are deleted and directories that contain all uploaded assignment files are moved to a new directory that is named after the values of the year and semester.

Another task of administrator is to set up course schedules for the coming semester. First, the AMS system displays a semester list of all existed data in the backup tables and a *Display Schedule* submit button below it. If the current semester's data are found in the current tables, a *Current Semester* item will be added to the list, otherwise, an *Add New Semester Schedule* button will be added to submit button list.

If one of the previous semesters is selected then backup information of it will be displayed, as shown in Figure 19. This page can also be used for searching course information of all backup data. Administrator can search courses by year, semester, course name, course number, and instructor information. The administrator can select the preferred display order. Pattern matches and partial information are also supported in searching index.

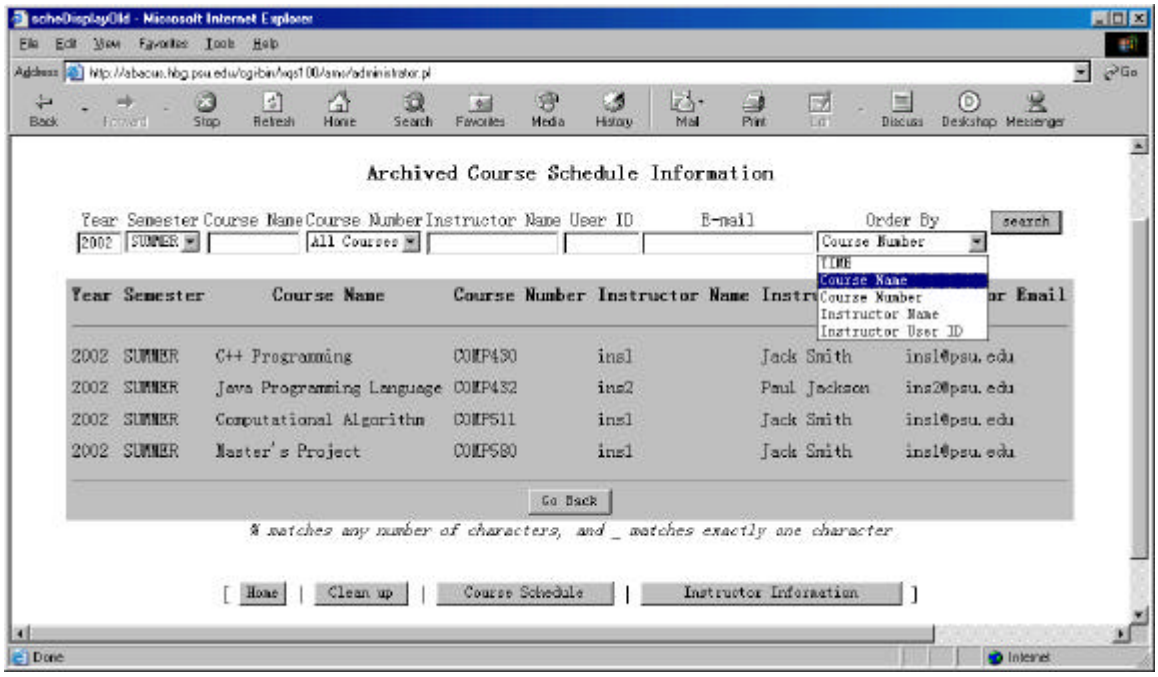

Figure 19. Example of an archive course schedule page

If the administrator selects the current semester from the semester list, the current semester information page will be displayed. The administrator can add new, edit, delete courses, and search course schedules by the course and instructor information. Adding and editing course information can be done easily by filling out a course information table. Two examples are shown in Figures 20 and 21, respectively. If no current semester data are found when entering schedule page, the administrator can use *Add New Semester Schedule* button to input whole semester schedules in batches.

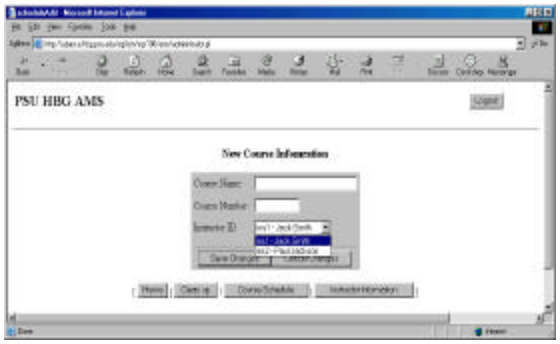

Figure 20. Example of adding a new course schedule

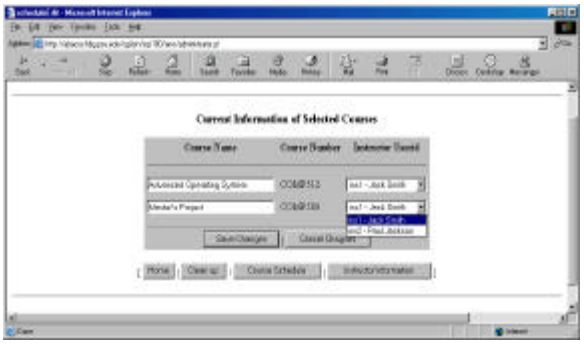

Figure 21. Example of editing course schedules

The administrator is also responsible for maintaining the information of instructors. AMS provides the administrator the functions for adding, searching, editing, and deleting instructor information. These operations are very similar to the maintenance of course schedules. The display order of searching results can be either instructor name or instructor user ID. Figure 22 shows an example of the instructor information page, which is the common interface of all other operations.

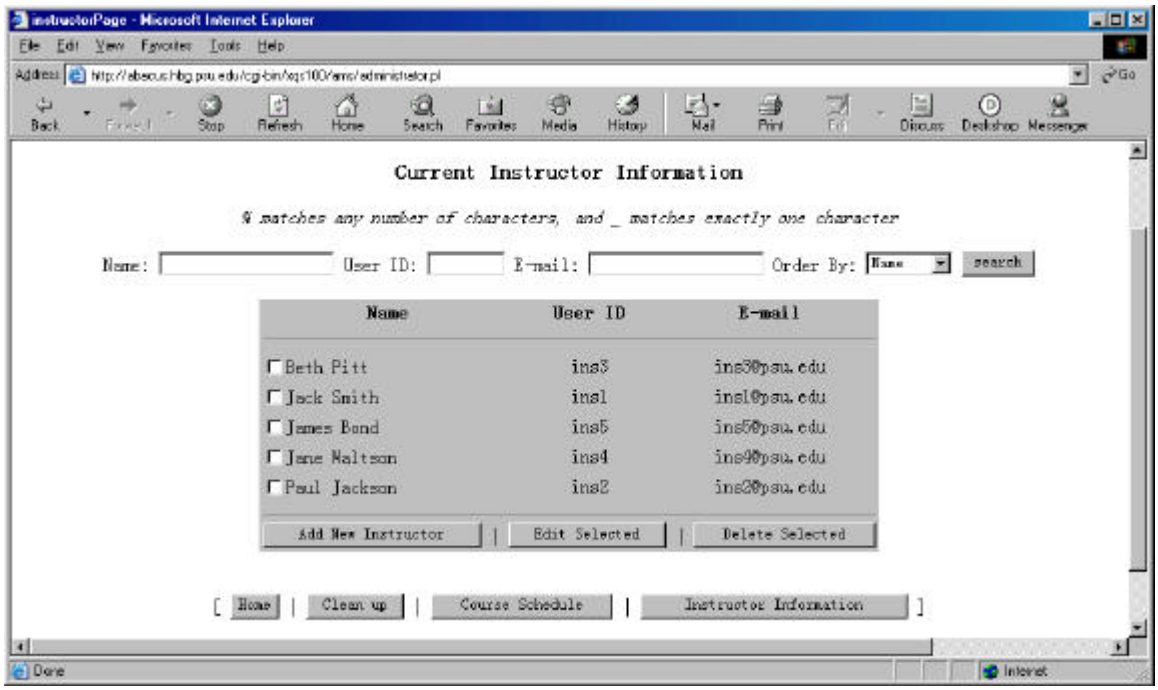

Figure 22. Example of an instructor information page

2.1.3 Student

The main features for student users are: checking information of classes, assignments, and grades, and submitting assignments online. Interfaces and operations are very similar to the instructor part. Figure 23 is an example of a student home page.

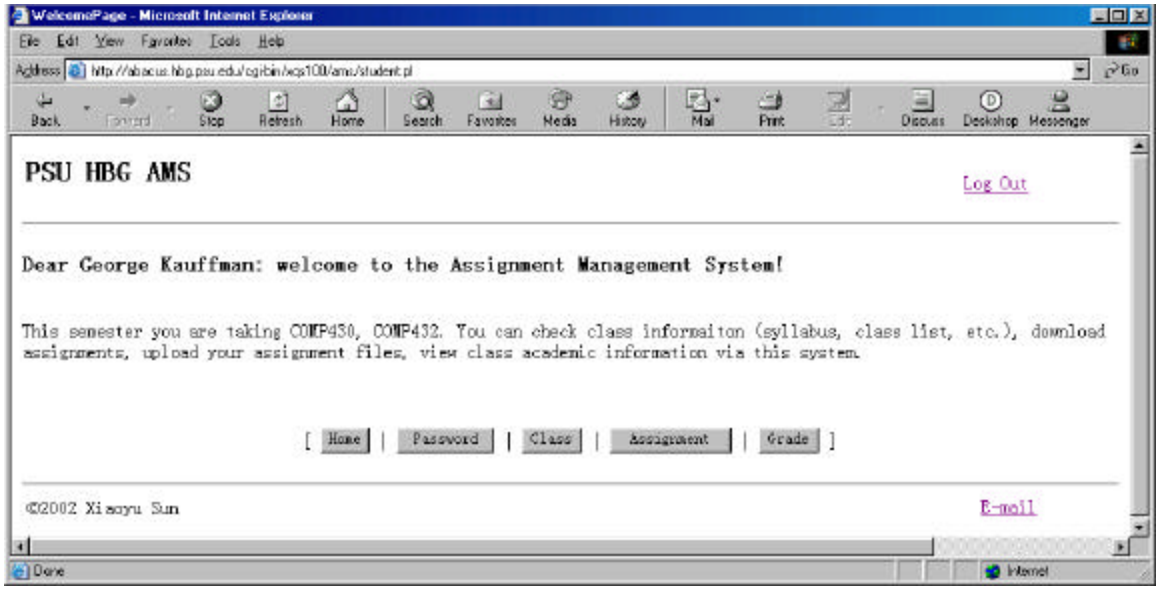

Figure 23. Example of a student home page

Students can check class information for any courses in the current semester. To enter the class information page, a student needs to select one course from a list that contains all the current courses. Then the instructor information, class syllabus, and class list of the selected course are displayed. The student names are shown as HTML links, which provide the student with an easy access to *the Penn State Directory*. An example is shown in Figure 24.

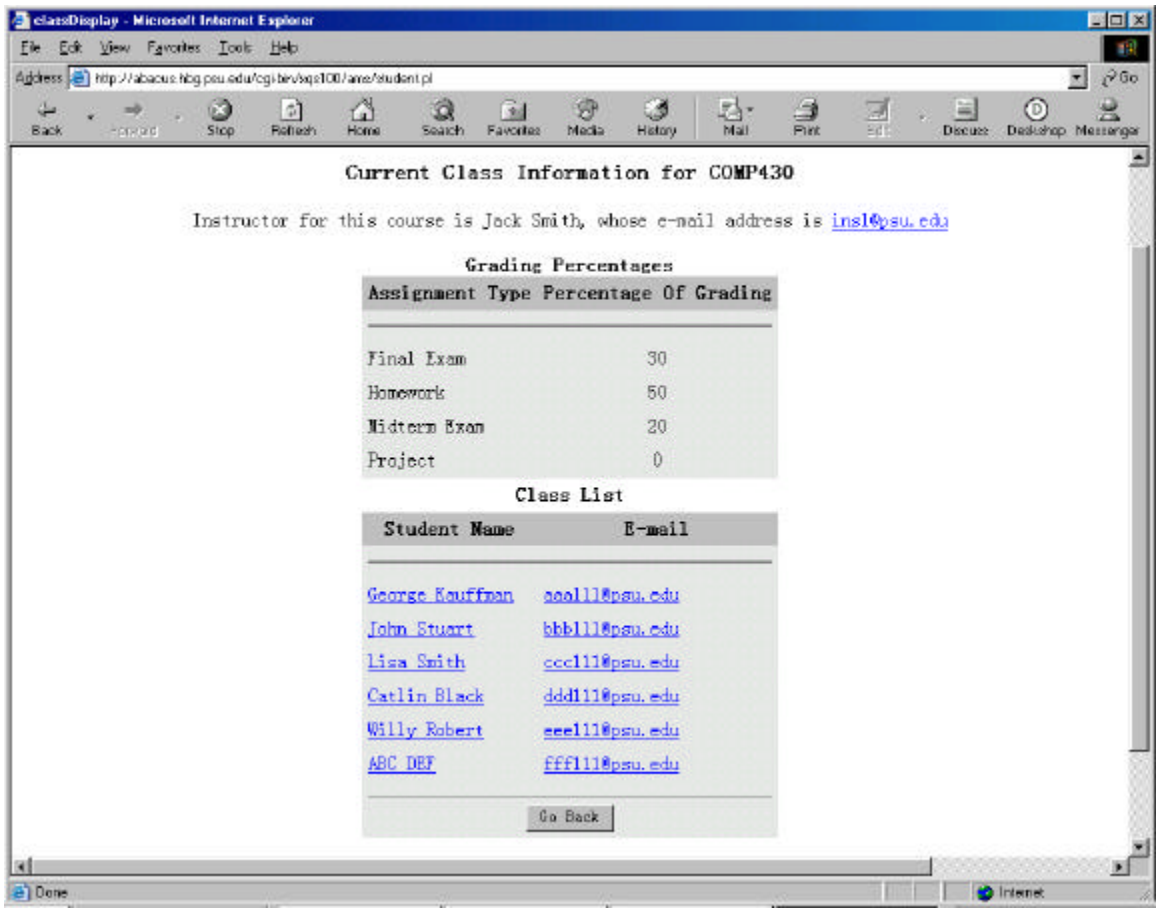

Figure 24. Example of a class information page

A student can check assignment information and upload assignment files for the courses he/she is taking. After selecting a course, the student enters the assignment information page. On this page, all the current assignment information for the selected course is displayed, which includes assignment name, total points, post date, due time, assignment files (if exist) and answer files (if exist). The student can press the *Submission Information* button to see his/her own current submission status for the selected assignment, and then upload assignment files. Uploading page will repeatedly display until the student indicates that all files have been uploaded. Functions for deleting or replacing uploaded files are provided to let students conveniently manage submissions. Each time a change is made, the AMS system automatically changes student's submission status and shows the student the new submission time, status (on time, late or N/A), and uploaded files. Figures 25 and 26 are the examples of such operations.

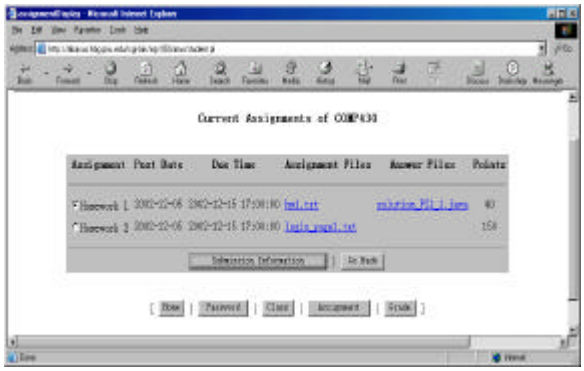

Figure 25. Example of a student assignment page

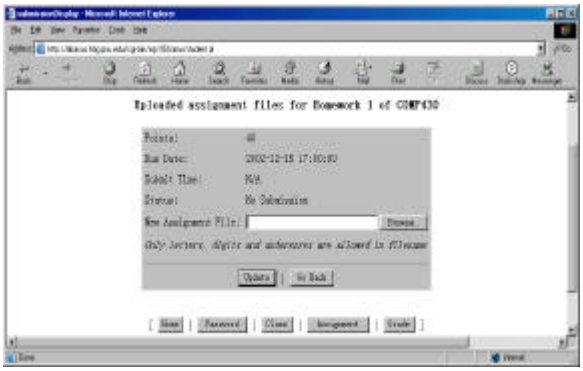

Figure 26. Example of a student submission page

Checking grade information is another important feature for student users. On the grade information page, a class grade list and a statistic report are displayed (see Figure 27). It looks very similar to the instructor's grade information page, but different concerns are involved. To protect other students' privacy, personal information such as name, student ID, and user ID are omitted from this page. The grades are grouped in assignments and are displayed in descending order, unlike the instructor part, where all the grades are grouped in users. Therefore, other students' information will not be disclosed.

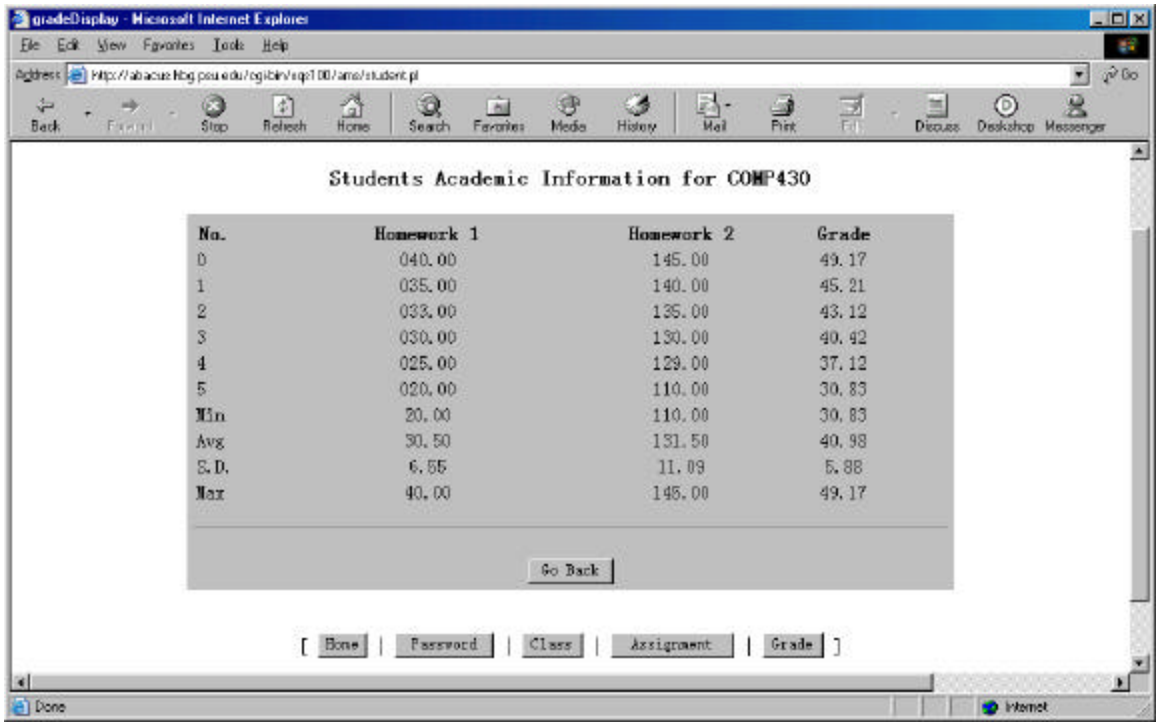

Figure 27. Example of a student grades information page

2.2 Design – From a Programmer's Perspective

2.2.1 Database Design

Providing user good interaction with system was a very important task during system design. One crucial thing of this is to let system be aware of as much as possible information, such as: instructor and student information, class information, and student academic information. Therefore, an information system is required for maintaining these data.

CGI script is one of several technologies used to generate dynamic web pages. Here is how it works. First, a server listens to port 80 on the host machine. When it gets the connection request from the client side, the server establishes an HTTP connection between itself and the browser. Second, when the server gets an HTTP request from a browser, it takes all the input and passes it as input to the CGI script. Once the CGI script terminates, the server returns its script output to the browser and closes the connection.

By studying the working flow carefully, we can see that each time an HTTP request is made to the CGI script, a new process for the program is invoked and then terminated. There is no way to store any information in variables within the program to be accessible next time. That's why we say that HTTP is a stateless protocol, i.e., "the server doesn't maintain or store any request-related information from one transaction to the next" [5]. Nevertheless, CGI scripts need to preserve their states for information sharing. How to make this happen? The solution is to use some external storage. Usually, there are three categories of such storage: browser/client side, flat files, and a database.

In the browser/client side strategy, three approaches can be used: query strings (with or without extra path information), hidden fields, and client-side cookies. Simplicity of implementation and no concurrency problems are the most benefits of these methods, but they are more appropriate for storing small amounts of information. Flat files can be used to store a relatively large amount of information and are fairly simple to implement, but concurrency control is difficult when multiple users are trying to write to the same file at the same time. Locking can be used to solve this problem, but it may lead to poor concurrency. Deadlock is another potential problem of using locks. It happens if a file is not successfully unlocked and other processes are all waiting for this lock. Increasing the number of files can reduce locking conflicts, but the resulting information inconsistency between different files may cause other troubles.

Compared to the other two strategies, using a database is more powerful in data management. Any concurrency issues can be handled by the DBMS. Databases appeared prior to the emergence of World Wide Web. It is not an exaggeration to say that databases have changed our world and made our lives much easier. Take library management system as example. With database management applications, one does not have to walk through all racks in the library to find the books he/she wants. Instead, he/she can use the computers at the library, input some simple information about the books, and then search all the books by title, author, ISBN number, or even a keyword. Until now, most information in the world has been stored in databases, so connecting web

pages to databases is a natural and necessary step for sharing information with the world. A database back-end can support many critical functions on the web. Let's think about the library system again. By integrating a database with web pages, users can search book information from any online libraries even at home, and librarians can update library information from any computer connected to the Internet. Therefore, a good assignment management system should be able to provide users with powerful, convenient services via good interaction. This is why we have chosen to use a DBMS in the AMS system.

Currently, there are many DBMS products in the market, including Oracle, DB2, Sybase, etc. They are all fully featured and highly optimized because they are designed to handle heavy tasks with very complex data relationships. That is why these systems are inevitably expensive. MySQL is suitable for small-scale database applications. Although compared to those large systems MySQL may lack some features, it is free, lightweight, fast and robust [6]. MySQL perfectly matches the needs of this project. We are using MySQL Version 3.22.32.

The interface between MySQL and Perl is DBI, which stands for Database Independent Interface. Before DBI, in order to connect any specific DBMS, Perl needs to use the corresponding Application Programming Interface (API), such as Mysql.pm for MySQL. This will cause problems when it is necessary to upgrade current system to a new DBMS or to connect different databases within one Perl script. DBI has been created to solve this problem. The name of Database Independent comes from two-layer interface structure. The bottom is the database dependent layer, which contains modules for each type of database accessible from Perl. Database independent layer lies on the top of these modules and is the real interface to access databases. By using such a common API, programmers only need to care about one interface, and new databases can be easily imported by adding its DBI module to the dependent layer, thus portability is enhanced. Furthermore, the DBI interface is robust and has become a standard in Perl programming [7].

The database created for this system is *ams*, which can be divided into two parts: current data and archived data. Both parts contain six tables with the same structure. Tables *ADMINISTRATOR*, *STUDENT*, *INSTRUCTOR*, *SEMESTER*, *COURSE*, *ASSIGNMENT* and *ACADEMY* contain all data for current semester, and tables *STUDENT_BACKUP*, *ADMINISTRATOR_BACKUP*, *INSTRUCTOR_BACKUP*, *SEMESTER_BACKUP*, *COURSE_BACKUP*, *ASSIGNMENT_BACKUP* and *ACADEMY_BACKUP* contain all backup data.

The *SEMESTER* table is used to store semester courses schedules. Instructors can get their course lists from this table. The *ADMINISTRATOR* table and the *INSTRUCTOR* table are used to store administrator and instructor users' personal information and AMS account information, respectively. They are both maintained by *ams_admin*. *STUDENT* table is populated automatically when instructors create class lists. It is used to maintain students' personal information and their AMS account information. When instructors set up syllabi for their teaching courses, the settings are saved to the *COURSE* table and can be retrieved later for computing student final grades. All assignment information is maintained in the *ASSIGNMENT* table, which is also used for creating an assignment list. *ACADEMY* contains student academic information. It is used for generating class academic statistics. The following figure shows the structure of the *ams* database.

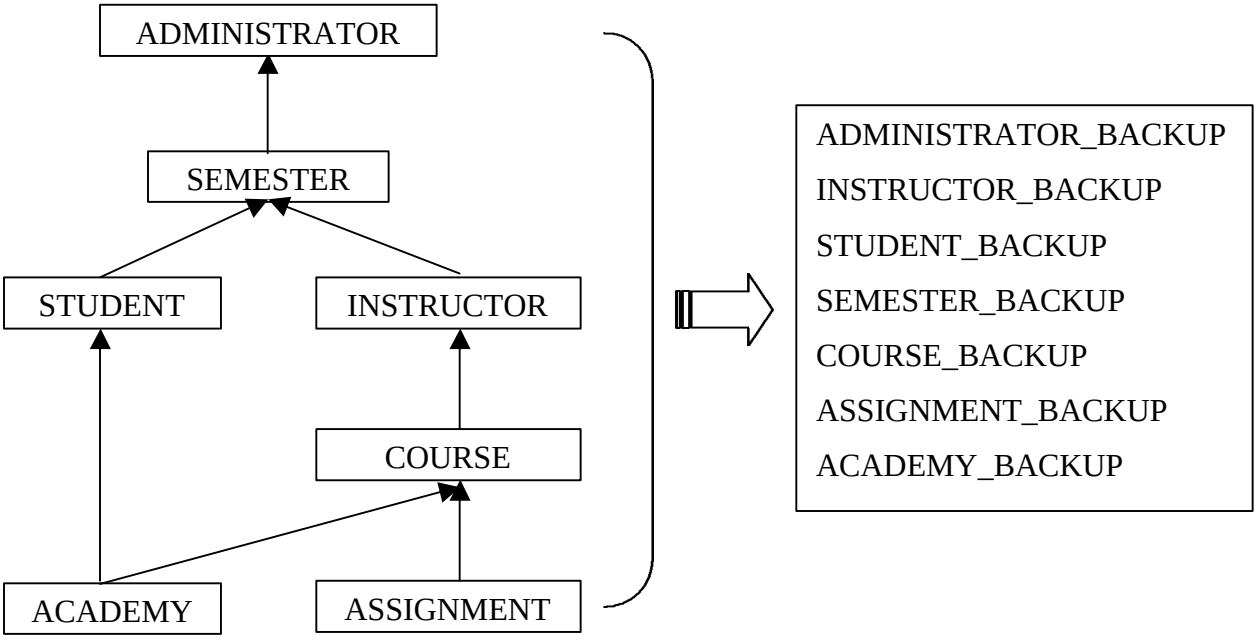

Figure 28. Database Structure

Primary keys are used in each table to ensure uniqueness and query efficiency. The following is an example of creating the *STUDENT* table. Detailed descriptions of all tables are attached as an appendix.

```
use ams;
create table STUDENT(
    stdID   CHAR(12)            NOT NULL,
    name    VARCHAR(40)         NOT NULL,
    userid  CHAR(8)             NOT NULL,
    pwd     CHAR(20)            NOT NULL,
    course  CHAR(10)            NOT NULL,
    status  CHAR(10)            NOT NULL,
    PRIMARY KEY(userid, course),
    UNIQUE (stdID, userid)
);
```

2.2.2 Software Implementation

There are several alternatives to CGI for handling dynamic requests: Perl, ASP, PHP, Java servlets, C, Python, etc. Because of the ease of use, powerful text handling abilities and being easily portable on many platforms, Perl has become one of the most widely used development platform for the Web programming. Countless open source modules contribute to its popularity. *CGI.pm* module is the standard tool used in Perl because it "provides a simple interface for the most of the common CGI tasks" [8]. Therefore, we chose Perl as the programming language and utilized CGI module to create CGI scripts. The main features of Perl used in the AMS system are: HTML processing, file and directory manipulation, database operation, sending emails, and string manipulation.

System applications are composed of five programs: *AMS.conf*, *AMS_common.pl*, *instructor.pl*, *administrator.pl*, and *student.pl*. *AMS.conf* is the system configuration file in which all common variables are defined, such as the upload directory, the maximum size of uploaded file and the maximum capacity of each assignment directory. *AMS_common.pl* takes care of all common tasks, such as printing a header and footer for each generated web page, uploading a file to the server, and various database operations. Specific tasks for different users are handled by *instructor.pl*, *administrator.pl*, and *student.pl*, respectively.

HTML processing includes form processing and dynamic HTML generation. User input information is usually stored in HTML forms and is sent to the server with an HTTP request. *GET* and *POST* are the two methods used in form transmission. Compared to *GET*, in which data is sent as part of the URL (Uniform Resource Locator) so form values are visible to users, *POST* is more secure because form data is placed in the environment variable *PATH_INFO* and can only be decoded by CGI programs. In this implementation, the decoding work becomes very easy by calling the *param()* function in the *CGI.pm* module. For example, retrieving the value of item "*userid*" can be done by using the command: param('userid') || ''. After processing user requests, CGI scripts generate some dynamic web pages to send to the browser. The *CGI.pm* module also provides many convenient functions for simplifying writing HTML code.

In the AMS system, both instructors and students need to upload assignment files to the server. It is impractical to put all these files in the database, not only because MySQL does not support such types of fields, but also because it may cause the database size to increase unexpectedly. We created a directory on the web server to store the files and the location is specified by variable $*UPLOAD_DIR*. Under this directory, subdirectories for all courses are created automatically when the system administrator sets up the semester course schedules. Each subdirectory is named after the course code. For example, *COMP432* is the name of subdirectory of course *COMP432*. All assignment files related to one course are saved under its subdirectory. When an instructor posts a new assignment, the system automatically creates an assignment directory under the course subdirectory, which will only be used to store files for this assignment. For example, when Homework 1 of the course *COMP432* is posted, directory *hw1* will be created under subdirectory *COMP432*.

All uploaded files are named in the format of user ID plus an underscore plus the original filename. While for assignment answer files, string “answer” is used instead of the user IDs. In this way, a CGI program can easily tell uploaded files apart and retrieve their original filenames for displaying. Since all files are shown as HTML links, the URLs of these files must be specified. They can be easily generated by the concatenation of the following information: the URL of *$UPLOAD_DIR*, course directory name, assignment directory name and the filename generated as previously.

Another important issue about uploading assignment files is the process of uploading. First, users select the required filename from their local computers. Then the AMS system reads the selected file remotely and creates a copy of it on the server. *FILEHANDLE* and *FILEDESCRIPTOR* are two different terms used in file accessing. They correspond to the Perl functions *open()* and *read()*. In this implementation, the *open()* function is used to retrieve the original filename (not the absolute path name), which creates a new file handle for the new opened file. This file is created under its corresponding directory defined as above and is open for writing. In Perl, the *open()* function can only read the content of local files, so the file handle is passed to the *read()*

function, which reads a chunk of data and writes it into a buffer. Then we write the contents of the buffer to the opened file handle each time a read is executed. Finally, the system closes the file handle after execution.

Sending emails is used in this system to inform users about their account information: user ID and password. Any Internet email message contains a header and a body separated by a blank line. The header contains the recipient's email address, the subject of the email, and the sender's email address; and the body contains the message. The AMS system sends emails using the Linux *sendmail* program.

Perl contains powerful string manipulation operators. They are very helpful in reading data files and constructing SQL queries. For example, retrieving data from each line of the class file can be easily implemented by using the command @fields = split(/,/, $line), which saves all values into an array with each element representing one value. In the implementation, some SQL query strings are constructed dynamically according to real-life conditions. This can be done by using the command $condition = join(" AND ", @searchIndex). In Perl, another scheme is playing an even more important role in string manipulation. That is *regular expression*. A *regular expression* is a pattern that is matched against strings [9]. By using it, a very simple command such as $file =~ /([^\\\/]+)$/ can be used to extract the filename from an absolute pathname.

2.2.3 System Security

Before the appearance of CGI technology, the World Wide Web had been a relatively secure way for cross-site resource sharing, because only static web pages could be sent in reply to a browser request. As soon as a CGI script is put online, anyone can run the application from almost anywhere in the world. This is amazing but also scary, because it means that users have gained higher authorities on the web server. Along with the development of CGI technology, network security has gained more and more attention of web programmers, since hackers have caused much damage to the web servers through

the Internet. Cross-site scripting is an attack that tricks a user into submitting scripting code to a dynamic form on the target web server, which can later cause all kinds of damage if the danger is not detected by the web server. As Paul Lindner said in his online article *Preventing Cross-site Scripting Attacks*, "The cross-site scripting attack is one of the most common security problems facing web developers today" [10]. There are also many other forms of attacks that work directly on the operating system or applications on the web server. The compromise of even a user account may cause risks to the entire network system [11]. Therefore, much attention has been paid to the security problem in the AMS implementation. Next we will discuss the security issues related to the AMS.

The first step for system security is the password protected user accounts. Each user has his/her own unique AMS account ID and a randomly generated password. To access the system, a user has to provide valid identification. Otherwise, any access request will be rejected.

The security of the database is as important as the server itself since it maintains all the useful information of the AMS system. MySQL uses its own database server *mysql* to ensure security. The database *mysql* is created with the installation of MySQL. It contains five tables: *db*, *host*, *user*, *tables_priv*, and *columns_priv*. The *user* table specifies what each user is allowed to do. The *db* table controls the accesses to individual databases and tables [12]. During implementation, three different types of users were created for the system: *ams_admin*, *ams_instructor*, and *ams_student*. They were assigned different authorities of accessing the database *ams*, as shown below:

```
use mysql;

INSERT INTO user VALUES ('localhost','ams_admin', PASSWORD('admin'),
'Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y');
INSERT INTO user VALUES ('localhost','ams_student', PASSWORD('student'),
'N','N','N','N','N','N','N','N','N','N','N','N','N','N');
INSERT INTO user VALUES ('localhost','ams_instructor', PASSWORD('instructor'),
'N','N','N','N','N','N','N','N','N','N','N','N','N','N');

INSERT INTO db VALUES ('localhost', 'ams','ams_admin',      'Y','Y','Y','Y','Y','Y','Y','Y','Y','Y');
INSERT INTO db VALUES ('localhost', 'ams','ams_student',    'Y','Y','Y','Y','N','N','N','N','N','N');
INSERT INTO db VALUES ('localhost', 'ams','ams_instructor', 'Y','Y','Y','Y','Y','N','N','N','N','N');

flush privileges;
```

As one can see from the *user* table, three types of users were created. They can access the MySQL server only via the web server *abacus*. User passwords are encrypted by the *PASSWORD()* function. This function is not reversible, which means no one can guess the real password by reversing the encrypted password into the original one. When users attempt to connect to the database with their real passwords, the server performs the same calculations and compares results with the values saved in the *user* table. If the comparison fails, the server rejects the connection request. Please notice that in the *user* table, we assigned instructor and student users no permissions. This scheme enables users to connect to the server but nothing else. All detailed permissions are defined in the *db* table. The rule of thumb for database security is: never give users more privileges than they have to have. We assigned these users access permissions only to database *ams*, and only select, insert, update, and delete operations can be done to its tables. The exception is to administrator users who have full control over the system.

Although MySQL provides database-level and table-level securities, deletion of users is still a problem for database security. Hackers may operate some malicious deletions and destroy everything on the server. The AMS implementation applied a fake deletion strategy to avoid this kind of attack. Fake deletion means that data are not really deleted from database although to users they appear to have been deleted. Take the assignment deletion as an example. In some cases, the instructor needs to delete an assignment, and everything related to this assignment, including assignment information and all related student academic records, will be deleted and will not be visible again to users after the deletion. It is a very dangerous operation because the potential exists to lose all information. In the AMS implementation, when a user sends such kind of requests to the server, the CGI script will warn user of the serious consequences. The real moves will be made to the database only after the user reconfirms his/her request. Furthermore, only assignment records can be deleted from the database. All student academic records will remain untouched. Later, when the user needs to check the class academic status, only the student records that related to the assignments currently existing in the *ASSIGNMENT* table will be presented. From the user's point of view, all deletions have been made. If an

instructor deleted an assignment mistakenly or if some hacker by whatever means entered the AMS system and maliciously deleted all assignments of a course, only the assignment information needs to be recovered. The instructor can use the *Add Assignment* function to re-input assignment information, thus all student information is visible again. The same strategy is also used in deletion of student information.

User input is always a potential problem of system security. Some users can always surprise people by their "creativity". Therefore, no assumptions should be made on the data from users. In other words, never trust the browser. User input can come from either URL addresses or HTML forms, so each time an HTTP request is sent to the server, CGI script needs to check both parts.

Let's talk about the URL check first. The content of URL is visible and can be modified by users. At the beginning of the system implementation, the current user's account ID was saved in the URLs of main function bar. In this way, the current user can easily access other users' information by changing the user ID from the browser address field. We have solved this problem by using hidden fields. Other users may try to break into the system by avoiding the login process. In this implementation, each action was assigned a unique name. When the CGI script is called, it retrieves the action name from the HTML form. If the action name is invalid or is empty (which is the case when a user calls it directly without going through the login page), the login page will be displayed and the user is forced to redo the log on transaction.

HTML form input can cause even more trouble if the input is not carefully checked. For example, to add a new student to the class list, an instructor needs to input the student information. Invalid or mistaken inputs can cause database errors, such as duplicate records, different records with same user ID, same user ID with different student IDs, or different user IDs with the same student ID. To avoid such errors, all user inputs are checked by system before any operation is done to the system or database. If the input is found to be invalid, the system terminates the current transaction and returns an error message to the user.

Allowing users to upload files to a web server may be a huge security hole because the upload directory has to be open to the users. To limit the access privileges to the smallest group of users, the upload directory was set to be accessible only by *nobody* users. The upload directory, including its descendant subdirectories, has to be fully opened to *nobody* users since they need to create subdirectories and files under it. During the uploading process, the system first creates a new file on the server, and then copies the contents of original file to the new file. Problems may happen if the filename on the server side is generated dynamically. Let's take a look at the following example. A user may be able to access the root of the web server file system by using "../../" path in the file field. Thus, some very important system files such as the system password file may be exposed. Or, a user may try to overwrite other files (both system files or other users' user files), by using the * character in the file field. In this system, filename restrictions are put on the inputs. If any dangerous symbols such as ../ or * are found, the input is considered insecure. In addition, the size of the uploading file and capacity of each assignment directory are also restricted to some reasonable range. If any user tries to upload an unreasonably large file or countless files with the intent to use up all free space on the server, the size restrictions can prevent the attacks.

For a secure system, it is important to maintain a system log file, which can keep track of users' operations on the server side. It also may be helpful for system administrators in identifying malicious users and dangerous operations. If any damage has been done to the system, a good log file can help with the system recovery. Although both Linux and MySQL systems have their own system log files, it is still necessary to create a particular log file for the AMS itself. In this system, three log files are created for three types of users. The content of the log files follows the format of the SQL language, so that the administrator can easily back up old data by executing these commands in order. File locks are used here to ensure the file consistency. Each process has to gain an exclusive lock before it executes writing to the log files. To avoid deadlock happening, the system assigns each process a maximum waiting time for gaining the lock. If the assigned

amount of time has passed and the process still cannot gain the lock, the process automatically sends an error report email to the administrator and jumps to the next step.

3. Conclusion

The currently implemented Assignment Management System provides instructors and students powerful features to handle assignments. The friendly HTML pages allow users to work easily and conveniently. The utilization of DBMS produces high system efficiency in data manipulation. Cross platform attributes of Perl and MySQL make it a portable system on most operating systems with slight modifications. In addition, the system security is strengthened by multiple security schemes.

The database design is very important during implementation because the database structure can significantly affect system efficiency and flexibility. Currently, the database structure is constructed in a very flexible manner, so that new data attributes or items can be easily added to the system without changing current structure significantly.

For further implementation, more features can be added to the system, such as the management of backup data or disaster recovery. More information could be explored according to users' requirements.

Appendix - Table Descriptions

```
create table ADMINISTRATOR (
        name   VARCHAR(40)      NOT NULL,
        userid CHAR(8)          NOT NULL,
        pwd    CHAR(20)         NOT NULL,
        PRIMARY KEY(userid)
);

create table STUDENT(
        stdID  CHAR(12)         NOT NULL,
        name   VARCHAR(40)      NOT NULL,
        userid CHAR(8)          NOT NULL,
        pwd    CHAR(20)         NOT NULL,
        course CHAR(10)         NOT NULL,
        status CHAR(10)         NOT NULL,
        PRIMARY KEY(userid, course),
        UNIQUE (stdID, userid)
);

create table INSTRUCTOR(
        name   VARCHAR(40)      NOT NULL,
        userid CHAR(8)          NOT NULL,
        pwd    CHAR(20)         NOT NULL,
        email  CHAR(30)         NOT NULL,
        PRIMARY KEY(userid)
);

create table SEMESTER(
        coursename    VARCHAR(40)      NOT NULL,
        course        CHAR(10)         NOT NULL,
        instructorid  CHAR(8)          NOT NULL,
        PRIMARY KEY(course)
);

create table COURSE(
        course CHAR(10)                 NOT NULL,
        item   CHAR(10)                 NOT NULL,
        value  TINYINT(3) UNSIGNED      NOT NULL,
        PRIMARY KEY(course, item)
);

create table ASSIGNMENT(
        course CHAR(10)     NOT NULL,
```

```
        item    CHAR(10)                    NOT NULL,
        points  SMALLINT(3) UNSIGNED  NOT NULL,
        post    DATE,
        due     DATETIME,
        PRIMARY KEY(course, item)
);

create table ACADEMY(
        course CHAR(10)                                 NOT NULL,
        userid  CHAR(8)                                 NOT NULL,
        item    CHAR(10)                                NOT NULL,
        time    DATETIME,
        grade   DECIMAL(6,2) UNSIGNED ZEROFILL    NOT NULL
        PRIMARY KEY(course, userid, item)
);

create table STUDENT_BACKUP(
        year            YEAR,
        semester        ENUM("SPRING", "SUMMER", "FALL"),
        stdID           CHAR(12),
        name            VARCHAR(40),
        userid          CHAR(8),
        pwd             CHAR(20),
        course          CHAR(10),
        status          CHAR(10)
);

create table INSTRUCTOR_BACKUP(
        year            YEAR,
        semester        ENUM("SPRING", "SUMMER", "FALL"),
        name            VARCHAR(40),
        userid          CHAR(8),
        pwd             CHAR(20),
        email           CHAR(30)
);

create table SEMESTER_BACKUP(
        year            YEAR,
        semester        ENUM("SPRING", "SUMMER", "FALL"),
        coursename      VARCHAR(40),
        course          CHAR(10),
        instructorid    CHAR(8)
);

create table COURSE_BACKUP(
        year            YEAR,
```

```
        semester        ENUM("SPRING", "SUMMER", "FALL"),
        course          CHAR(10),
        item            CHAR(10),
        value           TINYINT(3) UNSIGNED
);

create table ASSIGNMENT_BACKUP(
        year            YEAR,
        semester        ENUM("SPRING", "SUMMER", "FALL"),
        course          CHAR(10),
        item            CHAR(10),
        points          SMALLINT(3) UNSIGNED
        post            DATE,
        due             DATETIME
);

create table ACADEMY_BACKUP(
        year            YEAR,
        semester        ENUM("SPRING", "SUMMER", "FALL"),
        course          CHAR(10),
        userid          CHAR(8),
        item            CHAR(10),
        time            DATETIME,
        grade           DECIMAL(6,2) UNSIGNED ZEROFILL
);
```

References

1. http://cgi-pan.cqu.edu.an/david-jones/Publications/Papers_and_Books/97ascilites/.
2. http://www.cmu.edu/blackboard/help/index.html.
3. http://www.edu-link.com.
4. http://www.cyberlearninglabs.com/Products/AngelPortal/Tour.
5. Scott Guelich, Shishir Gundavaram, and Gunther Birznieks. CGI Programming with Perl. O'Reilly & Associates, Inc., 101 Morris Street, Sebastopol, CA 95472, 1996, pp. 265.
6. Randy Jay Yarger, George Reese, and Tim King. MySQL and mSQL. O'Reilly & Associates, Inc., 101 Morris Street, Sebastopol, CA 95472, 1999, pp. 7.
7. Randy Jay Yarger, George Reese, and Tim King. MySQL and mSQL. O'Reilly & Associates, Inc., 101 Morris Street, Sebastopol, CA 95472, 1999, pp. 154-155.
8. Scott Guelich, Shishir Gundavaram, and Gunther Birznieks. CGI Programming with Perl. O'Reilly & Associates, Inc., 101 Morris Street, Sebastopol, CA 95472, 1996, pp. 84.
9. Randal L. Schwartz and Tom Christiansen. Learning Perl, 2nd Edition. O'Reilly & Associates, Inc., 101 Morris Street, Sebastopol, CA 95472, 1997, pp. 76.
10. http://www.perl.com/pub/a/2002/02/20/css.html.
11. http://www.freebsd.org/doc/en_US.ISO8859-1/books/handbook/security-intro.html.
12. Randy Jay Yarger, George Reese, and Tim King. MySQL and mSQL. O'Reilly & Associates, Inc., 101 Morris Street, Sebastopol, CA 95472, 1999, pp. 51.