ecet4640-lab4

1.0

Generated by Doxygen 1.9.6

1 ecet4640-lab4 **1**

1.1 Intro 1

1.2 Contributions 1

1.3 Overview 2

1.4 Arguments for program 2

2 Compilation **3**

2.1 Compilation Pipelines 3

2.2 Compiling and Running 3

2.3 Screenshot of Compilation 4

2.4 Cleaning 4

3 Module Index **5**

3.1 Modules 5

4 Data Structure Index **7**

4.1 Data Structures 7

5 File Index **9**

5.1 File List 9

6 Module Documentation **11**

6.1 Build 11

6.1.1 Detailed Description 11

6.1.2 Function Documentation 11

6.1.2.1 PopulateStudents() 11

6.1.2.2 BuildStudentMap() 12

6.1.2.3 PipeWhoToStudentMap() 13

6.1.2.4 ProcessWhoLine() 14

6.1.2.5 SetAllStudentsInactive() 15

6.1.2.6 WriteStudentsToMemory() 15

6.1.2.7 ReadInitialCumulative() 16

6.1.2.8 PipeAcpToStudentMap() 17

6.1.2.9 ReadCumulativeFileLine() 18

6.1.2.10 ReadAcpPipeLine() 19

6.1.2.11 CalculateCumulative() 19

6.1.3 Variable Documentation 20

6.1.3.1 students 20

6.1.3.2 dirty 21

6.2 Data 21

6.2.1 Detailed Description 21

6.2.2 Macro Definition Documentation 21

6.2.2.1 DATA_NUM_RECORDS 21

6.2.2.2 DATA_ID_MAX_LENGTH 22

6.2.2.3 DATA_NAME_MAX_LENGTH . . . 22
6.2.2.4 DATA_SIZE . . . 22
6.2.3 Variable Documentation . . . 22
6.2.3.1 Data_Names [1/2] . . . 22
6.2.3.2 Data_Names [2/2] . . . 23
6.3 Files . . . 23
6.3.1 Detailed Description . . . 23
6.3.2 Macro Definition Documentation . . . 24
6.3.2.1 STATIC_USER_DATA_FILE . . . 24
6.3.2.2 STATIC_USER_CUMULATIVE_FILE . . . 24
6.3.2.3 LOCKFILE . . . 25
6.3.3 Function Documentation . . . 25
6.3.3.1 FileExists() . . . 25
6.3.3.2 CreateInitialUserDataFile() . . . 25
6.3.3.3 FillStudentMapFromFile() . . . 26
6.3.3.4 WriteStudentArrayToFile() . . . 27
6.3.3.5 CreateInitialCumulativeFile() . . . 28
6.3.3.6 DoesLockfileExist() . . . 29
6.3.3.7 CreateLockfile() . . . 29
6.3.3.8 DeleteLockfile() . . . 30
6.4 Map . . . 30
6.4.1 Detailed Description . . . 31
6.4.2 Function Documentation . . . 32
6.4.2.1 NewMap() . . . 32
6.4.2.2 Map_Set() . . . 32
6.4.2.3 Map_Get() . . . 33
6.4.2.4 Map_Delete() . . . 33
6.5 MemShare . . . 34
6.5.1 Detailed Description . . . 34
6.5.2 Macro Definition Documentation . . . 34
6.5.2.1 MEM_KEY . . . 35
6.5.2.2 MEM_PERMISSIONS . . . 35
6.5.2.3 MEM_SIZE . . . 35
6.5.3 Function Documentation . . . 35
6.5.3.1 CreateSharedMemory() . . . 35
6.5.3.2 DestroySharedMemory() . . . 36
6.5.3.3 GetMemoryPointer() . . . 36
6.5.3.4 ReleaseMemoryPointer() . . . 37
6.6 Process . . . 37
6.6.1 Detailed Description . . . 38
6.6.2 Function Documentation . . . 38
6.6.2.1 TerminateExistingServer() . . . 38

6.6.2.2 IndicateRereadNeeded() 39
6.6.2.3 IndicateRereadDone() 39
6.6.2.4 IsRereadNeeded() 40
6.6.2.5 SignalHandle() 40
6.6.2.6 Initialize() 41
6.6.2.7 Process() 42
6.6.2.8 HelpCommand() 43
6.6.2.9 RunCommand() 44
6.6.2.10 StopCommand() 45
6.6.2.11 ResetCommand() 46
6.6.2.12 RunHeadless() 47
6.6.3 Variable Documentation 48
6.6.3.1 is_stopping 48
6.7 Util 48
6.7.1 Detailed Description 49
6.7.2 Function Documentation 49
6.7.2.1 RandomInteger() 49
6.7.2.2 RandomFloat() 49
6.7.2.3 RandomFlag() 50
6.7.2.4 Trim() 51

7 Data Structure Documentation **53**
7.1 _map_bucket Struct Reference 53
7.1.1 Detailed Description 53
7.2 map Struct Reference 53
7.2.1 Detailed Description 54
7.3 map_result Struct Reference 54
7.3.1 Detailed Description 54
7.4 Student Struct Reference 54
7.4.1 Detailed Description 55

8 File Documentation **57**
8.1 Build.c 57
8.2 Build.h 60
8.3 Data.c 60
8.4 Data.h 61
8.5 Files.c 61
8.6 Files.h 63
8.7 src/server/main.c File Reference 63
8.7.1 Function Documentation 64
8.7.1.1 main() 64
8.8 main.c 65
8.9 map.c 66

8.10 map.h 68
8.11 memShare.c 69
8.12 memShare.h 69
8.13 Process.c 69
8.14 Process.h 73
8.15 util.c 73
8.16 util.h 74

Chapter 1

ecet4640-lab4

1.1 Intro

This program reads user information using the `who` and `ac -p` commands and publishes that information as an array of Student structures to shared virtual memory for client processes to read. It updates every second.

The main.c page is a good starting point for following the program control flow.

1.2 Contributions

- On 9/14 all group members collaborated on VSCode LiveShare to implement the program skeleton, including the testing framework.
- On 9/17 Karl started the client and worked on memshare.
- On 9/18 all group members collaborated to start Build, Data, and memShare.
- On 9/20 Christian worked on functions to randomize and create the initial data and started the processing of the who pipe.
- On 9/21 all group members collaborated to fix up memshare and read files.
- On 9/22 Karl added the map and worked to populate data structures from files.
- On 9/23 Karl finished the reading who process and the control flow.
- On 9/24 Paul worked on handling command line arguments.
- On 9/25 Karl implemented the ac -p cumulative times and headless running.
- On 9/26 Karl and Christian started documentation.
- On 9/27 all group members collaborated to finish the documentation.

1.3 Overview

The first time the program runs, it generates files containing static user data and the cumulative login times for each user. As the server runs, it will recheck the result of 'who' and 'ac -p' to update the student's cumulative login times, determine which students are actively logged in, and what time they last logged in. This information is stored in a read-shared memory segment so clients can access it using the shared memory key. If necessary, it also updates student information in the file.

Only one server process should be running at a given time. To that end, a running server creates a lockfile in the /tmp folder and deletes the lockfile when it is done. New servers will not be started if a lockfile exists, but the running server can be stopped by passing the command line argument 'stop' to the binary. There are other command line arguments available, as detailed below.

1.4 Arguments for program

Argument	Description	Calls
help	Prints usage of program.	HelpCommand()
reset	Resets and re-randomizes the static user data and restarts the cumulative time tracking.	ResetCommand()
stop	Stops an existing server process if it is running.	StopCommand()
headless	Runs the program headlessly in the background if it is not already running.	RunHeadless()
run	Runs the server in the current program if it is not already running.	RunCommand()

Author

`Karl Miller`

Paul Shriner

Christian Messmer

Chapter 2

Compilation

2.1 Compilation Pipelines

There are several compilation pipelines, which are described in more detail in the Makefile comments.

The first is for making and running the regular server process. Calling `make` executes this. It uses the files in `src/server` to generate the binary and runs it. This will output the help for the server command. Executing `make server` will make the server binary without running it.

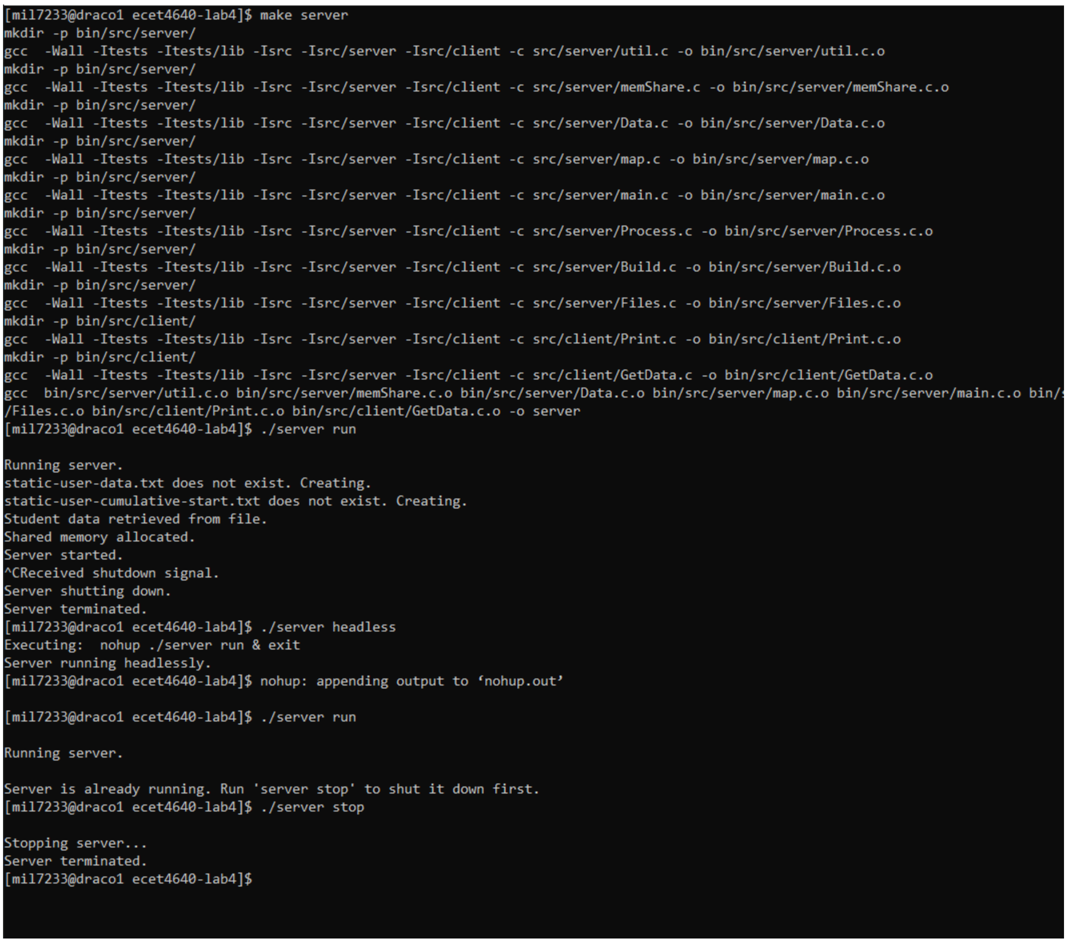
Second is for making the test client process with `make client`. This uses the files from `src/client`. The client process is not documented as it was not part of the program objective, and to avoid further documentation inflation.

Third is for making the test binary. This compiles the files in `tests` and the files in `src/server`, but excludes `src/main.c` so that `tests/main_test.c` will be the program entry point instead. The tests use `CuTest`. The tests are not documented here in order to not inflate the documentation size any further.

2.2 Compiling and Running

1. Copy the .zip file to the server
2. Extract the zip file.
3. Enter the unzipped folder.
4. Run `make server`
5. Run `./server run` to run the server in the shell.
6. Press ctrl-c to stop the server.
7. Run `./server headless` to run the server headlessly using nohup.
8. Run `./server reset` to re-randomize the user data and reset the login times.
9. Run `./server stop` to shut down the server. (You may want to leave it running so clients can connect to it)

2.3 Screenshot of Compilation



```
[mil7233@draco1 ecet4640-lab4]$ make server
mkdir -p bin/src/server/
gcc  -Wall -Itests -Itests/lib -Isrc -Isrc/server -Isrc/client -c src/server/util.c -o bin/src/server/util.c.o
mkdir -p bin/src/server/
gcc  -Wall -Itests -Itests/lib -Isrc -Isrc/server -Isrc/client -c src/server/memShare.c -o bin/src/server/memShare.c.o
mkdir -p bin/src/server/
gcc  -Wall -Itests -Itests/lib -Isrc -Isrc/server -Isrc/client -c src/server/Data.c -o bin/src/server/Data.c.o
mkdir -p bin/src/server/
gcc  -Wall -Itests -Itests/lib -Isrc -Isrc/server -Isrc/client -c src/server/map.c -o bin/src/server/map.c.o
mkdir -p bin/src/server/
gcc  -Wall -Itests -Itests/lib -Isrc -Isrc/server -Isrc/client -c src/server/main.c -o bin/src/server/main.c.o
mkdir -p bin/src/server/
gcc  -Wall -Itests -Itests/lib -Isrc -Isrc/server -Isrc/client -c src/server/Process.c -o bin/src/server/Process.c.o
mkdir -p bin/src/server/
gcc  -Wall -Itests -Itests/lib -Isrc -Isrc/server -Isrc/client -c src/server/Build.c -o bin/src/server/Build.c.o
mkdir -p bin/src/server/
gcc  -Wall -Itests -Itests/lib -Isrc -Isrc/server -Isrc/client -c src/server/Files.c -o bin/src/server/Files.c.o
mkdir -p bin/src/client/
gcc  -Wall -Itests -Itests/lib -Isrc -Isrc/server -Isrc/client -c src/client/Print.c -o bin/src/client/Print.c.o
mkdir -p bin/src/client/
gcc  -Wall -Itests -Itests/lib -Isrc -Isrc/server -Isrc/client -c src/client/GetData.c -o bin/src/client/GetData.c.o
gcc  bin/src/server/util.c.o bin/src/server/memShare.c.o bin/src/server/Data.c.o bin/src/server/map.c.o bin/src/server/main.c.o bin/
/Files.c.o bin/src/client/Print.c.o bin/src/client/GetData.c.o -o server
[mil7233@draco1 ecet4640-lab4]$ ./server run

Running server.
static-user-data.txt does not exist. Creating.
static-user-cumulative-start.txt does not exist. Creating.
Student data retrieved from file.
Shared memory allocated.
Server started.
^CReceived shutdown signal.
Server shutting down.
Server terminated.
[mil7233@draco1 ecet4640-lab4]$ ./server headless
Executing:  nohup ./server run & exit
Server running headlessly.
[mil7233@draco1 ecet4640-lab4]$ nohup: appending output to ‘nohup.out’

[mil7233@draco1 ecet4640-lab4]$ ./server run

Running server.

Server is already running. Run 'server stop' to shut it down first.
[mil7233@draco1 ecet4640-lab4]$ ./server stop

Stopping server...
Server terminated.
[mil7233@draco1 ecet4640-lab4]$
```

Figure 2.1 Compiling on draco1

2.4 Cleaning

There are two clean commands.

`make clean` will clean all .o files and binaries.

`make cleanf` will also remove the files generated on server initialization, such as the cumulative login file and user data file.

Chapter 3

Module Index

3.1 Modules

Here is a list of all modules:

Build 11
Data 21
Files 23
Map 30
MemShare 34
Process 37
Util 48

Chapter 4

Data Structure Index

4.1 Data Structures

Here are the data structures with brief descriptions:

_map_bucket
Map_bucket is an endpoint in the map. It is also a node in a linked list; if there were collisions, then the buckets are appended to the linked list at that location, then traversed until the matching key is found 53

map
A map. Stores key-value pairs for near constant lookup and insertion time 53

map_result
The result of a map retrieval 54

Student
The student data type 54

Chapter 5

File Index

5.1 File List

Here is a list of all documented files with brief descriptions:

src/server/Build.c 57

src/server/Build.h 60

src/server/Data.c 60

src/server/Data.h 61

src/server/Files.c 61

src/server/Files.h 63

src/server/main.c

Program entry point 63

src/server/map.c 66

src/server/map.h 68

src/server/memShare.c 69

src/server/memShare.h 69

src/server/Process.c 69

src/server/Process.h 73

src/server/util.c 73

src/server/util.h 74

Chapter 6

Module Documentation

6.1 Build

Functions that populate data structures.

Functions

- void PopulateStudents (char ∗∗studentIDs, char ∗∗studentNames, int arsize)
- void BuildStudentMap (map ∗stmap, Student ∗studentArr, int studentArrLength)
- int PipeWhoToStudentMap (map ∗stmap)
- int ProcessWhoLine (map ∗stmap, char ∗whoLine, int whoLineLength)
- void SetAllStudentsInactive (Student ∗stud_arr, int arr_len)
- void WriteStudentsToMemory (void ∗mem_ptr, Student ∗stud_arr, int arr_len)
- int ReadInitialCumulative (map ∗time_map, char ∗filename)
- int PipeAcpToStudentMap (map ∗st_map)
- void ReadCumulativeFileLine (map ∗time_map, char ∗acp_line)
- int ReadAcpPipeLine (map ∗stmap, char ∗acp_line)
- void CalculateCumulative (Student ∗stud_arr, int stud_arr_len, map ∗time_map)

Variables

- Student ∗ students
- short dirty = 1

6.1.1 Detailed Description

These functions perform actions that involve populating maps and arrays.

6.1.2 Function Documentation

6.1.2.1 PopulateStudents()

```
void PopulateStudents (
             char ** studentIDs,
             char ** studentNames,
             int arsize )
```

Allocate and populate the Students array with data.

Parameters

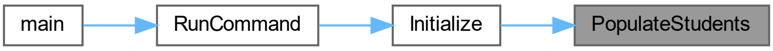
studentIDs	An array of student IDs.
studentNames	An array of student names.
arsize	The size of the array to allocate.

Warning

> studentIDs and studentNames must both be arsize in length.

Definition at line 17 of file Build.c.

Here is the caller graph for this function:



6.1.2.2 BuildStudentMap()

```
void BuildStudentMap (
            map * stmap,
            Student * studentArr,
            int studentArrLength )
```

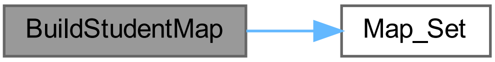
Given a student array, populates a student map, where the student IDs are the key, and the values are pointers to items in the array.

Parameters

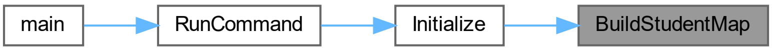
map	The map structure to populate.
studentArr	An array of student structures.
studentArrLength	The length of the students array.

Definition at line 28 of file Build.c.

Here is the call graph for this function:



Here is the caller graph for this function:



6.1.2.3 PipeWhoToStudentMap()

```
int PipeWhoToStudentMap (
            map * stmap )
```

Executes the 'who' command by reading from a file pipe. Calls ProcessWhoLine for each line, to realize updates in the user data from the who command.

Parameters

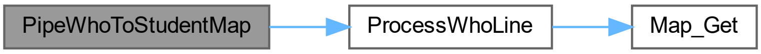
stmap	The student map.

Returns

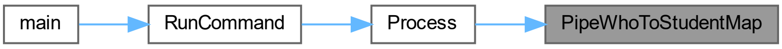
0 if succesful, otherwise nonzero.

Definition at line 41 of file Build.c.

Here is the call graph for this function:



Here is the caller graph for this function:



6.1.2.4 ProcessWhoLine()

```
int ProcessWhoLine (
            map * stmap,
            char * whoLine,
            int whoLineLength )
```

Processes a single line as read from the 'who' shell command. Uses that data to update the relevant student by retrieving them from the student map. Updates that students last login time. Also sets 'active' to 1 for the found student.

Attention

May set dirty to 1.

Parameters

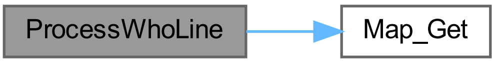
stmap	The student map.
whoLine	The line of text, such as returned from fgets
whoLineLength	The length of that text.

Returns

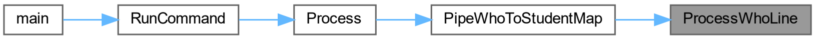
0 if success, -1 if the student was not found in the map.

Definition at line 61 of file Build.c.

Here is the call graph for this function:



Here is the caller graph for this function:



6.1.2.5 SetAllStudentsInactive()

```
void SetAllStudentsInactive (
             Student * stud_arr,
             int arr_len )
```

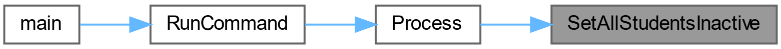
Sets the 'active' property on all students in the students array to 0.

Parameters

stud_arr	The students array.
arr_len	The length of the students array.

Definition at line 109 of file Build.c.

Here is the caller graph for this function:



6.1.2.6 WriteStudentsToMemory()

```
void WriteStudentsToMemory (
             void * mem_ptr,
             Student * stud_arr,
             int arr_len )
```

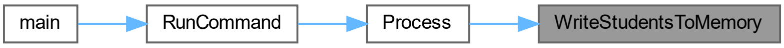
Writes the students array to the location specified by mem_ptr (eg. the shared memory segment).

Parameters

mem_ptr	The address to write at.
stud_arr	The students array to write.
arr_len	The length of the students array.

Definition at line 118 of file Build.c.

Here is the caller graph for this function:



6.1.2.7 ReadInitialCumulative()

```
int ReadInitialCumulative (
            map * time_map,
            char * filename )
```

Populates the cumulative map by reading from the initial cumulative file. The map will be of the form [userID] -> long seconds

The map will contain users who we don't care about, but it doesn't matter.

Parameters

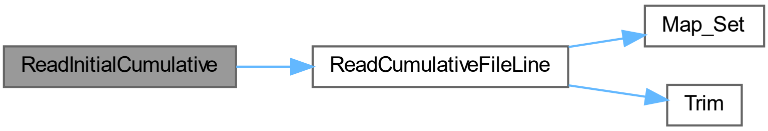
time_map	A map of cumulative times. Different from the students map.
filename	The filename where the initial cumulative times are located.

Returns

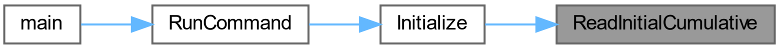
0 if success. -1 if it failed to find the file.

Definition at line 136 of file Build.c.

Here is the call graph for this function:



Here is the caller graph for this function:



6.1.2.8 PipeAcpToStudentMap()

```
int PipeAcpToStudentMap (
            map * st_map )
```

Pipes ac -p, then calls ReadCumulativeLine to update the student map.

Note

After this runs, the student map cumulative will be their total login time in the system. This total time must be subtracted from the cumulative map time to find the time they have been logged in since the program started.

Parameters

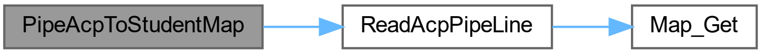
st_map	The students map.

Returns

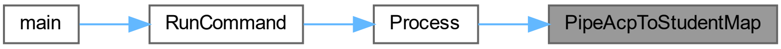
0 on success. -1 if the pipe could not be opened. Otherwise an error from ReadAcpPipeLine().

Definition at line 153 of file Build.c.

Here is the call graph for this function:



Here is the caller graph for this function:



6.1.2.9 ReadCumulativeFileLine()

```
void ReadCumulativeFileLine (
            map * time_map,
            char * acp_line )
```

Reads a single line from the initial cumulative file and updates the map so that userID maps to a long seconds value in the initial file.

Note

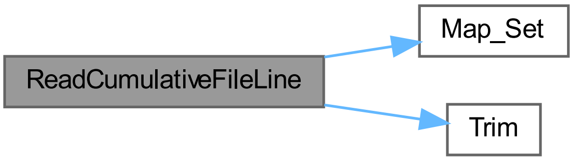
> A line is structured like this: `mes08346 10.06` It finishes with a line starting with `total`; this line should be disregarded.

Parameters

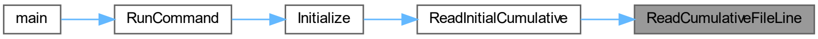
time_map	The cumulative map.
acp_line	A single line from ac -p.

Definition at line 177 of file Build.c.

Here is the call graph for this function:



Here is the caller graph for this function:



6.1.2.10 ReadAcpPipeLine()

```
int ReadAcpPipeLine (
            map * stmap,
            char * acp_line )
```

Reads a single line from the result of ac -p into the students map.

Parameters

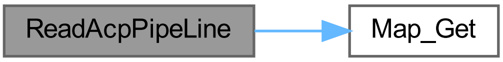
stmap	A map of students.
acp_line	A string representing 1 line result from ac -p.

Returns

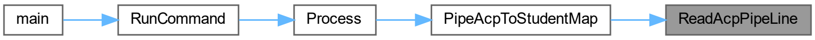
-1 if acp_line is NULL or length is less than 1, otherwise 0.

Definition at line 192 of file Build.c.

Here is the call graph for this function:



Here is the caller graph for this function:



6.1.2.11 CalculateCumulative()

```
void CalculateCumulative (
            Student * stud_arr,
            int stud_arr_len,
            map * time_map )
```

Calculates the cumulative time for each student by subtracting cum_map[studentID] from student.loginDuration.

Warning

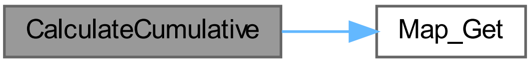
each student.loginDuration must have already been set to the total cumulative time logged in.

Parameters

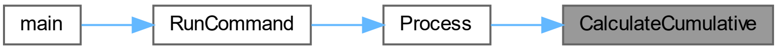
stud_arr	The student's array.
arr_len	The length of students array.
time_map	A map mapping studentIds to their cumulative login time when the server was started.

Definition at line 215 of file Build.c.

Here is the call graph for this function:



Here is the caller graph for this function:



6.1.3 Variable Documentation

6.1.3.1 students

```
Student* students
```

A pointer to the students array. It is heap allocated with malloc, when PopulateStudents() is called.

Note

> This array and its length are passed around via parameters, to decouple as much as possible and enable simple testing and dummy data, even though it is globally available.

Definition at line 15 of file Build.c.

6.1.3.2 dirty

```
short dirty = 1
```

Set to '1' if there are changes that should be written to a file.

Definition at line 39 of file Build.c.

6.2 Data

Declarations of types and macros.

Data Structures

- struct Student

  The student data type.

Macros

- #define DATA_NUM_RECORDS 17
- #define DATA_ID_MAX_LENGTH 9
- #define DATA_NAME_MAX_LENGTH 21
- #define DATA_SIZE 56

Variables

- char ∗ Data_Names [DATA_NUM_RECORDS]
- char ∗ Data_Names [ ]

6.2.1 Detailed Description

This module implements the data types required by the project specifications. The contents of this file should be shared with clients.

6.2.2 Macro Definition Documentation

6.2.2.1 DATA_NUM_RECORDS

```
#define DATA_NUM_RECORDS 17
```

The total count of records.

Definition at line 15 of file Data.h.

6.2.2.2 DATA_ID_MAX_LENGTH

```
#define DATA_ID_MAX_LENGTH 9
```

The amount of memory (bytes) required to be allocated for the ID field. Equal to the longest name in Data_IDs, "mes08346", plus the null terminator

Definition at line 20 of file Data.h.

6.2.2.3 DATA_NAME_MAX_LENGTH

```
#define DATA_NAME_MAX_LENGTH 21
```

The amount of memory (bytes) required to be allocated for the Name field. Equal to the longest name in Data_↩ Names, "Assefa Ayalew Yoseph", plus the null terminator

Definition at line 25 of file Data.h.

6.2.2.4 DATA_SIZE

```
#define DATA_SIZE 56
```

The size of one student record; the result of sizeof(Student).

Definition at line 36 of file Data.h.

6.2.3 Variable Documentation

6.2.3.1 Data_Names [1/2]

```
char* Data_Names[DATA_NUM_RECORDS]
```

Initial value:

```
= {
    "Weifeng Chen",
    "Christian Beatty",
    "Emily Bolles",
    "Cameron Calhoun",
    "Ty Kress",
    "Cody Long",
    "Caleb Massey",
    "Christian Messmer",
    "Karl Miller",
    "Jeremiah Neff",
    "Kaitlyn Novacek",
    "Joshua Panaro",
    "Caleb Rachocki",
    "Caleb Ruby",
    "Paul Shriner",
    "Alan Vayansky",
    "Assefa Ayalew Yoseph"}
```

Constant, all user's names.

Definition at line 26 of file Data.c.

6.2.3.2 Data_Names [2/2]

```
char* Data_Names[]  [extern]
```

Constant, all user's names.

Definition at line 26 of file Data.c.

6.3 Files

The Files module contains functions which operate on files.

Macros

- #define STATIC_USER_DATA_FILE "static-user-data.txt"
- #define STATIC_USER_CUMULATIVE_FILE "static-user-cumulative-start.txt"
- #define LOCKFILE "/tmp/ecet-server.lock"

Functions

- short FileExists (char *file_name_to_check)

  Determines whether a file exists.
- int CreateInitialUserDataFile (char *file_name, char **id_list, int id_list_len)

  Creates the initial user data file. This should be called only the first time the program runs, if it doesn't exist.
- int FillStudentMapFromFile (map *student_map, char *file_name, char **id_list, int id_list_len)

  Fills the student map with data from the file. It gets age, gpa, and lastLogin from this file.
- int WriteStudentArrayToFile (Student *students, int arr_len, char *file_name)

  Writes the student array to the file.
- int CreateInitialCumulativeFile (char *file_name)
- short DoesLockfileExist ()
- int CreateLockfile ()
- int DeleteLockfile ()

6.3.1 Detailed Description

Some program data needs to be stored in files, to preserve it in the case of early termination.

There are three files that are created if they don't exist when the program is first run.

- STATIC_USER_DATA_FILE contains a list of userIDs, ages, gpa, and last login time. Age and gpa are randomly generated on server start and when "reset" is run. The login time is updated when it changes as per the dirty flag.
- STATIC_USER_CUMULATIVE_FILE contains the results of 'ac -p' run when the server first starts. These values will be subtracted from later pipes of "ac -p" to determine the cumulative time since the server started.
- LOCKFILE contains a flag, 0 or 1, that indicates whether the STATIC_USER_DATA_FILE has been re-randomized and should be re-read. It contains the process ID of the running server process. It serves as an indicator to the process as to whether a server is already running and, when "close" is passed as a command line argument, which process to kill.

6.3.2 Macro Definition Documentation

6.3.2.1 STATIC_USER_DATA_FILE

```
#define STATIC_USER_DATA_FILE "static-user-data.txt"
```

File name for the text file that will store user data, namely, the age, gpa, and last login time.

Note

Each line contain in the created file contains:

(1) The ID from the students array, where the `line # - 1` == the index of the students array

(2) A tab character

(3) A random int between 18 and 22, for the age.

(4) A tab character

(5) A random float between 2.5 and 4.0, for the GPA.

(6) A tab character.

(7) A 0 (representing the last login time)

(8) A newline.

The order of entries in the file is the same as the order in the Data_IDs array from Data.c.

Definition at line 33 of file Files.h.

6.3.2.2 STATIC_USER_CUMULATIVE_FILE

```
#define STATIC_USER_CUMULATIVE_FILE "static-user-cumulative-start.txt"
```

File name for the text file that will store the cumulative login time for each user at the point in time when it was created.

The values in this file are subtracted from the result of running 'ac -p' later to get the cumulative time each user was logged in since the server started.

Note

Each line contains the following.

(1) A user ID

(2) An integer representing the minutes the user has been logged in.

Definition at line 43 of file Files.h.

6.3.2.3 LOCKFILE

```
#define LOCKFILE "/tmp/ecet-server.lock"
```

The lockfile serves as a signal to subsequent processes as to whether or not server is already running.

Note

File contains the following

(1) a 1 or a 0 indicating whether the data has been reset and must be re-read

(2) an integer correcponding to the PID of the process so that server close can end that process

Definition at line 53 of file Files.h.

6.3.3 Function Documentation

6.3.3.1 FileExists()

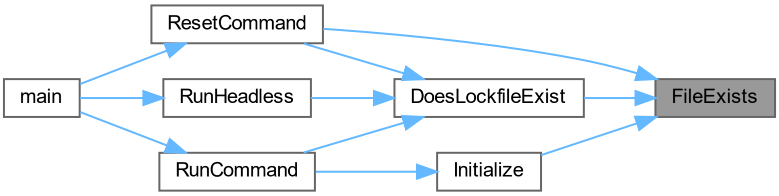
```
short FileExists (
             char * file_name_to_check )
```

Returns

1 if it exists. 0 if it does not.

Definition at line 12 of file Files.c.

Here is the caller graph for this function:



6.3.3.2 CreateInitialUserDataFile()

```
int CreateInitialUserDataFile (
             char * file_name,
             char ** id_list,
             int id_list_len )
```

Parameters

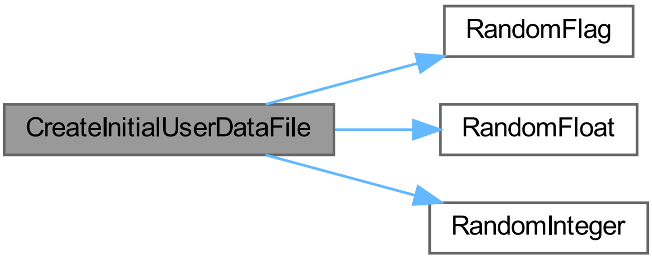
file_name	The file name to create.
id_list	An array containing the IDs. Eg. "Data_IDs" from Data.h
id_list_len	The length of the id_list. Eg. "DATA_NUM_RECORDS" from Data.h

Returns

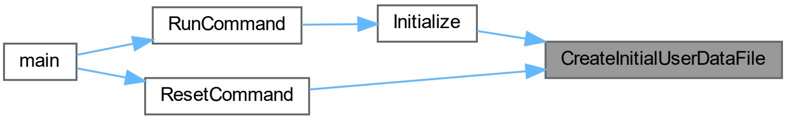
A 0 if the operation was succesful, otherwise nonzero.

Definition at line 27 of file Files.c.

Here is the call graph for this function:



Here is the caller graph for this function:



6.3.3.3 FillStudentMapFromFile()

```
int FillStudentMapFromFile (
             map * student_map,
             char * file_name,
             char ** id_list,
             int id_list_len )
```

Parameters

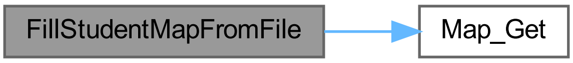
student_map	The map of student structs to be populated from the login.txt file
file_name	The name of the login.txt file.
id_list	An array containing the IDs. Eg. "Data_IDs" from Data.h
id_list_len	The length of the id_list. Eg. "DATA_NUM_RECORDS" from Data.h

Returns

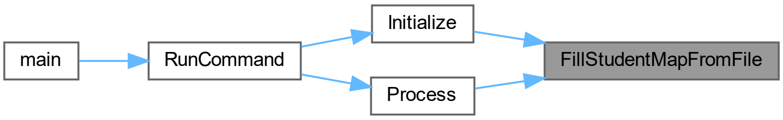
0 if succesful, 1 if there was an error.

Definition at line 53 of file Files.c.

Here is the call graph for this function:



Here is the caller graph for this function:



6.3.3.4 WriteStudentArrayToFile()

```
int WriteStudentArrayToFile (
            Student * students,
            int arr_len,
            char * file_name )
```

Parameters

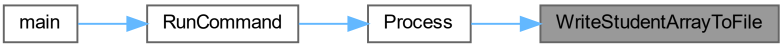
students	A pointer to the student array that will be read into the file.
arr_len	The length of the students array. e.g. DATA_NUM_RECORDS from Data.h.
file_name	The file name to write.

Returns

A 0 if the operation was succesful, otherwise a nonzero.

Definition at line 80 of file Files.c.

Here is the caller graph for this function:



6.3.3.5 CreateInitialCumulativeFile()

```
int CreateInitialCumulativeFile (
             char * file_name )
```

Creates the initial cumulative login time file.

It will hold the result of running 'ac -p'.

Parameters

file_name	The name of the file to created. EG STATIC_USER_CUMULATIVE_FILE

Warning

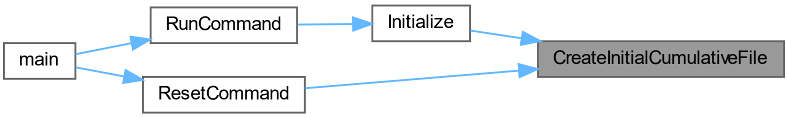
This file should already be validated to not exist.

Returns

0 if succesful, -1 if the file couldn't be opened, -2 if the pipe couldn't be opened, otherwise an error code.

Definition at line 96 of file Files.c.

Here is the caller graph for this function:



6.3.3.6 DoesLockfileExist()

```
short DoesLockfileExist ( )
```

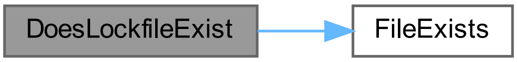
Determines if lockfile exists, which indicates that a server process is already running.

Returns

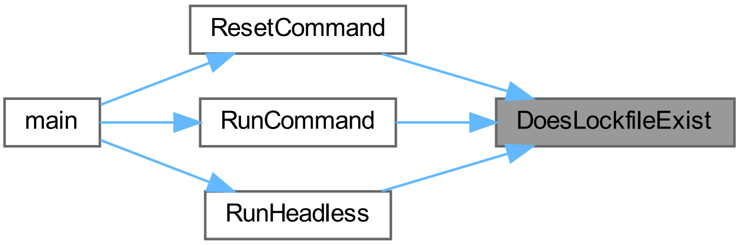
0 if lockfile does not exist, 1 if it does.

Definition at line 122 of file Files.c.

Here is the call graph for this function:



Here is the caller graph for this function:



6.3.3.7 CreateLockfile()

```
int CreateLockfile ( )
```

Creates a lockfile.

Warning

This should only be called by a running server process when a lockfile does not already exist.

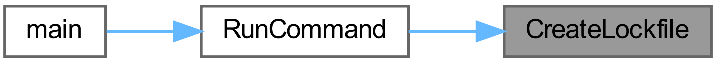
The lockfile will carry a 'data reset' signal and a process ID. CreateLockfile will write the current processes PID.

Returns

-1 if fopen failed, otherwise 0.

Definition at line 127 of file Files.c.

Here is the caller graph for this function:



6.3.3.8 DeleteLockfile()

```
int DeleteLockfile ( )
```

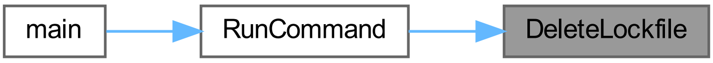
Deletes the lockfile.

Returns

0 on success, -1 on failure.

Definition at line 139 of file Files.c.

Here is the caller graph for this function:



6.4 Map

Functions that implement a hash map data structure.

Data Structures

- struct _map_bucket

  map_bucket is an endpoint in the map. It is also a node in a linked list; if there were collisions, then the buckets are appended to the linked list at that location, then traversed until the matching key is found.
- struct map

  A map. Stores key-value pairs for near constant lookup and insertion time.
- struct map_result

  The result of a map retrieval.

Functions

- map ∗ NewMap (int capacity)
- void Map_Set (map ∗a_map, char ∗key, void ∗value)

  Sets a value in the map.
- map_result Map_Get (map ∗a_map, char ∗key)

  Gets a value from the map. It will return a map_get_result describing whether it was succesful, and possibly containing the data sought, or NULL if it was unsuccesful.
- map_result Map_Delete (map ∗a_map, char ∗key, short free_it)

  Deletes a key from the map. Returns a map_get_result describing whether the delete was succesful and containing the removed data, if extant.

6.4.1 Detailed Description

Karl's take on a simple hash map structure, which maps strings to void pointers. You can use casting to convert the void pointers into most of whatever else is needed.

Example usage, casting an int into the data part of the map.

```
int myfunc() {
     map *mymap = NewMap(100);
     Map_Set(mymap, "age", (void*)55);
     map_result result = Map_Get(mymap, "age");
     int age;
     if(result.found) {
         age = (int) map_result.data;
     }
}
```

Note, with this simple implementation, the map cannot change its capacity. A change to its capacity would change the hashing.

Ultimately there are really only three things you need to do with the map.

Initialize it, with some capacity larger than you will use. EG map ∗ mymap = NewMap(100). The bigger it is, the fewer collisions (which are pretty rare anyway).

Set some values in it. Eg Map_Set(mymap, "key", &value);

You can cast numbers to void pointers to put them in the map, or you can use the pointers as references to, for example, strings malloced somewhere.

Get some values from it. Eg void∗ myval = Map_Get(mymap, "key");

Delete some values from it. For example Map_Delete(mymap, "key", 0);

Note that the last parameter, 'free it', tells the map whether it should call 'free' on the underlying data in memory. If this is 1, and the underyling data is not a reference to a malloced part of the heap, errors will result.

6.4.2 Function Documentation

6.4.2.1 NewMap()

```
map * NewMap (
             int capacity )
```

Creates a new map. The map capacity will be a power of 2 that is large enough to contain the estimated size.

Parameters

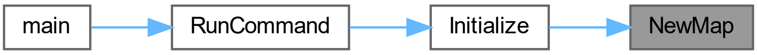
capacity	The estimated required capacity of the map.

Returns

A pointer to the heap allocated map.

Definition at line 49 of file map.c.

Here is the caller graph for this function:



6.4.2.2 Map_Set()

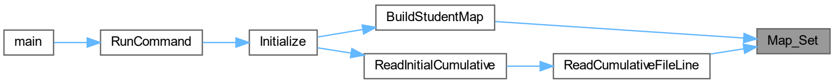
```
void Map_Set (
             map * a_map,
             char * key,
             void * value )
```

Parameters

map	The map to set a key in.
key	The key to use.
keylen	The length of the key.
value	The pointer to the data stored at that location.

Definition at line 89 of file map.c.

Here is the caller graph for this function:



6.4.2.3 Map_Get()

```
map_result Map_Get (
            map * a_map,
            char * key )
```

Parameters

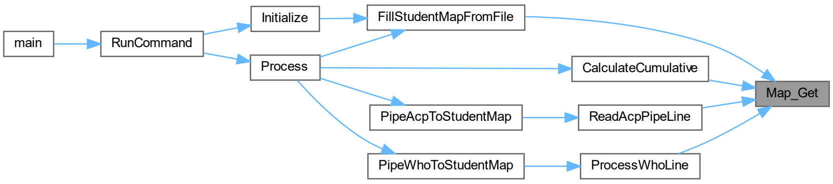
map	The map to retrieve from.
key	The key of the item.

Returns

A map_get_result containing the sought data.

Definition at line 119 of file map.c.

Here is the caller graph for this function:



6.4.2.4 Map_Delete()

```
map_result Map_Delete (
            map * a_map,
            char * key,
            short free_it )
```

Parameters

map	The map to delete the key from.
key	The key to delete.
free↩ _it	Whether to call free() on the underlying data.

Returns

A map_get_result with the data that was removed.

Definition at line 154 of file map.c.

6.5 MemShare

Functions that operate on a shared memory segment.

Macros

- #define MEM_KEY 0x727
- #define MEM_PERMISSIONS 0664
- #define MEM_SIZE DATA_SIZE ∗DATA_NUM_RECORDS

Functions

- int CreateSharedMemory ()
- int DestroySharedMemory ()
- void ∗ GetMemoryPointer (int shared_mem_id)
- int ReleaseMemoryPointer (void ∗shmaddr)

6.5.1 Detailed Description

To share data to clients program uses shared memory

- MEM_KEY is the key to access the shared memory and clients must have this info
- MEM_PERMISSIONS who has read, write permissions of the shared memory segment
- MEM_SIZE the total size of the shared memory allocation

6.5.2 Macro Definition Documentation

6.5.2.1 MEM_KEY

```
#define MEM_KEY 0x727
```

The shared memory key that clients and servers will use to identify the segment.

Definition at line 21 of file memShare.h.

6.5.2.2 MEM_PERMISSIONS

```
#define MEM_PERMISSIONS 0664
```

Memory permissions are: Self: RW 110 = 6 Group: R 100 = 4 Others: R 100 = 4

- All groups can read.
- Self can write.
- None can execute.

Definition at line 32 of file memShare.h.

6.5.2.3 MEM_SIZE

```
#define MEM_SIZE DATA_SIZE *DATA_NUM_RECORDS
```

The memory allocation must as large as the data size times the number of records.

Definition at line 37 of file memShare.h.

6.5.3 Function Documentation

6.5.3.1 CreateSharedMemory()

```
int CreateSharedMemory ( )
```

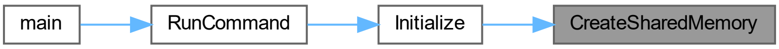
CreateSharedMemory retrieves a shared memory ID that can be used to access or delete shared memory.

Returns

A shared memory ID that can be used with other 'shm' commands to access shared memory, -1 if an error has occured

Definition at line 9 of file memShare.c.

Here is the caller graph for this function:



6.5.3.2 DestroySharedMemory()

```
int DestroySharedMemory ( )
```

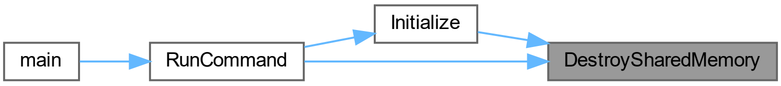
Flags the shared memory segment for deallocation. Returns the result of that operation.

Returns

0 if succesful. 1 if not succesful. Errno will be set.

Definition at line 14 of file memShare.c.

Here is the caller graph for this function:



6.5.3.3 GetMemoryPointer()

```
void * GetMemoryPointer (
            int shared_mem_id )
```

"Attaches" to the shared memory, returning a memory pointer to the shared memory.

Calls 'shmat(shared_mem_id, NULL, 0)`;

Parameters

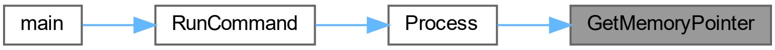
shared_mem← _id	The id of the shared memory

Returns

A pointer to the shared memory, or -1 if it fails.

Definition at line 23 of file memShare.c.

Here is the caller graph for this function:



6.5.3.4 ReleaseMemoryPointer()

```
int ReleaseMemoryPointer (
            void * shmaddr )
```

Release a shm memory pointer.

Parameters

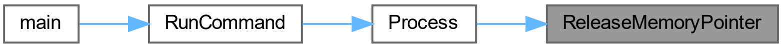
shmaddr	The memory pointer to release.

Returns

Whether the operation was succesful.

Definition at line 28 of file memShare.c.

Here is the caller graph for this function:



6.6 Process

Functions that manage control flow.

Functions

- int TerminateExistingServer ()
- int IndicateRereadNeeded ()
- int IndicateRereadDone ()

- short IsRereadNeeded ()
- void SignalHandle (int signo)
- int Initialize ()
- void Process (int shm_id)
- void HelpCommand ()
- void RunCommand ()

    Runs the server if it doesn't already exist.
- void StopCommand ()
- void ResetCommand ()
- void RunHeadless (char ∗processName)

Variables

- map ∗ **initial_cumulative_times**

    A map of userIDs to integer seconds. These values are subtracted from the current total cumulative time for each user to calculate their cumulative time since the server process started.
- short is_stopping = 0

6.6.1 Detailed Description

This module handles the processes that this server might execute. It calls functions from the other modules to realize program changes.

It contains the main update loop for a running server, Process(), as well as functions for implementing the different command line argument driven procedures.

6.6.2 Function Documentation

6.6.2.1 TerminateExistingServer()

```
int TerminateExistingServer ( )
```

Reads the lockfile to get the ID of the process that created it.

Sends a SIGTERM signal to that process.

Warning

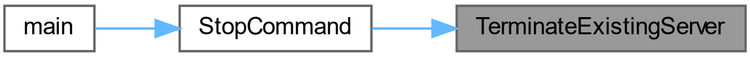
lockfile should be confirmed to exist

Returns

-1 if file doesn't exist, -2 if no valid process ID existed in the file, 1 if sending the kill signal failed.

Definition at line 22 of file Process.c.

Here is the caller graph for this function:



6.6.2.2 IndicateRereadNeeded()

```
int IndicateRereadNeeded ( )
```

If we reset the user data, we need to indicate to the running process that a re-read is needed. This changes the flag in the lockfile to 1, but keeps the same process ID as before there.

Warning

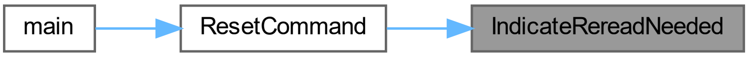
should only be called by non main processes

Returns

-1 if lockfile not found, 0 if success, or an error number if some other error

Definition at line 40 of file Process.c.

Here is the caller graph for this function:



6.6.2.3 IndicateRereadDone()

```
int IndicateRereadDone ( )
```

If we re-read the users file, we can indicate that we have done so by setting the re-read flag back to 0.

Warning

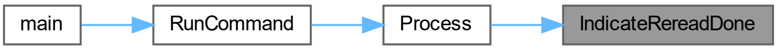
should only be called by main process.

Returns

0 on success, -1 if the file was not found, otherwise an error number produced by fclose.

Definition at line 57 of file Process.c.

Here is the caller graph for this function:



6.6.2.4 IsRereadNeeded()

```
short IsRereadNeeded ( )
```

Reads the lockfile for the re-read flag.

Warning

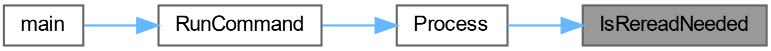
Lockfile should exist - should be called by the server in the main process loop

Returns

0 if the Lockfile starts with '0', 1 if the Lockfile starts with '1'.

Definition at line 74 of file Process.c.

Here is the caller graph for this function:



6.6.2.5 SignalHandle()

```
void SignalHandle (
            int signo )
```

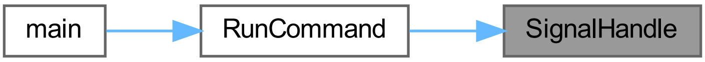
Called by a new server process, telling this server process to shut down. This sets 'is_stopping' to true, which shuts down the server gracefully, writing any necessary data to the user data file, then deleting the lockfile.

Parameters

signo	The signal number will be SIGTERM from the other server process or SIGINT if interrupted from the console.

Definition at line 82 of file Process.c.

Here is the caller graph for this function:



6.6.2.6 Initialize()

```
int Initialize ( )
```

Run once at program start. Calls functions from other modules to do the following:

Note

(1) - Create an initial user data file if it doesn't exist.

(2) - Initialize the students array.

(3) - Initialize the students map.

(4) - Read the data from the user data file into the map/array.

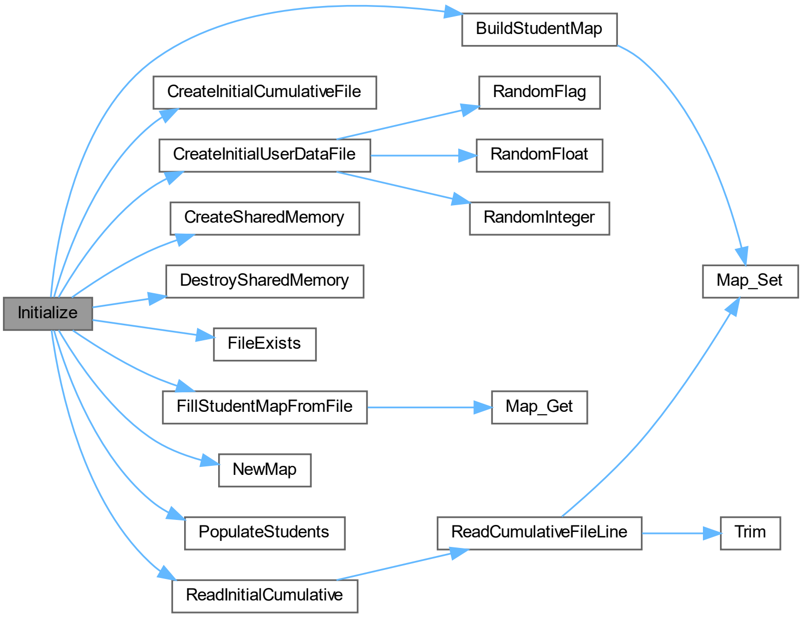
(5) - Initializes the shared memory segment.

Returns

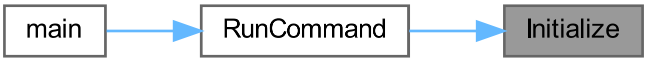
The ID of the shared memory segment or -1 if an error has occured.

Definition at line 97 of file Process.c.

Here is the call graph for this function:



Here is the caller graph for this function:



6.6.2.7 Process()

```
void Process (
             int shm_id )
```

Called repeatedly with a delay.

Note

(1) - Sets all users to inactive.

(2) - Reads the result of the `who` command, setting some users to active, and possibly changing 'dirty' and last login times.

(3) - Overwrites the user data file if we are dirty.

(4) - Sets dirty to false.

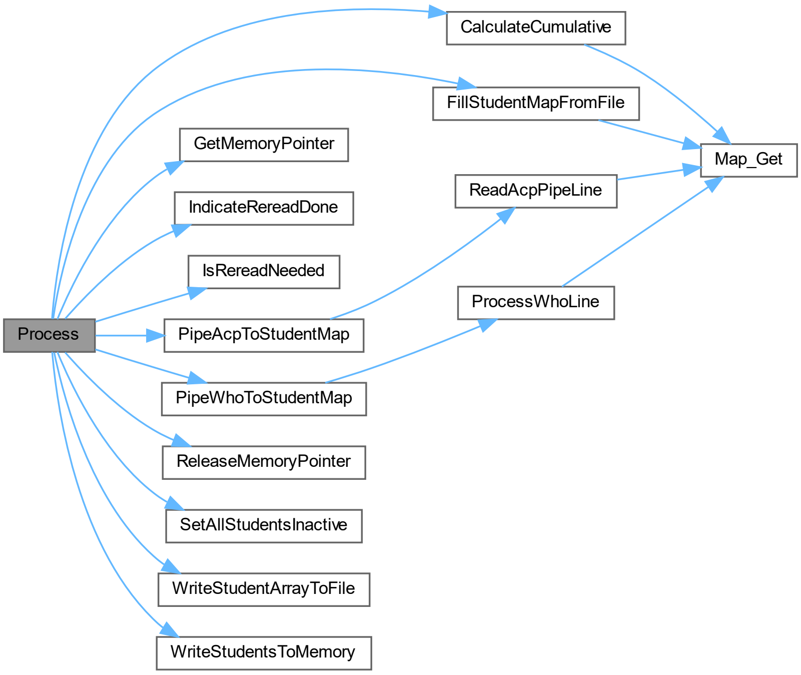
(5) - Rewrites the shared memory.

Parameters

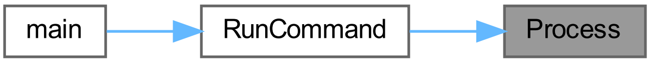
shm↩ _id	The ID of the shared memory segment.

Definition at line 147 of file Process.c.

Here is the call graph for this function:



Here is the caller graph for this function:



6.6.2.8 HelpCommand()

```
void HelpCommand ( )
```

Displays the commands available to the user.

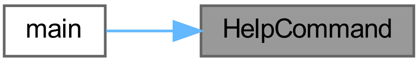
Note

To execute the command, pass "help" as an argument to the program.

This command will also run if arg num is incorrect or if invalid option is entered.

Definition at line 194 of file Process.c.

Here is the caller graph for this function:



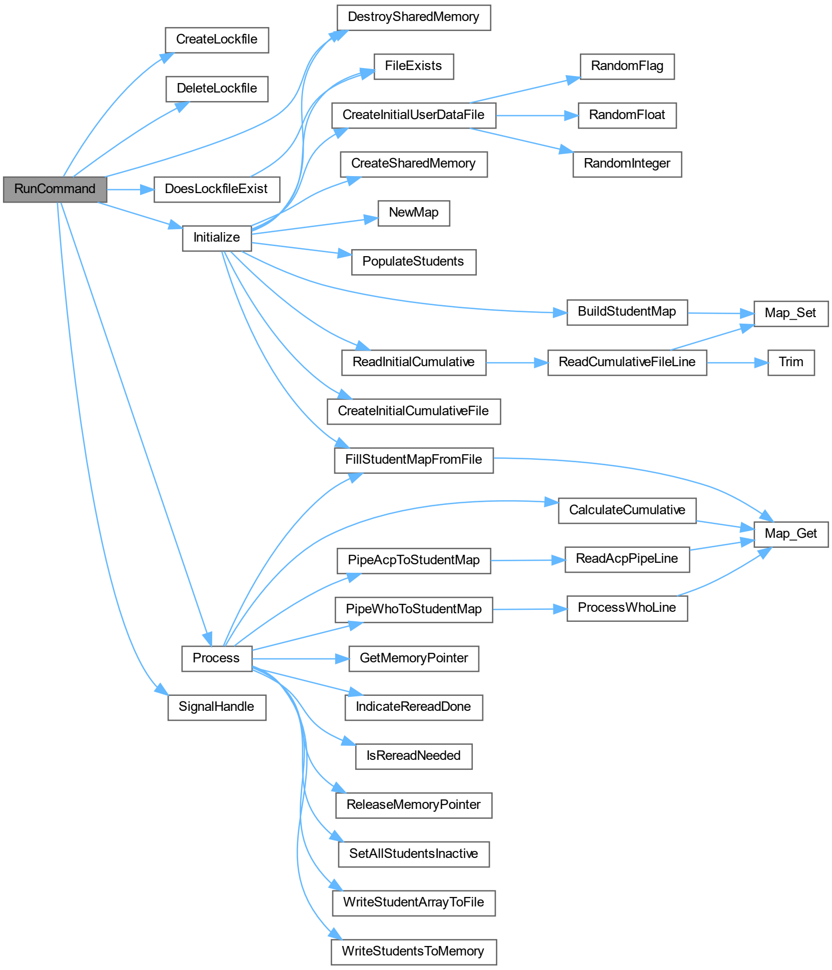
6.6.2.9 RunCommand()

```
void RunCommand ( )
```

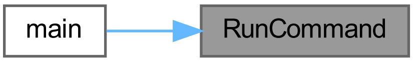
This function begins the Process() loop. It is ultimately called via two cli arguments; "run" and "headless".

Definition at line 205 of file Process.c.

Here is the call graph for this function:



Here is the caller graph for this function:



6.6.2.10 StopCommand()

```
void StopCommand ( )
```

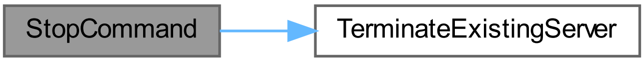
Stops an existing server process if it is running by calling `kill` on the pid stored in the Lockfile.

Note

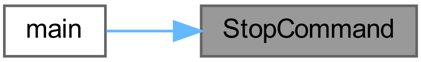
To execute the command, pass "stop" as an argument to the program.

Definition at line 243 of file Process.c.

Here is the call graph for this function:



Here is the caller graph for this function:



6.6.2.11 ResetCommand()

```
void ResetCommand ( )
```

Deletes and recreates the static-user-data file and cumulative login file.

Note

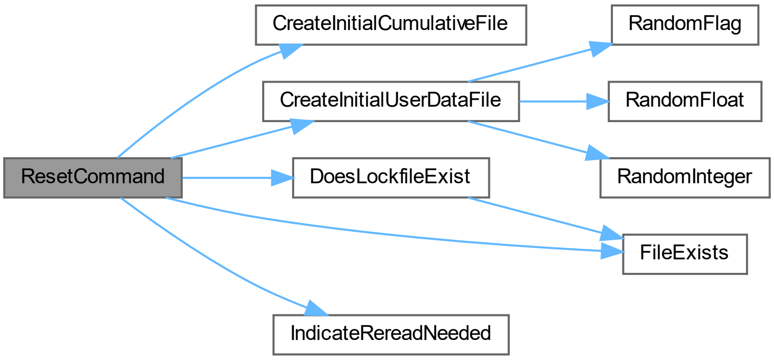
To execute the command, pass "reset" as an argument to the program.

Warning

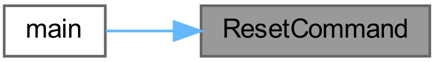
> This will clear login times.

Definition at line 268 of file Process.c.

Here is the call graph for this function:



Here is the caller graph for this function:



6.6.2.12 RunHeadless()

```
void RunHeadless (
            char * processName )
```

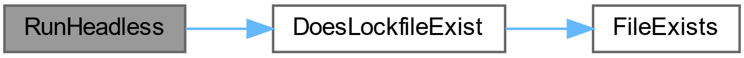
Uses nohup `./{processName} run` to run the process headlessly.

Parameters

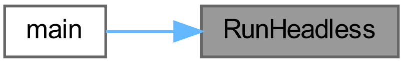
processName	The name of the currently running process, by default, 'server'.

Definition at line 307 of file Process.c.

Here is the call graph for this function:



Here is the caller graph for this function:



6.6.3 Variable Documentation

6.6.3.1 is_stopping

```
short is_stopping = 0
```

If 0, the server is running and looping, re-reading and writing every second. If 1, it is stopping and shutting down.

Definition at line 93 of file Process.c.

6.7 Util

Helper functions.

Functions

- int RandomInteger (int min, int max)
- float RandomFloat (float min, float max)
- short RandomFlag (float percentage_chance)
- void Trim (char ∗string)

6.7.1 Detailed Description

Contains utility functions that are not coupled to any other data or structures in the program. Contains randomization functions.

6.7.2 Function Documentation

6.7.2.1 RandomInteger()

```
int RandomInteger (
            int min,
            int max )
```

Returns an integer between min and max.

Parameters

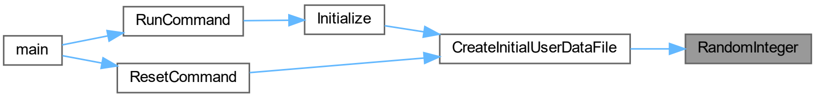
min	The minimum, inclusive.
max	The maximum, inclusive.

Returns

A random integer between min and max.

Definition at line 10 of file util.c.

Here is the caller graph for this function:



6.7.2.2 RandomFloat()

```
float RandomFloat (
            float min,
            float max )
```

Returns a float between min and max.

Parameters

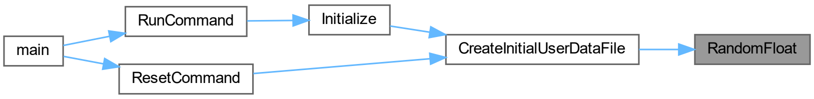
min	The minimum, inclusive.
max	The maximum, inclusive.

Returns

A random integer between min and max.

Definition at line 16 of file util.c.

Here is the caller graph for this function:



6.7.2.3 RandomFlag()

```
short RandomFlag (
             float percentage_chance )
```

Returns 1, percentage_chance of the time.

Parameters

percentage_chance	The chance to return 1.

Note

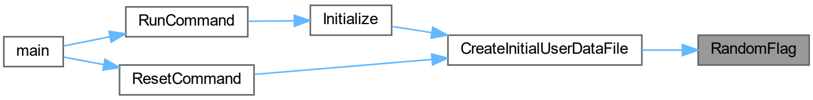
If percentage_chance > 1, this will always return true.

Returns

1 or 0

Definition at line 23 of file util.c.

Here is the caller graph for this function:



6.7.2.4 Trim()

```
void Trim (
             char * string )
```

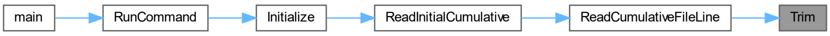
Trims a string by setting the first whitespace character found to the null-terminator.

Parameters

string	The string to trim.

Definition at line 33 of file util.c.

Here is the caller graph for this function:



Chapter 7

Data Structure Documentation

7.1 _map_bucket Struct Reference

map_bucket is an endpoint in the map. It is also a node in a linked list; if there were collisions, then the buckets are appended to the linked list at that location, then traversed until the matching key is found.

7.1.1 Detailed Description

Definition at line 81 of file map.h.

The documentation for this struct was generated from the following file:

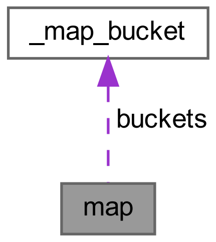
- src/server/map.h

7.2 map Struct Reference

A map. Stores key-value pairs for near constant lookup and insertion time.

```
#include <map.h>
```

Collaboration diagram for map:



7.2.1 Detailed Description

Note

Use NewMap() to create a new map.

Use Map_Set() to set a key in the map.

Use Map_Get() to get a value from the map.

The values stored are of type void pointer.

Definition at line 101 of file map.h.

The documentation for this struct was generated from the following file:

- src/server/map.h

7.3 map_result Struct Reference

The result of a map retrieval.

```
#include <map.h>
```

7.3.1 Detailed Description

Definition at line 111 of file map.h.

The documentation for this struct was generated from the following file:

- src/server/map.h

7.4 Student Struct Reference

The student data type.

```
#include <Data.h>
```

Data Fields

- char **userID** [DATA_ID_MAX_LENGTH]

  The unique user ID.
- char **fullName** [DATA_NAME_MAX_LENGTH]

  The user's full name.
- short **age**

  The user's age (randomized).
- float **gpa**

  The user's gpa (randomized).
- short **active**

  Whether the user is currently logged in (1) or not (0).
- time_t **lastLogin**

  The last time the user logged in.
- int **loginDuration**

  The cumulative time the user has been logged in since the server process started.

7.4.1 Detailed Description

Definition at line 41 of file Data.h.

The documentation for this struct was generated from the following file:

- src/server/Data.h

Chapter 8

File Documentation

8.1 Build.c

```c

#include "Build.h"
#include "memShare.h"
#include "util.h"
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <time.h>

// ~~~~~~~~  Data Structures ~~~~~~~~~

Student *students;

void PopulateStudents(char **studentIDs, char **studentNames, int arsize)
{
    students = malloc(sizeof(Student) * arsize);
    int i;
    for (i = 0; i < arsize; i++)
    {
        strcpy(students[i].userID, studentIDs[i]);
        strcpy(students[i].fullName, studentNames[i]);
    }
}

void BuildStudentMap(map *stmap, Student *studentArr, int studentArrLength)
{
    int i;
    for (i = 0; i < studentArrLength; i++)
    {
        Map_Set(stmap, studentArr[i].userID, (void *)(&studentArr[i]));
    }
}

// ~~~~~~~~  Processing ~~~~~~~~~

short dirty = 1; // start dirty

int PipeWhoToStudentMap(map *stmap)
{
    char command[4] = "who";
    char line[100];
    FILE *fpipe;
    fpipe = popen(command, "r");
    if (fpipe == NULL)
    {
        return -1;
    }

    while (fgets(line, sizeof(line), fpipe) != NULL)
    {
        ProcessWhoLine(stmap, line, strlen(line));
    }
    pclose(fpipe);

    return 0;
}

int ProcessWhoLine(map *stmap, char *whoLine, int whoLineLength)
```

```
{
    char userId[20];
    char dateString[50];
    char timeString[20];
    int read_total = 0;
    int read;
    sscanf(whoLine, " %s %n", userId, &read);
    read_total += read;

    map_result mr = Map_Get(stmap, userId);
    if (!mr.found)
    { // if we can’t find that person in the map, return early
        return -1;
    }
    Student *student = (Student *)mr.data;

    sscanf(whoLine + read_total, " %s %n", dateString, &read); // will be thrown away. eg ‘pts/1’
    read_total += read;
    sscanf(whoLine + read_total, " %s %n", dateString, &read); // read the date string
    read_total += read;
    sscanf(whoLine + read_total, " %s %n", timeString, &read); // read the time string
    strcat(dateString, " ");
    strcat(dateString, timeString); // catenate the time string back to the date string

    time_t now = time(NULL);
    struct tm dtime = *localtime(&now);
    dtime.tm_sec = 0;

    memset(&dtime, 0, sizeof(struct tm));

    sscanf(dateString, "%d-%d-%d %d:%d", &(dtime.tm_year), &(dtime.tm_mon), &(dtime.tm_mday), &(dtime.tm_hour), &(dtime.tm_min));

    dtime.tm_year -= 1900;
    dtime.tm_mon -= 1;
    dtime.tm_hour -= 1;

    time_t parsed_time = mktime(&dtime);

    if (student->lastLogin != parsed_time)
    {
        student->lastLogin = parsed_time;
        dirty = 1;
    }
    student->active = 1;
    return 0;
}

void SetAllStudentsInactive(Student *stud_arr, int arr_len)
{
    int i;
    for (i = 0; i < arr_len; i++)
    {
        stud_arr[i].active = 0;
    }
}

void WriteStudentsToMemory(void *mem_ptr, Student *stud_arr, int arr_len)
{
    Student *memloc = (Student *)mem_ptr;
    int i;
    for (i = 0; i < arr_len; i++)
    {
        strcpy(memloc[i].userID, stud_arr[i].userID);
        strcpy(memloc[i].fullName, stud_arr[i].fullName);
        memloc[i].age = stud_arr[i].age;
        memloc[i].gpa = stud_arr[i].gpa;
        memloc[i].active = stud_arr[i].active;
        memloc[i].lastLogin = stud_arr[i].lastLogin;
        memloc[i].loginDuration = stud_arr[i].loginDuration;
    }
}

// ~~~~~~~~  Cumulative Processing ~~~~~~~~~

int ReadInitialCumulative(map *time_map, char *filename)
{
    FILE *file = fopen(filename, "r");
    char line[100];
    if (file == NULL)
    {
        return -1;
    }
    while (fgets(line, sizeof(line), file) != NULL)
    {
        ReadCumulativeFileLine(time_map, line);
    }
```

```c

    fclose(file);
    return 0;
}

int PipeAcpToStudentMap(map *st_map)
{
    char command[6] = "ac -p";
    char line[300];
    FILE *fpipe;
    fpipe = popen(command, "r");
    if (fpipe == NULL)
    {
        return -1;
    }
    int err;
    while (fgets(line, sizeof(line), fpipe) != NULL)
    {
        err = ReadAcpPipeLine(st_map, line);
        if (err)
        {
            printf("\nError %d reading acp pipeline.", err);
            break;
        }
    }
    pclose(fpipe);
    return 0;
}

void ReadCumulativeFileLine(map *time_map, char *acp_line)
{
    char userId[20];
    float hours;
    sscanf(acp_line, " %s %f ", userId, &hours);
    long seconds = (long)(hours * 60 *60);
    // if(strcmp(userId, "mil7233") == 0) {
    //     printf("Cum file line for %s seconds = %ld\n", userId, seconds);
    // }
    Trim(userId);
    char* key = malloc( (strlen(userId)+1) * sizeof(char));
    strcpy(key, userId);
    Map_Set(time_map, userId, (void *)seconds);
}

int ReadAcpPipeLine(map *stmap, char *acp_line)
{
    if (acp_line == NULL || strlen(acp_line) < 1)
    {
        return -1;
    }
    char userId[40];
    float hours;
    sscanf(acp_line, "%s %f", userId, &hours);
    map_result result = Map_Get(stmap, userId);
    if (result.found)
    {
        Student *student = (Student *)result.data;
        int seconds = (int)(hours * 60*60);
        student->loginDuration = seconds;
        // if(strcmp(userId, "mil7233")==0) {
        //     printf("ACP pipe for %s student quant = %f\n", userId, hours);
        //     printf(" --- int seconds = %d\n", seconds);
        // }
    }
    return 0;
}

void CalculateCumulative(Student *stud_arr, int stud_arr_len, map *time_map)
{
    int i;
    for (i = 0; i < stud_arr_len; i++)
    {
        map_result result = Map_Get(time_map, stud_arr[i].userID);
        if (result.found)
        {

            long time_at_server_start = (long)result.data;
            stud_arr[i].loginDuration = stud_arr[i].loginDuration - time_at_server_start;
            // if(strcmp("mil7233", stud_arr->userID) == 0) {
            //     printf("calc cum: found user %s.\n", stud_arr[i].userID);
            //     printf(" tot time now: %ld\n", stud_arr[i].loginDuration);
            //     printf(" time at server start: %ld\n", time_at_server_start);
            //     printf(" new duration: %ld\n", stud_arr[i].loginDuration);
            // }
        }
    }
}
```

8.2 Build.h

```
#ifndef BUILD_H
#define BUILD_H
#include "Data.h"
#include "map.h"

// ~~~~~~~~  Data Structures ~~~~~~~~~

extern Student *students;

void PopulateStudents(char **studentIDs, char **studentNames, int arsize);

void BuildStudentMap(map *stmap, Student *studentArr, int studentArrLength);

// ~~~~~~~~  Processing ~~~~~~~~~

extern short dirty;

int PipeWhoToStudentMap(map *stmap);

int ProcessWhoLine(map *stmap, char *whoLine, int whoLineLength);

void SetAllStudentsInactive(Student *stud_arr, int arr_len);

void WriteStudentsToMemory(void *mem_ptr, Student *stud_arr, int arr_len);

// ~~~~~~~~  Cumulative Processing ~~~~~~~~~

int ReadInitialCumulative(map *time_map, char *filename);

int PipeAcpToStudentMap(map *st_map);

void ReadCumulativeFileLine(map *time_map, char *acp_line);

int ReadAcpPipeLine(map *stmap, char *acp_line);

void CalculateCumulative(Student *stud_arr, int stud_arr_len, map *time_map);

#endif
```

8.3 Data.c

```

#include "Data.h"

char *Data_IDs[DATA_NUM_RECORDS] = {
    "chen",
    "bea1389",
    "bol4559",
    "cal6258",
    "kre5277",
    "lon1150",
    "mas9309",
    "mes08346",
    "mil7233",
    "nef9476",
    "nov7488",
    "pan9725",
    "rac3146",
    "rub4133",
    "shr5683",
    "vay3083",
    "yos2327"};

char *Data_Names[DATA_NUM_RECORDS] = {
    "Weifeng Chen",
    "Christian Beatty",
    "Emily Bolles",
    "Cameron Calhoun",
    "Ty Kress",
    "Cody Long",
    "Caleb Massey",
    "Christian Messmer",
    "Karl Miller",
    "Jeremiah Neff",
    "Kaitlyn Novacek",
    "Joshua Panaro",
    "Caleb Rachocki",
    "Caleb Ruby",
    "Paul Shriner",
    "Alan Vayansky",
    "Assefa Ayalew Yoseph"};

```

8.4 Data.h

```
#ifndef Data_h
#define Data_h
#include <time.h>
#include <sys/types.h>

#define DATA_NUM_RECORDS 17
#define DATA_ID_MAX_LENGTH 9
#define DATA_NAME_MAX_LENGTH 21

/* Constant, all user IDs. */
extern char *Data_IDs[];

extern char *Data_Names[];

#define DATA_SIZE 56

typedef struct
{
    char userID[DATA_ID_MAX_LENGTH];
    char fullName[DATA_NAME_MAX_LENGTH];
    short age;
    float gpa;
    short active;
    time_t lastLogin;
    int loginDuration;
} Student;
#endif
```

8.5 Files.c

```

#include "Files.h"
#include "util.h"
#include <stdlib.h>
#include <stdio.h>
#include <strings.h>
#include <unistd.h>

short FileExists(char *file_name_to_check)
{
    FILE *file = fopen(file_name_to_check, "r");
    short result = 1;
    if (file == NULL)
    {
        result = 0;
    }
    else
    {
        fclose(file);
    }
    return result;
}

int CreateInitialUserDataFile(char *file_name, char **id_list, int id_list_len)
{
    FILE *file = fopen(file_name, "w");
    if (file == NULL)
    {
        return -1;
    }
    int i;
    for (i = 0; i < id_list_len; i++)
    {
        int rand_age = RandomInteger(18, 22);
        float gpa;
        if (RandomFlag(0.42))
        {
            gpa = 4.0; // 42% of the time, make the GPA 4.0
        }
        else
        {
            gpa = RandomFloat(2.5, 4.0);
        }
        fprintf(file, "%s\t%d\t%.2f\t%d\n", id_list[i], rand_age, gpa, 0);
    }
    fclose(file);
    return 0;
}

int FillStudentMapFromFile(map *student_map, char *file_name, char **id_list, int id_list_len)
{
    FILE *file = fopen(file_name, "r");
    if (file == NULL)
    {
        return -1;
    }
    // id buffer
    char user_id[9];
    int age;
    float gpa;
    long time;
    while (fscanf(file, "%9s\t%d\t%f\t%ld", user_id, &age, &gpa, &time) == 4)
    {
        map_result result = Map_Get(student_map, user_id);
        if (result.found == 0)
        {
            continue;
        }
        ((Student *)result.data)->age = age;
        ((Student *)result.data)->gpa = gpa;
        ((Student *)result.data)->lastLogin = time;
    }
    fclose(file);
    return 0;
}

int WriteStudentArrayToFile(Student *students, int arr_len, char *file_name)
{
    FILE *file = fopen(file_name, "w");
    if (file == NULL)
    {
        return -1;
    }
    int i;
    for (i = 0; i < arr_len; i++)
    {
        fprintf(file, "%s\t%d\t%.2f\t%ld\n", students[i].userID, students[i].age, students[i].gpa, students[i].lastLogin);
    }
    fclose(file);
    return 0;
}

int CreateInitialCumulativeFile(char *file_name)
{
    FILE *file = fopen(file_name, "w");
    if (file == NULL)
    {
        return -1;
    }
    FILE *pipe = popen("ac -p", "r");
    if (pipe == NULL)
    {
        fclose(file);
        return -2;
    }

    char line[100];
    while (fgets(line, sizeof(line), pipe) != NULL)
    {
        fputs(line, file);
    }
    pclose(pipe);
    fclose(file);
    return 0;
}

// ~~~~~~~~~~~~~~~ Lockfile Commands ~~~~~~~~~~~~~~~~~~~~~

short DoesLockfileExist()
{
    return FileExists(LOCKFILE);
}

int CreateLockfile()
{
    FILE *file = fopen(LOCKFILE, "w");
    if (file == NULL)
    {
        return -1;
    }
    fprintf(file, "0 %d", getpid());
    fclose(file);
    return 0;
}

int DeleteLockfile()
{
```

```
    return remove(LOCKFILE);
}
```

8.6 Files.h

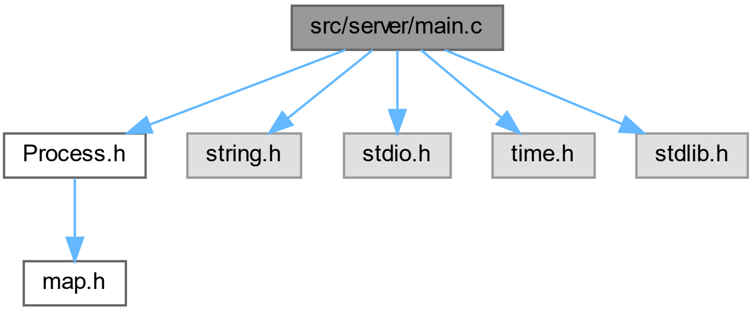
```
#ifndef Files_H
#define Files_H
#include "Data.h"
#include "map.h"

#define STATIC_USER_DATA_FILE "static-user-data.txt"

#define STATIC_USER_CUMULATIVE_FILE "static-user-cumulative-start.txt"

#define LOCKFILE "/tmp/ecet-server.lock"

short FileExists(char *file_name_to_check);

int CreateInitialUserDataFile(char *file_name, char **id_list, int id_list_len);

int WriteStudentArrayToFile(Student *students, int arr_len, char *file_name);

int FillStudentMapFromFile(map *student_map, char *file_name, char **id_list, int id_list_len);

int CreateInitialCumulativeFile(char *file_name);



// ~~~~~~~~~~~~~~~ Lockfile Commands ~~~~~~~~~~~~~~~~~~~~~

short DoesLockfileExist();

int CreateLockfile();

int DeleteLockfile();
#endif
```

8.7 src/server/main.c File Reference

Program entry point.

```
#include "Process.h"
#include <string.h>
#include <stdio.h>
#include <time.h>
#include <stdlib.h>
```

Include dependency graph for main.c:



Functions

- int main (int argc, char **argv)

  Program entry.

8.7.1 Function Documentation

8.7.1.1 main()

```
int main (
            int argc,
            char ** argv )
```

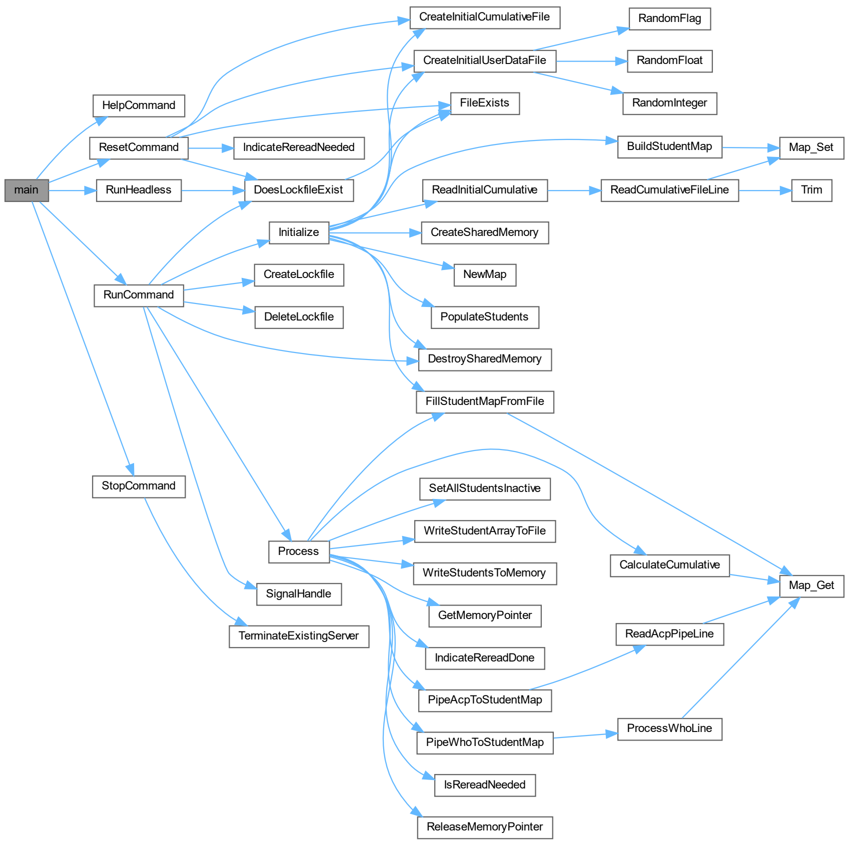
Parses arguments and calls the appropriate Process.h function.

Parameters

argc	The argument count.
argv	The argument values.

Definition at line 104 of file main.c.

Here is the call graph for this function:



8.8 main.c

Go to the documentation of this file.

```

#include "Process.h"
#include <string.h>
#include <stdio.h>
#include <time.h>
#include <stdlib.h>
int main(int argc, char **argv)
{
    srand(time(NULL)); // seed the randomizer

    if (argc <= 1 || argc >= 3)
    {
        printf("Too few or many options!\n");
        HelpCommand();
    }
    else if (strcmp(argv[1], "help") == 0)
    {
        HelpCommand();
    }
    else if (strcmp(argv[1], "reset") == 0)
    {
        ResetCommand();
    }
    else if (strcmp(argv[1], "stop") == 0 || strcmp(argv[1], "end") == 0 || strcmp(argv[1], "close") == 0 || strcmp(argv[1], "exit") == 0)
    {
        StopCommand();
    }
    else if (strcmp(argv[1], "headless") == 0)
    {
        RunHeadless(argv[0]);
    }
    else if (strcmp(argv[1], "run") == 0 || strcmp(argv[1], "start") == 0)
    {
        RunCommand();
    }
    else
    {
        printf("Unknown option!\n");
        HelpCommand();
    }
    return 0;
}
```

8.9 map.c

```

#include "stdlib.h"
#include "string.h"
#include "map.h"
#include "math.h"

int hash_log2(int num_to_log)
{
    int t = 1;
    int i = 0;
    do
    {
        num_to_log = num_to_log & ~t;
        t = t « 1;
        i++;
    } while (num_to_log > 0);
    return i;
}

int hash_upperLimit(int bitsize)
{
    return 1 « bitsize;
}

int char_ratio = (int)(sizeof(int) / sizeof(char));

int hash_string(int hash_table_size, char *string, int strlen)
{
    int i, hash = 2166136261;
    for (i = 0; i < strlen; i += 1)
    {
        hash *= 16777619;
        hash ^= string[i];
    }
    if (hash < 0)
    {
        hash *= -1;
    }
    return hash % hash_table_size;
}

map *NewMap(int capacity)
{
    int log2 = hash_log2(capacity);
    int capac = hash_upperLimit(log2);
    int sz = sizeof(struct _map_bucket) * capac;
    struct _map_bucket *buckets = malloc(sz);
    memset(buckets, 0, sz);
    int i;
    for (i = 0; i < capac; i++)
    {
        buckets[i] = (struct _map_bucket){NULL, NULL, NULL};
    }
    map newm = (map){capac, buckets};
    map *map_p = malloc(sizeof(map));
    *map_p = newm;
    return map_p;
}

void _bucket_insert(struct _map_bucket *bucket, char *key, void *value)
{
    struct _map_bucket *check = bucket;
    while (check->key != NULL)
    {
        if (strcmp(check->key, key) == 0)
        {
            check->data = value;
            return;
        }
        if (check->next == NULL)
        {
            check->next = malloc(sizeof(struct _map_bucket));
            *(check->next) = (struct _map_bucket){NULL, NULL, NULL};
        }
        check = check->next;
    }
    check->key = key;
    check->data = value;
}

void Map_Set(map *a_map, char *key, void *value)
{
    int keyl = (int)strlen(key);
    int hash = hash_string(a_map->size, key, keyl);
    _bucket_insert(&(a_map->buckets[hash]), key, value);
}
void _bucket_get(struct _map_bucket *bucket, char *key, map_result *result)
{
    struct _map_bucket *check = bucket;
    while (check->key != NULL)
    {
        if (strcmp(check->key, key) == 0)
        {
            result->found = 1;
            result->data = check->data;
            return;
        }
        else if (check->next != NULL)
        {
            check = check->next;
        }
        else
        {
            result->found = 0;
            break;
        }
    }
}

map_result Map_Get(map *a_map, char *key)
{
    map_result res = (map_result){0, NULL};
    int keyl = (int)strlen(key);
    int hash = hash_string(a_map->size, key, keyl);
    _bucket_get(&(a_map->buckets[hash]), key, &res);
    return res;
}

void _bucket_delete(struct _map_bucket *bucket, char *key, short free_it, map_result *result)
{
    struct _map_bucket *last = bucket;
    struct _map_bucket *next = bucket->next;
    while (next != NULL)
    {
        if (strcmp(next->key, key) == 0)
        {
            result->found = 1;
            result->data = next->data;
            if (free_it)
            {
                free(next->data);
                result->data = NULL;
            }
            last->next = next->next;
            free(next);
        }
        else
        {
            last = next;
            next = next->next;
        }
    }
}

map_result Map_Delete(map *a_map, char *key, short free_it)
{
    map_result res = (map_result){0, NULL};
    int keyl = (int)strlen(key);
    int hash = hash_string(a_map->size, key, keyl);

    struct _map_bucket top = a_map->buckets[hash];
    if (top.key == NULL)
    {
        return res;
    }
    if (strcmp(top.key, key) == 0)
    {
        res.found = 1;
        res.data = top.data;
        if (free_it)
        {
            free(top.data);
            res.data = NULL;
        }
        if (top.next != NULL)
        {
            a_map->buckets[hash] = *(top.next);
            free(top.next);
        }
        else
        {
            a_map->buckets[hash] = (struct _map_bucket){NULL, NULL, NULL};
        }
        return res;
    }
    if (top.next == NULL)
    {
        return res;
    }
    _bucket_delete(&(a_map->buckets[hash]), key, free_it, &res);

    return res;
}
```

8.10 map.h

```
#ifndef map_h
#define map_h

// ------------------------------
//          Hashing Math
// ------------------------------

int hash_log2(int number_to_log);

int hash_string(int hash_table_capacity, char *string, int strlen);

int hash_upperLimit(int bitsize);

// -------------------------------------
//          General Map Operations
// -------------------------------------

struct _map_bucket
{
    char *key;
    void *data;
    struct _map_bucket *next;
};

typedef struct
{
    int size;
    struct _map_bucket *buckets;
} map;

typedef struct
{

    short found;
    void *data;
} map_result;

map *NewMap(int capacity);

void Map_Set(map *a_map, char *key, void *value);

map_result Map_Get(map *a_map, char *key);

map_result Map_Delete(map *a_map, char *key, short free_it);

#endif
```

8.11 memShare.c

```
#include "memShare.h"
#include <string.h>
#include <stdio.h>

int CreateSharedMemory()
{
    return shmget(MEM_KEY, MEM_SIZE, IPC_CREAT | MEM_PERMISSIONS);
}

int DestroySharedMemory()
{
    int shm_id = shmget(MEM_KEY, MEM_SIZE, 0);
    int control_result = shmctl(shm_id, IPC_RMID, 0);
    if (control_result != -1)
        return 0;
    return control_result;
}

void *GetMemoryPointer(int shared_mem_id)
{
    return shmat(shared_mem_id, NULL, 0);
}

int ReleaseMemoryPointer(void *shmaddr)
{
    return shmdt(shmaddr);
}
```

8.12 memShare.h

```
#ifndef MEM_SHARE_H
#define MEM_SHARE_H
#include <sys/shm.h>
#include <sys/ipc.h>
#include "Data.h"

#define MEM_KEY 0x727

#define MEM_PERMISSIONS 0664

#define MEM_SIZE DATA_SIZE *DATA_NUM_RECORDS

int CreateSharedMemory();

int DestroySharedMemory();

void *GetMemoryPointer(int shared_mem_id);

int ReleaseMemoryPointer(void *shmaddr);
#endif
```

8.13 Process.c

```
#include "Process.h"
#include "Files.h"
#include "Data.h"
#include "Build.h"
#include "memShare.h"
#include <errno.h>
#include <stdio.h>
#include <unistd.h>
#include <signal.h>
#include <stdlib.h>
#include <string.h>

map *student_map;
map *initial_cumulative_times;

int TerminateExistingServer()
{
    FILE *file = fopen(LOCKFILE, "r");
    if (file == NULL)
    {
        return -1;
    }
    int need_rewrite;
    int pid = 0;
    fscanf(file, "%d %d", &need_rewrite, &pid);
    fclose(file);
    if (pid > 0)
    {
        return kill(pid, SIGTERM);
    }
    return -2;
}

int IndicateRereadNeeded()
{
    FILE *file = fopen(LOCKFILE, "r+");
    if (file == NULL)
    {
        return -1;
    }
    int err = 0;
    err = fseek(file, 0, SEEK_SET);
    if (!err)
    {
        fputc('1', file);
    }
    err = fclose(file);
    return err;
}

int IndicateRereadDone()
{
    FILE *file = fopen(LOCKFILE, "r+");
    if (file == NULL)
    {
        return -1;
    }
    int err = 0;
    err = fseek(file, 0, SEEK_SET);
    if (!err)
    {
        fputc('0', file);
    }
    err = fclose(file);
    return err;
}

short IsRereadNeeded()
{
    FILE *file = fopen(LOCKFILE, "r");
    char firstc = fgetc(file);
    fclose(file);
    return firstc == '1';
}

void SignalHandle(int signo)
{
    printf("Received shutdown signal.\n");
    if (signo == SIGINT || signo == SIGTERM)
    {
        is_stopping = 1;
    }
    // possible feature: add a timeout terminate emergency exit (with graceful shutdown)

}

short is_stopping = 0;

// ~~~~~~~~~~~~~~~ CLI Commands ~~~~~~~~~~~~~~~~~~~~~~~~~~

int Initialize()
{
    int err;
    if (!FileExists(STATIC_USER_DATA_FILE))
    {
        printf("%s does not exist. Creating.\n", STATIC_USER_DATA_FILE);
        err = CreateInitialUserDataFile(STATIC_USER_DATA_FILE, Data_IDs, DATA_NUM_RECORDS);
        if (err)
        {
            printf("Problem creating %s!\n", STATIC_USER_DATA_FILE);
        }
    }
    if (!FileExists(STATIC_USER_CUMULATIVE_FILE))
    {
        printf("%s does not exist. Creating.\n", STATIC_USER_CUMULATIVE_FILE);
        err = CreateInitialCumulativeFile(STATIC_USER_CUMULATIVE_FILE);
        if (err)
        {
            printf("Problem creating %s!\n", STATIC_USER_CUMULATIVE_FILE);
```

```
        }
    }
    PopulateStudents(Data_IDs, Data_Names, DATA_NUM_RECORDS);
    student_map = NewMap(50);
    BuildStudentMap(student_map, students, DATA_NUM_RECORDS);
    err = FillStudentMapFromFile(student_map, STATIC_USER_DATA_FILE, Data_IDs, DATA_NUM_RECORDS);
    if (err)
    {
        printf("Problem filling student map from %s!\n", STATIC_USER_DATA_FILE);
    }
    printf("Student data retrieved from file.\n");

    initial_cumulative_times = NewMap(50);
    err = ReadInitialCumulative(initial_cumulative_times, STATIC_USER_CUMULATIVE_FILE);
    if (err)
    {
        printf("Failed to read %s. Cumulative times may be wrong!", STATIC_USER_CUMULATIVE_FILE);
    }

    dirty = 0;

    int shmid = CreateSharedMemory();
    if (shmid == -1)
    {
        DestroySharedMemory();
        shmid = CreateSharedMemory();
    }
    printf("Shared memory allocated.\n");
    return shmid;
}

void Process(int shm_id)
{
    if (IsRereadNeeded())
    {
        printf("\nReread indicated - rechecking user data file.");
        FillStudentMapFromFile(student_map, STATIC_USER_DATA_FILE, Data_IDs, DATA_NUM_RECORDS);
        IndicateRereadDone();
    }
    SetAllStudentsInactive(students, DATA_NUM_RECORDS);
    int err = PipeAcpToStudentMap(student_map);
    if (err)
    {
        printf("Error piping ac -p command! \n");
    }
    else
    {
        CalculateCumulative(students, DATA_NUM_RECORDS, initial_cumulative_times);
    }
    err = PipeWhoToStudentMap(student_map);
    if (err)
    {
        perror("Error updating from who!");
    }
    if (dirty)
    {
        err = WriteStudentArrayToFile(students, DATA_NUM_RECORDS, STATIC_USER_DATA_FILE);
        if (err)
        {
            printf("\nError updating %s!", STATIC_USER_DATA_FILE);
        }
        else
        {
            dirty = 0;
        }
    }
    void *ptr = GetMemoryPointer(shm_id);
    if (ptr == (void *)-1)
    {
        perror("Error attaching to shared memory");
    }
    else
    {
        WriteStudentsToMemory(ptr, students, DATA_NUM_RECORDS);
        ReleaseMemoryPointer(ptr);
    }
}

void HelpCommand()
{
    printf("\nUsage: server [OPTION]\n\n");
    printf("Options: \n");
    printf("\thelp\t\t\tShows the possible program commands\n");
    printf("\treset\t\t\tRegenerates the user data file\n");
    printf("\tstop\t\t\tStops an existing server process if it is running\n");
    printf("\trun\t\t\tCreates a new server with output to the shell if a server isn't already running.\n");
    printf("\theadless\t\tCreates a new headless server if a server isn’t already running.\n\n");
}

void RunCommand()
{
    printf("\nRunning server.\n");
    if (DoesLockfileExist())
    {
        printf("\nServer is already running. Run ’server stop’ to shut it down first.\n");
        return;
    }
    int err = CreateLockfile();
    if (err)
    {
        printf("\nFailed to create lockfile! Exiting.\n");
        return;
    }
    int shm_id = Initialize();
    signal(SIGTERM, SignalHandle);
    signal(SIGINT, SignalHandle);
    printf("Server started.\n");
    fflush(stdout);
    while (!is_stopping)
    {
        Process(shm_id);
        sleep(1);
    }
    printf("Server shutting down.\n");
    err = DeleteLockfile();
    if (err)
    {
        printf("Failed to delete lockfile!\n");
    }
    err = DestroySharedMemory();
    if (err)
    {
        printf("Failed to destroy shared memory!\n");
    }
    printf("Server terminated.\n");
}

void StopCommand()
{
    printf("\nStopping server...\n");
    int err = TerminateExistingServer();
    if (err != 0)
    {
        if (err == -1)
        {
            printf("Server isn’t running.\n");
        }
        else if (err == -2)
        {
            printf("Lockfile did not contain a valid process id!\n");
        }
        else
        {
            printf("Sending terminate signal failed!\n");
        }
    }
    else
    {
        printf("Server terminated.\n");
    }
}

void ResetCommand()
{
    int err;

    if (FileExists(STATIC_USER_DATA_FILE))
    {
        printf("User data file exists. Deleting...\n");
        remove(STATIC_USER_DATA_FILE);
    }

    printf("Creating new data file.\n");
    err = CreateInitialUserDataFile(STATIC_USER_DATA_FILE, Data_IDs, DATA_NUM_RECORDS);
    if (err)
    {
        printf("Problem creating %s!\n", STATIC_USER_DATA_FILE);
    }
    else
    {
        printf("%s created.\n", STATIC_USER_DATA_FILE);
    }

    printf("Creating new cumulative file.\n");
    err = CreateInitialCumulativeFile(STATIC_USER_CUMULATIVE_FILE);
    if (err)
    {
        printf("Problem creating %s!\n", STATIC_USER_CUMULATIVE_FILE);
    }
    else
    {
        printf("%s created.\n", STATIC_USER_CUMULATIVE_FILE);
    }

    if (DoesLockfileExist())
    {
        printf("Indicated re-read to running server process.\n");
        IndicateRereadNeeded();
    }
}

void RunHeadless(char *processName)
{
    if (DoesLockfileExist())
    {
        printf("Server process already running.\n");
        return;
    }
    char commandFront[] = " nohup ";
    char commandEnd[] = " run & exit";
    size_t comm_length = strlen(commandFront) + strlen(commandEnd) + strlen(processName) + 1;
    char *commandFull = malloc(comm_length * sizeof(char));
    memset(commandFull, 0, comm_length * sizeof(char));
    strcpy(commandFull, commandFront);
    strcat(commandFull, processName);
    strcat(commandFull, commandEnd);

    printf("Executing: %s\n", commandFull);
    popen(commandFull, "we");
    printf("Server running headlessly.\n");
}
```

8.14 Process.h

```
#ifndef Process_h
#define Process_h
#include "map.h"

int TerminateExistingServer();

int IndicateRereadNeeded();

int IndicateRereadDone();

short IsRereadNeeded();

void SignalHandle(int signo);

extern short is_stopping;

// ~~~~~~~~~~~~~~~ CLI Commands ~~~~~~~~~~~~~~~~~~~~~~~~~~

extern map *student_map;

int Initialize();

void Process(int shm_id);

void ResetCommand();

void StopCommand();

void RunCommand();

void HelpCommand();

void RunHeadless(char *processName);
#endif
```

8.15 util.c

```

```

```
#include "util.h"

#include <stdlib.h>
#include <string.h>

int RandomInteger(int min, int max)
{
    int r_add = rand() % (max - min + 1);
    return r_add + min;
}

float RandomFloat(float min, float max)
{
    float dif = max - min;
    int rand_int = rand() % (int)(dif * 10000);
    return min + (float)rand_int / 10000.0;
}

short RandomFlag(float percentage_chance)
{
    float random_value = (float)rand() / RAND_MAX;
    if (random_value < percentage_chance)
    {
        return 1;
    }
    return 0;
}

void Trim(char * string)
{
    size_t len = strlen(string);
    int i;
    for(i = 0; i < len; i++)
    {
        if(string[i] == ’ ’ || string[i] == ’\t’ || string[i] == ’\n’)
        {
            string[i] = ’\0’;
            break;
        }
    }
}

```

8.16 util.h

```
#ifndef util_h
#define util_h
int RandomInteger(int min, int max);

float RandomFloat(float min, float max);

short RandomFlag(float percentage_chance);

void Trim(char * string);
#endif
```

Index

_map_bucket, 53

Build, 11
    BuildStudentMap, 12
    CalculateCumulative, 19
    dirty, 20
    PipeAcpToStudentMap, 17
    PipeWhoToStudentMap, 13
    PopulateStudents, 11
    ProcessWhoLine, 14
    ReadAcpPipeLine, 18
    ReadCumulativeFileLine, 18
    ReadInitialCumulative, 16
    SetAllStudentsInactive, 15
    students, 20
    WriteStudentsToMemory, 15
BuildStudentMap
    Build, 12

CalculateCumulative
    Build, 19
CreateInitialCumulativeFile
    Files, 28
CreateInitialUserDataFile
    Files, 25
CreateLockfile
    Files, 29
CreateSharedMemory
    MemShare, 35

Data, 21
    DATA_ID_MAX_LENGTH, 21
    DATA_NAME_MAX_LENGTH, 22
    Data_Names, 22
    DATA_NUM_RECORDS, 21
    DATA_SIZE, 22
DATA_ID_MAX_LENGTH
    Data, 21
DATA_NAME_MAX_LENGTH
    Data, 22
Data_Names
    Data, 22
DATA_NUM_RECORDS
    Data, 21
DATA_SIZE
    Data, 22
DeleteLockfile
    Files, 30
DestroySharedMemory
    MemShare, 35
dirty
    Build, 20
DoesLockfileExist
    Files, 28

FileExists
    Files, 25
Files, 23
    CreateInitialCumulativeFile, 28
    CreateInitialUserDataFile, 25
    CreateLockfile, 29
    DeleteLockfile, 30
    DoesLockfileExist, 28
    FileExists, 25
    FillStudentMapFromFile, 26
    LOCKFILE, 24
    STATIC_USER_CUMULATIVE_FILE, 24
    STATIC_USER_DATA_FILE, 24
    WriteStudentArrayToFile, 27
FillStudentMapFromFile
    Files, 26

GetMemoryPointer
    MemShare, 36

HelpCommand
    Process, 43

IndicateRereadDone
    Process, 39
IndicateRereadNeeded
    Process, 38
Initialize
    Process, 41
is_stopping
    Process, 48
IsRereadNeeded
    Process, 39

LOCKFILE
    Files, 24

main
    main.c, 64
main.c
    main, 64
Map, 30
    Map_Delete, 33
    Map_Get, 33
    Map_Set, 32
    NewMap, 32

map, 53
Map_Delete
    Map, 33
Map_Get
    Map, 33
map_result, 54
Map_Set
    Map, 32
MEM_KEY
    MemShare, 34
MEM_PERMISSIONS
    MemShare, 35
MEM_SIZE
    MemShare, 35
MemShare, 34
    CreateSharedMemory, 35
    DestroySharedMemory, 35
    GetMemoryPointer, 36
    MEM_KEY, 34
    MEM_PERMISSIONS, 35
    MEM_SIZE, 35
    ReleaseMemoryPointer, 37

NewMap
    Map, 32

PipeAcpToStudentMap
    Build, 17
PipeWhoToStudentMap
    Build, 13
PopulateStudents
    Build, 11
Process, 37
    HelpCommand, 43
    IndicateRereadDone, 39
    IndicateRereadNeeded, 38
    Initialize, 41
    is_stopping, 48
    IsRereadNeeded, 39
    Process, 42
    ResetCommand, 46
    RunCommand, 44
    RunHeadless, 47
    SignalHandle, 40
    StopCommand, 45
    TerminateExistingServer, 38
ProcessWhoLine
    Build, 14

RandomFlag
    Util, 50
RandomFloat
    Util, 49
RandomInteger
    Util, 49
ReadAcpPipeLine
    Build, 18
ReadCumulativeFileLine
    Build, 18
ReadInitialCumulative
    Build, 16
ReleaseMemoryPointer
    MemShare, 37
ResetCommand
    Process, 46
RunCommand
    Process, 44
RunHeadless
    Process, 47

SetAllStudentsInactive
    Build, 15
SignalHandle
    Process, 40
src/server/Build.c, 57
src/server/Build.h, 60
src/server/Data.c, 60
src/server/Data.h, 61
src/server/Files.c, 61
src/server/Files.h, 63
src/server/main.c, 63, 65
src/server/map.c, 66
src/server/map.h, 68
src/server/memShare.c, 69
src/server/memShare.h, 69
src/server/Process.c, 69
src/server/Process.h, 73
src/server/util.c, 73
src/server/util.h, 74
STATIC_USER_CUMULATIVE_FILE
    Files, 24
STATIC_USER_DATA_FILE
    Files, 24
StopCommand
    Process, 45
Student, 54
students
    Build, 20

TerminateExistingServer
    Process, 38
Trim
    Util, 51

Util, 48
    RandomFlag, 50
    RandomFloat, 49
    RandomInteger, 49
    Trim, 51

WriteStudentArrayToFile
    Files, 27
WriteStudentsToMemory
    Build, 15