**Class Tracker :: Class Audit System**

**CPRE 308 Final Project**

**Mike Croskey**

1. **Summary**

Class Tracker Server holds data passed using IPC from the client. It allows the user to create an audit that contains a list of information about courses attended at university. The user is able to call tasks from the server using the client program, each task is received by the server using a job parsing method. The server is able to generate a .txt file which contains a pretty text version of all the entered classes, credits and grades. The server's .txt audit includes a GPA calculation along with current number of completed credits and corresponding class information.

Class Tracker Client offers a friendly text based user interface for inputting degree and course information. The client has a menu system that allows the user to select which task is needed to be performed. The interface prompts the user for information pertinent to the selection menu, i.e. Student name, course name, credits, grade. The client also uses IPC communication to read and write from shared memory and the server monitors this for new jobs and information. The client offers the user the ability to create a new audit, add/remove courses and view a pretty .txt version of the audit.

I decided to select this project because I wanted to create a program that I might actually use in the future. I thought about what types of programs may be useful to my life style and I began to list different possibilities. I finally decided that creating a program that was useful to my academic career and the idea dawn on me for creating a personal audit system. I wanted a method of keeping track of my classes that was slightly simpler than ISU’s class audit system. Hence, this influenced me to create the class tracker audit system.

1. **Design**

As briefly stated in the summary I began designing this project by asking myself what I could create that would be useful to me in the future. So, with this in mind I began laying out the function I wanted my audit system to be able to handle.

I began with recognizing the immediate need for some function capable of printing the completed document to some sort of pretty text format in a user friendly file format. I continued to examine how my program should function and decided it needed some way of communicating with the user to request/receive information. So, I created the UI that is capable of navigating between different text menus, prompts user for information and displays entered information for user conformation.

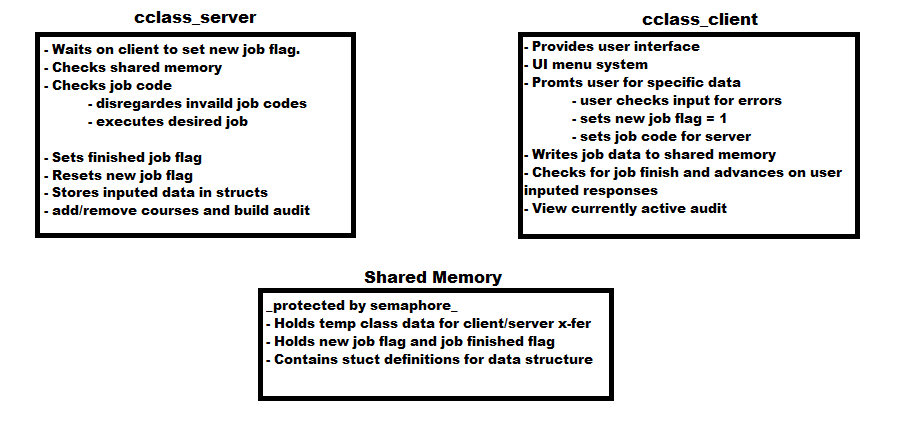
I decided to create the program as a two part system of server and client pair because I wanted to eventually expand the program to multiple users. Unfortunately, time restrictions prevented me from expanding the program beyond a single user. This style of design still allows for the use of the program as a daemon that communicates to the running server via the ipc communication style of shared memory.

During the design phase I ran into multiple issues that I had to figure out work around in order to make the program work correctly. The first major issue I ran into dealt with the shared memory structures I implemented for the transfer of information between server and client. The issue arose when I attempted to use a pointer, representing a string in the client for accessing data on the server side. I realized the pointer doesn’t work outside of the running process. In order to avoid this issue I simple declared the full size of every array within the shared memory structs. This allowed me to directly copy the string into the shared memory and then copy the string back out of the shared memory. A pointer can then be created that represents the shared memory structure array on the server side. This avoided pointer arithmetic errors and pointers referring to memory locations outside the active process.

About the time I figured out the issue with passing a string via the shared memory, I also figured out my second major issue of how to store the data passed from the client to server. I had to decide the best method for storing class information collected by the server. I decided to use a simple declaration of structures in the server to act as a data storage structure. This also provided modularity and enabled an easy transition to a multiple user structure as I originally intended on implementing.

The last major issue was discovered during the first testing run of my client and server. I realized a methods was needed for indicating “job” details to the server capable of activating certain functions on the server. To overcome this minor setback I decide to create a simple flagging job parsing method that would handle incoming job codes. The client received information from the user and upon conformation the client sets the information in the shared memory. The client also sets a flag indicating a new job is now active and set the job code. The server continuously checks the shared memory for the new job flag to appear and reads the job code. If the new job flag is set the server then references the job code and executes the appropriate job. In order to prevent any access issues I implemented a shared semaphore to protect the shared memory between client and server writes. I have included a diagram to help explain server to client communication (Diagram 1).

I quickly relies most issues before I begin programing but occasionally during coding I find unknowns that must be overcome. The final project had a number of challenges including shared memory issues, data structures, pointer variable scope and string formatting. But, despite these issues creating a few time setbacks I was still able to create a fully functional class auditing system.

****

1. **Results / Discussion**

I felt the program was completed to a point at which it satisfied the requirements and I began to test the program for proper function. I began testing the program by creating a new audit:

**CLIENT OUTPUT**

Welcome to Class Tracker! Your personal system for keeping records on all of you courses.

Please select from the menu options:

1) Create an audit

2) Exit

Enter 1 or 2:

1

Create a new Degree Audit:

--------------------------------

Enter Student Name: (max length 50)

mike

Enter Type of Degree: (max length 200)

Cpre

Enter total credits required for degree completion: (max 3 digit)

134

Is the following information correct?

Name = mike

Degree = Cpre

Total Credits = 134

Enter 1 (yes) or 2 (no) or 3 (exit):

1

To verify that the program is running correctly run the server in verbose mode using the –v argument in command line.

**SERVER OUTPUT:**

./class\_server -v

Verbose Enabled

Job Received : Job Code = 1

Created new audit:

student name = mike :: degree = Engr :: student credits = 132

Job Received : Job Code = 1

The verbose output confirms the server has received the job and indicates the job code received. It the outputs a verbose indicator of the job results for debugging. In this case the job code 3 is received telling the server to create a new audit and it confirms this is done correctly. I continued to test the server’s functionality by attempting to add a ccourse to a newly created audit:

**CLIENT OUTPUT**

Please select the desiered Action:

1) Add/Remove courses

2) View Curent Audit

3) Exit

Please Enter 1 or 2 or 3 :

1

Here you can Add and/or Remove Courses from the current audit:

1) Add

2) Remove

3) Exit

Enter 1 or 2 or 3 :

1

Add a new Course to the current Class Tracker Degree Audit:

Enter Student Name: (max length 50)

mike

Enter Class Name : (max length 200)

Cpre 308

Enter Instructor Name: (max length 50)

Vens

Enter Class type: ex -- general education (max length 50)

Computer Engineering

Enter Semester of Completion: ex -- Fall 2015 (max length 30)

fall 2015

Enter credits earn for course completion: (max 1 digit)

4

Enter Grade Recived: (A,B,C,D,F,I no(+/-))

A

Enter Brief Course Description: (max length 3000 characters)

it was awsome

Is the following information correct?

Student = mike

Class Name = Cpre 308

Instructor Name = Vens

Class Type = Computer Engineering

Semester Completed = fall 2015

Credits = 4

Grade =A

Description = it was awsome

Enter 1 (yes) or 2 (no) or 3 (exit):

1

**SERVER OUTPUT**

Job Recived : Job Code = 3

begin class add @ server

class add completed @ server

Class = Cpre 308

Instructor = Vens

Semster Completed = fall 2015

Grade = A

Class Type = CPRE

Class Descr. = it was fun

Class Credits = 4

class\_count = 1

The client prompts the user and the server confirms the course has been added as seen in the debugging output.

I tested the feature for removing a course from an active audit and it work successfully. I am omitting the results because the remove function works correctly and there is no need to display. The last test I confirmed was the request for view a finalized audit from the client. I navigated to the proper menu and selected the view audit option:

**CLIENT OUTPUT**

Waiting for audit to complete building.

Waiting for audit to complete building.

Waiting for audit to complete building.

Recieved File :: ./cclass\_data.txt

0 :: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1 ::

2 :: ---------------------- CLASS TRACKER--------------------------

3 ::

4 :: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

5 ::

6 :: Student Name :: mike

7 :: Degree :: Cpre

8 :: Credits Required = 134

9 :: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

10 ::

11 :: Course Name :: Cpre 308

12 ::

13 :: Instructor :: Vens

14 ::

15 :: Semester Completed :: fall 2015

16 ::

17 :: Grade :: A

18 ::

19 :: Class Type :: Computer Engineering

20 ::

21 :: Credits :: 4

22 ::

23 :: Class Description :: it was awesome

24 ::

25 :: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

26 ::

27 :: Course Name :: Engr 314

28 ::

29 :: Instructor :: todey

30 ::

31 :: Semester Completed :: fall 2015

32 ::

33 :: Grade :: A

34 ::

35 :: Class Type :: English

36 ::

37 :: Credits :: 4

38 ::

39 :: Class Description :: it was difficult

40 ::

41 :: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

42 ::

43 :: Total Credits Earned :: 8

44 :: GPA :: 4

45 :: END

**SERVER OUTPUT**

Job Received : Job Code = 5

Begin class audit creation @ server: mike

degree @ server: Engr

credits @ server: 132

Finished class audit creation @ server: mike

The server indicates it has received a job code 5 from the client and created an audit output that is then displayed by the client. The server also creates a new .txt file called class\_data.txt containing a copy of the generated audit. This file can then be printed, emailed or saved on external drive. The project is fully functional for a single user setup and the multi-user capability would have been implemented given a few more weeks to finish.

1. **Conclusion**

In conclusion, this project offered an array of challenges ranging from program structure to application of theory. I was confident the program would function correctly using the shared memory and semaphore setup to transfer information between two processes. Despite this confidence I was very worried this project might not work and relied on theory learned from class to design the functionality of the two programs. If I was able to do this project over again then I would probably do the exact same thing I did this time around. The project worked exactly as I had intended it to function. All of the abilities I originally wanted to incorporate are included in the project except for the multi-user function and loading stored audits from file. Thus, I feel given more time for completion I would easily have been able to create a multi-user setup and possibly an API for others to write UI for the server.

The Class Tracker project applies to this course because it uses concepts related to modern operating systems. This includes the use of a parsing system for handling incoming jobs, Shared memory as a form of IPC and use of semaphores to protect portions of shared memory. The project also implements many of the c programing concepts I have learned over previous years. Hence, I feel this project correlates with ECPE curriculum and applies the concepts learned throughout many of my courses.