

# CS 570 - Assignment 1

Concepts: interprocess communication through shared memory,  
Synchronization using semaphores

System calls: shmget, shmctl, shmat, shmdt, semget, semctl, semop, etc.

Maximum points 150

In this assignment, you will write several separate programs for creating and maintaining a Student database. For simplicity, you can assume that at most 20 entries will be in the database at any given time. The users of this database are the student advisors and students. The advisors have the right to change the entries in the database. Students only have the right to read the database. Several students and advisors are allowed to access the database concurrently. Each student entry in the database has the following fields: Name, Student ID, Address, and Telephone Number.

Your program should support the following operations:

**Load:** Loads the database. It should read the initial database from a file and load it into shared memory.

**Print:** Prints the contents of the shared memory to the screen with proper formatting.

**Change:** Lets a student advisor change the data related to any student with the given Student ID. A student advisor is required to have a password to change the database. You can assume the password is “000”, for simplicity.

**Query:** Lets anyone query the database. Anyone should be able to retrieve the record of any student by typing in the student’s ID.

**Clean:** Lets a student advisor save the contents of the shared memory to a file in a format that is suitable for loading into memory later and also deletes the shared memory segments and the semaphores created for synchronization.

**Hints and requirements:** Write five separate programs, to support these five functions. A program “Load.c” to load the database. i.e., it should create the shared memory segment required for loading the database and load the database into shared memory; it should also create the appropriate number of semaphores needed to synchronize concurrent access to the database. A program Print.c to print the contents of the shared memory to the screen, a program “Query.c” to query the database, a program “Change.c” to change the database and a program “Clean.c” to save the database to a file (in a format suitable for loading later) and delete the shared memory segments and the semaphores. Your program should provide a user-friendly user interface and provide appropriate error messages on wrong input. Note that once the database is loaded into memory several students and advisors will be accessing the database for modifying (using the Change program) and reading (using Print and Query). So, to maintain consistency of the database, you need to synchronize concurrent access. Use semaphores to synchronize concurrent access to the database. i.e., You should allow multiple readers (Print, Query) and writers (Change) to access the database concurrently. Readers and writers should be mutually exclusive. (i.e., two or more readers can concurrently access the database; a reader and a writer cannot concurrently access the database; two or more writers cannot concurrently access the database.) Your program should not deadlock and should satisfy the progress property. You **must** use shared memory and semaphores in your program to get credit. **You can assume that a user will not run Print, Query or Change before the Load program is executed or after Clean program has been executed.** You can also assume that the data file used by the Load program has no errors. **Make sure you have the line of code “sleep(10)” inside the critical section of each program so the grader as well as you can test if the synchronization is done correctly**

Before you logout from any machine, make sure you have deleted all the shared memory segments and semaphores created. You can look at ids of the shared memory segments and semaphores created (that have not been deleted) using the command `ipcs`. You can delete the shared memory segments and semaphores using the command `ipcrm`. See the *man* pages of these commands for more information.

**What to submit?** You should submit the tar file of the following files:

1. The program files, fully documented internally.
2. A makefile for compiling the program files; note that your make file should produce five executables Load, Print, Query, Change and Clean.
3. A “README” file containing instructions on how to compile and run your program.

Grading will be as follows:

Makefile : 10 points

Program compiles and all functions have been implemented (but may not work correctly) : 10 points

Load and Print work correctly: 20 points

Query works correctly: 20 points

Change works correctly: 20 points

Query and Change together work correctly: 30 points

Clean works correctly: 10 points

Documentation and modularity: 10 points

Robustness of your program: 20 points

What to submit: A tar (zip) file containing the makefile, README file and the source files.