MIT Unified Submission and Grading System (MUGS)

Shannon Hwang, Kate Nelson, Jason Paulos

1. Introduction

6.033 is an MIT class that covers the fundamentals of computer system engineering and design, yet its grading system is severely lacking. Currently, several key system functionalities are haphazardly split between a submission site, Gradescope, and staff personal machines. 6.033's current system also fails keep track of files before submission, or facilitate group collaboration or submission for the team assignments that make up nearly half of the course.

We present an updated system design, the MIT Unified Submission and Grading System (MUGS), that solves these problems while prioritizing scalability, simplicity, reliability, and security. We achieve scalability by defining abstract elements such as users, groups, and assignment directories that can be easily adjusted to the course structure of other classes. We achieve simplicity by building these elements into the file system structure, on which we base system security and functionalities. We achieve reliability – critical for both students and staff in a grading system – by keeping uniquely-named records of all files and submissions and ensuring reliable file transmission across potentially unreliable networks. We achieve security by defining a hierarchy of user roles and permissions, ensuring that files are only accessible by the correct users. Scalability, simplicity, reliability, and security are closely intertwined, allowing our design decisions to intentionally align with all four of our goals.

2. System Overview

Our system consists of individual components that communicate with each other to execute varied user requests, as shown in Figure 1.

The MUGS Server is our system's central hub. It receives users' requests from the front-end, determines which actions to take, and executes these actions on the MIT File System. All of our system's custom logic resides here.

The MIT File System (MFS) stores all persistent information in our system. This includes user information, assignment submissions, grades, and more. MFS' permission system contributes to system security by determining which actions users can take.

The MUGS Front-end is the primary interface where users interact with our system. The front-end is responsible for passing user authentication information securely to the MIT ID service (MIDS), which authenticates users by granting them a Kerberos ID. This Kerberos ID is included in requests to the MUGS Server to identify users and their requests. The front-end is also responsible for uploading and downloading files to an instance of MFS using the MIT Sync Service (MSS).

Gradescope interacts with both end users and the MUGS Server. Students submit assignments to Gradescope, the assignments are graded, and the MUGS Server synchronizes Gradescope grades with MFS.

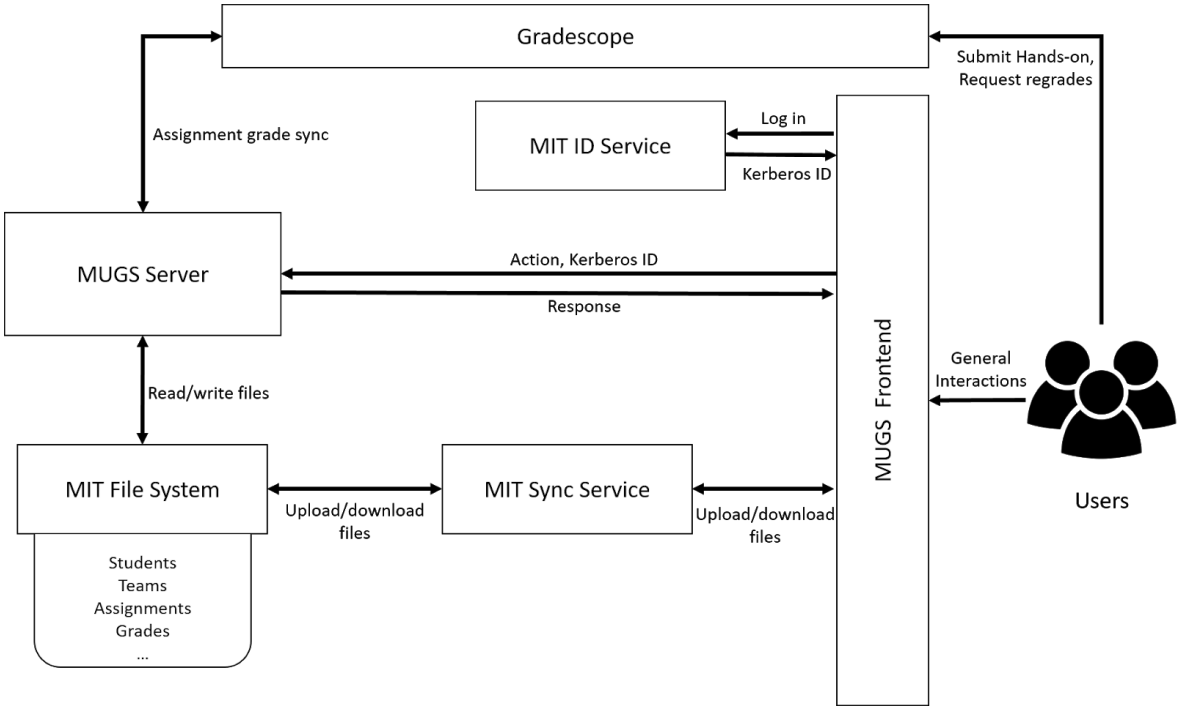

Figure 1: An overview of the different components and interactions in MUGS. Users interface with a front-end, which handles routing requests to the MIT ID Service, MIT Sync Service, and MUGS Server. The MUGS Server and MIT Sync Service also share access to the MIT File System, which stores the majority of information about the course. Gradescope has interactions with both the users and MUGS Server.

3. Roles

We use Kerberos IDs to represent every student and staff member. In order to simply implement security policies, we also assign a group ID to various groupings that arise naturally in the course such as recitations, tutorials, and teams, organized using a hierarchical model that reflects the various levels of permissions that the groups naturally have. With group and user IDs, controlling security and user access to various files and directories simply involves setting permissions in the file system corresponding to different user IDs. Figure 2 shows an example hierarchy of users and groups in the course. Notice that members of a group can be specific users as well members of other groups.

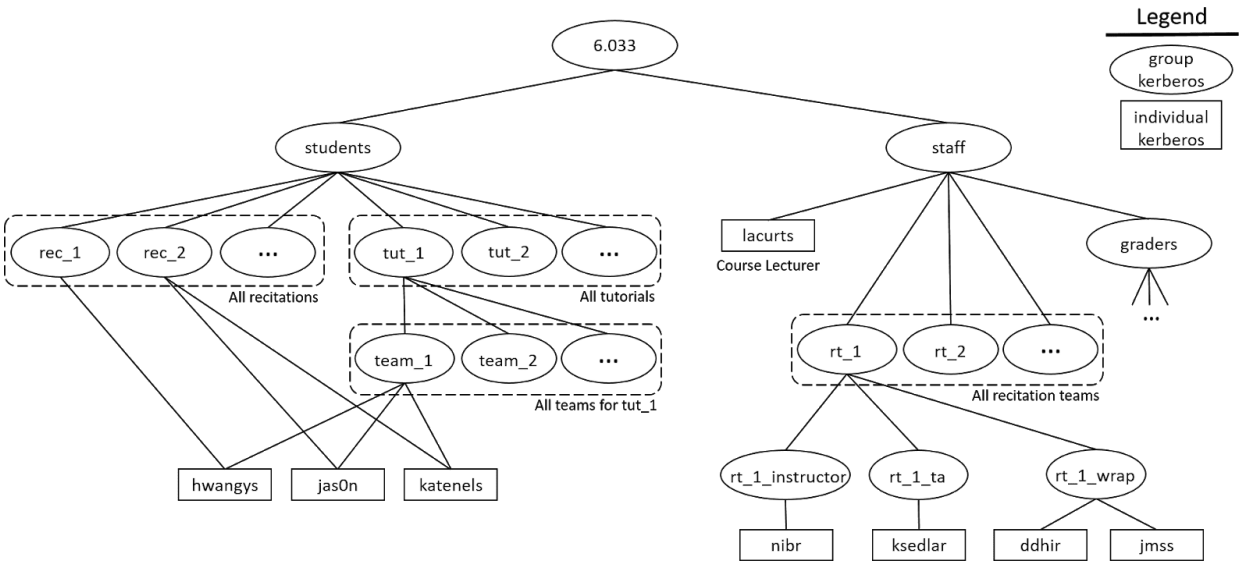

Figure 2: The hierarchy of users and groups in MUGS, shown with specific users as an example. 6.033 is the common ancestor of all roles, splitting users into the students and staff groups.

3.1 Users

A *role* in our system can be either an individual user or a group. Every role has a unique Kerberos, which is essential for identification in our system. The *parents* of a role are all of the groups to which it immediately belongs. Note that 6.033 is the only role that does not have a parent, and individual students have more than one parent. The *family* of a role are all of the groups to which it belongs at any depth, and itself. For example, in Figure 2, the parents of `jas0n` are `rec_2` and `team_1`, and the family of `jas0n` is `jas0n`, `rec_2`, `team_1`, `tut_1`, `students`, and `6.033`.

3.2 Permissions

An individual can use their Kerberos to perform actions on behalf of any members of their family. This functionality is implemented by examining all Kerberos belonging to the user's family in a well-ordered manner. The system will attempt to use each Kerberos to complete the action until either it succeeds in finding a Kerberos that can perform the action, or every family member's Kerberos is tried (in which case the action fails). The system examines parent links in each role's home directory (described in further detail in Section 4) to find a role's parents and compile its family members.

We chose to sacrifice simplicity for scalability in designing the permission system because we believe the ability to represent an arbitrarily complex grouping system with permission lookup allows us to abstract away much of the complexity from the rest of our system. The permission

lookup procedure actually results in increased overall security simplicity, since other components are not responsible for verifying the many relationships between users and groups.

4. Filesystem

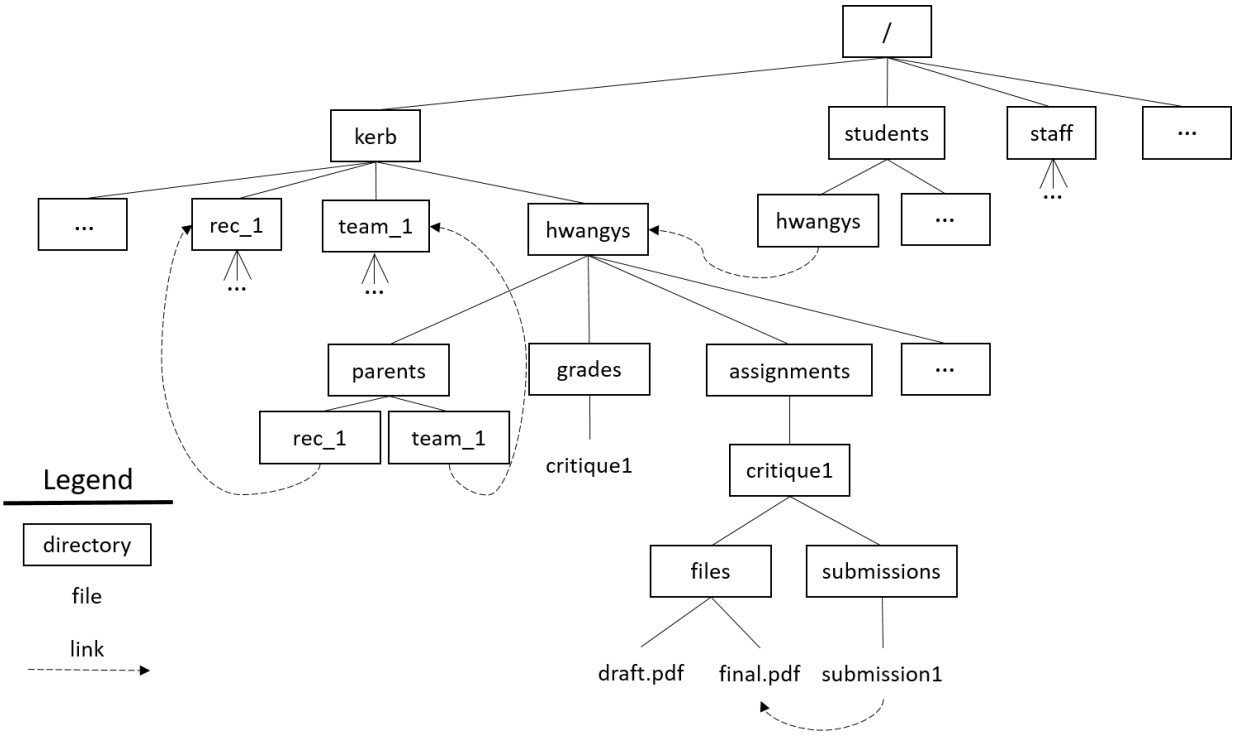

Figure 3: A subsection of the MUGS filesystem. The directory `kerb` contains the home directories of all users. Symbolic links are used throughout the file system to relate different entities and to ensure information is accessible to multiple users in well-defined locations.

Our filesystem structure follows our hierarchical group model to similarly support scalability. The filesystem enforces roles, relationships, and permissions by defining links between directories that facilitate access for students, teams, and staff.

The filesystem's `/kerb` directory contains UNIX-style home directories for all users and groups. Individual home directories are linked to another directory within `/students` or `/staff` as appropriate. For example, `/kerb/jas0n` and `/students/jas0n` both represent the home directory for the student `jas0n`. We also define parent relationships through links within home directories. For example, `/kerb/jas0n/parents` contains links to `jas0n`'s group folders: `/kerb/jas0n/parents/rec_2` links to `/kerb/rec_2` and `/kerb/jas0n/parents/team_1` links to `/kerb/team_1`. For security purposes, we ensure that only course staff can modify or create any parent links.

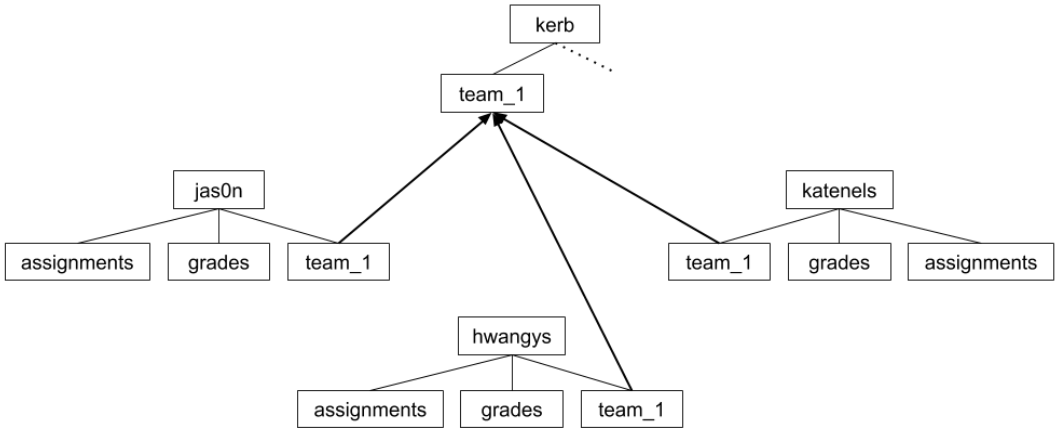

Figure 4: Our linking scheme is also used to coordinate files and submissions within teams. We create a link between a user's team folder and the high level team directory so that all team members can access the same files.

Team directories are linked similarly, as illustrated in Figure 4. Each student's home directory has a link to their team directory, in which they can share files with their team.

At the beginning of the term, staff will configure directories for assignments and roles as outlined in Figure 3. We leave this as a manual process so that the roles and directories can be simply scaled to other classes. Since we envision that each staff member will create directories for groups and users directly under them in the user hierarchy (for example, the course administrative TA creates directories for recitation TAs, who create directories for their recitations and corresponding students, and so on), thus dividing directory creation efficiently, we believe manual directory creation does not significantly hinder system utility.

For reliability, we keep copies of all submitted files and allow staff to grade any previous submission. We differentiate submissions using by naming files uniquely based on version number, submitting student, and timestamps (i.e. submission1_jas0n, submission2_katenels, etc.).

We also utilize the MIT Sync Service (MSS) and MIT Lock Service (MLS) when interfacing with the filesystem to prevent any concurrent access errors. MSS allows individual users to upload files from their machine to any of their directories, as well as download any accessible files or directories to their machine. We use MLS to handle concurrent accesses to files in MFS by blocking users from reading or writing to files that other users are writing to.

5. Network Protocol

Since we assume that users of the system are heavily invested in ensuring that their files are transmitted correctly, our network prioritizes reliability by following a TCP-like packet acknowledgement protocol when transmitting uploaded or downloaded files. For additional

reliability and fault tolerance, the network has an additional procedure for gracefully closing connections in the event of user cancelation of file uploads or downloads.

5.1 Packet Acknowledgement

Like in TCP, when users upload or download files from MUGS, the file is transmitted as a numbered sequence of data packets (1, 2, 3, ...). When the receiver gets a packet, it sends a numbered acknowledgement packet (*ack*) back confirming receipt of all packets up to the ack number. If the sender does not receive an ack for a transmitted packet *p* within a given time period, it retransmits *p* after a timeout proportional to the time it takes packets to make a round trip between sender and receiver. For reliability, the receiver maintains a buffer so it can order the packets it gets correctly, and keeps track of the last packet it has received to avoid sending duplicate acks.

The ack system imposes some overhead on the system by increasing the number of transmitted packets, but we believe that any performance or simplicity decrease from extra ack packets is worth the increased file transfer reliability.

5.2 Graceful Kill

Because networks are prone to “brownouts” and periods of variable latency and bandwidth, we anticipate that users will need to be able to cancel file uploads or downloads while they are in progress. In order to ensure that both MUGS and the user can reliably verify that a file transfer has been cancelled, we implement the following *graceful kill* procedure:

1. The user sends a packet with a “kill” header (retransmitting if necessary).
2. MUGS receives the “kill” packet and stops accepting packets from the user (if the user was uploading) or transmitting packets to the user (if the user was downloading).
3. MUGS sends a “kill ack” packet.
4. The user receives the “kill ack” packet and closes their connection.

6. Use Cases

6.1 Students

6.1.1 Submissions

Students have directories associated with each assignment, such as `/students/jas0n/assignments/critique1`, that contain assignment instructions and

`files` and `submissions` subdirectories. Students may upload intermediate work into `files`. When a student marks a file in `files` for submission, MUGS creates a link to that file in `submissions`. A student can submit and resubmit assignments as many times as they wish; we maintain a history of all submissions. Only the `submissions` folder is linked to the appropriate staff folders (e.g. the student's TA's folder) for grading.

6.1.2 Viewing Grades

TAs upload grades and comments into a separate folder (e.g. `/tas/ddhir/critique1/jas0n/`). Each student assignment directory also contains a `grades` folder. Once grade release is approved by staff, MUGS creates a link between the TA's grade file and the student's `grades` folder so students can view their grades.

6.1.3 Voting

Our design accommodates special assignments in which students submit ballots ranking videos created by other project teams. When students submit ballots, the system can screen ballots for possible voting abuses (such as voting for the same team twice, submitting multiple ballots, etc.) and tally total vote counts in a file located in an appropriate staff member's folder.

6.2 Teams

One of 6.033's core components is the team-based final design project which students complete in teams of three. Teams must share files, turn in group submissions, be graded as a group, and be able to see and vote on files from other teams. Our design enables secure, simple, and scalable implementation of team-based functionalities that are not supported by 6.033's current grading system.

6.2.1 Submissions and Grades

Since each team has their own team Kerberos, team file uploads and assignment submissions follow the functionality described for students in Section 6.1 (now using a shared folder like `/teams/teamKerb/assignments/DPPR/`). Similarly, viewing grades and comments as a team closely follows the functionalities described for students, except for a given grade, students now reference a grade file shared among members of the team in a folder like `teamKerb/assignments/DPPR/grades/`.

6.2.2 Forming Teams

Teams must be composed of students attending the same tutorial; thus, team formation naturally takes place in a student's tutorial folder, which is linked to their MUGS home folder. Since all students in the tutorial have access to the tutorial folder, the tutorial TA can use it to

coordinate team formation and create team directories as subdirectories of the tutorial directory, setting permissions and creating links to student home directories as necessary.

Our approach to team formation necessitates some work and potential for fault on the TA's part; however, we believe this sacrifice is reasonable because teams are small and there are only around 5 or 6 teams per tutorial, while this procedure maintains the simplicity of the system and is general enough to scale to arbitrary groupings in other classes.

6.2.3 Coordinating Files

Team assignment directories (e.g. `teamKerb/assignments/DPPR/`) are accessible by all team members, who can use the directory's files to collaborate on assignments. For reliability, MUGS uniquely names each uploaded file so that users can not silently overwrite other users' files.

Through naming, the system can keep track of which user uploaded the most recent submission for a given assignment and display that information when students try to submit a given team assignment. Though this approach sacrifices a bit of simplicity on the system's part, it greatly increases reliability as it prevents team members from unknowingly overwriting other members' submissions.

6.2.4 Sharing Work With Other Teams

Two team assignments – a report and video – need to be viewed by other teams. This is made possible by linking the submissions for these assignments to read-only folders within `/kerb/6.033`, which everyone can access. This linking process can be initiated by the course lecturer once all teams have submitted the appropriate assignment.

6.3 Staff

Course staff are responsible for grading student submissions, publishing grades for assignments, and monitoring each student's performance. Our filesystem makes these actions simple to support.

6.3.1 Assigning Grades

For reliability, our filesystem maintains separate grade files for each party responsible for grading or commenting on an assignment. Each party simply submits their evaluation to MUGS, and our system creates the appropriate grade file without any risk of overwriting grades given by other parties. Additionally, our system makes no assumptions about how to penalize students for late submissions. This allows graders to manually determine late penalties on a case by case basis, which keeps our system simple, flexible to individual circumstances, and scalable to different classes.

6.3.2 Publicizing Grades

Staff-uploaded grades are initially inaccessible to students. The course instructor must manually initiate the publication process, which links all grade files to their appropriate student assignment directories while ensuring that each student has read-only access to their grade files. The system then sends an email to all students notifying them that their grades have been released.

6.3.3 Viewing Grades

Staff can also generate a report of individual student grades. This report can be filtered in a few different ways, all of which are easily supported by our filesystem through the use of the wildcard (`*`) search symbol. For example, recitation teams can view grades of their students for individual assignments by the system collecting all grades matching a path such as `/kerb/rec_1/students/*/grades/critique1`. The course instructor can similarly collect all students' grades for all assignments by the system searching the path `/students/*/grades/*`.

6.3.4 Gradescope

In order to maintain records of assignments submitted through Gradescope, the MUGS Server performs a sync operation using Gradescope's API at the same time every day. If there has been a change in Gradescope's set of grades in the last 24 hours, MUGS pulls the set of all Gradescope grades, compares it to grades in MFS, and modifies any differing MFS grade files to reflect the grades and comments provided by Gradescope. We believe that a propagation time of at most 24 hours between Gradescope and MUGS is an acceptable trade off for the added simplicity of only synchronizing once per day.

7. Conclusion

MUGS uses Kerberos IDs and a corresponding hierarchical filesystem to enable 6.033 students and staff to collaborate on, submit, and grade a variety of assignments. MUGS is designed to prioritize four interwoven design goals: simplicity, scalability, reliability, and security. The simplicity of assigning abstract users to real-world analogues (staff members, students, and teams) and directories to real-world groupings (recitations, tutorials, etc.) ensures that security is also simple to implement by setting file and directory permissions granted to different users to reflect real-world privileges. In addition, our system prioritizes reliability and simplicity by simply tracking file history through uniquely-named copies of all uploaded files, and utilizing a simple ack system when transmitting files over unreliable network. Our current design prioritizes simplicity and reliability over performance, which may impact scalability in terms of class size (and the number of files that can be uploaded for a class). However, we believe that our design

can handle most standard class sizes and file management needs, and the overall simplicity of the system allows it to scale to a more important dimension: different types of classes and assignments.