ecet4640-lab4

1.0

Generated by Doxygen 1.9.6

1 ecet4640-lab4	1
	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 1	1
6.1 Build	1
6.1.1 Detailed Description	1
6.1.2 Function Documentation	1
6.1.2.1 PopulateStudents()	1
6.1.2.2 BuildStudentMap()	2
6.1.2.3 PipeWhoToStudentMap()	3
6.1.2.4 ProcessWhoLine()	4
6.1.2.5 SetAllStudentsInactive()	5
6.1.2.6 WriteStudentsToMemory()	5
6.1.2.7 ReadInitialCumulative()	6
6.1.2.8 PipeAcpToStudentMap()	7
6.1.2.9 ReadCumulativeFileLine()	8
6.1.2.10 ReadAcpPipeLine()	9
6.1.2.11 CalculateCumulative()	9
6.1.3 Variable Documentation	0
6.1.3.1 students	0
6.1.3.2 dirty	1
6.2 Data	1
6.2.1 Detailed Description	1
6.2.2 Macro Definition Documentation	1
6.2.2.1 DATA_NUM_RECORDS	1
6.2.2.2 DATA_ID_MAX_LENGTH	2

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 F	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
	7.4.1 Detailed Description	55
8 F	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.

2 ecet4640-lab4

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)

4 Compilation

2.3 Screenshot of Compilation

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
Мар .								 			 								 						30
MemS	ha	re						 			 								 						34
Proces	s							 			 								 						37
Util											 								 						48

6 Module Index

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_r	result	
	The result of a map retrieval	54
Studer	nt ·	
	The student data type	54

8 Data Structure Index

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	
src/server/Data.c	. 60
src/server/Data.h	
src/server/Files.c	. 61
src/server/Files.h	. 63
src/server/main.c	
Program entry point	
src/server/map.c	
src/server/map.h	
src/server/memShare.c	. 69
src/server/memShare.h	
src/server/Process.c	. 69
src/server/Process.h	_
src/server/util.c	. 73
src/server/util h	. 74

10 File Index

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()

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()

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

тар	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.

6.1 Build 13

Here is the call graph for this function:



Here is the caller graph for this function:

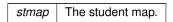


6.1.2.3 PipeWhoToStudentMap()

```
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



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:



6.1 Build 15

Here is the caller graph for this function:



6.1.2.5 SetAllStudentsInactive()

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.	

Generated by Doxygen

Definition at line 118 of file Build.c.

Here is the caller graph for this function:



6.1.2.7 ReadInitialCumulative()

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

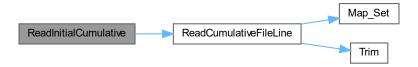
time_map	A map of cumulative times. Different from the students map.
filename	The filename where the initial cumulative times are located.

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:



6.1 Build 17

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

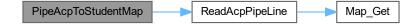
st_map	The students map.

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()

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 Build 19

6.1.2.10 ReadAcpPipeLine()

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()

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.

6.2 Data 21

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.

6.3 Files 23

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 rerandomized 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 Files 25

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

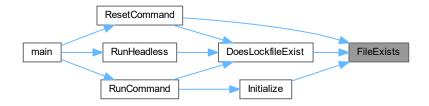
6.3.3.1 FileExists()

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()

Parameters

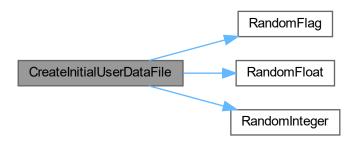
file_name	The file name to create.
id_list	An array containing the IDs. Eg. "Data_IDs" from Data.h
id_list_len	The length of the id_list. Eg. "DATA_NUM_RECORDS" from Data.h

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()

6.3 Files 27

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()

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()

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 Files 29

6.3.3.6 DoesLockfileExist()

```
short DoesLockfileExist ( )
```

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

Returns

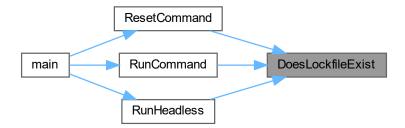
0 if lockfile does not exist, 1 if it does.

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.

6.4 Map 31

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 successful, and possibly containing the data sought, or NULL if it was unsuccessful.

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 successful 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 underyling 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()

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()

Parameters

тар	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.

6.4 Map 33

Here is the caller graph for this function:



6.4.2.3 Map_Get()

Parameters

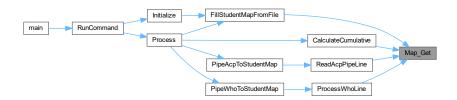
тар	The map to retrieve from.
key	The key of the item.

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()

Parameters

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

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 MemShare 35

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()

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

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

Parameters

shared_mem↔	The id of the shared memory
_id	

Returns

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

Definition at line 23 of file memShare.c.

6.6 Process 37

Here is the caller graph for this function:



6.5.3.4 ReleaseMemoryPointer()

```
int ReleaseMemoryPointer ( \mbox{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 Process 39

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()

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.

6.6 Process 41

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

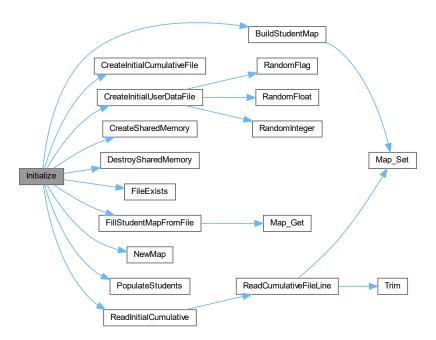
- (1) Create an initial user data file if it doesn't exist.
- (2) Initialize the students array.
- (3) Initialize the students map.
- (4) Read the data from the user data file into the map/array.
- (5) Initializes the shared memory segment.

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 \mbox{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.

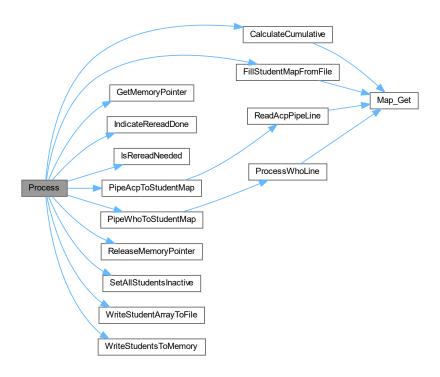
6.6 Process 43

Parameters

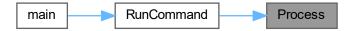
shm⊷	The ID of the shared memory segment.
_id	

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()

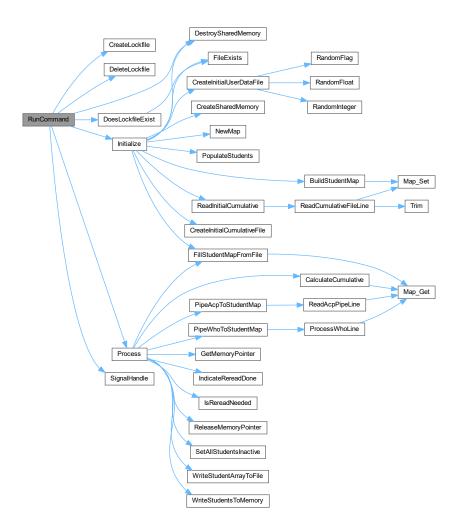
void RunCommand ()

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.

6.6 Process 45

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.

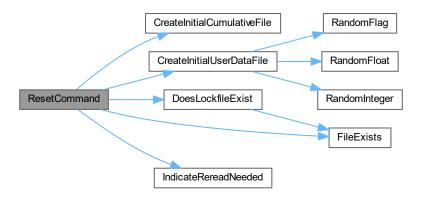
6.6 Process 47

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 ( {\tt 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 Util 49

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

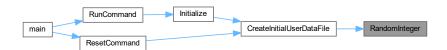
min	The minimum, inclusive.
max	The maximum, inclusive.

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()

Returns a float between min and max.

Parameters

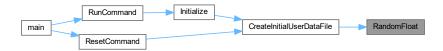
min	The minimum, inclusive.
max	The maximum, inclusive.

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()

Returns 1, percentage_chance of the time.

Parameters

percentage_chance	The chance to return 1.

Note

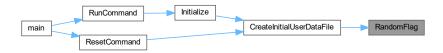
If $percentage_chance > 1$, this will always return true.

Returns

1 or 0

Definition at line 23 of file util.c.

Here is the caller graph for this function:



6.7 Util 51

6.7.2.4 Trim()

```
void Trim ( {\tt char} \ * \ {\it 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:

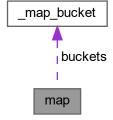
· src/server/map.h

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

```
00005 #include "Build.h"
00006 #include "memShare.h"
00007 #include "util.h"
00008 #include <string.h>
00009 #include <stdlib.h>
00010 #include <stdio.h>
00011 #include <time.h>
00012
00013 // ~~~~~ Data Structures ~~~~~~
00014
00015 Student *students:
00016
00017 void PopulateStudents(char **studentIDs, char **studentNames, int arsize)
00018 {
00019
          students = malloc(sizeof(Student) * arsize);
00020
          int i;
          for (i = 0; i < arsize; i++)</pre>
00021
00022
          {
               strcpy(students[i].userID, studentIDs[i]);
00024
              strcpy(students[i].fullName, studentNames[i]);
00025
00026 }
00027
00028 void BuildStudentMap(map *stmap, Student *studentArr, int studentArrLength)
00029 {
00030
00031
          for (i = 0; i < studentArrLength; i++)</pre>
00032
               Map_Set(stmap, studentArr[i].userID, (void *)(&studentArr[i]));
00033
00034
00035 }
00036
00037 // ~~~~~ Processing ~~~~~~
00038
00039 short dirty = 1; // start dirty
00040
00041 int PipeWhoToStudentMap(map *stmap)
00042 {
00043
          char command[4] = "who";
00044
          char line[100];
00045
          FILE *fpipe;
          fpipe = popen(command, "r");
if (fpipe == NULL)
00046
00047
00048
          {
00049
              return -1;
00050
00051
          while (fgets(line, sizeof(line), fpipe) != NULL)
00052
00053
00054
              ProcessWhoLine(stmap, line, strlen(line));
00055
00056
          pclose(fpipe);
00057
00058
          return 0;
00059 }
00060
00061 int ProcessWhoLine(map *stmap, char *whoLine, int whoLineLength)
```

```
00062 {
00063
          char userId[20];
00064
          char dateString[50];
00065
          char timeString[20];
00066
          int read_total = 0;
00067
          int read:
          sscanf(whoLine, " %s %n", userId, &read);
00068
00069
          read_total += read;
00070
00071
          map_result mr = Map_Get(stmap, userId);
00072
          if (!mr.found)
          { // if we can't find that person in the map, return early
00073
00074
              return -1;
00075
00076
          Student *student = (Student *)mr.data;
00077
          sscanf(whoLine + read_total, " %s %n", dateString, &read); // will be thrown away. eg 'pts/1'
00078
00079
          read total += read;
00080
          sscanf(whoLine + read_total, " %s %n", dateString, &read); // read the date string
00081
          read_total += read;
00082
          sscanf(whoLine + read_total, " %s %n", timeString, &read); // read the time string
00083
          strcat(dateString, " ");
          strcat(dateString, timeString); // catenate the time string back to the date string
00084
00085
00086
          time_t now = time(NULL);
          struct tm dtime = *localtime(&now);
00087
00088
          dtime.tm_sec = 0;
00089
00090
          memset(&dtime, 0, sizeof(struct tm));
00091
          sscanf(dateString, "%d-%d-%d-%d.%d", &(dtime.tm_year), &(dtime.tm_mon), &(dtime.tm_mon),
00092
     &(dtime.tm_hour), &(dtime.tm_min));
00093
00094
          dtime.tm_year -= 1900;
00095
          dtime.tm_mon -= 1;
          dtime.tm_hour -= 1;
00096
00097
00098
          time_t parsed_time = mktime(&dtime);
00099
00100
          if (student->lastLogin != parsed_time)
00101
00102
              student->lastLogin = parsed_time;
00103
              dirty = 1;
00104
00105
          student->active = 1;
00106
00107 }
00108
00109 void SetAllStudentsInactive(Student *stud arr, int arr len)
00110 {
00111
          int i;
00112
          for (i = 0; i < arr_len; i++)</pre>
00113
00114
              stud_arr[i].active = 0;
00115
00116 }
00117
00118 void WriteStudentsToMemory(void *mem_ptr, Student *stud_arr, int arr_len)
00119 {
00120
          Student *memloc = (Student *)mem_ptr;
00121
          int i;
          for (i = 0; i < arr_len; i++)</pre>
00122
00123
          {
00124
              strcpy(memloc[i].userID, stud_arr[i].userID);
00125
              strcpy(memloc[i].fullName, stud_arr[i].fullName);
              memloc[i].age = stud_arr[i].age;
memloc[i].gpa = stud_arr[i].gpa;
00126
00127
              memloc[i].active = stud_arr[i].active;
00128
00129
              memloc[i].lastLogin = stud_arr[i].lastLogin;
              memloc[i].loginDuration = stud_arr[i].loginDuration;
00130
00131
          }
00132 }
00133
00134 // ~~~~~~ Cumulative Processing ~~~~~~~
00135
00136 int ReadInitialCumulative(map *time_map, char *filename)
00137 {
00138
          FILE *file = fopen(filename, "r");
00139
          char line[100];
          if (file == NULL)
00140
00141
          {
00142
              return -1:
00143
00144
          while (fgets(line, sizeof(line), file) != NULL)
00145
              ReadCumulativeFileLine(time map, line);
00146
00147
          }
```

8.1 Build.c 59

```
00149
            fclose(file);
00150
            return 0;
00151 }
00152
00153 int PipeAcpToStudentMap(map *st_map)
00155
            char command[6] = "ac -p";
00156
            char line[300];
00157
           FILE *fpipe;
           fpipe = popen(command, "r");
00158
00159
            if (fpipe == NULL)
00160
            {
00161
                return -1;
00162
00163
            while (fgets(line, sizeof(line), fpipe) != NULL)
00164
00165
00166
                err = ReadAcpPipeLine(st_map, line);
00167
                if (err)
00168
                {
                     printf("\nError %d reading acp pipeline.", err);
00169
00170
00171
                }
00172
00173
           pclose(fpipe);
00174
00175 }
00176
00177 void ReadCumulativeFileLine(map *time_map, char *acp_line)
00178 {
00179
            char userId[20];
00180
            float hours;
            sscanf(acp_line, " %s %f ", userId, &hours);
00181
            long seconds = (long) (hours * 60 *60); 
// if(strcmp(userId, "mil7233") == 0) { 
// printf("Cum file line for %s seconds = %ld\n", userId, seconds);
00182
00183
00184
00185
00186
            Trim(userId);
            char* key = malloc( (strlen(userId)+1) * sizeof(char));
strcpy(key, userId);
00187
00188
00189
           Map_Set(time_map, userId, (void *)seconds);
00190 }
00191
00192 int ReadAcpPipeLine(map *stmap, char *acp_line)
00193 {
00194
            if (acp_line == NULL || strlen(acp_line) < 1)</pre>
00195
00196
                return -1;
00197
00198
           char userId[40];
00199
            float hours;
00200
            sscanf(acp_line, "%s %f", userId, &hours);
00201
            map_result result = Map_Get(stmap, userId);
00202
            if (result.found)
00203
00204
                Student *student = (Student *)result.data;
                seconds = (int) (hours * 60*60);
student->loginDuration = seconds;
// if(strcmp(userId, "mi17233")==0) {
// printf("ACP pine for "
00205
00206
00207
                        printf("ACP pipe for %s student quant = %f\n", userId, hours);
printf(" --- int seconds = %d\n", seconds);
00208
                //
00209
00210
                // }
00211
            return 0;
00212
00213 }
00214
00215 void CalculateCumulative(Student *stud arr, int stud arr len, map *time map)
00216 {
00217
00218
            for (i = 0; i < stud_arr_len; i++)</pre>
00219
00220
                map_result result = Map_Get(time_map, stud_arr[i].userID);
00221
                if (result.found)
00222
00223
00224
                     long time_at_server_start = (long)result.data;
                     stud_arr[i].loginDuration = stud_arr[i].loginDuration - time_at_server_start;
// if(strcmp("mi17233", stud_arr->userID) == 0) {
00225
00226
                     11
                             printf("calc cum: found user %s.\n", stud_arr[i].userID);
00227
                             printf(" tot time now: %ld\n", stud_arr[i].loginDuration;
printf(" time at server start: %ld\n", time_at_server_start);
                     11
00228
00229
00230
                              printf(" new duration: %ld\n", stud_arr[i].loginDuration);
00231
00232
                }
00233
           }
00234 }
```

8.2 Build.h

```
00001 #ifndef BUILD_H
00002 #define BUILD_H
00009 #include "Data.h"
00010 #include "map.h"
00011
00012 // ~~~~~ Data Structures ~~~~~
00013
00019 extern Student *students;
00020
00028 void PopulateStudents (char **studentIDs, char **studentNames, int arsize);
00036 void BuildStudentMap(map *stmap, Student *studentArr, int studentArrLength);
00037
00038 // ~~~~~ Processing ~~~~~~
00039
00041 extern short dirty;
00042
00049 int PipeWhoToStudentMap(map *stmap);
00050
00064 int ProcessWhoLine(map *stmap, char *whoLine, int whoLineLength);
00065
00072 void SetAllStudentsInactive(Student *stud arr, int arr len);
00073
00082 void WriteStudentsToMemory(void *mem_ptr, Student *stud_arr, int arr_len);
00083
00084 // ~~~~~ Cumulative Processing ~~~~~~
00085
00095 int ReadInitialCumulative(map *time_map, char *filename);
00096
00105 int PipeAcpToStudentMap(map *st_map);
00106
00115 void ReadCumulativeFileLine(map *time_map, char *acp_line);
00116
00124 int ReadAcpPipeLine(map *stmap, char *acp line);
00125
00135 void CalculateCumulative(Student *stud_arr, int stud_arr_len, map *time_map);
00136
00141 #endif
```

8.3 Data.c

00001

```
00005 #include "Data.h"
00007 char *Data_IDs[DATA_NUM_RECORDS] = {
          "chen",
80000
          "bea1389",
00009
00010
           "bol4559".
00011
          "cal6258",
00012
          "kre5277",
00013
          "lon1150",
00014
          "mas9309"
          "mes08346"
00015
           "mi17233",
00016
           "nef9476",
00017
00018
          "nov7488",
00019
          "pan9725",
00020
          "rac3146",
          "rub4133"
00021
          "shr5683",
00022
00023
          "vay3083",
00024
          "yos2327"};
00025
00026 char *Data_Names[DATA_NUM_RECORDS] = {
00027
          "Weifeng Chen",
          "Christian Beatty",
00028
           "Emily Bolles",
00029
00030
          "Cameron Calhoun",
00031
          "Ty Kress",
          "Cody Long",
00032
00033
          "Caleb Massey",
           "Christian Messmer",
00034
          "Karl Miller",
00035
           "Jeremiah Neff"
00036
00037
          "Kaitlyn Novacek",
00038
          "Joshua Panaro",
00039
          "Caleb Rachocki",
00040
          "Caleb Ruby",
00041
           "Paul Shriner"
00042
           "Alan Vayansky",
00043
          "Assefa Ayalew Yoseph"};
00044
```

8.4 Data.h 61

8.4 Data.h

```
00001 #ifndef Data_h
00002 #define Data_h
00009 #include <time.h>
00010 #include <sys/types.h>
00011
00015 #define DATA_NUM_RECORDS 17
00020 #define DATA_ID_MAX_LENGTH 9
00025 #define DATA_NAME_MAX_LENGTH 21
00026
00027 /* Constant, all user IDs. */
00028 extern char *Data_IDs[];
00029
00031 extern char *Data_Names[];
00032
00036 #define DATA SIZE 56
00037
00041 typedef struct
00042 {
00044
           char userID[DATA_ID_MAX_LENGTH];
00046
           char fullName[DATA_NAME_MAX_LENGTH];
00048
          short age;
00050
          float gpa;
short active;
00052
00054
          time_t lastLogin;
00056
           int loginDuration;
00057 } Student;
00061 #endif
```

8.5 Files.c

```
00001
00005 #include "Files.h"
00006 #include "util.h"
00007 #include <stdlib.h>
00008 #include <stdio.h>
00009 #include <strings.h>
00010 #include <unistd.h>
00011
00012 short FileExists(char *file_name_to_check)
00013 {
          FILE *file = fopen(file_name_to_check, "r");
00014
00015
          short result = 1;
          if (file == NULL)
00016
00017
00018
              result = 0;
00019
00020
          else
00021
          {
00022
              fclose(file);
00023
00024
          return result;
00025 }
00026
00027 int CreateInitialUserDataFile(char *file_name, char **id_list, int id_list_len)
00028 {
00029
          FILE *file = fopen(file_name, "w");
00030
          if (file == NULL)
00031
00032
              return -1;
00033
00034
          int i:
00035
          for (i = 0; i < id_list_len; i++)</pre>
00036
00037
              int rand_age = RandomInteger(18, 22);
00038
              float gpa;
              if (RandomFlag(0.42))
00039
00040
              {
                  gpa = 4.0; // 42% of the time, make the GPA 4.0
00041
00042
00043
              else
00044
              {
00045
                  gpa = RandomFloat(2.5, 4.0);
00046
00047
              fprintf(file, "%s\t%d\t%.2f\t%d\n", id_list[i], rand_age, gpa, 0);
00048
00049
          fclose(file);
00050
          return 0;
00051 }
00052
00053 int FillStudentMapFromFile(map *student_map, char *file_name, char **id_list, int id_list_len)
00054 {
```

```
FILE *file = fopen(file_name, "r");
00056
          if (file == NULL)
00057
00058
              return -1:
00059
          // id buffer
00060
00061
          char user_id[9];
00062
          int age;
00063
          float gpa;
00064
          long time;
          while (fscanf(file, "%9s\t%d\t%f\t%ld", user_id, &age, &gpa, &time) == 4)
00065
00066
00067
              map_result result = Map_Get(student_map, user_id);
00068
              if (result.found == 0)
00069
              {
00070
                  continue;
00071
              ((Student *)result.data)->age = age;
((Student *)result.data)->gpa = gpa;
00072
00073
00074
              ((Student *)result.data)->lastLogin = time;
00075
00076
          fclose(file);
00077
          return 0;
00078 }
00079
00080 int WriteStudentArrayToFile(Student *students, int arr_len, char *file_name)
00081 {
          FILE *file = fopen(file_name, "w");
if (file == NULL)
00082
00083
00084
00085
              return -1:
00086
00087
          int i;
00088
          for (i = 0; i < arr_len; i++)</pre>
00089
              fprintf(file, "%s\t%d\t%.2f\t%ld\n", students[i].userID, students[i].age, students[i].gpa,
00090
      students[i].lastLogin);
00091
          fclose(file);
00092
00093
          return 0;
00094 }
00095
00096 int CreateInitialCumulativeFile(char *file name)
00097 {
00098
          FILE *file = fopen(file_name, "w");
00099
          if (file == NULL)
00100
00101
              return -1;
00102
00103
          FILE *pipe = popen("ac -p", "r");
          if (pipe == NULL)
00104
00105
00106
              fclose(file);
00107
              return -2;
00108
          }
00109
00110
          char line[100];
00111
          while (fgets(line, sizeof(line), pipe) != NULL)
00112
00113
              fputs(line, file);
00114
00115
          pclose(pipe);
00116
          fclose(file);
00117
          return 0;
00118 }
00119
00121
00122 short DoesLockfileExist()
00123 {
00124
          return FileExists(LOCKFILE);
00125 }
00126
00127 int CreateLockfile()
00128 {
00129
          FILE *file = fopen(LOCKFILE, "w");
00130
          if (file == NULL)
00131
00132
              return -1:
00133
          fprintf(file, "0 %d", getpid());
00134
00135
          fclose(file);
00136
          return 0;
00137 }
00138
00139 int DeleteLockfile()
00140 {
```

8.6 Files.h 63

```
00141     return remove(LOCKFILE);
00142 }
```

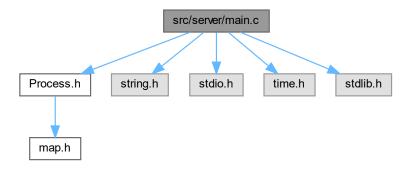
8.6 Files.h

```
00001 #ifndef Files_H
00002 #define Files_H
00015 #include "Data.h"
00016 #include "map.h"
00017
00033 #define STATIC_USER_DATA_FILE "static-user-data.txt"
00034
00043 #define STATIC_USER_CUMULATIVE_FILE "static-user-cumulative-start.txt"
00044
00053 #define LOCKFILE "/tmp/ecet-server.lock"
00054
00059 short FileExists(char *file_name_to_check);
00060
00070 int CreateInitialUserDataFile(char *file_name, char **id_list, int id_list_len);
00071
00081 int WriteStudentArrayToFile(Student *students, int arr_len, char *file_name);
00082
00093 int FillStudentMapFromFile(map *student_map, char *file_name, char **id_list, int id_list_len);
00094
00104 int CreateInitialCumulativeFile(char *file_name);
00105
00106
00107
00108 // ~~~~~~ Lockfile Commands ~~~~~~~~
00109
00115 short DoesLockfileExist();
00116
00125 int CreateLockfile();
00126
00131 int DeleteLockfile();
00136 #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 ( \label{eq:int_argc} \text{int } \textit{argc,} \text{char } ** \textit{argv} \text{ })
```

Parses arguments and calls the appropriate Process.h function.

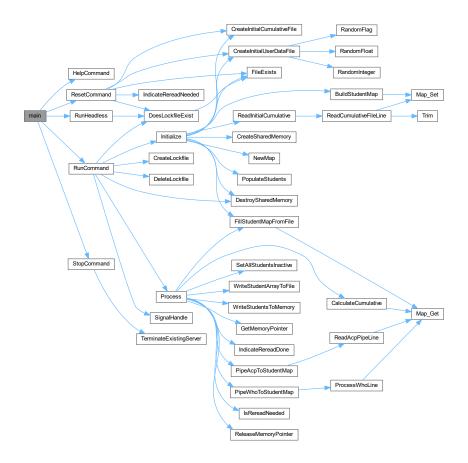
Parameters

argc	The argument count.
argv	The argument values.

Definition at line 104 of file main.c.

8.8 main.c 65

Here is the call graph for this function:



8.8 main.c

Go to the documentation of this file.

```
00001
00005 #include "Process.h"
00006 #include <string.h>
00007 #include <stdio.h>
00008 #include <time.h>
00009 #include <stdlib.h>
00104 int main(int argc, char **argv)
00105 {
00106
           srand(time(NULL)); // seed the randomizer
00107
00108
           if (argc <= 1 || argc >= 3)
00109
00110
               printf("Too few or many options!\n");
00111
               HelpCommand();
00112
00113
           else if (strcmp(argv[1], "help") == 0)
00114
00115
               HelpCommand();
00116
           else if (strcmp(argv[1], "reset") == 0)
00117
00118
               ResetCommand();
00119
00120
00121
           else if (\text{strcmp}(\text{argv}[1], "\text{stop"}) == 0 \mid | \text{strcmp}(\text{argv}[1], "\text{end"}) == 0 \mid | \text{strcmp}(\text{argv}[1], "\text{close"})
        = 0 || strcmp(argv[1], "exit") == 0)
00122
00123
               StopCommand();
00124
00125
           else if (strcmp(argv[1], "headless") == 0)
00126
```

```
RunHeadless(argv[0]);
00128
          else if (strcmp(argv[1], "run") == 0 || strcmp(argv[1], "start") == 0)
00129
00130
00131
              RunCommand():
00132
00133
          else
00134
          {
00135
              printf("Unknown option!\n");
00136
              HelpCommand();
00137
00138
          return 0:
00139 }
```

8.9 map.c

```
00001
00005 #include "stdlib.h"
00006 #include "string.h"
00007 #include "map.h"
00008 #include "math.h"
00009
00011 int hash_log2(int num_to_log)
00012 {
00013
          int t = 1:
          int i = 0;
00014
00015
          do
00016
          {
00017
              num_to_log = num_to_log & ~t;
              t = t « 1;
i++;
00018
00019
00020
          } while (num_to_log > 0);
          return i;
00021
00022 }
00023
00025 int hash_upperLimit(int bitsize)
00026 {
00027
          return 1 « bitsize;
00028 }
00029
00031 int char_ratio = (int)(sizeof(int) / sizeof(char));
00032
00034 int hash_string(int hash_table_size, char *string, int strlen)
00035 {
          int i, hash = 2166136261;
00037
          for (i = 0; i < strlen; i += 1)</pre>
00038
00039
              hash *= 16777619;
              hash ^= string[i];
00040
00041
00042
          if (hash < 0)
00043
          {
00044
              hash *= -1;
00045
00046
          return hash % hash_table_size;
00047 }
00048
00049 map *NewMap(int capacity)
00050 {
          int log2 = hash_log2(capacity);
int capac = hash_upperLimit(log2);
00051
00052
00053
          int sz = sizeof(struct _map_bucket) * capac;
          struct _map_bucket *buckets = malloc(sz);
memset(buckets, 0, sz);
00054
00056
          int i;
00057
          for (i = 0; i < capac; i++)</pre>
00058
00059
              buckets[i] = (struct _map_bucket) {NULL, NULL, NULL};
00060
00061
          map newm = (map) {capac, buckets};
00062
          map *map_p = malloc(sizeof(map));
          *map_p = newm;
00063
00064
          return map_p;
00065 }
00066
00068 void _bucket_insert(struct _map_bucket *bucket, char *key, void *value)
00069 {
00070
          struct _map_bucket *check = bucket;
00071
          while (check->key != NULL)
00072
          {
00073
               if (strcmp(check->key, key) == 0)
00074
              {
00075
                   check->data = value;
```

8.9 map.c 67

```
00076
                  return;
00077
00078
              if (check->next == NULL)
00079
              {
                  check->next = malloc(sizeof(struct _map_bucket));
00080
00081
                  *(check->next) = (struct _map_bucket) {NULL, NULL};
00083
              check = check->next;
00084
00085
          check->key = key;
00086
          check->data = value;
00087 }
00088
00089 void Map_Set(map *a_map, char *key, void *value)
00090 {
          int keyl = (int)strlen(key);
int hash = hash_string(a_map->size, key, keyl);
00091
00092
00093
          _bucket_insert(&(a_map->buckets[hash]), key, value);
00094 }
00096 void _bucket_get(struct _map_bucket *bucket, char *key, map_result *result)
00097 {
00098
          struct _map_bucket *check = bucket;
00099
          while (check->key != NULL)
00100
00101
              if (strcmp(check->key, key) == 0)
00102
              {
00103
                  result->found = 1;
00104
                  result->data = check->data;
00105
                  return;
00106
00107
              else if (check->next != NULL)
00108
              {
00109
                  check = check->next;
00110
00111
              else
00112
              {
00113
                  result->found = 0;
00114
                  break;
00115
00116
          }
00117 }
00118
00119 map_result Map_Get(map *a_map, char *key)
00120 {
00121
          map_result res = (map_result) {0, NULL};
00122
          int keyl = (int)strlen(key);
00123
          int hash = hash_string(a_map->size, key, keyl);
00124
          _bucket_get(&(a_map->buckets[hash]), key, &res);
00125
          return res:
00126 }
00127
00128 void _bucket_delete(struct _map_bucket *bucket, char *key, short free_it, map_result *result)
00129 {
          struct _map_bucket *last = bucket;
struct _map_bucket *next = bucket->next;
00130
00131
          while (next != NULL)
00132
00134
              if (strcmp(next->key, key) == 0)
00135
              {
00136
                  result->found = 1;
                  result->data = next->data;
00137
00138
                  if (free_it)
00139
                  {
00140
                      free(next->data);
00141
                      result->data = NULL;
00142
                  last->next = next->next;
00143
00144
                  free(next);
00145
00146
              else
00147
00148
                  last = next;
                  next = next->next;
00149
00150
00151
          }
00152 }
00153
00154 map_result Map_Delete(map *a_map, char *key, short free_it)
00155 {
00156
          map result res = (map result) {0, NULL};
          int keyl = (int)strlen(key);
00157
          int hash = hash_string(a_map->size, key, keyl);
00158
00159
00160
          struct _map_bucket top = a_map->buckets[hash];
00161
          if (top.key == NULL)
00162
00163
              return res:
```

```
00165
          if (strcmp(top.key, key) == 0)
00166
00167
              res.found = 1;
              res.data = top.data;
if (free_it)
00168
00169
00170
             {
00171
                  free(top.data);
00172
                 res.data = NULL;
00173
00174
              if (top.next != NULL)
00175
              {
00176
                  a_map->buckets[hash] = *(top.next);
00177
                  free(top.next);
00178
00179
              else
00180
              {
00181
                  a_map->buckets[hash] = (struct _map_bucket){NULL, NULL};
              }
00182
00183
              return res;
00184
00185
          if (top.next == NULL)
00186
00187
              return res:
00188
00189
          _bucket_delete(&(a_map->buckets[hash]), key, free_it, &res);
00190
00191
          return res;
00192 }
```

8.10 map.h

```
00001 #ifndef map_h
00002 #define map_h
00003
00041 //
00042 // Hashing Math
00043 // -----
00044
00051 int hash_log2(int number_to_log);
00052
00062 int hash_string(int hash_table_capacity, char *string, int strlen);
00063
00070 int hash_upperLimit(int bitsize);
00072 // -
00073 //
              General Map Operations
00074 // -----
00075
00081 struct _map_bucket
00082 {
         char *key;
00086
         void *data;
00088
         struct _map_bucket *next;
00089 };
00090
00101 typedef struct
00102 {
         int size;
00103
00104
         struct _map_bucket *buckets;
00105 } map;
00106
00111 typedef struct
00112 {
00113
00114
        short found;
         void *data;
00115
00116 } map_result;
00117
00124 map *NewMap(int capacity);
00125
00133 void Map_Set(map *a_map, char *key, void *value);
00134
00141 map_result Map_Get(map *a_map, char *key);
00142
00150 map_result Map_Delete(map *a_map, char *key, short free_it);
00151
00152 #endif
```

8.11 memShare.c 69

8.11 memShare.c

```
00001
00005 #include "memShare.h"
00006 #include <string.h>
00007 #include <stdio.h>
80000
00009 int CreateSharedMemory()
00010 {
00011
          return shmget(MEM_KEY, MEM_SIZE, IPC_CREAT | MEM_PERMISSIONS);
00012 }
00013
00014 int DestroySharedMemory()
00015 {
00016
          int shm_id = shmget(MEM_KEY, MEM_SIZE, 0);
00017
          int control_result = shmctl(shm_id, IPC_RMID, 0);
         if (control_result != -1)
00018
00019
             return 0:
00020
         return control_result;
00021 }
00022
00023 void *GetMemoryPointer(int shared_mem_id)
00024 {
00025
          return shmat(shared mem id, NULL, 0);
00026 }
00027
00028 int ReleaseMemoryPointer(void *shmaddr)
00029 {
00030
          return shmdt(shmaddr);
00031 }
```

8.12 memShare.h

```
00001 #ifndef MEM SHARE H
00002 #define MEM SHARE H
00014 #include <sys/shm.h>
00015 #include <sys/ipc.h>
00016 #include "Data.h"
00017
00021 #define MEM_KEY 0x727
00022
00032 #define MEM_PERMISSIONS 0664
00033
00037 #define MEM_SIZE DATA_SIZE *DATA_NUM_RECORDS
00038
00044 int CreateSharedMemory();
00045
00051 int DestroySharedMemory();
00062 void *GetMemoryPointer(int shared_mem_id);
00063
00069 int ReleaseMemoryPointer(void *shmaddr);
00073 #endif
```

8.13 Process.c

```
00001
00006 #include "Process.h"
00007 #include "Files.h"
00008 #include "Data.h"
00009 #include "Build.h"
00010 #include "memShare.h"
00011 #include <errno.h>
00012 #include <stdio.h>
00013 #include <unistd.h>
00014 #include <signal.h>
00015 #include <stdlib.h>
00016 #include <string.h>
00017
00018 map *student_map;
00020 map *initial_cumulative_times;
00021
00022 int TerminateExistingServer()
00023 {
00024
            FILE *file = fopen(LOCKFILE, "r");
00025
            if (file == NULL)
00026
            {
00027
                 return -1;
00028
            }
```

```
00029
         int need_rewrite;
         int pid = 0;
fscanf(file, "%d %d", &need_rewrite, &pid);
00030
00031
00032
         fclose(file);
00033
         if (pid > 0)
00034
00035
             return kill(pid, SIGTERM);
00036
00037
         return -2;
00038 }
00039
00040 int IndicateRereadNeeded()
00041 {
00042
         FILE *file = fopen(LOCKFILE, "r+");
00043
         if (file == NULL)
00044
00045
             return -1:
00046
00047
         int err = 0;
00048
         err = fseek(file, 0, SEEK_SET);
00049
         if (!err)
00050
         {
00051
             fputc('1', file);
00052
00053
         err = fclose(file);
00054
         return err;
00055 }
00056
00057 int IndicateRereadDone()
00058 {
00059
         FILE *file = fopen(LOCKFILE, "r+");
00060
         if (file == NULL)
00061
00062
             return -1;
00063
         int err = 0;
00064
00065
         err = fseek(file, 0, SEEK_SET);
         if (!err)
00066
00067
         {
00068
             fputc('0', file);
00069
00070
         err = fclose(file):
00071
         return err;
00072 }
00073
00074 short IsRereadNeeded()
00075 {
         FILE *file = fopen(LOCKFILE, "r");
00076
         char firstc = fgetc(file);
00077
00078
         fclose(file);
00079
         return firstc == '1';
00080 }
00081
00082 void SignalHandle(int signo) 00083 {
00084
         printf("Received shutdown signal.\n");
00085
          if (signo == SIGINT || signo == SIGTERM)
00086
00087
             is_stopping = 1;
00088
00089
          // possible feature: add a timeout terminate emergency exit (with graceful shutdown)
00090
00091 }
00092
00093 short is_stopping = 0;
00094
00096
00097 int Initialize()
00098 {
00099
          int err;
00100
         if (!FileExists(STATIC_USER_DATA_FILE))
00101
             printf("%s does not exist. Creating.\n", STATIC_USER_DATA_FILE);
00102
00103
             err = CreateInitialUserDataFile(STATIC_USER_DATA_FILE, Data_IDs, DATA_NUM_RECORDS);
00104
00105
             {
00106
                 printf("Problem creating %s!\n", STATIC_USER_DATA_FILE);
00107
00108
         if (!FileExists(STATIC_USER_CUMULATIVE_FILE))
00109
00110
             00111
00112
00113
             if (err)
00114
             {
00115
                 printf("Problem creating %s!\n", STATIC_USER_CUMULATIVE_FILE);
```

8.13 Process.c 71

```
00116
              }
00117
00118
          PopulateStudents(Data_IDs, Data_Names, DATA_NUM_RECORDS);
00119
          student_map = NewMap(50);
          BuildStudentMap(student_map, students, DATA_NUM_RECORDS);
00120
00121
          err = FillStudentMapFromFile(student_map, STATIC_USER_DATA_FILE, Data_IDs, DATA_NUM_RECORDS);
00122
          if (err)
00123
          {
00124
              printf("Problem filling student map from %s!\n", STATIC_USER_DATA_FILE);
00125
          printf("Student data retrieved from file.\n");
00126
00127
00128
          initial_cumulative_times = NewMap(50);
00129
          err = ReadInitialCumulative(initial_cumulative_times, STATIC_USER_CUMULATIVE_FILE);
00130
          if (err)
00131
              printf("Failed to read %s. Cumulative times may be wrong!", STATIC_USER_CUMULATIVE_FILE);
00132
00133
          }
00134
00135
          dirty = 0;
00136
00137
          int shmid = CreateSharedMemory();
00138
          if (shmid == -1)
00139
          {
00140
              DestroySharedMemory();
00141
              shmid = CreateSharedMemory();
00142
00143
          printf("Shared memory allocated.\n");
00144
          return shmid;
00145 }
00146
00147 void Process(int shm_id)
00148 {
00149
          if (IsRereadNeeded())
00150
              printf("\nReread indicated - rechecking user data file.");
00151
00152
              FillStudentMapFromFile(student_map, STATIC_USER_DATA_FILE, Data_IDs, DATA_NUM_RECORDS);
00153
              IndicateRereadDone();
00154
00155
          SetAllStudentsInactive(students, DATA_NUM_RECORDS);
00156
          int err = PipeAcpToStudentMap(student_map);
00157
          if (err)
00158
          {
00159
              printf("Error piping ac -p command! \n");
00160
00161
          else
00162
00163
              CalculateCumulative(students, DATA_NUM_RECORDS, initial_cumulative_times);
00164
00165
          err = PipeWhoToStudentMap(student map);
00166
          if (err)
00167
          {
00168
              perror("Error updating from who!");
00169
00170
          if (dirty)
00171
          {
00172
              err = WriteStudentArrayToFile(students, DATA_NUM_RECORDS, STATIC_USER_DATA_FILE);
00173
00174
00175
                  printf("\nError updating %s!", STATIC_USER_DATA_FILE);
00176
              }
00177
              else
00178
              {
00179
                  dirty = 0;
00180
              }
00181
          void *ptr = GetMemoryPointer(shm_id);
00182
          if (ptr == (void *)-1)
00183
00184
          {
00185
              perror("Error attaching to shared memory");
00186
00187
          else
00188
          {
00189
              WriteStudentsToMemory(ptr, students, DATA_NUM_RECORDS);
00190
              ReleaseMemoryPointer(ptr);
00191
00192 }
00193
00194 void HelpCommand()
00195 {
          printf("\nUsage: server [OPTION]\n\n");
00196
          printf("Options: \n");
printf("\thelp\t\t\tShows the possible program commands\n");
00197
00198
00199
          printf("\treset\t\t\tRegenerates the user data file\n");
           printf("\tstop\t\ttstops an existing server process if it is running\n"); \\
00200
          \texttt{printf("} \\ \texttt{trun}\\ \texttt{t}\\ \texttt{tCreates a new server with output to the shell if a server isn't already}
00201
      running.\n");
```

```
printf("\theadless\t\tCreates a new headless server if a server isn't already running.\n\n");
00203 }
00204
00205 void RunCommand()
00206 {
00207
          printf("\nRunning server.\n");
          if (DoesLockfileExist())
00208
00209
00210
              printf("\nServer is already running. Run 'server stop' to shut it down first.\n");
00211
00212
00213
          int err = CreateLockfile();
00214
          if (err)
00215
00216
              printf("\nFailed to create lockfile! Exiting.\n");
00217
00218
00219
          int shm id = Initialize();
          signal(SIGTERM, SignalHandle);
signal(SIGINT, SignalHandle);
00220
00221
00222
          printf("Server started.\n");
00223
          fflush(stdout);
00224
          while (!is_stopping)
00225
          {
00226
              Process(shm_id);
00227
              sleep(1);
00228
00229
          printf("Server shutting down.\n");
00230
          err = DeleteLockfile();
00231
          if (err)
00232
00233
              printf("Failed to delete lockfile!\n");
00234
00235
          err = DestroySharedMemory();
00236
          if (err)
00237
00238
              printf("Failed to destroy shared memory!\n");
00239
00240
          printf("Server terminated.\n");
00241 }
00242
00243 void StopCommand()
00244 {
00245
          printf("\nStopping server...\n");
00246
          int err = TerminateExistingServer();
00247
          if (err != 0)
00248
00249
              if (err == -1)
00250
00251
                  printf("Server isn't running.\n");
00252
00253
              else if (err == -2)
00254
00255
                   printf("Lockfile did not contain a valid process id! \n");
00256
00257
              else
00258
00259
                   printf("Sending terminate signal failed!\n");
00260
00261
00262
          else
00263
          {
00264
              printf("Server terminated.\n");
00265
00266 }
00267
00268 void ResetCommand()
00269 {
00270
00271
00272
          if (FileExists(STATIC_USER_DATA_FILE))
00273
              printf("User data file exists. Deleting...\n");
remove(STATIC_USER_DATA_FILE);
00274
00275
00276
          }
00277
00278
          printf("Creating new data file.\n");
00279
          err = CreateInitialUserDataFile(STATIC_USER_DATA_FILE, Data_IDs, DATA_NUM_RECORDS);
00280
          if (err)
00281
          {
              printf("Problem creating %s!\n", STATIC_USER_DATA_FILE);
00282
00283
00284
00285
00286
              printf("%s created.\n", STATIC_USER_DATA_FILE);
00287
          }
00288
```

8.14 Process.h 73

```
printf("Creating new cumulative file.\n");
00290
           err = CreateInitialCumulativeFile(STATIC_USER_CUMULATIVE_FILE);
00291
           if (err)
00292
           {
00293
               printf("Problem creating %s!\n", STATIC_USER_CUMULATIVE_FILE);
00294
00295
          else
00296
          {
00297
               printf("%s created.\n", STATIC_USER_CUMULATIVE_FILE);
00298
           }
00299
00300
           if (DoesLockfileExist())
00301
           {
00302
               printf("Indicated re-read to running server process.\n");
00303
               IndicateRereadNeeded();
00304
           }
00305 }
00306
00307 void RunHeadless (char *processName)
00308 {
00309
           if (DoesLockfileExist())
00310
00311
               printf("Server process already running.\n");
00312
               return;
00313
00314
          char commandFront[] = " nohup ";
char commandEnd[] = " run & exit";
00315
00316
           size_t comm_length = strlen(commandFront) + strlen(commandEnd) + strlen(processName) + 1;
           char *commandFull = malloc(comm_length * sizeof(char));
00317
00318
          memset(commandFull, 0, comm_length * sizeof(char));
          strcpy(commandFull, commandFront);
00319
          strcat(commandFull, processName);
strcat(commandFull, commandEnd);
00320
00321
00322
          printf("Executing: %s\n", commandFull);
popen(commandFull, "we");
00323
00324
00325
           printf("Server running headlessly.\n");
00326 }
```

8.14 Process.h

```
00001 #ifndef Process_h
00002 #define Process h
00010 #include "map.h"
00011
00020 int TerminateExistingServer();
00021
00029 int IndicateRereadNeeded();
00030
00037 int IndicateRereadDone();
00038
00045 short IsRereadNeeded();
00046
00052 void SignalHandle(int signo);
00053
00057 extern short is_stopping;
00058
00059 // ~~~~~~ CLI Commands ~~~~~~~
00060
00061 extern map *student_map;
00062
00073 int Initialize():
00074
00087 void Process(int shm_id);
00088
00095 void ResetCommand();
00096
00102 void StopCommand();
00103
00108 void RunCommand();
00116 void HelpCommand();
00117
00122 void RunHeadless(char *processName);
00127 #endif
```

8.15 util.c

00001

```
00005 #include "util.h"
00006
00007 #include <stdlib.h>
00008 #include <string.h>
00009
00010 int RandomInteger(int min, int max)
00011 {
00012
          int r_add = rand() % (max - min + 1);
00013
         return r_add + min;
00014 }
00015
00016 float RandomFloat (float min, float max)
00017 {
00018
          float dif = max - min;
          int rand_int = rand() % (int) (dif * 10000);
return min + (float)rand_int / 10000.0;
00019
00020
00021 }
00022
00023 short RandomFlag(float percentage_chance)
00024 {
00025
          float random_value = (float)rand() / RAND_MAX;
00026
          if (random_value < percentage_chance)</pre>
00027
00028
              return 1;
00029
         }
00030
          return 0;
00031 }
00032
00033 void Trim(char * string)
00034 {
00035
          size_t len = strlen(string);
00036
          int i;
00037
          for (i = 0; i < len; i++)</pre>
00038
              00039
00040
00041
                  string[i] = ' \setminus 0';
00042
                  break;
00043
              }
00044
          }
00045 }
00046
```

8.16 util.h

```
00001 #ifndef util_h
00002 #define util_h
00017 int RandomInteger(int min, int max);
00018
00025 float RandomFloat(float min, float max);
00026
00033 short RandomFlag(float percentage_chance);
00034
00039 void Trim(char * string);
00044 #endif
```

Index

Build, 11 BuildStudentMap, 12 CalculateCumulative, 19 dirty, 20 PipeAcpToStudentMap, 17 PipeWhoToStudentMap, 17 PipeWhoToStudentMap, 17 PipeWhoToStudentMap, 17 PipeWhoToStudentMap, 17 PipeWhoToStudentMap, 18 PeadCumulativeFileLine, 18 ReadCumulativeFileLine, 18 ReadCumulativeFileLine, 18 ReadInitialCumulative, 16 SetAllStudentsInactive, 15 students, 20 WriteStudentsToMemory, 15 Build, 19 Build, 19 CreateInitialCumulative Build, 19 CreateInitialCumulative Files, 28 CreateInitialCumulative Files, 26 CreateInitialCumulative Files, 26 CreateInitialCumulative Files, 26 CreateLockfile, 30 DoesLockfileExist, 28 FileExists Files, 29 CreateInitialCumulative Files, 26 CreateInitialCumulative Files, 26 CreateInitialCumulative Files, 26 CreateInitialCumulative Files, 26 CreateInitialCumulative Files, 28 GetMemoryPointer Files, 26 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_NUM_RECORDS Data, 21 DATA_NUM_RECORDS Data, 21 DATA_NAME_MAX_LENGTH Data, 22 DATA_NUM_RECORDS Data, 21 DATA_SIZE DATA_DATA_SIZE DATA_SIZE	_map_bucket, 53	dirty
BuildStudentMap, 12 CalculateCumulative, 19 dirty, 20 PipeAcpToStudentMap, 17 PipeWhoToStudentMap, 13 PopulateStudents, 11 ProcessWhoLine, 14 ReadAcpPipeLine, 18 ReadCumulativeFileLine, 18 ReadCumulativeFileLine, 18 ReadInitialCumulative, 16 SetAllStudentsInactive, 15 students, 20 WriteStudentsInactive, 15 students, 20 WriteStudentsFoMemory, 15 BuildStudentMap Build, 12 CalculateCumulative Build, 19 CreateInitialCumulativeFile Files, 28 FileSxists Files, 23 CreateLockfile, 29 CreateLockfile, 30 DoesLockfileExist, 28 FileSxists, 28 FileSxists, 28 FileStists, 28 FileStists, 28 FileStists, 28 FileStist, 28 FileStists, 29 FileStists, 29 FileStists, 29 FileStists, 29 FileStists, 29 FileStists, 29 FileStists CreateLockfile, 29 CreateIntialCumulativeFile Files, 25 FileStists CreateLockfile, 29 FileStists CreateLockfile, 25 FileStists CreateLockfile, 29 FileStists CreateLockfile, 29 FileStists CreateLockfile, 29 FileStists CreateLockfile, 25 FileStists CreateLockfile, 26 Files, 28 FileStists CreateLockfile, 25 FileStists CreateLockfile, 25 FileStists CreateLockfile, 26 Files, 28 FileStists CreateLockfile, 25 FileStists CreateLockfile, 25 FileStists CreateLockfile, 26 CockFile, 24 WriteStudentsTownorthy Files, 27 FillStudentMapFromFile, 26 LOCKFile, 24 WriteStudentsTownorthy Files, 29 FileStientMapFromFile, 26 LOCKFile, 24 WriteStudentsTownorthy FileStigentMapFromFile, 26 Files, 25 FillStudentMapFromFile, 26 LOCKFile, 24 WriteStudentsTownorthy FileStidentsTownorthy FileStidentsTownorthy FileStidentsTownorthy FileStidentsTownorthy FileStidentsTownorthy FileStidentsTownorthy FileStidentsTownorthy FileStidentsTownorthy FileStidentsTownorthy FileStidentsTo	Build 11	•
CalculateCumulative, 19 diry, 20 PipeAcpToStudentMap, 17 PipeWhoToStudentMap, 13 PopulateStudents, 11 ProcessWhoLine, 14 ReadAcpPipeLine, 18 ReadCumulativeFileLine, 18 ReadCumulativeFileLine, 18 ReadCitialCumulative, 16 SetAllStudentsInactive, 15 students, 20 WriteStudentsToMemory, 15 BuildStudentMap Build, 19 CreateInitialCumulativeFile Files, 28 CreateInitialCumulative Build, 19 CreateInitialCumulative Files, 28 CreateInitialCumulative Build, 19 CreateInitialCumulative Files, 28 CreateInitialCumulative Files, 29 CreateSharedMemory MemShare, 35 Data, 21 DATA_NAME_MAX_LENGTH, 21 DATA_NAME_MAX_LENGTH Data, 21 DATA_NAME_MAX_LENGTH Data, 22 DATA_NAME_MAX_LENGTH DATA_SIZE DA	•	
dirty, 20 PipeAcpToStudentMap, 17 PipeWhoToStudentMap, 13 PopulateStudents, 11 ProcessWhoLine, 14 ReadAcpPipeLine, 18 ReadCumulativeFileLine, 18 ReadCumulativeFileLine, 18 ReadInitialCumulative, 16 SetAllStudentsInactive, 15 students, 20 WriteStudentsToMemory, 15 BuildStudentMap Build, 12 CalculateCumulative Build, 19 CreateInitialCumulativeFile Files, 28 CreateSharedMemory MemShare, 35 Data, 21 DATA_NAME_MAX_LENGTH, 21 DATA_NAME_MAX_LENGTH, 22 DATA_NUM_RECORDS, 21 DATA_NAME_MAX_LENGTH Data, 22 DATA_NAME_MAX_LENGTH Data, 21 DATA_NAME_MAX_LENGTH Data, 21 DATA_NAME_MAX_LENGTH Data, 22 DATA_NUM_RECORDS Data, 31 DATA_NAME_MAX_LENGTH Data, 22 DATA_NAME_MAX_LENGTH Data, 21 DATA_NAME_MAX_LENGTH Data, 21 DATA_NAME_MAX_LENGTH Data, 22 DATA_NAME_MAX_LENGTH Data, 21 DATA_NAME_MAX_LENGTH DATA_SIZE	•	riies, 20
PipeAcpToStudentMap, 17 PipeWhoToStudentMap, 13 PopulateStudents, 11 ProcessWhoLine, 14 ReadAcpPipeLine, 18 ReadInitialCumulative, 16 SetAllStudents, 20 WriteStudentsToMemory, 15 BuildStudentMap Build, 12 CalculateCumulative Build, 19 CreateInitialUserDataFile Files, 25 FillStudentMapFromFile, 26 LOCKFILE, 24 STATIC_USER_DATA_FILE, 24 WriteStudentArrayToFile, 27 FillStudentMapFromFile Files, 26 CreateLockfile Files, 27 FillStudentMapFromFile, 26 LOCKFILE, 24 STATIC_USER_DATA_FILE, 24 WriteStudentArrayToFile, 27 FillStudentMapFromFile Files, 26 CreateLockfile Files, 27 FillStudentMapFromFile, 26 LOCKFILE, 24 WriteStudentArrayToFile, 27 FillStudentMapFromFile Files, 26 GetMemoryPointer MemShare, 36 HelpCommand Process, 43 IndicateRereadDone Process, 39 IndicateRereadNeeded Process,		FileFxists
PipeWhoToStudentMap, 13 PopulateStudents, 11 ProcessWhoLine, 14 ReadAcpPipeLine, 18 ReadCurnulativeFileLine, 18 ReadCurnulativeFileLine, 18 ReadCurnulativeFileLine, 15 SetAllStudentsInactive, 15 Students, 20 WriteStudentsToMemory, 15 BuildStudentMap Build, 12 CalculateCumulative Build, 12 CreateInitialUserDataFile Files, 28 CreateLockfile, 29 DeleteLockfile, 30 DoestockfileExist, 28 FileExists, 25 FillStudentMapFromFile, 26 LOCKFILE, 24 STATIC_USER_CUMULATIVE_FILE, 24 WriteStudentArpToFile, 27 FillStudentMapFromFile Files, 28 GetMemoryPointer MemShare, 36 Files, 29 CreateSharedMemory MemShare, 35 Data, 21 DATA_ID_MAX_LENGTH, 21 DATA_NAME_MAX_LENGTH, 22 DATA_NAME_MAX_LENGTH Data, 21 DATA_ID_MAX_LENGTH Data, 21 DATA_NAME_MAX_LENGTH Data, 21 DATA_NAME_MAX_LENGTH Data, 21 DATA_NAME_MAX_LENGTH Data, 21 DATA_NAME_MAX_LENGTH Data, 22 DATA_NAME_MAX_LENGTH Data, 21 DATA_NAME_MAX_LENGTH Data, 21 DATA_NAME_MAX_LENGTH Data, 21 DATA_NAME_MAX_LENGTH Data, 22 DATA_NAME_MAX_LENGTH Data, 21 DATA_NAME_MAX_LENGTH Data, 22 DATA_NUM_RECORDS Data, 21 DATA_NUM_RECORDS Data, 21 DATA_SIZE DATA_NUM_RECORDS Data, 21 DATA_SIZE DATA_SIZ	•	
PopulateStudents, 11 ProcessWhoLine, 14 ReadAcpPipeLine, 18 ReadCumulativeFileLine, 18 ReadCumulativeFileLine, 18 ReadCumulative, 16 SetAllStudentsInactive, 15 students, 20 WriteStudentsToMemory, 15 BuildStudentMap Build, 12 CreateInitialUserDataFile, 28 FileExists, 25 FileStudentMapFromFile, 26 LOCKFILE, 24 STATIC_USER_CUMULATIVE_FILE, 24 WriteStudentArrayToFile, 27 FillStudentMapFromFile Files, 28 GreateInitialUserDataFile Files, 25 CreateLockfile Files, 25 CreateLockfile Files, 29 CreateSharedMemory MemShare, 35 Data, 21 DATA_NAME_MAX_LENGTH, 21 DATA_NAME_MAX_LENGTH, 22 DATA_NUM_RECORDS, 21 DATA_SIZE, 22 DATA_NUM_RECORDS Data, 21 DATA_NAME_MAX_LENGTH Data, 22 DATA_NUM_RECORDS Data, 21 DATA_NUM_RECORDS Data, 21 DATA_SIZE Data, 22 DATA_NUM_RECORDS Data, 21 DATA_SIZE Data, 22 DATA_NUM_RECORDS Data, 21 DATA_SIZE Data, 22 DATA_NUM_RECORDS Data, 21 DATA_SIZE DATA_SI	·	
ProcessWhoLine, 14 ReadAcpPipeLine, 18 ReadAcpPipeLine, 18 ReadComulativeFileLine, 18 ReadInitialCumulative, 16 SetAllStudentsInactive, 15 students, 20 WriteStudentsToMemory, 15 BuildStudentMap Build, 12 CreateInitialCumulative Build, 19 CreateInitialCumulative Files, 28 CreateLockfile, 29 DeteLockfile, 30 DoesLockfileExist, 28 FileSxists, 25 FillStudentMapFromFile, 26 LOCKFILE, 24 STATIC_USER_CUMULATIVE_FILE, 24 WriteStudentArrayToFile, 27 FillStudentMapFromFile, 26 CockellateCumulative Build, 19 CreateInitialCumulativeFile Files, 28 CreateInitialUserDataFile Files, 25 CreateLockfile Files, 25 CreateLockfile Files, 25 CreateLockfile Files, 25 CreateAcplane Files, 25 CreateAcplane Files, 25 CreateLockfile Files, 25 CreateAcplane Files, 25 CreateAcplane Files, 26 GettMemoryPointer MemShare, 36 Files, 26 FillStudentMapFromFile, 26 FillStudentMapFromFile Files, 26 FillStudentMapFro	·	
ReadAcpPipeLine, 18 ReadCumulativeFileLine, 18 ReadCumulativeFileLine, 18 ReadIlStudentsInactive, 15 SetAllStudentsInactive, 15 Students, 20 WriteStudentsToMemory, 15 BuildStudentMap Build, 12 CalculateCumulative Build, 19 CreateInitialCumulativeFile Files, 28 CreateInitialCumulativeFile Files, 25 CreateLockfile Files, 29 CreateSharedMemory MemShare, 35 Data, 21 DATA_NAME_MAX_LENGTH, 22 DATA_NIM_RECORDS, 21 DATA_NAME_MAX_LENGTH Data, 22 DATA_NUM_RECORDS Data, 23 DATA_NUM_RECORDS Data, 25 DATA_NUM_RECORDS Data, 26 DATA_NUM_RECORDS Data, 27 DATA_NUM_RECORDS Data, 27 DATA_NUM_RECORDS Data, 28 DATA_NUM_RECORDS Data, 29 DATA_NUM_RECORDS Data, 29 DATA_NUM_RECORDS Data, 21 DATA_NUM_RECORDS Data, 21 DATA_NUM_RECORDS Data, 21 DATA_NUM_RECORDS Data, 22 DATA_NUM_RECORDS Data, 23 DATA_NUM_RECORDS Data, 21 DATA_NUM_RECORDS DATA_SIZE DATA_NUM_RECORDS DATA_SIZE DATA_	•	
ReadCumulativeFileLine, 18 ReadInitialCumulative, 16 SetAllStudentsInactive, 15 students, 20 WriteStudentsToMemory, 15 BuildStudentMap Build, 12 CreateInitialCumulative Build, 19 CreateInitialCumulativeFile Files, 28 CreateInitialUserDataFile Files, 25 CreateLockfile Files, 29 CreateSharedMemory MemShare, 35 Data, 21 DATA_NAME_MAX_LENGTH Data, 22 DATA_NUM_RECORDS Data, 21 DATA_SIZE DATA_SI		•
ReadInitialCumulative, 16 SetAllStudentsInactive, 15 students, 20 WriteStudents ToMemory, 15 BuildStudentMap Build, 12 Build, 12 CreateInitialCumulativeFile Files, 28 CreateInitialUserDataFile Files, 29 CreateSharedMemory MemShare, 35 Data, 21 DATA_NAME_MAX_LENGTH, 21 DATA_NAME_MAX_LENGTH Data, 22 DATA_NUM_RECORDS Data, 22 DATA_SIZE DAT		
SetAllStudentsInactive, 15 students, 20 WriteStudentsToMemory, 15 BuildStudentMap Build, 12 CalculateCumulative Build, 19 CreateInitialCumulativeFile Files, 28 CreateLockfile Files, 25 CreateLockfile Files, 29 CreateSharedMemory MemShare, 35 Data, 21 DATA_NAME_MAX_LENGTH, 22 DATA_NUM_RECORDS, 21 DATA_NAME_MAX_LENGTH Data, 22 DATA_NAME_MAX_LENGTH Data, 21 DATA_NAME_MAX_LENGTH Data, 22 DATA_NUM_RECORDS Data, 21 DATA_SIZE DAT	•	•
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_NAME_MAX_LENGTH, 22 DATA_NUM_RECORDS, 21 DATA_NAME_MAX_LENGTH Data, 22 DATA_NAME_MAX_LENGTH Data, 21 DATA_NAME_MAX_LENGTH Data, 22 DATA_NUM_RECORDS Data, 21 DATA_NIM_RECORDS Data, 21 DATA_NIM_RECORDS Data, 21 DATA_SIZE DATA_SI		
WriteStudentsToMemory, 15 BuildStudentMap Build, 12 STATIC_USER_CUMULATIVE_FILE, 24 STATIC_USER_DATA_FILE, 24 WriteStudentArrayToFile, 27 CalculateCumulative Build, 19 CreateInitialCumulativeFile Files, 28 CreateLockfile Files, 25 CreateLockfile Files, 29 CreateSharedMemory MemShare, 35 Data, 21 DATA_ID_MAX_LENGTH, 21 DATA_NAME_MAX_LENGTH, 22 DATA_NUM_RECORDS, 21 DATA_ID_MAX_LENGTH Data, 21 DATA_ID_MAX_LENGTH Data, 21 DATA_NAME_MAX_LENGTH Data, 21 DATA_NAME_MAX_LENGTH Data, 22 DATA_NUM_RECORDS Data, 21 DATA_SIZE DATA_SIZ		
BuildStudentMap Build, 12 STATIC_USER_CUMULATIVE_FILE, 24 STATIC_USER_DATA_FILE, 24 WriteStudentArrayToFile, 27 FillstudentMapFromFile Files, 28 CreateInitialCumulativeFile Files, 25 CreateLockfile Files, 29 CreateSharedMemory MemShare, 35 Data, 21 DATA_ID_MAX_LENGTH, 21 DATA_NAME_MAX_LENGTH, 22 DATA_NUM_RECORDS, 21 DATA_SIZE, 22 DATA_NAME_MAX_LENGTH Data, 21 DATA_NAME_MAX_LENGTH Data, 21 DATA_NAME_MAX_LENGTH Data, 22 DATA_NAME_MAX_LENGTH Data, 23 DATA_SIZE DATA		·
Build, 12 CalculateCumulative Build, 19 CreateInitialCumulativeFile Files, 28 CreateInitialUserDataFile Files, 25 CreateLockfile Files, 29 CreateSharedMemory MemShare, 35 Data, 21 DATA_INAME_MAX_LENGTH, 21 DATA_NAME_MAX_LENGTH Data, 21 DATA_NAME_MAX_LENGTH Data, 21 DATA_NAME_MAX_LENGTH Data, 21 DATA_NAME_MAX_LENGTH Data, 21 DATA_NAME_MAX_LENGTH Data, 21 DATA_NAME_MAX_LENGTH Data, 21 DATA_NAME_MAX_LENGTH Data, 22 DATA_NUM_RECORDS Data, 21 DATA_SIZE Data_Names Data, 21 DATA_SIZE Data_AC DATA_SIZE Data, 22 DeleteLockfile Files, 30 Map_Delete, 33 Map_Set, 32		
CalculateCumulative Build, 19 CreateInitialCumulativeFile Files, 28 CreateLockfile Files, 25 CreateLockfile Files, 29 CreateSharedMemory MemShare, 35 Data, 21 DATA_NAME_MAX_LENGTH, 22 DATA_NUM_RECORDS, 21 DATA_NAME_MAX_LENGTH Data, 22 DATA_NAME_MAX_LENGTH Data, 21 DATA_NAME_MAX_LENGTH Data, 22 DATA_NAME_MAX_LENGTH Data, 22 Data_Names Data, 22 DATA_NAME_MAX_LENGTH Data, 22 DeteleLockfile Files, 30 DestroySharedMemory Map_Delete, 33 Map_Set, 32	•	
CalculateCumulative Build, 19 CreateInitialCumulativeFile Files, 28 CreateInitialUserDataFile Files, 25 CreateLockfile Files, 29 CreateSharedMemory MemShare, 35 Data, 21 DATA_NAME_MAX_LENGTH, 22 DATA_ID_MAX_LENGTH Data, 21 DATA_NAME_MAX_LENGTH Data, 22 DATA_NAME_MAX_LENGTH Data, 23 DATA_NAME_MAX_LENGTH Data, 23 DATA_NAME_MAX_LENGTH Data, 23 DATA_NAME_MAX_LENGTH Data, 23 DATA_NAME_MAX_LENGTH DATA_SIZE DATA_NAME_MAX_LENGTH Map_Delete, 33 Map_Delete, 33 Map_Set, 32	Build, 12	·
Build, 19 CreateInitialCumulativeFile Files, 28 CreateInitialUserDataFile Files, 25 CreateLockfile Files, 29 CreateSharedMemory MemShare, 35 Data, 21 DATA_ID_MAX_LENGTH, 21 DATA_SIZE, 22 DATA_NAME_MAX_LENGTH Data, 21 DATA_NAME_MAX_LENGTH Data, 22 DATA_NAME_CORDS Data, 21 DATA_SIZE Data_1Names Data, 22 DATA_NUM_RECORDS Data, 21 DATA_SIZE Data_22 DATA_SIZE Data, 22 DATA_SIZE Data, 22 DATA_SIZE Data, 22 DATA_SIZE Data, 22 DeleteLockfile Files, 30 DestroySharedMemory MemShare, 36 GetMemoryPointer MemShare, 36 Files, 26 GetMemoryPointer MemShare, 36 Files, 36 GetMemoryPointer MemShare, 36 Files, 26 GetMemoryPointer MemShare, 36 Files, 26 GetMemoryPointer MemShare, 36 Files, 36 GetMemoryPointer MemShare, 36 Files, 26 GetMemoryPointer MemShare, 36 Files, 36 Files, 36 GetMemoryPointer MemShare, 36 Files, 36 GetMemoryPointer MemShare, 36 Files, 26 Files, 26 GetMemoryPointer MemShare MemShare Files, 26 Files, 26 Files, 26 Files, 26 Files, 27 Files, 26 Fi	CalculateCumulative	
CreateInitialCumulativeFile Files, 28 CreatesInitialUserDataFile Files, 25 CreateLockfile Files, 29 CreateSharedMemory MemShare, 35 Data, 21 DATA_ID_MAX_LENGTH, 21 DATA_NUM_RECORDS, 21 DATA_ID_MAX_LENGTH Data, 21 DATA_ID_MAX_LENGTH Data, 21 DATA_NAME_MAX_LENGTH Data, 21 DATA_NAME_MAX_LENGTH Data, 21 DATA_NAME_MAX_LENGTH Data, 22 DATA_NOM_RECORDS Data, 21 DATA_SIZE Data, 22 DATA_SIZE Data, 22 DATA_SIZE Data, 22 Detectockfile Files, 30 DestroySharedMemory Map_Delete, 33 Map_Set, 32		•
Files, 28 CreateInitialUserDataFile Files, 25 CreateLockfile Files, 29 CreateSharedMemory MemShare, 35 Data, 21 DATA_ID_MAX_LENGTH, 21 DATA_SIZE Data_Names Data, 21 DATA_NAME_MAX_LENGTH Data, 22 DATA_NAME_MAX_LENGTH DATA_SIZE DATA_NOM_RECORDS Data, 21 DATA_SIZE DATA_NOM_RECORDS Data, 21 DATA_SIZE DATA	•	1 1163, 20
CreateInitialUserDataFile Files, 25 CreateLockfile Files, 29 CreateSharedMemory MemShare, 35 Data, 21 DATA_NAME_MAX_LENGTH, 22 DATA_NUM_RECORDS, 21 DATA_SIZE Data_Names Data, 22 DATA_NUM_RECORDS Data, 21 DATA_NUM_RECORDS Data, 21 DATA_NUM_RECORDS Data, 22 DATA_NUM_RECORDS Data, 21 DATA_SIZE DATA_NUM_RECORDS Data, 21 DATA_SIZE DATA_NUM_RECORDS Data, 21 DATA_SIZE DATA_NUM_RECORDS Data, 21 DATA_SIZE DATA_NUM_RECORDS DATA_SIZE D		GetMemoryPointer
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_ID_MAX_LENGTH Data, 21 DATA_ID_MAX_LENGTH DATA_SIZE, 22 DATA_NUM_RECORDS, 21 DATA_SIZE, 22 DATA_NAME_MAX_LENGTH Data, 21 DATA_NAME_MAX_LENGTH Data, 22 DATA_NUM_RECORDS Data, 21 DATA_SIZE Data_Names Data, 22 DATA_NUM_RECORDS Data, 21 DATA_SIZE Data_SIZ	•	
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_ID_MAX_LENGTH Data, 21 DATA_ID_MAX_LENGTH DATA_SIZE, 22 DATA_NUM_RECORDS, 21 DATA_SIZE, 22 DATA_NAME_MAX_LENGTH Data, 21 DATA_NAME_MAX_LENGTH Data, 22 DATA_NAME_MAX_LENGTH Data, 22 DATA_NAME_MAX_LENGTH Data, 22 DATA_NAME_MAX_LENGTH Data, 22 DATA_NUM_RECORDS Data, 21 DATA_SIZE DAT		,
Files, 29 CreateSharedMemory MemShare, 35 Data, 21 DATA_ID_MAX_LENGTH, 21 DATA_NAME_MAX_LENGTH, 22 Data_Names, 22 DATA_ID_MAX_LENGTH DATA_ID_MAX_LENGTH DATA_SIZE, 22 DATA_NAME_MAX_LENGTH Data, 21 DATA_NAME_MAX_LENGTH Data, 22 DATA_NUM_RECORDS Data_Names Data, 22 DATA_NUM_RECORDS Data, 21 DATA_SIZE DA	•	HelpCommand
CreateSharedMemory MemShare, 35 Data, 21 DATA_ID_MAX_LENGTH, 21 DATA_NAME_MAX_LENGTH, 22 Data_Names, 22 DATA_NUM_RECORDS, 21 DATA_ID_MAX_LENGTH Data, 21 DATA_ID_MAX_LENGTH Data, 21 DATA_SIZE, 22 DATA_NAME_MAX_LENGTH Data, 22 DATA_NUM_RECORDS Data_Names Data, 22 DATA_NUM_RECORDS Data_SIZE Data_Names Data, 22 DATA_NUM_RECORDS Data, 21 DATA_SIZE Data_SIZE Data, 22 DeleteLockfile Files, 30 DestroySharedMemory MemShare, 35 IndicateRereadDone Process, 39 IndicateRereadDne Process, 38 Initialize Process, 41 Is_stopping Process, 48 Ical Select Is_stopping Process, 48 Ical Select Is_stopping Process, 48 Ical Select Is_stopping Process, 39 Process, 48 Ical Select Is_stopping Process, 41 Ical Select Process, 48 Ical Select		Process, 43
MemShare, 35 Data, 21 DATA_ID_MAX_LENGTH, 21 DATA_NAME_MAX_LENGTH, 22 Data_Names, 22 DATA_NUM_RECORDS, 21 DATA_ID_MAX_LENGTH Data, 21 DATA_SIZE, 22 DATA_NAME_MAX_LENGTH Data, 22 DATA_NAME_MAX_LENGTH Data, 22 DATA_NUM_RECORDS Data_Names Data, 22 DATA_NUM_RECORDS Data_Size, 22 DATA_NUM_RECORDS Data_Size, 22 DATA_NAME_MAX_LENGTH Data, 22 DATA_NUM_RECORDS Data_Size Map_Size, 33 Map_Get, 33 Map_Set, 32		
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_NAME_MAX_LENGTH Data, 22 DATA_NAME_MAX_LENGTH Data, 22 DATA_NUM_RECORDS Data_Names Data, 22 DATA_NUM_RECORDS Data, 21 DATA_NUM_RECORDS Data, 21 DATA_NUM_RECORDS Data, 21 DATA_SIZE Data_22 DATA_NUM_RECORDS Data, 21 DATA_SIZE Data, 22 DeleteLockfile Files, 30 DestroySharedMemory MamShare 35 Initialize Process, 38 Initialize Process, 41 is_stopping Process, 41 is_stopping Process, 41 Is_stopping Process, 42 Initialize Process, 41 Is_stopping Process, 41 Is_stopping Process, 41 Is_stopping Process, 42 Process, 42 Initialize Process, 41 Is_stopping Process, 42 Initialize Process, 42 Process, 42 Process, 41 Is_stopping Process, 42 Initialize Process, 42 Process, 41 Is_stopping Process, 42 Process, 42 Initialize Process, 42 Process, 42 Process, 41 Is_stopping Process, 42 Process, 42 Process, 41 Initialize Process, 42 Process, 41 Is_stopping Process, 42 Process, 41 Is_stopping Process, 42 Process, 41 Is_stopping Process, 42 Process, 42 Process, 42 Process, 42 Process, 42 And		
DATA_ID_MAX_LENGTH, 21 DATA_NAME_MAX_LENGTH, 22 Data_Names, 22 DATA_NUM_RECORDS, 21 DATA_ID_MAX_LENGTH Data, 21 DATA_NAME_MAX_LENGTH Data, 22 DATA_NAME_MAX_LENGTH Data, 22 DATA_NAME_MAX_LENGTH Data, 22 DATA_NAME_Stare Data_SIZE DATA_NUM_RECORDS Data_SIZE Map_SizE Map_Delete, 33 Map_Get, 33 Map_Set, 32	,	
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_NUM_RECORDS Data, 21 DATA_NUM_RECORDS Data, 21 DATA_SIZE DeleteLockfile Files, 30 DestroySharedMemory Map_Share_35 Map_Set, 32	Data, 21	
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_NUM_RECORDS Data, 21 DATA_SIZE Data_SIZE Data_Size Data, 22 DeleteLockfile Files, 30 DestroySharedMemory MomShero, 25	DATA_ID_MAX_LENGTH, 21	
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_NUM_RECORDS Data, 21 DATA_SIZE Data_SIZE Data_SizE Data, 22 DeleteLockfile Files, 30 DestroySharedMemory MamShare, 25 Is_stopping Process, 48 IsRereadNeeded Process, 39 LOCKFILE Files, 24 main main.c main.c, 64 main.c main, 64 Map, 30 Map_Delete, 33 Map_Get, 33 Map_Set, 32	DATA_NAME_MAX_LENGTH, 22	
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 Map_Slare_SIZE Map_Delete, 33 Map_Delete, 33 Map_Set, 32 Map_Set, 32	Data_Names, 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 MamShare, 25 ISRereadNeeded Process, 39 LOCKFILE Files, 24 main main.c main.c, 64 main.c main, 64 Map, 30 Map_Delete, 33 Map_Get, 33 Map_Set, 32	DATA_NUM_RECORDS, 21	
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 MamShare, 35 DATA_NUM_RECORDS Main.c main.c main.c main.c Map_Delete, 33 Map_Get, 33 Map_Set, 32	DATA_SIZE, 22	
DATA_NAME_MAX_LENGTH Data, 22 Data_Names Data, 22 DATA_NUM_RECORDS Data, 21 DATA_SIZE Data, 22 DeleteLockfile Files, 30 DestroySharedMemory MamShare, 35 LOCKFILE Files, 24 main main.c main.c, 64 main.c main, 64 Map, 30 Map_Delete, 33 Map_Get, 33 Map_Set, 32	DATA_ID_MAX_LENGTH	
Data, 22 EOCKFILE Data_Names Files, 24 Data, 22 main DATA_NUM_RECORDS main.c, 64 Data, 21 main.c DATA_SIZE main, 64 Data, 22 Map, 30 DeleteLockfile Map_Delete, 33 Files, 30 Map_Get, 33 DestroySharedMemory Map_Set, 32	Data, 21	Process, 39
Data, 22 Data_Names Data, 22 DATA_NUM_RECORDS Data, 21 DATA_SIZE Data, 22 DeleteLockfile Files, 30 DestroySharedMemory MamShare, 35 Files, 24 main main.c main.c, 64 main.c Map_Delete, 33 Map_Delete, 33 Map_Get, 33 Map_Set, 32	DATA_NAME_MAX_LENGTH	LOCKELLE
Data_Names Data, 22 DATA_NUM_RECORDS Data, 21 DATA_SIZE Data, 22 DeleteLockfile Files, 30 DestroySharedMemory MamShare, 35 main main.c main, 64 Map, 30 Map_Delete, 33 Map_Get, 33 Map_Get, 33 Map_Set, 32	Data, 22	
DATA_NUM_RECORDS Data, 21 DATA_SIZE Data, 22 DeleteLockfile Files, 30 DestroySharedMemory Map_Stare 35 main.c, 64 main.c main, 64 Map, 30 Map_Delete, 33 Map_Delete, 33 Map_Get, 33 Map_Set, 32	Data_Names	FileS, 24
DATA_NUM_RECORDS Data, 21 DATA_SIZE Data, 22 DeleteLockfile Files, 30 DestroySharedMemory Map_Share, 35 main.c, 64 main.c main, 64 Map, 30 Map_Delete, 33 Map_Delete, 33 Map_Get, 33 Map_Set, 32	Data, 22	main
Data, 21 DATA_SIZE Data, 22 DeleteLockfile Files, 30 DestroySharedMemory Map_Share, 35 main.c main, 64 Map, 30 Map_Delete, 33 Map_Get, 33 Map_Get, 33 Map_Set, 32	DATA_NUM_RECORDS	
DATA_SIZE main, 64 Data, 22 Map, 30 DeleteLockfile Map_Delete, 33 Files, 30 Map_Get, 33 DestroySharedMemory Map_Set, 32	Data, 21	
Data, 22 Map, 30 DeleteLockfile Files, 30 DestroySharedMemory Map_Stare, 35 Map_Get, 33 Map_Set, 32	DATA_SIZE	
DeleteLockfile Map_Delete, 33 Files, 30 Map_Get, 33 DestroySharedMemory Map_Set, 32	Data, 22	
Files, 30 Map_Get, 33 DestroySharedMemory Map_Set, 32	DeleteLockfile	•
DestroySharedMemory Map_Set, 32	Files, 30	• —
MomChara 25	DestroySharedMemory	
	MemShare, 35	NewMap, 32

76 INDEX

map, 53	ReadInitialCumulative
Map_Delete	Build, 16
Map, 33	ReleaseMemoryPointer
Map_Get	MemShare, 37
Map, 33	ResetCommand
map_result, 54	Process, 46
Map_Set	RunCommand
Map, 32	Process, 44
MEM_KEY	RunHeadless
MemShare, 34	Process, 47
MEM_PERMISSIONS	O IANO I I I I
MemShare, 35	SetAllStudentsInactive
MEM_SIZE	Build, 15
MemShare, 35	SignalHandle
MemShare, 34	Process, 40
CreateSharedMemory, 35	src/server/Build.c, 57
DestroySharedMemory, 35	src/server/Build.h, 60
GetMemoryPointer, 36	src/server/Data.c, 60
MEM_KEY, 34	src/server/Data.h, 61
MEM_PERMISSIONS, 35	src/server/Files.c, 61
MEM_SIZE, 35	src/server/Files.h, 63
ReleaseMemoryPointer, 37	src/server/main.c, 63, 65
	src/server/map.c, 66
NewMap	src/server/map.h, 68
Map, 32	src/server/memShare.c, 69
Din - A - a T- Ob - d - a + M - a	src/server/memShare.h, 69
PipeAcpToStudentMap	src/server/Process.c, 69
Build, 17	src/server/Process.h, 73
PipeWhoToStudentMap	src/server/util.c, 73
Build, 13	src/server/util.h, 74
PopulateStudents	STATIC_USER_CUMULATIVE_FILE
Build, 11	Files, 24
Process, 37	STATIC_USER_DATA_FILE
HelpCommand, 43	Files, 24
IndicateRereadDone, 39	StopCommand
IndicateRereadNeeded, 38	Process, 45
Initialize, 41	Student, 54
is_stopping, 48	students
IsRereadNeeded, 39	Build, 20
Process, 42	
ResetCommand, 46	TerminateExistingServer
RunCommand, 44	Process, 38
RunHeadless, 47	Trim
SignalHandle, 40	Util, 51
StopCommand, 45	Util, 48
TerminateExistingServer, 38	
ProcessWhoLine	RandomFlag, 50 RandomFloat, 49
Build, 14	RandomInteger, 49
DandamElag	
RandomFlag	Trim, 51
Util, 50 RandomFloat	WriteStudentArrayToFile
	Files, 27
Util, 49	WriteStudentsToMemory
RandomInteger	Build, 15
Util, 49	Dana, 10
ReadAcpPipeLine	
Build, 18 ReadCumulativeFileLine	
Build, 18	
Dulla, 10	