ecet4640-lab4

Generated by Doxygen 1.9.3

1 ecet4640-lab4	1
1.0.1 About	2
2 Data Structure Index	3
2.1 Data Structures	3
3 File Index	5
3.1 File List	5
4 Data Structure Documentation	7
4.1 map Struct Reference	7
4.1.1 Detailed Description	7
4.1.2 Field Documentation	8
4.1.2.1 buckets	8
4.1.2.2 size	8
4.2 map_result Struct Reference	8
4.2.1 Detailed Description	8
4.2.2 Field Documentation	8
4.2.2.1 data	9
4.2.2.2 found	9
4.3 Student Struct Reference	9
4.3.1 Detailed Description	9
4.3.2 Field Documentation	9
4.3.2.1 active	10
4.3.2.2 age	10
4.3.2.3 fullName	10
4.3.2.4 gpa	10
4.3.2.5 lastLogin	10
4.3.2.6 loginDuration	10
4.3.2.7 userID	10
5 File Documentation	11
5.1 src/server/Build.c File Reference	
5.1.1 Function Documentation	
5.1.1.1 BuildStudentMap()	
5.1.1.2 CalculateCumulative()	
5.1.1.3 PopulateStudents()	
5.1.1.4 ProcessWhoLine()	
5.1.1.5 ReadACP()	
5.1.1.6 ReadAcpPipeLine()	
5.1.1.7 ReadCumulativeFileLine()	
5.1.1.8 ReadInitialCumulative()	
5.1.1.9 SetAllStudentsInactive()	
5.1.1.10 UpdateFromWho()	
Similar Spaces Similar Significant Signifi	0

5.1.1.11 WriteStudentsToMemory()	20
5.1.2 Variable Documentation	21
5.1.2.1 dirty	21
5.1.2.2 students	21
5.2 Build.c	22
5.3 src/server/Build.h File Reference	24
5.3.1 Function Documentation	26
5.3.1.1 BuildStudentMap()	26
5.3.1.2 CalculateCumulative()	27
5.3.1.3 PopulateStudents()	28
5.3.1.4 ProcessWhoLine()	28
5.3.1.5 ReadACP()	29
5.3.1.6 ReadAcpPipeLine()	30
5.3.1.7 ReadCumulativeFileLine()	31
5.3.1.8 ReadInitialCumulative()	32
5.3.1.9 SetAllStudentsInactive()	33
5.3.1.10 UpdateFromWho()	34
5.3.1.11 WriteStudentsToMemory()	34
5.3.2 Variable Documentation	35
5.3.2.1 dirty	35
5.3.2.2 students	35
5.4 Build.h	36
5.5 src/server/Data.c File Reference	36
5.5.1 Variable Documentation	37
5.5.1.1 Data_IDs	37
5.5.1.2 Data_Names	37
5.6 Data.c	38
5.7 src/server/Data.h File Reference	38
5.7.1 Macro Definition Documentation	39
5.7.1.1 DATA_ID_MAX_LENGTH	39
5.7.1.2 DATA_NAME_MAX_LENGTH	40
5.7.1.3 DATA_NUM_RECORDS	40
5.7.1.4 DATA_SIZE	40
5.7.2 Variable Documentation	40
5.7.2.1 Data_IDs	40
5.7.2.2 Data_Names	40
5.8 Data.h	41
5.9 src/server/Files.c File Reference	41
5.9.1 Function Documentation	42
5.9.1.1 CreateInitialCumulativeFile()	42
5.9.1.2 CreateInitialUserDataFile()	43
5.9.1.3 FileExists()	44

5.9.1.4 FillStudentMapFromFile()	44
5.9.1.5 WriteStudentArrayToFile()	45
5.10 Files.c	46
5.11 src/server/Files.h File Reference	47
5.11.1 Macro Definition Documentation	49
5.11.1.1 STATIC_USER_CUMULATIVE_FILE	49
5.11.1.2 STATIC_USER_DATA_FILE	49
5.11.2 Function Documentation	49
5.11.2.1 CreateInitialCumulativeFile()	49
5.11.2.2 CreateInitialUserDataFile()	50
5.11.2.3 FileExists()	51
5.11.2.4 FillStudentMapFromFile()	52
5.11.2.5 WriteStudentArrayToFile()	53
5.12 Files.h	53
5.13 src/server/main.c File Reference	54
5.13.1 Function Documentation	54
5.13.1.1 main()	55
5.14 main.c	55
5.15 src/server/map.c File Reference	56
5.15.1 Function Documentation	57
5.15.1.1 _bucket_delete()	57
5.15.1.2 _bucket_get()	58
5.15.1.3 _bucket_insert()	58
5.15.1.4 hash_log2()	58
5.15.1.5 hash_string()	59
5.15.1.6 hash_upperLimit()	60
5.15.1.7 Map_Delete()	60
5.15.1.8 Map_Get()	61
5.15.1.9 Map_Set()	62
5.15.1.10 NewMap()	63
5.15.2 Variable Documentation	64
5.15.2.1 char_ratio	64
5.16 map.c	64
5.17 src/server/map.h File Reference	67
5.17.1 Function Documentation	67
5.17.1.1 hash_log2()	68
5.17.1.2 hash_string()	69
5.17.1.3 hash_upperLimit()	70
5.17.1.4 Map_Delete()	71
5.17.1.5 Map_Get()	71
5.17.1.6 Map_Set()	72
5.17.1.7 NewMap()	73

5.18 map.h	74
5.19 src/server/memShare.c File Reference	75
5.19.1 Function Documentation	75
5.19.1.1 CreateSharedMemory()	76
5.19.1.2 DestroySharedMemory()	76
5.19.1.3 GetMemoryPointer()	76
5.19.1.4 ReleaseMemoryPointer()	77
5.20 memShare.c	77
5.21 src/server/memShare.h File Reference	78
5.21.1 Macro Definition Documentation	79
5.21.1.1 MEM_KEY	79
5.21.1.2 MEM_PERMISSIONS	79
5.21.1.3 MEM_SIZE	79
5.21.2 Function Documentation	79
5.21.2.1 CreateSharedMemory()	80
5.21.2.2 DestroySharedMemory()	80
5.21.2.3 GetMemoryPointer()	80
5.21.2.4 ReleaseMemoryPointer()	81
5.22 memShare.h	81
5.23 src/server/Process.c File Reference	82
5.23.1 Function Documentation	83
5.23.1.1 CreateLockfile()	83
5.23.1.2 DeleteLockfile()	84
5.23.1.3 DoesLockfileExist()	84
5.23.1.4 HelpCommand()	85
5.23.1.5 IndicateRereadDone()	86
5.23.1.6 IndicateRereadNeeded()	86
5.23.1.7 Initialize()	87
5.23.1.8 IsRereadNeeded()	88
5.23.1.9 Process()	88
5.23.1.10 ResetCommand()	89
5.23.1.11 RunCommand()	90
5.23.1.12 RunHeadless()	92
5.23.1.13 SignalHandle()	92
5.23.1.14 StopCommand()	93
5.23.1.15 TerminateExistingServer()	94
5.23.2 Variable Documentation	94
5.23.2.1 initial_cumulative_times	94
5.23.2.2 is_stopping	94
5.23.2.3 student_map	95
5.24 Process.c	95
5.25 erc/server/Process h File Reference	ac

Index	121
5.30 util.h	. 119
5.29.1.3 RandomInteger()	
5.29.1.2 RandomFloat()	
5.29.1.1 RandomFlag()	
5.29.1 Function Documentation	
5.29 src/server/util.h File Reference	
5.28 util.c	
5.27.1.3 RandomInteger()	
5.27.1.2 RandomFloat()	
5.27.1.1 RandomFlag()	
5.27.1 Function Documentation	
5.27 src/server/util.c File Reference	
5.26 Process.h	. 113
5.25.3.2 student_map	. 112
5.25.3.1 is_stopping	. 112
5.25.3 Variable Documentation	. 112
5.25.2.16 TerminateExistingServer()	. 112
5.25.2.15 StopCommand()	. 111
5.25.2.14 SignalHandle()	. 110
5.25.2.13 RunHeadless()	. 110
5.25.2.12 RunCommand()	. 108
5.25.2.11 ResetCommand()	. 107
5.25.2.10 Process()	. 106
5.25.2.9 IsRereadNeeded()	. 106
5.25.2.8 Initialize()	. 105
5.25.2.7 IndicateRereadNeeded()	. 104
5.25.2.6 IndicateRereadDone()	. 104
5.25.2.5 HelpCommand()	. 103
5.25.2.4 DoesLockfileExist()	. 102
5.25.2.3 DeleteLockfile()	. 102
5.25.2.2 CreateLockfile()	. 101
5.25.2.1 ClearCommand()	. 101
5.25.2 Function Documentation	
5.25.1.1 LOCKFILE	
5.25.1 Macro Definition Documentation	. 100

## **Chapter 1**

## ecet4640-lab4

This program reads user information using the who command and publishes that information 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 call graph.

Here is a general call graph at a glance:

2 ecet4640-lab4

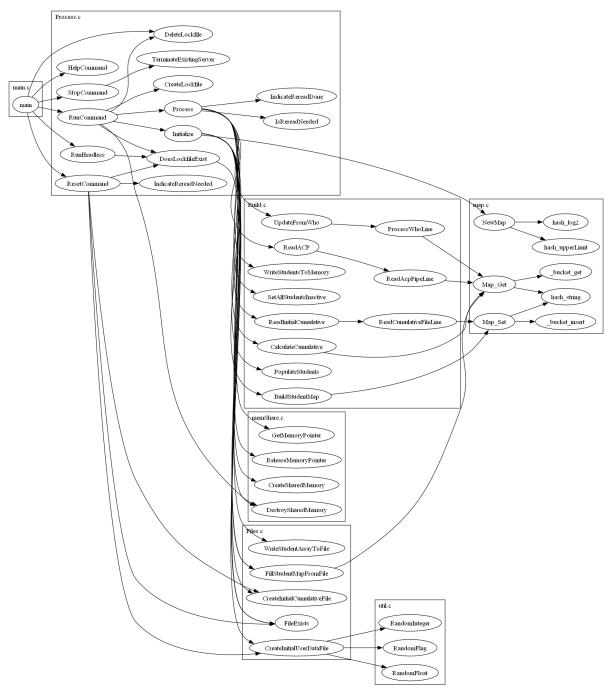


Figure 1.1 Program Call graph

## 1.0.1 About

Created for Computer Networking, September 2023, by Karl Miller, Paul Shriner, and Christian Messmer.

# **Chapter 2**

# **Data Structure Index**

## 2.1 Data Structures

Here are the data structures with brief descriptions:

map		
	A map. Stores key-value pairs for near constant lookup and insertion time	7
map_res	sult control of the second	
	The result of a map retrieval	8
Student		
	The student data type	9

4 Data Structure Index

# **Chapter 3**

# File Index

## 3.1 File List

Here is a list of all files with brief descriptions:

src/server/Build.c	
src/server/Build.h	24
src/server/Data.c	
src/server/Data.h	
src/server/Files.c	
src/server/Files.h	
src/server/main.c	
src/server/map.c	
src/server/map.h	
src/server/memShare.c	
src/server/memShare.h	
src/server/Process.c	
src/server/Process.h	
src/server/util.c	
src/server/util.h	116

6 File Index

## **Chapter 4**

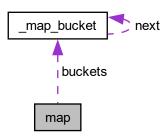
## **Data Structure Documentation**

## 4.1 map Struct Reference

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

#include <map.h>

Collaboration diagram for map:



#### **Data Fields**

- int size
- struct \_map\_bucket \* buckets

### 4.1.1 Detailed Description

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

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 99 of file map.h.

#### 4.1.2 Field Documentation

#### 4.1.2.1 buckets

```
struct _map_bucket* buckets
```

Definition at line 104 of file map.h.

#### 4.1.2.2 size

int size

Definition at line 102 of file map.h.

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

• src/server/map.h

## 4.2 map\_result Struct Reference

The result of a map retrieval.

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

#### **Data Fields**

- short found
- void \* data

#### 4.2.1 Detailed Description

The result of a map retrieval.

Definition at line 110 of file map.h.

### 4.2.2 Field Documentation

#### 4.2.2.1 data

void\* data

Definition at line 115 of file map.h.

#### 4.2.2.2 found

short found

Definition at line 113 of file map.h.

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

src/server/map.h

### 4.3 Student Struct Reference

The student data type.

#include <Data.h>

#### **Data Fields**

- char userID [DATA\_ID\_MAX\_LENGTH]
- char fullName [DATA\_NAME\_MAX\_LENGTH]
- short age
- float gpa
- short active
- time\_t lastLogin
- · int loginDuration

## 4.3.1 Detailed Description

The student data type.

Definition at line 38 of file Data.h.

#### 4.3.2 Field Documentation

#### 4.3.2.1 active

short active

Definition at line 44 of file Data.h.

#### 4.3.2.2 age

short age

Definition at line 42 of file Data.h.

#### 4.3.2.3 fullName

char fullName[DATA\_NAME\_MAX\_LENGTH]

Definition at line 41 of file Data.h.

#### 4.3.2.4 gpa

float gpa

Definition at line 43 of file Data.h.

#### 4.3.2.5 lastLogin

time\_t lastLogin

Definition at line 45 of file Data.h.

#### 4.3.2.6 loginDuration

int loginDuration

Definition at line 46 of file Data.h.

### 4.3.2.7 userID

char userID[DATA\_ID\_MAX\_LENGTH]

Definition at line 40 of file Data.h.

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

• src/server/Data.h

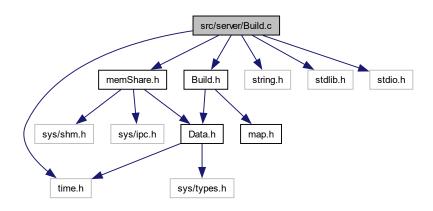
## **Chapter 5**

## **File Documentation**

#### 5.1 src/server/Build.c File Reference

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

Include dependency graph for Build.c:



#### **Functions**

- void PopulateStudents (char \*\*studentIDs, char \*\*studentNames, int arsize)
- void BuildStudentMap (map \*stmap, Student \*studentArr, int studentArrLength)
- int UpdateFromWho (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 ReadACP (map \*st\_map)
- void ReadCumulativeFileLine (map \*cum\_map, char \*acp\_line)
- int ReadAcpPipeLine (map \*stmap, char \*acp\_line)
- void CalculateCumulative (Student \*stud\_arr, int stud\_arr\_len, map \*cum\_map)

#### **Variables**

• Student \* students

Definitions for functions that populate data structures.

• short dirty = 1

#### 5.1.1 Function Documentation

#### 5.1.1.1 BuildStudentMap()

Given a student array, populates a student map, where the student IDs are the key, and the values are pointers to the 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 27 of file Build.c.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 5.1.1.2 CalculateCumulative()

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

#### Warning

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.	
cum_map	A map mapping studentIds to their cumulative login time when the server was started.	

Definition at line 205 of file Build.c.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 5.1.1.3 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 15 of file Build.c.

Here is the caller graph for this function:



#### 5.1.1.4 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

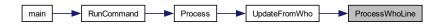
0 if success, -1 if the student was not found in the map.

Definition at line 60 of file Build.c.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 5.1.1.5 ReadACP()

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**

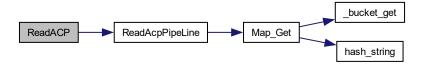
st_map	The students map.

Returns

0 on success.

Definition at line 152 of file Build.c.

Here is the call graph for this function:



Here is the caller graph for this function:



### 5.1.1.6 ReadAcpPipeLine()

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

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

#### **Parameters**

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

## Returns

-1 if acp\_line is NULL or length is less than 1, otherwise 0.

Definition at line 186 of file Build.c.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 5.1.1.7 ReadCumulativeFileLine()

Reads a single line from the initial cumulative file and updates the map so that userID maps to a float 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**

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

#### Returns

-1 ...

Definition at line 176 of file Build.c.

Here is the call graph for this function:



Here is the caller graph for this function:



### 5.1.1.8 ReadInitialCumulative()

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

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

#### Parameters

cum_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 135 of file Build.c.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 5.1.1.9 SetAllStudentsInactive()

Sets the 'active' member 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 108 of file Build.c.

Here is the caller graph for this function:



#### 5.1.1.10 UpdateFromWho()

```
int UpdateFromWho (  \label{eq:map * stmap} \mbox{ } \mbox{$\mathsf{map}$ * stmap $\mathsf{p}$ } \mbox{} \mbox{}
```

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

#### **Parameters**

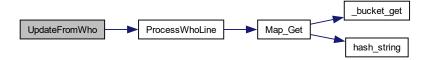
```
stmap The student map.
```

#### Returns

0 if succesful, otherwise nonzero.

Definition at line 40 of file Build.c.

Here is the call graph for this function:



Here is the caller graph for this function:



## 5.1.1.11 WriteStudentsToMemory()

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

Writes the students array to the location specified by mem\_ptr (eg. the shared memory segment).

#### **Parameters**

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

Definition at line 117 of file Build.c.

Here is the caller graph for this function:



#### 5.1.2 Variable Documentation

#### 5.1.2.1 dirty

short dirty = 1

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

Definition at line 38 of file Build.c.

#### 5.1.2.2 students

Student\* students

Definitions for functions that populate data structures.

Declarations for functions that populate data structures.

Definition at line 13 of file Build.c.

#### 5.2 Build.c

#### Go to the documentation of this file.

```
00001
00004 #include "Build.h"
00005 #include "memShare.h"
00006 #include <string.h>
00007 #include <stdlib.h>
00008 #include <stdio.h>
00009 #include <time.h>
00010
00011 // ~~~~~~ Data Structures ~~~~~~
00012
00013 Student *students;
00014
00015 void PopulateStudents(char **studentIDs, char **studentNames, int arsize)
00016 {
00017
          students = malloc(sizeof(Student) * arsize);
00018
          int i;
00019
          for (i = 0; i < arsize; i++)</pre>
00020
         {
00021
              strcpy(students[i].userID, studentIDs[i]);
00022
              strcpy(students[i].fullName, studentNames[i]);
00023
              // students[i].age = randAge(18, 22);
00024
00025 }
00026
00027 void BuildStudentMap(map *stmap, Student *studentArr, int studentArrLength)
00028 {
00029
00030
          for (i = 0; i < studentArrLength; i++)</pre>
00031
00032
              Map_Set(stmap, studentArr[i].userID, (void *)(&studentArr[i]));
00033
00034 }
00035
00036 // ~~~~~~ Processing ~~~~~~
00037
00038 short dirty = 1; // start dirty
00039
00040 int UpdateFromWho(map *stmap)
00041 {
00042
          char command[4] = "who";
          char line[100];
00044
          FILE *fpipe;
00045
          fpipe = popen(command, "r");
00046
          if (fpipe == NULL)
00047
         {
00048
              return -1:
00049
          }
00050
00051
          while (fgets(line, sizeof(line), fpipe) != NULL)
00052
          {
00053
              ProcessWhoLine(stmap, line, strlen(line));
00054
00055
          pclose(fpipe);
00056
00057
00058 }
00059
00060 int ProcessWhoLine(map *stmap, char *whoLine, int whoLineLength)
00061 {
00062
          char userId[20];
00063
          char dateString[50];
00064
          char timeString[20];
00065
          int read_total = 0;
00066
          int read;
          sscanf(whoLine, " %s %n", userId, &read);
00067
00068
          read_total += read;
00069
          map_result mr = Map_Get(stmap, userId);
00070
00071
             (!mr.found)
          { // if we can't find that person in the map, return early
00072
00073
             return -1:
00074
00075
          Student *student = (Student *)mr.data;
00076
00077
          sscanf(whoLine + read_total, " %s %n", dateString, &read); // will be thrown away. eg 'pts/1'
00078
          read_total += read;
          sscanf(whoLine + read_total, " %s %n", dateString, &read); // read the date string
00079
08000
          read total += read;
          sscanf(whoLine + read_total, " %s %n", timeString, &read); // read the time string
00081
00082
          strcat(dateString, " ");
00083
          strcat(dateString, timeString); // catenate the time string back to the date string
00084
```

5.2 Build.c 23

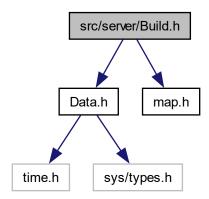
```
time_t now = time(NULL);
00086
          struct tm dtime = *localtime(&now);
00087
          dtime.tm_sec = 0;
00088
00089
          memset(&dtime, 0, sizeof(struct tm));
00090
          00091
      &(dtime.tm_hour), &(dtime.tm_min));
00092
00093
          dtime.tm_year -= 1900;
00094
          dtime.tm\_mon -= 1;
00095
          dtime.tm hour -= 1;
00096
00097
          time_t parsed_time = mktime(&dtime);
00098
00099
          if (student->lastLogin != parsed_time)
00100
00101
              student->lastLogin = parsed_time;
00102
              dirty = 1;
00103
00104
          student->active = 1;
00105
          return 0;
00106 }
00107
00108 void SetAllStudentsInactive(Student *stud_arr, int arr_len)
00109 {
00110
00111
          for (i = 0; i < arr_len; i++)</pre>
00112
00113
              stud_arr[i].active = 0;
00114
00115 }
00116
00117 void WriteStudentsToMemory(void *mem_ptr, Student *stud_arr, int arr_len)
00118 {
          Student *memloc = (Student *)mem_ptr;
00119
00120
          int i;
00121
          for (i = 0; i < arr_len; i++)</pre>
00122
          {
00123
              strcpy(memloc[i].userID, stud_arr[i].userID);
00124
              strcpy(memloc[i].fullName, stud_arr[i].fullName);
             memloc[i].age = stud_arr[i].age;
memloc[i].gpa = stud_arr[i].gpa;
00125
00126
00127
              memloc[i].active = stud_arr[i].active;
00128
              memloc[i].lastLogin = stud_arr[i].lastLogin;
00129
              memloc[i].loginDuration = stud_arr[i].loginDuration;
00130
          }
00131 }
00132
00133 // ~~~~~~ Cumulative Processing ~~~~~~
00134
00135 int ReadInitialCumulative(map *time_map, char *filename)
00136 {
00137
          FILE *file = fopen(filename, "r");
          char line[100];
00138
00139
          if (file == NULL)
00140
00141
00142
00143
          while (fgets(line, sizeof(line), file) != NULL)
00144
00145
              ReadCumulativeFileLine(time map, line);
00146
          }
00147
00148
          fclose(file);
00149
          return 0;
00150 }
00151
00152 int ReadACP (map *st_map)
00153 {
00154
          char command[6] = "ac -p";
00155
          char line[300];
00156
          FILE *fpipe;
          fpipe = popen(command, "r");
if (fpipe == NULL)
00157
00158
00159
00160
00161
00162
          int err;
          while (fgets(line, sizeof(line), fpipe) != NULL)
00163
00164
00165
              err = ReadAcpPipeLine(st_map, line);
00166
              if (err)
00167
              {
00168
                  printf("\nError %d reading acp pipeline.", err);
00169
                  break;
00170
              }
```

```
00171
00172
           pclose(fpipe);
00173
           return 0;
00174 }
00175
00176 void ReadCumulativeFileLine(map *cum_map, char *acp_line)
00177 {
00178
00179
           float minutes;
           sscanf(acp_line, " %s %f ", userId, &minutes);
00180
           // int seconds = (int) (minutes * 60)
long seconds = (long) (minutes * 60);
00181
00182
00183
           Map_Set (cum_map, userId, (void *) seconds);
00184 }
00185
00186 int ReadAcpPipeLine(map *stmap, char *acp_line)
00187 {
00188
           if (acp_line == NULL || strlen(acp_line) < 1)</pre>
00189
00190
                return -1;
00191
00192
           char userId[40];
00193
           float minutes;
           sscanf(acp_line, "%s %f", userId, &minutes);
map_result result = Map_Get(stmap, userId);
00194
00195
00196
           if (result.found)
00197
00198
                Student *student = (Student *)result.data;
               int seconds = (int) (minutes * 60);
student->loginDuration = seconds;
00199
00200
00201
00202
           return 0;
00203 }
00204
00205 void CalculateCumulative(Student *stud_arr, int stud_arr_len, map *cum_map)
00206 {
00207
           int i;
           for (i = 0; i < stud_arr_len; i++)</pre>
00209
           {
00210
                map_result result = Map_Get(cum_map, stud_arr[i].userID);
00211
                if (result.found)
00212
               {
00213
                    long time_at_server_start = (long)result.data;
00214
                    stud_arr[i].loginDuration = stud_arr[i].loginDuration - time_at_server_start;
00215
00216
           }
00217 }
```

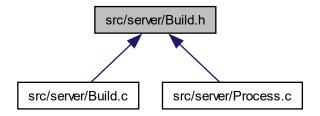
## 5.3 src/server/Build.h File Reference

```
#include "Data.h"
#include "map.h"
```

Include dependency graph for Build.h:



This graph shows which files directly or indirectly include this file:



#### **Functions**

- void PopulateStudents (char \*\*studentIDs, char \*\*studentNames, int arsize)
- void BuildStudentMap (map \*stmap, Student \*studentArr, int studentArrLength)
- int UpdateFromWho (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 ReadACP (map \*st\_map)
- void ReadCumulativeFileLine (map \*cum map, char \*acp line)
- int ReadAcpPipeLine (map \*stmap, char \*acp\_line)
- void CalculateCumulative (Student \*stud\_arr, int stud\_arr\_len, map \*cum\_map)

#### **Variables**

• Student \* students

Declarations for functions that populate data structures.

short dirty

#### 5.3.1 Function Documentation

#### 5.3.1.1 BuildStudentMap()

Given a student array, populates a student map, where the student IDs are the key, and the values are pointers to the 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 27 of file Build.c.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 5.3.1.2 CalculateCumulative()

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

#### Warning

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.
cum_map	A map mapping studentIds to their cumulative login time when the server was started.

Definition at line 205 of file Build.c.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 5.3.1.3 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 15 of file Build.c.

Here is the caller graph for this function:



#### 5.3.1.4 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

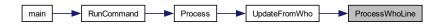
0 if success, -1 if the student was not found in the map.

Definition at line 60 of file Build.c.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 5.3.1.5 ReadACP()

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**

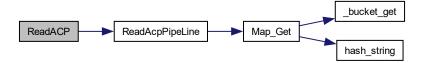
st_map	The students map.

Returns

0 on success.

Definition at line 152 of file Build.c.

Here is the call graph for this function:



Here is the caller graph for this function:



## 5.3.1.6 ReadAcpPipeLine()

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

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

### **Parameters**

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

## Returns

-1 if acp\_line is NULL or length is less than 1, otherwise 0.

Definition at line 186 of file Build.c.

Here is the call graph for this function:



Here is the caller graph for this function:



### 5.3.1.7 ReadCumulativeFileLine()

Reads a single line from the initial cumulative file and updates the map so that userID maps to a float 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**

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

## Returns

-1 ...

Definition at line 176 of file Build.c.

Here is the call graph for this function:



Here is the caller graph for this function:



## 5.3.1.8 ReadInitialCumulative()

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

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

## Parameters

cum_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 135 of file Build.c.

Here is the call graph for this function:



Here is the caller graph for this function:



## 5.3.1.9 SetAllStudentsInactive()

Sets the 'active' member 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 108 of file Build.c.

Here is the caller graph for this function:



### 5.3.1.10 UpdateFromWho()

```
int UpdateFromWho (  \label{eq:map * stmap} \mbox{ } \mbox{$\mathsf{map}$ * stmap $\mathsf{p}$ } \mbox{} \mbox{}
```

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

#### **Parameters**

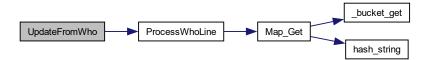
```
stmap The student map.
```

#### Returns

0 if succesful, otherwise nonzero.

Definition at line 40 of file Build.c.

Here is the call graph for this function:



Here is the caller graph for this function:



## 5.3.1.11 WriteStudentsToMemory()

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

Writes the students array to the location specified by mem\_ptr (eg. the shared memory segment).

#### **Parameters**

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

Definition at line 117 of file Build.c.

Here is the caller graph for this function:



## 5.3.2 Variable Documentation

### 5.3.2.1 dirty

```
short dirty [extern]
```

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

Definition at line 38 of file Build.c.

## 5.3.2.2 students

```
Student* students [extern]
```

Declarations for functions that populate data structures.

The underlying students array. Will be heap allocated with malloc, after PopulateStudents is called.

Generally this array and its length are still passed around via parameters, to decouple as much as possible and enable simple testing and dummy data.

Declarations for functions that populate data structures.

Definition at line 13 of file Build.c.

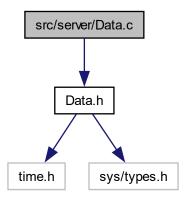
## 5.4 Build.h

### Go to the documentation of this file.

```
00001 #ifndef BUILD_H
00002 #define BUILD H
00007 #include "Data.h
00008 #include "map.h"
00009
00010 // ~~~~~ Data Structures ~~~~~
00011
00017 extern Student *students;
00018
00026 void PopulateStudents (char **studentIDs, char **studentNames, int arsize);
00034 void BuildStudentMap(map *stmap, Student *studentArr, int studentArrLength);
00035
00036 // ~~~~~ Processing ~~~~~~
00037
00039 extern short dirty;
00040
00047 int UpdateFromWho(map *stmap);
00048
00062 int ProcessWhoLine(map *stmap, char *whoLine, int whoLineLength);
00063
00070 void SetAllStudentsInactive(Student *stud_arr, int arr_len);
00080 void WriteStudentsToMemory(void *mem_ptr, Student *stud_arr, int arr_len);
00081
00082 // ~~~~~ Cumulative Processing ~~~~~~
00083
00093 int ReadInitialCumulative(map *time_map, char *filename);
00094
00103 int ReadACP(map *st_map);
00104
00114 void ReadCumulativeFileLine(map *cum_map, char *acp_line);
00115
00123 int ReadAcpPipeLine(map *stmap, char *acp_line);
00124
00134 void CalculateCumulative(Student *stud_arr, int stud_arr_len, map *cum_map);
00135
00136 #endif
```

## 5.5 src/server/Data.c File Reference

#include "Data.h"
Include dependency graph for Data.c:



## **Variables**

```
    char * Data_IDs [DATA_NUM_RECORDS]
        Data structures and constants.

    char * Data_Names [DATA_NUM_RECORDS]
```

#### 5.5.1 Variable Documentation

#### 5.5.1.1 Data\_IDs

```
char* Data_IDs[DATA_NUM_RECORDS]

Initial value:
= {
    "chen",
    "beal389",
    "bol4559",
    "cal6258",
    "kre5277",
    "lon1150",
    "mas9309",
    "mes08346",
    "mi17233",
    "nef9476",
    "Nov-88",
    "pan9725",
    "rac3146",
    "rub4133",
    "shr5683",
```

Data structures and constants.

"vay3083",
"yos2327"}

Definition at line 6 of file Data.c.

### 5.5.1.2 Data\_Names

```
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 25 of file Data.c.

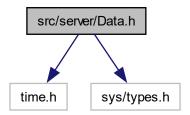
## 5.6 Data.c

### Go to the documentation of this file.

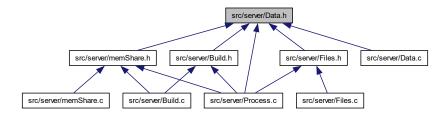
```
00001
00004 #include "Data.h"
00005
00006 char *Data_IDs[DATA_NUM_RECORDS] = {
00007
            "chen",
           "bea1389",
80000
           "bol4559",
00009
            "cal6258",
00010
           "kre5277",
00011
00012
           "lon1150",
           "mas9309",
"mes08346",
00013
00014
00015
00016
            "mi17233",
           "nef9476",
           "Nov-88", "pan9725",
00017
00018
           "rac3146",
00019
00020
           "rub4133",
           "shr5683",
00021
00022
00023
           "yos2327"};
00024
00025 char *Data_Names[DATA_NUM_RECORDS] = {
00026
           "Weifeng Chen",
           "Christian Beatty",
00027
00028
           "Emily Bolles",
            "Cameron Calhoun",
00029
00030
           "Ty Kress",
00031
           "Cody Long",
00032
           "Caleb Massey",
00033
           "Christian Messmer",
00034
           "Karl Miller",
            "Jeremiah Neff",
00035
            "Kaitlyn Novacek",
00036
           "Joshua Panaro",
"Caleb Rachocki",
00037
00038
00039
           "Caleb Ruby",
           "Paul Shriner",
"Alan Vayansky",
"Assefa Ayalew Yoseph"};
00040
00041
00042
```

## 5.7 src/server/Data.h File Reference

```
#include <time.h>
#include <sys/types.h>
Include dependency graph for Data.h:
```



This graph shows which files directly or indirectly include this file:



### **Data Structures**

struct Student

The student data type.

## **Macros**

- #define DATA\_NUM\_RECORDS 17
  - Declarations of types and macros.
- #define DATA\_ID\_MAX\_LENGTH 9
- #define DATA NAME MAX LENGTH 21
- #define DATA\_SIZE 56

## **Variables**

- char \* Data\_IDs []
  - Data structures and constants.
- char \* Data\_Names []

### 5.7.1 Macro Definition Documentation

### 5.7.1.1 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 17 of file Data.h.

### 5.7.1.2 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 22 of file Data.h.

## 5.7.1.3 DATA\_NUM\_RECORDS

```
#define DATA_NUM_RECORDS 17
```

Declarations of types and macros.

The total count of records.

Definition at line 12 of file Data.h.

## 5.7.1.4 DATA\_SIZE

```
#define DATA_SIZE 56
```

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

Definition at line 33 of file Data.h.

## 5.7.2 Variable Documentation

### 5.7.2.1 Data\_IDs

```
char* Data_IDs[] [extern]
```

Data structures and constants.

Definition at line 6 of file Data.c.

## 5.7.2.2 Data\_Names

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

Constant, all user's names.

Definition at line 25 of file Data.c.

5.8 Data.h 41

## 5.8 Data.h

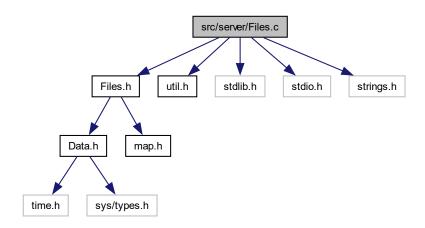
### Go to the documentation of this file.

```
00001 #ifndef Data_h
00002 #define Data_h
00006 #include <time.h>
00007 #include <sys/types.h>
80000
00012 #define DATA_NUM_RECORDS 17
00017 #define DATA_ID_MAX_LENGTH 9
00022 #define DATA_NAME_MAX_LENGTH 21
00023
00024 /* Constant, all user IDs. */
00025 extern char *Data_IDs[];
00026
00028 extern char *Data_Names[];
00029
00033 #define DATA_SIZE 56
00034
00038 typedef struct
00039 {
00040
              char userID[DATA_ID_MAX_LENGTH];
             char fullName[DATA_NAME_MAX_LENGTH];
short age;
00042
             float gpa;
short active;
00043
00044
            time_t lastLogin;
int loginDuration;
00045
00046
00047 } Student;
00048
00049 #endif
```

## 5.9 src/server/Files.c File Reference

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

Include dependency graph for Files.c:



# **Functions**

short FileExists (char \*file\_name\_to\_check)

Declarations of functions that operate on files..

• 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)

#### 5.9.1 Function Documentation

#### 5.9.1.1 CreateInitialCumulativeFile()

Creates the initial cumulative login time file.

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

#### **Parameters**

1	file_name	The name of the file to created. EG STATIC_USER_CUMULATIVE_FILE	
---	-----------	---	--

#### Warning

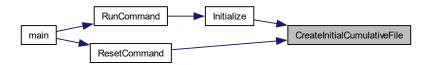
This file should already be validated to not exist.

#### 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 94 of file Files.c.

Here is the caller graph for this function:



## 5.9.1.2 CreateInitialUserDataFile()

Creates the initial user data file. This should be called only the first time the program runs, if it doesn't exist.

#### **Parameters**

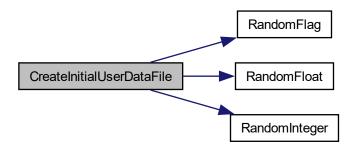
file_name	The file name to create.
id_list	An array containing the IDs. Eg. "Data_IDs" from Data.h
id_list_len	The length of the id_list. Eg. "DATA_NUM_RECORDS" from Data.h

#### Returns

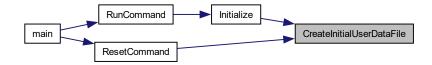
A 0 if the operation was succesful, otherwise nonzero.

Definition at line 25 of file Files.c.

Here is the call graph for this function:



Here is the caller graph for this function:



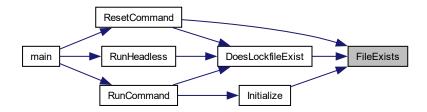
### 5.9.1.3 FileExists()

Declarations of functions that operate on files..

Determines whether a file exists.

Definition at line 10 of file Files.c.

Here is the caller graph for this function:



## 5.9.1.4 FillStudentMapFromFile()

```
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.

### **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

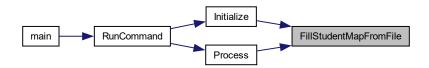
0 if succesful, 1 if there was an error.

Definition at line 51 of file Files.c.

Here is the call graph for this function:



Here is the caller graph for this function:



## 5.9.1.5 WriteStudentArrayToFile()

Writes the student array to the file.

### **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 78 of file Files.c.

Here is the caller graph for this function:



## 5.10 Files.c

#### Go to the documentation of this file.

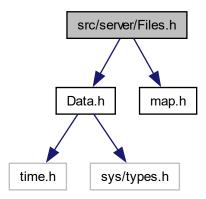
```
00004 #include "Files.h"
00005 #include "util.h"
00006 #include <stdlib.h>
00007 #include <stdio.h>
00008 #include <strings.h>
00009
00010 short FileExists(char *file_name_to_check)
00011 {
          FILE *file = fopen(file_name_to_check, "r");
short result = 1;
00012
00013
00014
          if (file == NULL)
00015
00016
              result = 0;
00017
00018
          else
00019
          {
00020
              fclose(file):
00021
00022
          return result;
00023 }
00024
00025 int CreateInitialUserDataFile(char *file_name, char **id_list, int id_list_len)
00026 {
00027
          FILE *file = fopen(file_name, "w");
00028
          if (file == NULL)
00029
00030
              return -1;
00031
00032
          int i:
00033
          for (i = 0; i < id_list_len; i++)</pre>
00034
00035
              int rand_age = RandomInteger(18, 22);
00036
               if (RandomFlag(0.42))
00037
00038
              {
                  gpa = 4.0; // 42% of the time, make the GPA 4.0
00039
00040
              }
00041
              else
00042
              {
00043
                   gpa = RandomFloat(2.5, 4.0);
00044
              fprintf(file, "%s\t%d\t%.2f\t%d\n", id_list[i], rand_age, gpa, 0);
00045
00046
00047
          fclose(file);
00048
          return 0;
00049 }
00050
00051 int FillStudentMapFromFile(map *student_map, char *file_name, char **id_list, int id_list_len)
00052 {
          FILE *file = fopen(file_name, "r");
00053
00054
          if (file == NULL)
00055
00056
              return -1:
00057
          // id buffer
00058
00059
          char user_id[9];
00060
          int age;
00061
          float gpa;
00062
          long time;
00063
          while (fscanf(file, "%9s\t%d\t%f\t%ld", user_id, &age, &gpa, &time) == 4)
00064
          {
00065
              map_result result = Map_Get(student_map, user_id);
00066
              if (result.found == 0)
```

```
00067
                                           {
 00068
                                                       continue;
00069
                                            ((Student *)result.data)->age = age;
((Student *)result.data)->gpa = gpa;
((Student *)result.data)->lastLogin = time;
00070
00071
00072
 00073
 00074
                               fclose(file);
00075
                               return 0;
00076 }
00077
00078 int WriteStudentArrayToFile(Student *students, int arr_len, char *file_name)
 00079 {
 08000
                               FILE *file = fopen(file_name, "w");
                               if (file == NULL)
 00081
 00082
00083
                                           return -1:
00084
 00085
                              int i;
 00086
                               for (i = 0; i < arr_len; i++)</pre>
 00087
                                            fprintf(file, \ "%s\t%d\t%.2f\t%ld\n", \ students[i].userID, \ students[i].age, \ students[i].gpa, \ stude
00088
                 students[i].lastLogin);
00089
00090
                               fclose(file);
 00091
                              return 0;
00092 }
00093
00094 int CreateInitialCumulativeFile(char *file_name)
00095 {
00096
                               FILE *file = fopen(file_name, "w");
 00097
                               if (file == NULL)
 00098
00099
                                           return -1;
00100
                              FILE *pipe = popen("ac -p", "r");
if (pipe == NULL)
00101
00102
 00103
 00104
                                            fclose(file);
 00105
                                           return -2;
00106
00107
00108
                              char line[100];
 00109
                              while (fgets(line, sizeof(line), pipe) != NULL)
 00110
00111
                                            fputs(line, file);
00112
00113
                               pclose(pipe);
00114
                               fclose(file);
00115
                               return 0;
00116 }
```

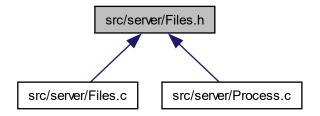
## 5.11 src/server/Files.h File Reference

```
#include "Data.h"
#include "map.h"
```

Include dependency graph for Files.h:



This graph shows which files directly or indirectly include this file:



### **Macros**

- #define STATIC\_USER\_DATA\_FILE "static-user-data.txt"
   Definitions for functions that operate on files.
- #define STATIC\_USER\_CUMULATIVE\_FILE "static-user-cumulative-start.txt"

#### **Functions**

- short FileExists (char \*file\_name\_to\_check)
  - Determines whether a file exists.
- $\bullet \ \ \text{int CreateInitialUserDataFile (char *file\_name, char **id\_list, int id\_list\_len)}\\$ 
  - Creates the initial user data file. This should be called only the first time the program runs, if it doesn't exist.
- int WriteStudentArrayToFile (Student \*students, int arr\_len, char \*file\_name)
  - Writes the student array to the file.
- 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 CreateInitialCumulativeFile (char \*file\_name)

#### 5.11.1 Macro Definition Documentation

#### 5.11.1.1 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.

Definition at line 38 of file Files.h.

### 5.11.1.2 STATIC\_USER\_DATA\_FILE

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

Definitions for functions that operate on files.

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

There are two files that are created if they don't exist when the program is first run. One has userIDs linked to their age, gpa, and last login time. The other has userIDs linked to the cumulative login time as determined by ac -p in order to determine the time logged in since program first ran. 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 31 of file Files.h.

### 5.11.2 Function Documentation

## 5.11.2.1 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

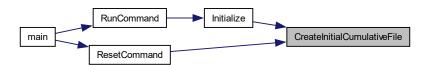
This file should already be validated to not exist.

### 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 94 of file Files.c.

Here is the caller graph for this function:



# 5.11.2.2 CreateInitialUserDataFile()

Creates the initial user data file. This should be called only the first time the program runs, if it doesn't exist.

## **Parameters**

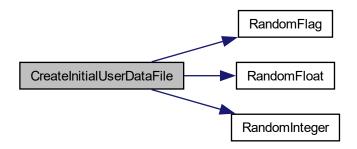
file_name	The file name to create.
id_list	An array containing the IDs. Eg. "Data_IDs" from Data.h
id_list_len	The length of the id_list. Eg. "DATA_NUM_RECORDS" from Data.h

#### Returns

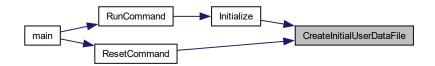
A 0 if the operation was succesful, otherwise nonzero.

Definition at line 25 of file Files.c.

Here is the call graph for this function:



Here is the caller graph for this function:



## 5.11.2.3 FileExists()

Determines whether a file exists.

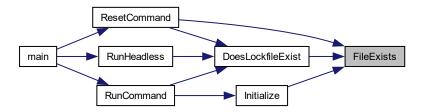
Returns

1 if it exists. 0 if it does not.

Determines whether a file exists.

Definition at line 10 of file Files.c.

Here is the caller graph for this function:



## 5.11.2.4 FillStudentMapFromFile()

```
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.

#### **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.

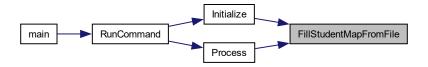
Definition at line 51 of file Files.c.

Here is the call graph for this function:



5.12 Files.h 53

Here is the caller graph for this function:



# 5.11.2.5 WriteStudentArrayToFile()

Writes the student array to the file.

#### **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 78 of file Files.c.

Here is the caller graph for this function:



## 5.12 Files.h

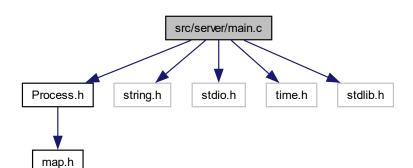
Go to the documentation of this file.

```
00001 #ifndef Files_H
00002 #define Files_H
```

```
00006 #include "Data.h"
00007 #include "map.h"
80000
00031 #define STATIC_USER_DATA_FILE "static-user-data.txt"
00032
00038 #define STATIC_USER_CUMULATIVE_FILE "static-user-cumulative-start.txt"
00039
00044 short FileExists(char *file_name_to_check);
00045
00055 int CreateInitialUserDataFile(char *file_name, char **id_list, int id_list_len);
00056
00066 int WriteStudentArrayToFile(Student *students, int arr_len, char *file_name);
00067
00078 int FillStudentMapFromFile(map *student_map, char *file_name, char **id_list, int id_list_len);
00079
00089 int CreateInitialCumulativeFile(char *file_name);
00090
00091 #endif
```

## 5.13 src/server/main.c File Reference

```
#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.

## 5.13.1 Function Documentation

5.14 main.c 55

# 5.13.1.1 main()

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

Program entry.

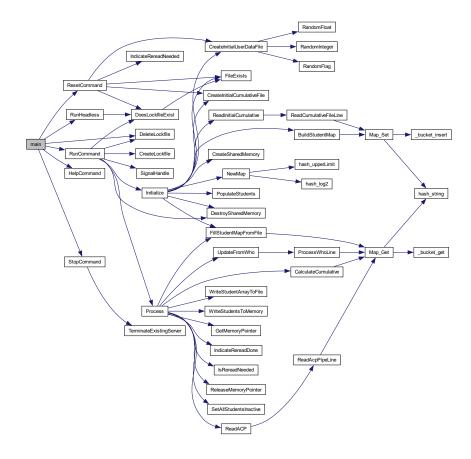
Parses arguments and calls the appropriate Process.h function.

### **Parameters**

argc	The argument count.
argv	The argument values.

Definition at line 30 of file main.c.

Here is the call graph for this function:



## 5.14 main.c

Go to the documentation of this file.

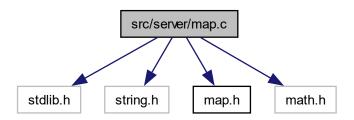
```
00001 #include "Process.h"
00002 #include <string.h>
```

```
00003 #include <stdio.h>
00004 #include <time.h>
00005 #include <stdlib.h>
00030 int main(int argc, char **argv)
00031 {
00032
          srand(time(NULL)); // seed the randomizer
00034
          if (argc <= 1 || argc >= 3)
00035
              printf("You entered too few or many options!\n");
00036
00037
              HelpCommand();
00038
00039
          else if (strcmp(argv[1], "help") == 0)
00040
00041
              HelpCommand();
00042
          else if (strcmp(argv[1], "reset") == 0)
00043
00044
              ResetCommand();
00046
00047
          else if (strcmp(argv[1], "stop") == 0 || strcmp(argv[1], "end") == 0 || strcmp(argv[1], "close")
      == 0)
00048
00049
              StopCommand();
00050
00051
          else if (strcmp(argv[1], "headless") == 0)
00052
00053
              RunHeadless(argv[0]);
00054
          else if (strcmp(argv[1], "run") == 0 || strcmp(argv[1], "start") == 0)
00055
00056
00057
              RunCommand();
00058
00059
          else if (strcmp(argv[1], "delete-lockfile") == 0)
          { // an admin debug command; not meant to be called
    printf("Deleting lockfile.\n");
00060
00061
00062
              DeleteLockfile();
00063
00064
          else
00065
00066
              printf("Unknown option!\n");
00067
              HelpCommand();
00068
00069
          return 0;
00070 }
```

# 5.15 src/server/map.c File Reference

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

Include dependency graph for map.c:



#### **Functions**

int hash\_log2 (int num\_to\_log)

Definitions for functions relating to a hashmap data structure.

• int hash upperLimit (int bitsize)

This calculates what the actual capacity of the map will be. Given a result from hash\_log2, it gets the maximum storable for that many bits. For example, hash\_upperLimit(3) returns the maximum that 3 bits can hold, which is 8. hash upperLimit(4) returns 16.

- int hash\_string (int hash\_table\_size, char \*string, int strlen)
- map \* NewMap (int capacity)
- void \_bucket\_insert (struct \_map\_bucket \*bucket, char \*key, void \*value)
- void Map\_Set (map \*a\_map, char \*key, void \*value)

Sets a value in the map.

- void \_bucket\_get (struct \_map\_bucket \*bucket, char \*key, map\_result \*result)
- map\_result Map\_Get (map \*a\_map, char \*key)

Gets a value from the map. It will return a map\_get\_result describing whether it was succesful, and possibly containing the data sought, or NULL if it was unsuccesful.

- void \_bucket\_delete (struct \_map\_bucket \*bucket, char \*key, short free\_it, map\_result \*result)
- 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.

### **Variables**

• int char\_ratio = (int)(sizeof(int) / sizeof(char))

### 5.15.1 Function Documentation

### 5.15.1.1 bucket delete()

Definition at line 123 of file map.c.

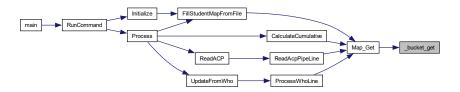
Here is the caller graph for this function:



## 5.15.1.2 \_bucket\_get()

Definition at line 91 of file map.c.

Here is the caller graph for this function:



### 5.15.1.3 \_bucket\_insert()

```
void _bucket_insert (
          struct _map_bucket * bucket,
          char * key,
          void * value )
```

Definition at line 63 of file map.c.

Here is the caller graph for this function:



### 5.15.1.4 hash\_log2()

```
int hash_log2 ( int \ num\_to\_log \ )
```

Definitions for functions relating to a hashmap data structure.

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

Definition at line 9 of file map.c.

Here is the caller graph for this function:



### 5.15.1.5 hash\_string()

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

Uses some clever, prime-number-multiplication, ORing, and bitwise operations to generate a number than, when modulused with the hash\_table\_size, will produce numbers ('buckets') of even distribution, to minimize the number of collisions. This function contains the meat of the hashing algorithm; it converts a key-string to an array index.

### See also

http://isthe.com/chongo/tech/comp/fnv/

## Parameters

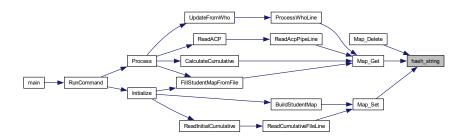
hash_table_capacity	The number of buckets this table holds.
string	The key to hash.
strlen	The length of the key.

#### Returns

The index of the bucket that should be used.

Definition at line 30 of file map.c.

Here is the caller graph for this function:



### 5.15.1.6 hash\_upperLimit()

This calculates what the actual capacity of the map will be. Given a result from hash\_log2, it gets the maximum storable for that many bits. For example, hash\_upperLimit(3) returns the maximum that 3 bits can hold, which is 8. hash\_upperLimit(4) returns 16.

#### **Parameters**

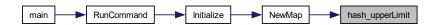
bitsize	The number of bits to calculate the max from.
---------	---

#### Returns

The max value that number of bits can hold.

Definition at line 22 of file map.c.

Here is the caller graph for this function:



#### 5.15.1.7 Map\_Delete()

Deletes a key from the map. Returns a map\_get\_result describing whether the delete was successful and containing the removed data, if extant.

### **Parameters**

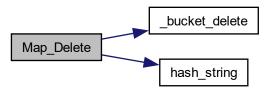
тар	The map to delete the key from.
key	The key to delete.
free←	Whether to call free() on the underlying data.
_it	

### Returns

A map\_get\_result with the data that was removed.

Definition at line 149 of file map.c.

Here is the call graph for this function:



## 5.15.1.8 Map\_Get()

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.

## **Parameters**

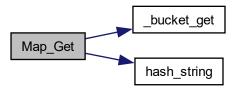
тар	The map to retrieve from.
key	The key of the item.

### Returns

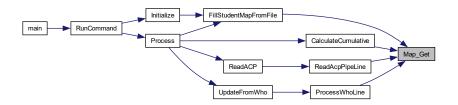
A map\_get\_result containing the sought data.

Definition at line 114 of file map.c.

Here is the call graph for this function:



Here is the caller graph for this function:



## 5.15.1.9 Map\_Set()

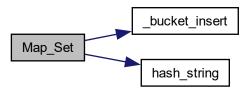
Sets a value in the map.

### **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 84 of file map.c.

Here is the call graph for this function:



Here is the caller graph for this function:



### 5.15.1.10 NewMap()

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

## **Parameters**

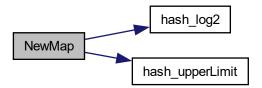
capacity	The estimated required capacity of the map.

## Returns

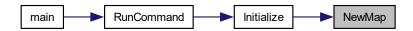
A pointer to the heap allocated map.

Definition at line 45 of file map.c.

Here is the call graph for this function:



Here is the caller graph for this function:



## 5.15.2 Variable Documentation

## 5.15.2.1 char\_ratio

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

Definition at line 27 of file map.c.

# 5.16 map.c

### Go to the documentation of this file.

```
00001
00004 #include "stdlib.h"
00004 #include "stdilb."

00005 #include "string.h"

00006 #include "map.h"

00007 #include "math.h"
80000
00009 int hash_log2(int num_to_log)
00010 {
00011
              int t = 1;
00012
              int i = 0;
00013
              do
             {
00014
00015
00016
                  num_to_log = num_to_log & ~t;
             t = t « 1;
i++;
} while (num_to_log > 0);
00017
00018
00019
              return i;
```

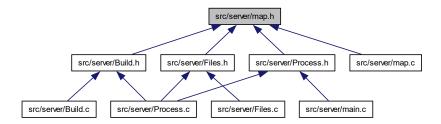
5.16 map.c 65

```
00020 }
00021
00022 int hash_upperLimit(int bitsize)
00023 {
00024
          return 1 « bitsize;
00025 }
00026
00027 int char_ratio = (int)(sizeof(int) / sizeof(char));
00028
00029 // Modified some stuff from : http://isthe.com/chongo/tech/comp/fnv/
00030 int hash_string(int hash_table_size, char *string, int strlen)
00031 {
00032
          int i, hash = 2166136261;
          for (i = 0; i < strlen; i += 1)
00033
00034
          {
              hash *= 16777619;
hash ^= string[i];
00035
00036
00037
00038
          if (hash < 0)
00039
          {
00040
              hash \star = -1;
00041
00042
          return hash % (hash_table_size - 1) + 1;
00043 }
00044
00045 map *NewMap(int capacity)
00046 {
00047
          int log2 = hash_log2(capacity);
00048
          int capac = hash_upperLimit(log2);
          int sz = sizeof(struct _map_bucket) * capac;
struct _map_bucket *buckets = malloc(sz);
00049
00050
00051
          memset (buckets, 0, sz);
00052
00053
          for (i = 0; i < capac; i++)</pre>
00054
00055
              buckets[i] = (struct _map_bucket) {NULL, NULL, NULL};
00056
00057
          map newm = (map) {capac, buckets};
00058
          map *map_p = malloc(sizeof(map));
00059
          *map_p = newm;
00060
          return map_p;
00061 }
00062
00063 void _bucket_insert(struct _map_bucket *bucket, char *key, void *value)
00064 {
00065
          struct _map_bucket *check = bucket;
00066
          while (check->key != NULL)
00067
00068
              if (strcmp(check->key, key) == 0)
00069
              {
00070
                  check->data = value;
00071
                   return;
00072
00073
               if (check->next == NULL)
00074
              {
00075
                  check->next = malloc(sizeof(struct map bucket));
00076
                   *(check->next) = (struct _map_bucket) {NULL, NULL};
00077
00078
              check = check->next;
00079
08000
          check->kev = kev:
00081
          check->data = value;
00082 }
00083
00084 void Map_Set(map *a_map, char *key, void *value)
00085 {
00086
          int keyl = (int) strlen(key);
          int hash = hash_string(a_map->size, key, keyl);
00087
00088
          _bucket_insert(&(a_map->buckets[hash]), key, value);
00089 }
00090
00091 void _bucket_get(struct _map_bucket *bucket, char *key, map_result *result)
00092 {
          struct _map_bucket *check = bucket;
00093
00094
          while (check->key != NULL)
00095
00096
               if (strcmp(check->key, key) == 0)
00097
              {
00098
                  result->found = 1;
00099
                   result->data = check->data;
00100
                   return;
00101
00102
              else if (check->next != NULL)
00103
00104
                   check = check->next;
00105
00106
              else
```

```
{
00108
                  result->found = 0;
00109
                  break;
00110
00111
          }
00112 }
00113
00114 map_result Map_Get(map *a_map, char *key)
00115 {
          map_result res = (map_result) {0, NULL};
00116
00117
          int keyl = (int)strlen(key);
          int hash = hash_string(a_map->size, key, keyl);
00118
00119
          _bucket_get(&(a_map->buckets[hash]), key, &res);
00120
00121 }
00122
00123 void _bucket_delete(struct _map_bucket *bucket, char *key, short free_it, map_result *result)
00124 {
          struct _map_bucket *last = bucket;
00126
          struct _map_bucket *next = bucket->next;
00127
          while (next != NULL)
00128
              if (strcmp(next->key, key) == 0)
00129
00130
              {
00131
                  result->found = 1;
00132
                  result->data = next->data;
00133
                  if (free_it)
00134
00135
                      free(next->data);
                      result->data = NULL;
00136
00137
00138
                  last->next = next->next;
00139
                  free(next);
00140
00141
              else
00142
                  last = next;
next = next->next;
00143
00144
00145
00146
         }
00147 }
00148
00149 map_result Map_Delete(map *a_map, char *key, short free_it)
00150 {
00151
          map_result res = (map_result) {0, NULL};
00152
          int keyl = (int)strlen(key);
00153
          int hash = hash_string(a_map->size, key, key1);
00154
00155
          struct _map_bucket top = a_map->buckets[hash];
          if (top.key == NULL)
00156
00157
          {
00158
00159
00160
          if (strcmp(top.key, key) == 0)
00161
00162
              res.found = 1;
              res.data = top.data;
00164
              if (free_it)
00165
00166
                  free(top.data);
00167
                  res.data = NULL;
00168
00169
              if (top.next != NULL)
00170
00171
                  a_map->buckets[hash] = *(top.next);
00172
                  free(top.next);
00173
00174
              else
00175
              {
00176
                  a_map->buckets[hash] = (struct _map_bucket){NULL, NULL};
00177
00178
              return res;
00179
          if (top.next == NULL)
00180
00181
00182
              return res;
00183
00184
          _bucket_delete(&(a_map->buckets[hash]), key, free_it, &res);
00185
00186
          return res:
00187 }
```

# 5.17 src/server/map.h File Reference

This graph shows which files directly or indirectly include this file:



## **Data Structures**

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**

int hash\_log2 (int number\_to\_log)

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

- int hash\_string (int hash\_table\_capacity, char \*string, int strlen)
- int hash\_upperLimit (int bitsize)

This calculates what the actual capacity of the map will be. Given a result from hash\_log2, it gets the maximum storable for that many bits. For example, hash\_upperLimit(3) returns the maximum that 3 bits can hold, which is 8. hash\_upperLimit(4) returns 16.

- 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.

## 5.17.1 Function Documentation

## 5.17.1.1 hash\_log2()

Definitions for functions that operate on a hash map data structure. Karl's take on a simple hashmap 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 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.

--- Some Improvements.

- 1. Map can free on delete. We could have a Map\_Set(map, key, char \*) that will automatically malloc a string on set, to simplify string to string maps.
- 2. We could have convenience methods that auto-cast for various types of data. Eg. Map\_GetTime...

Get's a log2 ceiling. Eg, hash\_log2(5) == 3.

#### **Parameters**

number_to_log	The number to calculate the log of.
---------------	-------------------------------------

# Returns

The log ceiling; eg, the lowest exponent to raise 2 with which would yield a number greater or equal to number\_to\_log.

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

Definition at line 9 of file map.c.

Here is the caller graph for this function:



## 5.17.1.2 hash\_string()

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

Uses some clever, prime-number-multiplication, ORing, and bitwise operations to generate a number than, when modulused with the hash\_table\_size, will produce numbers ('buckets') of even distribution, to minimize the number of collisions. This function contains the meat of the hashing algorithm; it converts a key-string to an array index.

#### See also

http://isthe.com/chongo/tech/comp/fnv/

# **Parameters**

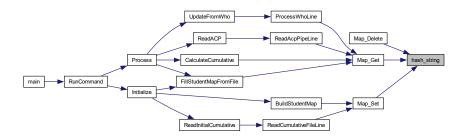
hash_table_capacity	The number of buckets this table holds.
string	The key to hash.
strlen	The length of the key.

#### Returns

The index of the bucket that should be used.

Definition at line 30 of file map.c.

Here is the caller graph for this function:



# 5.17.1.3 hash\_upperLimit()

This calculates what the actual capacity of the map will be. Given a result from hash\_log2, it gets the maximum storable for that many bits. For example, hash\_upperLimit(3) returns the maximum that 3 bits can hold, which is 8. hash\_upperLimit(4) returns 16.

#### **Parameters**

bitsize The number of bits to calculate the max from.
---

## Returns

The max value that number of bits can hold.

Definition at line 22 of file map.c.



#### 5.17.1.4 Map\_Delete()

Deletes a key from the map. Returns a map\_get\_result describing whether the delete was successful and containing the removed data, if extant.

#### **Parameters**

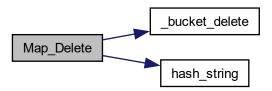
тар	The map to delete the key from.
key	The key to delete.
free⊷	Whether to call free() on the underlying data.
_it	

#### Returns

A map\_get\_result with the data that was removed.

Definition at line 149 of file map.c.

Here is the call graph for this function:



# 5.17.1.5 Map\_Get()

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.

#### **Parameters**

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

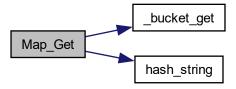
Generated by Doxygen

#### Returns

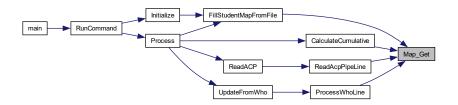
A map\_get\_result containing the sought data.

Definition at line 114 of file map.c.

Here is the call graph for this function:



Here is the caller graph for this function:



# 5.17.1.6 Map\_Set()

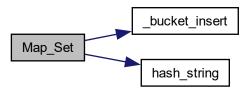
Sets a value in the map.

## **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 84 of file map.c.

Here is the call graph for this function:



Here is the caller graph for this function:



## 5.17.1.7 NewMap()

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

# **Parameters**

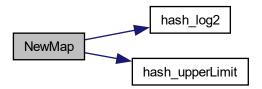
capacity	The estimated required capacity of the map.
----------	---

## Returns

A pointer to the heap allocated map.

Definition at line 45 of file map.c.

Here is the call graph for this function:



Here is the caller graph for this function:



# 5.18 map.h

#### Go to the documentation of this file.

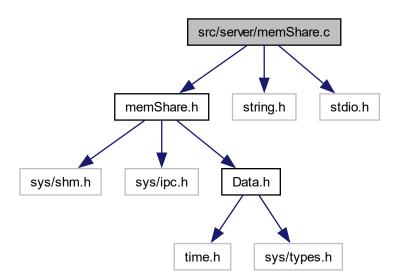
```
00001 #ifndef map_h
00002 #define map_h
00003
00043 //
00044 //
               Hashing Math
00045 //
00046
00052 int hash_log2(int number_to_log);
00053
00062 int hash_string(int hash_table_capacity, char *string, int strlen);
00063
00069 int hash_upperLimit(int bitsize);
00070
00071 // -
00072 //
              General Map Operations
00073 // -
00074
00080 struct _map_bucket
00081 {
          \ensuremath{//} The key associated with this bucket.
00082
00083
          char *key;
00084
          // The data this bucket holds.
00085
          void *data;
00086
          // The next node in this linked list, or NULL if it is a leaf.
00087
          struct _map_bucket *next;
00088 };
00089
00099 typedef struct
00100 {
00101
          // The number of base buckets in this map.
00102
          int size;
00103
          // The buckets for this map.
00104
          struct _map_bucket *buckets;
00105 } map;
00106
00110 typedef struct
00111 {
```

```
00112
          // 1 if successfully found. 0 if not found.
          short found;
          // The data linked with that key; indeterminate if found == 0.
00114
         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
```

# 5.19 src/server/memShare.c File Reference

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

Include dependency graph for memShare.c:



## **Functions**

- int CreateSharedMemory ()
  - Definitions for functions that operate on a shared memory segment.
- int DestroySharedMemory ()
- void \* GetMemoryPointer (int shared\_mem\_id)
- int ReleaseMemoryPointer (void \*shmaddr)

## 5.19.1 Function Documentation

## 5.19.1.1 CreateSharedMemory()

```
int CreateSharedMemory ( )
```

Definitions for functions that operate on a shared memory segment.

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.

Definition at line 12 of file memShare.c.

Here is the caller graph for this function:



# 5.19.1.2 DestroySharedMemory()

```
int DestroySharedMemory ( )
```

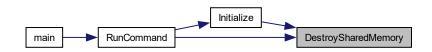
Flags the shared memory segment for deallocation. Returns the result of that operation.

Returns

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

Definition at line 17 of file memShare.c.

Here is the caller graph for this function:



# 5.19.1.3 GetMemoryPointer()

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

Calls 'shmat(shared\_mem\_id, NULL, 0)`;

5.20 memShare.c 77

#### **Parameters**

shared_mem←	The id of the shared memory	l
_id		l

#### Returns

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

Definition at line 26 of file memShare.c.

Here is the caller graph for this function:



# 5.19.1.4 ReleaseMemoryPointer()

```
int ReleaseMemoryPointer ( \mbox{void} \ * \ shmaddr \ )
```

Definition at line 31 of file memShare.c.

Here is the caller graph for this function:



# 5.20 memShare.c

## Go to the documentation of this file.

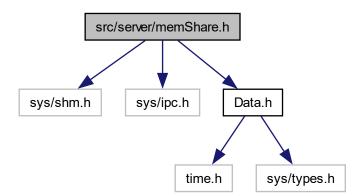
```
00001
00004 #include "memShare.h"
00005 #include <string.h>
00006 #include <stdio.h>
00007
00008 /*
00009 Todo: Error handling and printing
00010 */
00011
00012 int CreateSharedMemory()
00013 {
```

```
return shmget(MEM_KEY, MEM_SIZE, IPC_CREAT | MEM_PERMISSIONS);
00015 }
00016
00017 int DestroySharedMemory()
00018 {
          int shm_id = shmget(MEM_KEY, MEM_SIZE, 0);
00019
          int control_result = shmctl(shm_id, IPC_RMID, 0);
if (control_result != -1)
00020
00021
00022
              return 0;
00023
          return control_result;
00024 }
00025
00026 void *GetMemoryPointer(int shared_mem_id)
00027 {
00028
          return shmat(shared_mem_id, NULL, 0);
00029 }
00030
00031 int ReleaseMemoryPointer(void *shmaddr)
00032 {
00033
          return shmdt(shmaddr);
00034 }
```

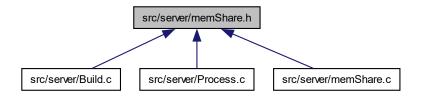
# 5.21 src/server/memShare.h File Reference

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

Include dependency graph for memShare.h:



This graph shows which files directly or indirectly include this file:



## **Macros**

- #define MEM\_KEY 0x727
  - Declarations for functions that operate on a shared memory segment.
- #define MEM\_PERMISSIONS 0664
- #define MEM\_SIZE DATA\_SIZE \*DATA\_NUM\_RECORDS

## **Functions**

- int CreateSharedMemory ()
  - Definitions for functions that operate on a shared memory segment.
- int DestroySharedMemory ()
- void \* GetMemoryPointer (int shared\_mem\_id)
- int ReleaseMemoryPointer (void \*shmaddr)

#### 5.21.1 Macro Definition Documentation

# 5.21.1.1 MEM\_KEY

#define MEM\_KEY 0x727

Declarations for functions that operate on a shared memory segment.

Definition at line 14 of file memShare.h.

## 5.21.1.2 MEM\_PERMISSIONS

#define MEM\_PERMISSIONS 0664

Definition at line 25 of file memShare.h.

## 5.21.1.3 MEM\_SIZE

#define MEM\_SIZE DATA\_SIZE \*DATA\_NUM\_RECORDS

Definition at line 30 of file memShare.h.

# 5.21.2 Function Documentation

#### 5.21.2.1 CreateSharedMemory()

```
int CreateSharedMemory ( )
```

Definitions for functions that operate on a shared memory segment.

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.

Definition at line 12 of file memShare.c.

Here is the caller graph for this function:



## 5.21.2.2 DestroySharedMemory()

```
int DestroySharedMemory ( )
```

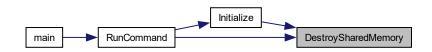
Flags the shared memory segment for deallocation. Returns the result of that operation.

Returns

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

Definition at line 17 of file memShare.c.

Here is the caller graph for this function:



# 5.21.2.3 GetMemoryPointer()

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

Calls 'shmat(shared\_mem\_id, NULL, 0)`;

5.22 memShare.h

#### **Parameters**

shared_mem←	The id of the shared memory	l
_id		l

#### Returns

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

Definition at line 26 of file memShare.c.

Here is the caller graph for this function:



# 5.21.2.4 ReleaseMemoryPointer()

```
int ReleaseMemoryPointer ( {\tt void} \ * \ shmaddr \ )
```

Definition at line 31 of file memShare.c.

Here is the caller graph for this function:



# 5.22 memShare.h

## Go to the documentation of this file.

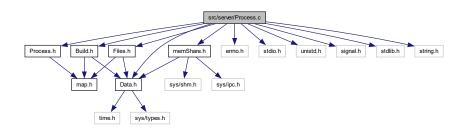
```
00001 #ifndef MEM_SHARE_H
00002 #define MEM_SHARE_H
00007 #include <sys/shm.h>
00008 #include <sys/ipc.h>
00009 #include "Data.h"
00010
00011 /*
00012 The shared memory key that clients and servers will use to identify the segment.
00013 */
00014 #define MEM_KEY 0x727
00015
```

```
00016 /*
00017 Memory permissions are:
              RW 110 = 6
R 100 = 4
00018 Self:
00019 Group:
00020 Others: R
                      100 = 4
00021 - All groups can read.
00022 - Self can write.
00023 - None can execute.
00024 */
00025 #define MEM_PERMISSIONS 0664
00026
00027 /*
00028 The memory allocation must as large as the data size times the number of records.
00029 */
00030 #define MEM_SIZE DATA_SIZE *DATA_NUM_RECORDS
00031
00037 int CreateSharedMemory();
00038
00044 int DestroySharedMemory();
00055 void *GetMemoryPointer(int shared_mem_id);
00056
00057 /*
00058 Release a shm memory pointer.
00059 */
00060 int ReleaseMemoryPointer(void *shmaddr);
00061
00062 #endif
```

#### 5.23 src/server/Process.c File Reference

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

Include dependency graph for Process.c:



## **Functions**

- short DoesLockfileExist ()
- int CreateLockfile ()
- int DeleteLockfile ()
- int TerminateExistingServer ()
- int IndicateRereadNeeded ()

- int IndicateRereadDone ()
- short IsRereadNeeded ()
- void SignalHandle (int signo)
- int Initialize ()
- void Process (int shm\_id)
- void HelpCommand ()
- void RunCommand ()
- void StopCommand ()
- void ResetCommand ()
- void RunHeadless (char \*processName)

#### **Variables**

- map \* student\_map
  - Definitions for functions that manage control flow.
- map \* initial\_cumulative\_times
- short is\_stopping = 0

## 5.23.1 Function Documentation

#### 5.23.1.1 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 27 of file Process.c.



# 5.23.1.2 DeleteLockfile()

```
int DeleteLockfile ( )
```

Deletes the lockfile.

Returns

0 on success, -1 on failure.

Definition at line 39 of file Process.c.

Here is the caller graph for this function:



# 5.23.1.3 DoesLockfileExist()

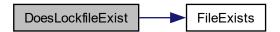
```
short DoesLockfileExist ( )
```

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

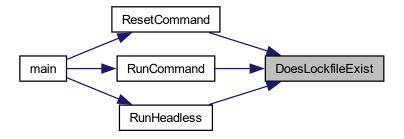
Returns

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

Definition at line 22 of file Process.c.



Here is the caller graph for this function:



# 5.23.1.4 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 214 of file Process.c.



## 5.23.1.5 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.

Definition at line 79 of file Process.c.

Here is the caller graph for this function:



#### 5.23.1.6 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.

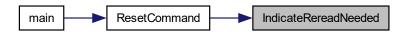
## Warning

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 62 of file Process.c.



#### 5.23.1.7 Initialize()

```
int Initialize ( )
```

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

## Note

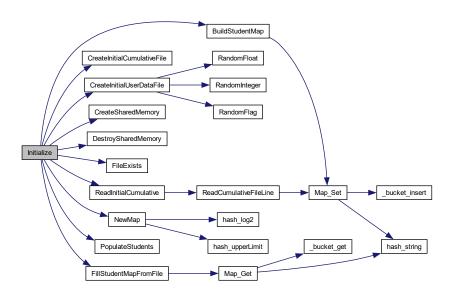
- (1) Create an initial user data file if it doesn't exist
- (2) Initialize the students array.
- (3) Initialize the students map.
- (4) Read the data from the user data file into the map/array.
- (5) Initializes the shared memory segment.

#### Returns

The ID of the shared memory segment. If -1, there was an error.

Definition at line 117 of file Process.c.

Here is the call graph for this function:





## 5.23.1.8 IsRereadNeeded()

```
short IsRereadNeeded ( )
```

Reads the lockfile for the reread flag.

## Warning

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

Definition at line 96 of file Process.c.

Here is the caller graph for this function:



## 5.23.1.9 Process()

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

Called repeatedly with a delay.

## Note

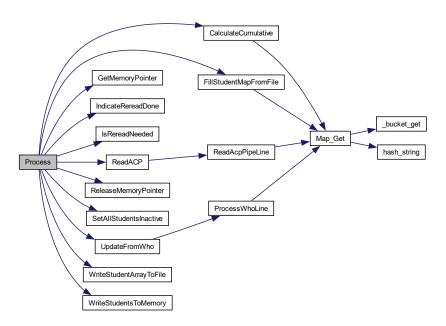
- (1) Sets all users to inactive.
- (2) Reads the result of the  ${\tt who}$  command, setting some users to active, and possibly changing 'dirty' and last login times.
- (3) Overwrites the user data file if we are dirty.
- (4) Sets dirty to false.
- (5) Re-writes the shared memory.

## **Parameters**

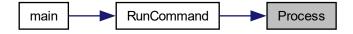
shm⊷	The ID of the shared memory segment.
_id	

Definition at line 167 of file Process.c.

Here is the call graph for this function:



Here is the caller graph for this function:



# 5.23.1.10 ResetCommand()

void ResetCommand ( )

Deletes and re-creates the static-user-data file and cumulative login file.

Note

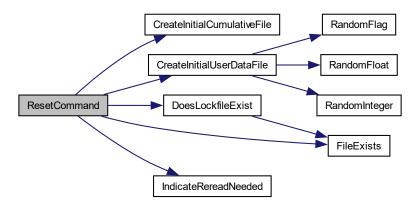
To execute the command, pass "reset" as an argument to the program.

Warning

This will clear login times.

Definition at line 288 of file Process.c.

Here is the call graph for this function:



Here is the caller graph for this function:



# 5.23.1.11 RunCommand()

void RunCommand ( )

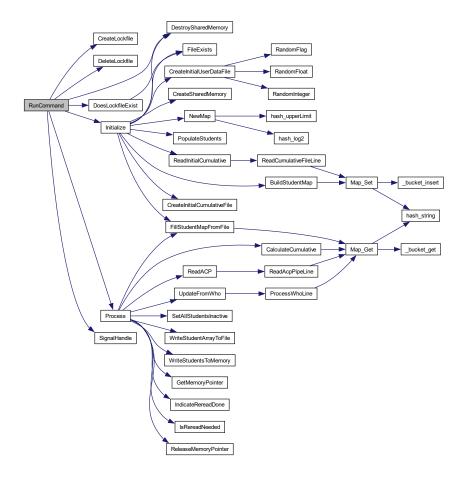
If a server exists, stops it. Begins the process loop.

Note

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

Definition at line 225 of file Process.c.

Here is the call graph for this function:





## 5.23.1.12 RunHeadless()

Uses nohup ./{processName} run to run the prodess headlessly.

## **Parameters**

processName
-------------

Definition at line 327 of file Process.c.

Here is the call graph for this function:



Here is the caller graph for this function:



# 5.23.1.13 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.

Returns

0 on success, -1 on error

Definition at line 104 of file Process.c.

Here is the caller graph for this function:



# 5.23.1.14 StopCommand()

void StopCommand ( )

Stops an existing server process if it is running.

Note

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

Definition at line 263 of file Process.c.

Here is the call graph for this function:





## 5.23.1.15 TerminateExistingServer()

```
int TerminateExistingServer ( )
```

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

Sends a SIGTERM signal to that process.

#### Warning

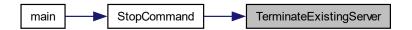
lockfile should be confirmed to exist

# 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 44 of file Process.c.

Here is the caller graph for this function:



# 5.23.2 Variable Documentation

#### 5.23.2.1 initial\_cumulative\_times

```
map* initial_cumulative_times
```

Definition at line 18 of file Process.c.

## 5.23.2.2 is\_stopping

```
short is\_stopping = 0
```

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

Definition at line 113 of file Process.c.

5.24 Process.c 95

#### 5.23.2.3 student\_map

```
map* student_map
```

Definitions for functions that manage control flow.

Definition at line 17 of file Process.c.

# 5.24 Process.c

Go to the documentation of this file.

```
00001
00005 #include "Process.h"
00006 #include "Files.h"
00007 #include "Data.h"
00008 #include "Build.h"
00009 #include "memShare.h"
00010 #include <errno.h>
00011 #include <stdio.h>
00012 #include <unistd.h>
00013 #include <signal.h>
00014 #include <stdlib.h>
00015 #include <string.h>
00016
00017 map *student_map;
00018 map *initial_cumulative_times;
00019
00020 // ~~~~~ Lockfile Commands ~~~~~~~
00021
00022 short DoesLockfileExist()
00023 {
00024
          return FileExists(LOCKFILE);
00025 }
00026
00027 int CreateLockfile()
00028 {
          FILE *file = fopen(LOCKFILE, "w");
00029
          if (file == NULL)
00030
00031
00032
              return -1;
00033
00034
          fprintf(file, "0 %d", getpid());
00035
          fclose(file);
00036
          return 0;
00037 }
00038
00039 int DeleteLockfile()
00040 {
00041
          return remove (LOCKFILE);
00042 }
00043
00044 int TerminateExistingServer()
00045 {
          FILE *file = fopen(LOCKFILE, "r");
if (file == NULL)
{
00046
00047
00048
00049
              return -1;
00050
00051
           int need_rewrite;
00052
           int pid = 0;
           fscanf(file, "%d %d", &need_rewrite, &pid);
00053
00054
           fclose(file):
00055
           if (pid > 0)
00056
          {
00057
              return kill(pid, SIGTERM);
00058
00059
           return -2;
00060 }
00061
00062 int IndicateRereadNeeded()
00063 {
          FILE *file = fopen(LOCKFILE, "r+");
if (file == NULL)
00064
00065
00066
00067
               return -1:
00068
00069
          int err = 0;
```

```
err = fseek(file, 0, SEEK_SET);
          if (!err)
00071
00072
00073
              fputc('1', file);
00074
00075
          err = fclose(file);
00076
          return err;
00077 }
00078
00079 int IndicateRereadDone()
00080 {
          FILE *file = fopen(LOCKFILE, "r+");
00081
          if (file == NULL)
00082
00083
00084
              return -1;
00085
          int err = 0;
00086
          err = fseek(file, 0, SEEK_SET);
00087
          if (!err)
00088
00089
          {
00090
              fputc('0', file);
00091
00092
          err = fclose(file);
00093
          return err;
00094 }
00095
00096 short IsRereadNeeded()
00097 {
          FILE *file = fopen(LOCKFILE, "r");
char firstc = fgetc(file);
00098
00099
00100
          fclose(file);
00101
          return firstc == '1';
00102 }
00103
00104 void SignalHandle(int signo)
00105 {
00106
          printf("Received shutdown signal.\n");
          if (signo == SIGINT || signo == SIGTERM)
00108
          {
00109
              is_stopping = 1;
00110
00111 }
00112
00113 short is_stopping = 0;
00114
00115 // ----- CLI Commands -----
00116
00117 int Initialize()
00118 {
00119
          int err:
00120
          if (!FileExists(STATIC_USER_DATA_FILE))
00121
00122
              printf("%s does not exist. Creating.\n", STATIC_USER_DATA_FILE);
00123
              err = CreateInitialUserDataFile(STATIC_USER_DATA_FILE, Data_IDs, DATA_NUM_RECORDS);
00124
              if (err)
00125
              {
00126
                  printf("Problem creating %s!\n", STATIC_USER_DATA_FILE);
00127
00128
          if (!FileExists(STATIC_USER_CUMULATIVE_FILE))
00129
00130
              printf("%s does not exist. Creating.\n", STATIC_USER_CUMULATIVE_FILE);
err = CreateInitialCumulativeFile(STATIC_USER_CUMULATIVE_FILE);
00131
00132
00133
00134
              {
00135
                  printf("Problem creating %s!\n", STATIC_USER_CUMULATIVE_FILE);
00136
              }
00137
00138
          PopulateStudents(Data_IDs, Data_Names, DATA_NUM_RECORDS);
          student_map = NewMap(50);
00139
00140
          BuildStudentMap(student_map, students, DATA_NUM_RECORDS);
00141
          err = FillStudentMapFromFile(student_map, STATIC_USER_DATA_FILE, Data_IDs, DATA_NUM_RECORDS);
00142
          if (err)
00143
          {
00144
              printf("Problem filling student map from %s!\n", STATIC USER DATA FILE);
00145
00146
          printf("Student data retrieved from file.\n");
00147
00148
          initial cumulative times = NewMap(50);
00149
          err = ReadInitialCumulative(initial cumulative times, STATIC USER CUMULATIVE FILE);
00150
          if (err)
00151
          {
00152
              printf("Failed to read %s. Cumulative times may be wrong!", STATIC_USER_CUMULATIVE_FILE);
00153
          }
00154
00155
          dirty = 0;
00156
```

5.24 Process.c 97

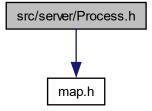
```
int shmid = CreateSharedMemory();
00158
          if (shmid == -1)
00159
              DestroySharedMemory();
shmid = CreateSharedMemory();
00160
00161
00162
00163
          printf("Shared memory allocated.\n");
00164
          return shmid;
00165 }
00166
00167 void Process(int shm_id)
00168 {
00169
          if (IsRereadNeeded())
00170
00171
              printf("\nReread\ indicated\ -\ rechecking\ user\ data\ file.");
00172
              FillStudentMapFromFile(student_map, STATIC_USER_DATA_FILE, Data_IDs, DATA_NUM_RECORDS);
00173
              IndicateRereadDone():
00174
00175
          SetAllStudentsInactive(students, DATA_NUM_RECORDS);
00176
          int err = ReadACP(student_map);
00177
00178
00179
              printf("Error piping ac -p command! \n");
00180
00181
          else
00182
         {
00183
              CalculateCumulative(students, DATA_NUM_RECORDS, initial_cumulative_times);
00184
00185
          err = UpdateFromWho(student_map);
00186
          if (err)
00187
          {
00188
              perror("Error updating from who!");
00189
00190
          if (dirty)
00191
              err = WriteStudentArrayToFile(students, DATA_NUM_RECORDS, STATIC_USER_DATA_FILE);
00192
00193
              if (err)
00194
              {
00195
                  printf("\nError updating %s!", STATIC_USER_DATA_FILE);
00196
00197
              else
00198
              {
00199
                  dirtv = 0;
00200
00201
00202
          void *ptr = GetMemoryPointer(shm_id);
00203
          if (ptr == (void *)-1)
00204
00205
              perror("Error attaching to shared memory");
00206
          }
00207
          else
00208
          {
00209
              WriteStudentsToMemory(ptr, students, DATA_NUM_RECORDS);
00210
              ReleaseMemoryPointer(ptr);
00211
          }
00212 }
00213
00214 void HelpCommand()
00215 {
          printf("\nUsage: server [OPTION]\n\n");
00216
          printf("Options: \n");
00217
          printf("\thelp\t\t\tShows the possible program commands\n");
00218
         00219
00220
00221
          printf("\trun\t\t\tCreates a new server with output to the shell if a server isn't already
     running.\n");
00222
          printf("\theadless\t\tCreates a new headless server if a server isn't already running.\n\n");
00223 }
00224
00225 void RunCommand()
00226 {
00227
          printf("\nRunning server.\n");
00228
          if (DoesLockfileExist())
00229
00230
              printf("\nServer is already running. Run 'server stop' to shut it down first.\n");
00231
00232
00233
          int err = CreateLockfile();
00234
          if (err)
00235
          {
              printf("\nFailed to create lockfile! Exiting.\n");
00236
00237
              return;
00238
00239
          int shm_id = Initialize();
          signal(SIGTERM, SignalHandle);
signal(SIGINT, SignalHandle);
00240
00241
00242
          printf("Server started.\n");
```

```
00243
          fflush(stdout);
00244
          while (!is_stopping)
00245
00246
              Process (shm_id);
00247
              sleep(1);
00248
00249
          printf("Server shutting down.\n");
00250
          err = DeleteLockfile();
00251
          if (err)
00252
              printf("Failed to delete lockfile!\n");
00253
00254
00255
          err = DestroySharedMemory();
00256
          if (err)
00257
          {
00258
              printf("Failed to destroy shared memory!\n");
00259
00260
          printf("Server terminated.\n");
00261 }
00262
00263 void StopCommand()
00264 {
          printf("\nStopping server...\n");
00265
00266
          int err = TerminateExistingServer();
00267
          if (err)
00268
          {
00269
              if (err == -1)
00270
              {
                  printf("Server isn't running.\n");
00271
00272
00273
              else if (err == -2)
00274
              {
00275
                  printf("Lockfile \ did \ not \ contain \ a \ valid \ process \ id! \ ");
00276
00277
              else
00278
              {
00279
                  printf("Sending terminate signal failed!\n");
00280
00281
00282
          else
00283
          {
              printf("Server terminated.\n");
00284
00285
00286 }
00287
00288 void ResetCommand()
00289 {
00290
          int err;
00291
00292
          if (FileExists(STATIC_USER_DATA_FILE))
00293
          {
00294
              printf("User data file exists. Deleting...\n");
00295
              remove(STATIC_USER_DATA_FILE);
00296
          }
00297
00298
          printf("Creating new data file.\n");
00299
          err = CreateInitialUserDataFile(STATIC_USER_DATA_FILE, Data_IDs, DATA_NUM_RECORDS);
00300
00301
              \label{lem:printf}  \mbox{printf("Problem creating $s:\n", STATIC_USER_DATA_FILE);} 
00302
00303
00304
          else
00305
          {
00306
              printf("%s created.\n", STATIC_USER_DATA_FILE);
00307
00308
00309
          printf("Creating new cumulative file.\n");
          err = CreateInitialCumulativeFile(STATIC_USER_CUMULATIVE_FILE);
00310
00311
          if (err)
00312
          {
00313
              printf("Problem creating %s!\n", STATIC_USER_CUMULATIVE_FILE);
00314
00315
          else
00316
00317
              printf("%s created.\n", STATIC_USER_CUMULATIVE_FILE);
00318
00319
00320
          if (DoesLockfileExist())
00321
              printf("Indicated re-read to running server process.\n");
00322
00323
              IndicateRereadNeeded();
00324
          }
00325 }
00326
00327 void RunHeadless(char *processName)
00328 {
00329
          if (DoesLockfileExist())
```

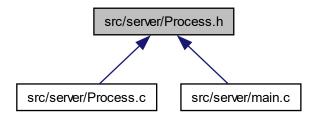
```
00330
            {
00331
                  printf("Server process already running.\n");
00332
00333
            char commandFront[] = " nohup ";
char commandEnd[] = " run & exit";
size_t comm_length = strlen(commandFront) + strlen(commandEnd) + strlen(processName) + 1;
00334
00335
00336
00337
            char *commandFull = malloc(comm_length * sizeof(char));
00338
            memset(commandFull, 0, comm_length * sizeof(char));
00339
             strcpy(commandFull, commandFront);
            strcat(commandFull, processName);
strcat(commandFull, commandEnd);
00340
00341
00342
            printf("Executing: %s\n", commandFull);
popen(commandFull, "we");
00343
00344
00345
            printf("Server running headlessly.\n");
00346 }
```

# 5.25 src/server/Process.h File Reference

#include "map.h"
Include dependency graph for Process.h:



This graph shows which files directly or indirectly include this file:



#### **Macros**

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

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

#### **Functions**

- short DoesLockfileExist ()
- int CreateLockfile ()
- int DeleteLockfile ()
- int TerminateExistingServer ()
- int IndicateRereadNeeded ()
- int IndicateRereadDone ()
- short IsRereadNeeded ()
- void SignalHandle (int signo)
- int Initialize ()
- void Process (int shm id)
- void ClearCommand ()
- void ResetCommand ()
- void StopCommand ()
- void RunCommand ()
- void HelpCommand ()
- void RunHeadless (char \*processName)

#### **Variables**

- short is\_stopping
- map \* student\_map

Definitions for functions that manage control flow.

## 5.25.1 Macro Definition Documentation

#### 5.25.1.1 LOCKFILE

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

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

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

It carries two pieces of data. The first is a 0 or 1, telling the server whether user data has been reset and needs to be re-read.

The second is an int corresponding to the process ID, so a close signal can be sent to the running process when a user enters server close.

Definition at line 17 of file Process.h.

# 5.25.2 Function Documentation

# 5.25.2.1 ClearCommand()

```
void ClearCommand ( )
```

Clears / Deallocates the shared virtual memory segment.

Note

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

#### 5.25.2.2 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 27 of file Process.c.



# 5.25.2.3 DeleteLockfile()

```
int DeleteLockfile ( )
```

Deletes the lockfile.

Returns

0 on success, -1 on failure.

Definition at line 39 of file Process.c.

Here is the caller graph for this function:



# 5.25.2.4 DoesLockfileExist()

```
short DoesLockfileExist ( )
```

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

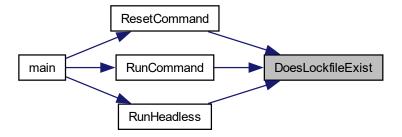
Returns

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

Definition at line 22 of file Process.c.



Here is the caller graph for this function:



# 5.25.2.5 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 214 of file Process.c.



#### 5.25.2.6 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.

Definition at line 79 of file Process.c.

Here is the caller graph for this function:



#### 5.25.2.7 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.

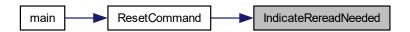
#### Warning

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 62 of file Process.c.



#### 5.25.2.8 Initialize()

```
int Initialize ( )
```

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

#### Note

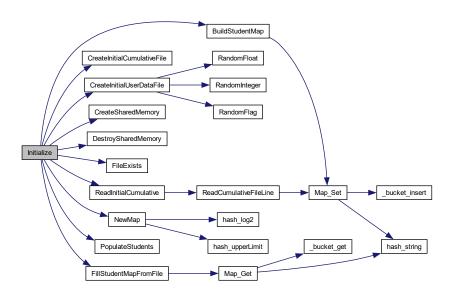
- (1) Create an initial user data file if it doesn't exist
- (2) Initialize the students array.
- (3) Initialize the students map.
- (4) Read the data from the user data file into the map/array.
- (5) Initializes the shared memory segment.

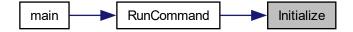
#### Returns

The ID of the shared memory segment. If -1, there was an error.

Definition at line 117 of file Process.c.

Here is the call graph for this function:





#### 5.25.2.9 IsRereadNeeded()

```
short IsRereadNeeded ( )
```

Reads the lockfile for the reread flag.

#### Warning

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

Definition at line 96 of file Process.c.

Here is the caller graph for this function:



#### 5.25.2.10 Process()

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

Called repeatedly with a delay.

# Note

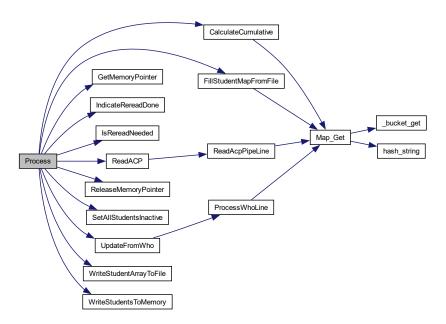
- (1) Sets all users to inactive.
- (2) Reads the result of the  ${\tt who}$  command, setting some users to active, and possibly changing 'dirty' and last login times.
- (3) Overwrites the user data file if we are dirty.
- (4) Sets dirty to false.
- (5) Re-writes the shared memory.

#### **Parameters**

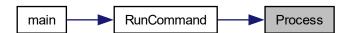
shm⊷	The ID of the shared memory segment.
_id	

Definition at line 167 of file Process.c.

Here is the call graph for this function:



Here is the caller graph for this function:



# 5.25.2.11 ResetCommand()

void ResetCommand ( )

Deletes and re-creates the static-user-data file and cumulative login file.

Note

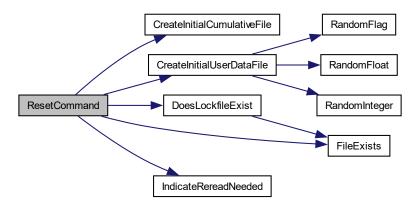
To execute the command, pass "reset" as an argument to the program.

Warning

This will clear login times.

Definition at line 288 of file Process.c.

Here is the call graph for this function:



Here is the caller graph for this function:



# 5.25.2.12 RunCommand()

void RunCommand ( )

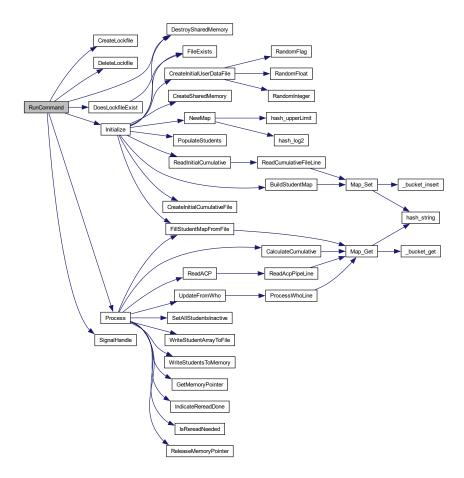
If a server exists, stops it. Begins the process loop.

Note

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

Definition at line 225 of file Process.c.

Here is the call graph for this function:





#### 5.25.2.13 RunHeadless()

Uses nohup ./{processName} run to run the prodess headlessly.

#### **Parameters**

ſ	processName	The name of the currently running process.	1
---	-------------	--	---

Definition at line 327 of file Process.c.

Here is the call graph for this function:



Here is the caller graph for this function:



# 5.25.2.14 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**

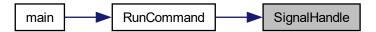
signo	The signal number. Will be SIGTERM from the other server process or SIGINT if interrupted from the
	console.

Returns

0 on success, -1 on error

Definition at line 104 of file Process.c.

Here is the caller graph for this function:



# 5.25.2.15 StopCommand()

void StopCommand ( )

Stops an existing server process if it is running.

Note

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

Definition at line 263 of file Process.c.

Here is the call graph for this function:





#### 5.25.2.16 TerminateExistingServer()

```
int TerminateExistingServer ( )
```

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

Sends a SIGTERM signal to that process.

#### Warning

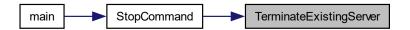
lockfile should be confirmed to exist

# 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 44 of file Process.c.

Here is the caller graph for this function:



# 5.25.3 Variable Documentation

# 5.25.3.1 is\_stopping

```
short is_stopping [extern]
```

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

Definition at line 113 of file Process.c.

# 5.25.3.2 student\_map

```
map* student_map [extern]
```

Definitions for functions that manage control flow.

Definition at line 17 of file Process.c.

5.26 Process.h 113

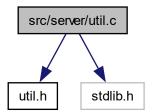
# 5.26 Process.h

#### Go to the documentation of this file.

```
00001 #ifndef Process_h
00002 #define Process_h
00007 #include "map.h"
00017 #define LOCKFILE "/tmp/ecet-server.lock"
00018
00019 // ----- Lockfile Commands -----
00020
00026 short DoesLockfileExist();
00027
00036 int CreateLockfile();
00037
00042 int DeleteLockfile();
00043
00052 int TerminateExistingServer();
00053
00061 int IndicateRereadNeeded();
00062
00068 int IndicateRereadDone();
00069
00075 short IsRereadNeeded();
00076
00084 void SignalHandle(int signo);
00085
00089 extern short is_stopping;
00090
00091 // ~~~~~~ CLI Commands ~~~~~~~
00092
00093 extern map *student_map;
00094
00105 int Initialize();
00106
00119 void Process(int shm_id);
00120
00126 void ClearCommand();
00134 void ResetCommand();
00135
00141 void StopCommand();
00142
00148 void RunCommand();
00156 void HelpCommand();
00157
00162 void RunHeadless(char *processName);
00163
00164 #endif
```

# 5.27 src/server/util.c File Reference

```
#include "util.h"
#include <stdlib.h>
Include dependency graph for util.c:
```



# **Functions**

- int RandomInteger (int min, int max)
  - Definitions for helper functions.
- float RandomFloat (float min, float max)
- short RandomFlag (float percentage\_chance)

# 5.27.1 Function Documentation

# 5.27.1.1 RandomFlag()

Returns 1, percentage\_chance of the time.

#### **Parameters**

percentage_chance	The chance to return 1.
-------------------	-------------------------

Note

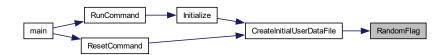
If percentage\_chance > 1, this will always return true.

Returns

1 or 0

Definition at line 21 of file util.c.

Here is the caller graph for this function:



# 5.27.1.2 RandomFloat()

Returns a float between min and max.

#### **Parameters**

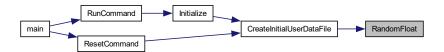
min	The minimum, inclusive.
max	The maximum, inclusive.

#### Returns

A random integer between min and max.

Definition at line 14 of file util.c.

Here is the caller graph for this function:



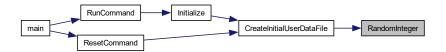
# 5.27.1.3 RandomInteger()

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

Definitions for helper functions.

Declarations for helper functions.

Definition at line 8 of file util.c.



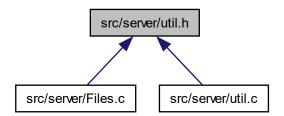
# 5.28 util.c

#### Go to the documentation of this file.

```
00004 #include "util.h"
00005
00006 #include <stdlib.h>
00007
00008 int RandomInteger(int min, int max)
00009 {
00010
          int r_add = rand() % (max - min + 1);
00011
          return r_add + min;
00012 }
00013
00014 float RandomFloat(float min, float max)
00015 {
00016
          float dif = max - min;
00017
          int rand_int = rand() % (int)(dif * 10000);
00018
          return min + (float)rand_int / 10000.0;
00019 }
00020
00021 short RandomFlag(float percentage_chance)
00022 {
00023
          float random_value = (float)rand() / RAND_MAX;
00024
          if (random_value < percentage_chance)</pre>
00025
00026
              return 1:
00027
00028
          return 0;
00029 }
```

# 5.29 src/server/util.h File Reference

This graph shows which files directly or indirectly include this file:



# **Functions**

- int RandomInteger (int min, int max)
  - Declarations for helper functions.
- float RandomFloat (float min, float max)
- short RandomFlag (float percentage chance)

#### 5.29.1 Function Documentation

# 5.29.1.1 RandomFlag()

Returns 1, percentage\_chance of the time.

#### **Parameters**

percentage_chance	The chance to return 1.
-------------------	-------------------------

Note

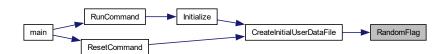
If  $percentage\_chance > 1$ , this will always return true.

Returns

1 or 0

Definition at line 21 of file util.c.

Here is the caller graph for this function:



# 5.29.1.2 RandomFloat()

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

Returns a float between min and max.

#### **Parameters**

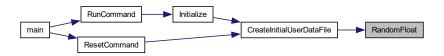
min	The minimum, inclusive.
max	The maximum, inclusive.

# Returns

A random integer between min and max.

Definition at line 14 of file util.c.

Here is the caller graph for this function:



#### 5.29.1.3 RandomInteger()

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

Declarations for helper functions.

Contains utility functions that are not coupled to any other data or structures in the program. Returns an integer between min and max.

#### **Parameters**

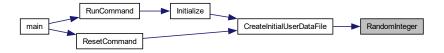
min	The minimum, inclusive.
max	The maximum, inclusive.

# Returns

A random integer between min and max.

Declarations for helper functions.

Definition at line 8 of file util.c.



5.30 util.h 119

# 5.30 util.h

```
Go to the documentation of this file.

00001 #ifndef util_h

00002 #define util_h

00015 int RandomInteger(int min, int max);
00015 int RandomInteger(int min, int max);
00016
00023 float RandomFloat(float min, float max);
00024
00031 short RandomFlag(float percentage_chance);
00032
00033 #endif
```

# Index

_bucket_delete	map.c, 64
map.c, 57	ClearCommand
bucket get	Process.h, 100
map.c, 57	CreateInitialCumulativeFile
bucket insert	Files.c, 42
map.c, 58	Files.h, 49
ap.0, 00	CreateInitialUserDataFile
active	Files.c, 42
Student, 9	Files.h, 50
age	CreateLockfile
Student, 10	Process.c, 83
	Process.h, 101
buckets	CreateSharedMemory
map, 8	memShare.c, 75
Build.c	memShare.h, 79
BuildStudentMap, 12	memerial em, 10
CalculateCumulative, 13	data
dirty, 21	map result, 8
PopulateStudents, 13	Data.c
ProcessWhoLine, 14	Data_IDs, 37
ReadACP, 15	Data_Names, 37
ReadAcpPipeLine, 16	Data.h
ReadCumulativeFileLine, 17	DATA_ID_MAX_LENGTH, 39
ReadInitialCumulative, 18	Data_IDs, 40
SetAllStudentsInactive, 19	DATA_NAME_MAX_LENGTH, 39
students, 21	Data_Names, 40
UpdateFromWho, 19	DATA_NUM_RECORDS, 40
WriteStudentsToMemory, 20	DATA_SIZE, 40
Build.h	DATA_ID_MAX_LENGTH
BuildStudentMap, 26	Data.h, 39
CalculateCumulative, 27	Data_IDs
dirty, 35	Data.c, 37
PopulateStudents, 27	Data.h, 40
ProcessWhoLine, 28	DATA_NAME_MAX_LENGTH
ReadACP, 29	Data.h, 39
ReadAcpPipeLine, 30	Data_Names
ReadCumulativeFileLine, 31	Data.c, 37
ReadInitialCumulative, 32	Data.h, 40
SetAllStudentsInactive, 33	DATA_NUM_RECORDS
students, 35	Data.h, 40
UpdateFromWho, 33	DATA_SIZE
WriteStudentsToMemory, 34	Data.h, 40
BuildStudentMap	DeleteLockfile
Build.c, 12	Process.c, 83
Build.h, 26	Process.h, 101
	DestroySharedMemory
CalculateCumulative	memShare.c, 76
Build.c, 13	memShare.h, 80
Build.h, 27	dirty
char_ratio	Build.c, 21

122 INDEX

Build.h, 35	Process.h, 104
DoesLockfileExist	is_stopping
Process.c, 84	Process.c, 94
Process.h, 102	Process.h, 112
	IsRereadNeeded
FileExists	Process.c, 87
Files.c, 43	Process.h, 105
Files.h, 51	,
Files.c	lastLogin
CreateInitialCumulativeFile, 42	Student, 10
CreateInitialUserDataFile, 42	LOCKFILE
FileExists, 43	Process.h, 100
FillStudentMapFromFile, 44	loginDuration
WriteStudentArrayToFile, 45	Student, 10
Files.h	Stadent, 10
CreateInitialCumulativeFile, 49	main
CreateInitialUserDataFile, 50	main.c, 54
FileExists, 51	main.c
FillStudentMapFromFile, 52	main, 54
·	map, 7
STATIC_USER_CUMULATIVE_FILE, 49	buckets, 8
STATIC_USER_DATA_FILE, 49	,
WriteStudentArrayToFile, 53	size, 8
FillStudentMapFromFile	map.c
Files.c, 44	_bucket_delete, 57
Files.h, 52	_bucket_get, 57
found	_bucket_insert, 58
map_result, 9	char_ratio, 64
fullName	hash_log2, 58
Student, 10	hash_string, 59
	hash_upperLimit, 60
GetMemoryPointer	Map_Delete, 60
memShare.c, 76	Map_Get, 61
memShare.h, 80	Map_Set, 62
gpa	NewMap, 63
Student, 10	map.h
·	hash_log2, 67
hash_log2	hash string, 69
map.c, 58	hash_upperLimit, 70
map.h, 67	Map_Delete, 70
hash_string	Map_Get, 71
map.c, 59	Map_Set, 71 Map_Set, 72
map.h, 69	NewMap, 73
hash_upperLimit	Map_Delete
map.c, 60	
map.h, 70	map.c, 60
HelpCommand	map.h, 70
Process.c, 85	Map_Get
,	map.c, 61
Process.h, 103	map.h, 71
IndicateRereadDone	map_result, 8
	data, 8
Process.c, 85	found, 9
Process.h, 103	Map_Set
IndicateRereadNeeded	map.c, <mark>62</mark>
Process.c, 86	map.h, 72
Process.h, 104	MEM_KEY
initial_cumulative_times	memShare.h, 79
Process.c, 94	MEM PERMISSIONS
Initialize	memShare.h, 79
Process.c, 86	MEM_SIZE

INDEX 123

memShare.h, 79	RunHeadless, 109
memShare.c	SignalHandle, 110
CreateSharedMemory, 75	StopCommand, 111
DestroySharedMemory, 76	student_map, 112
GetMemoryPointer, 76	TerminateExistingServer, 111
ReleaseMemoryPointer, 77	ProcessWhoLine
memShare.h	Build.c, 14
CreateSharedMemory, 79	Build.h, 28
DestroySharedMemory, 80	Bana.11, 20
GetMemoryPointer, 80	RandomFlag
MEM KEY, 79	util.c, 114
	util.h, 116
MEM_PERMISSIONS, 79	RandomFloat
MEM_SIZE, 79	util.c, 114
ReleaseMemoryPointer, 81	util.h, 117
Name	
NewMap	RandomInteger
map.c, 63	util.c, 115
map.h, 73	util.h, 118
Danielata Cherdanta	ReadACP
PopulateStudents	Build.c, 15
Build.c, 13	Build.h, 29
Build.h, 27	ReadAcpPipeLine
Process	Build.c, 16
Process.c, 88	Build.h, 30
Process.h, 106	ReadCumulativeFileLine
Process.c	Build.c, 17
CreateLockfile, 83	Build.h, 31
DeleteLockfile, 83	ReadInitialCumulative
DoesLockfileExist, 84	Build.c, 18
HelpCommand, 85	Build.h, 32
IndicateRereadDone, 85	ReleaseMemoryPointer
IndicateRereadNeeded, 86	memShare.c, 77
initial_cumulative_times, 94	memShare.h, 81
Initialize, 86	ResetCommand
is_stopping, 94	
IsRereadNeeded, 87	Process.c, 89
Process, 88	Process.h, 107
ResetCommand, 89	RunCommand
	Process.c, 90
RunCommand, 90	Process.h, 108
RunHeadless, 91	RunHeadless
SignalHandle, 92	Process.c, 91
StopCommand, 93	Process.h, 109
student_map, 94	0.480.
TerminateExistingServer, 93	SetAllStudentsInactive
Process.h	Build.c, 19
ClearCommand, 100	Build.h, 33
CreateLockfile, 101	SignalHandle
DeleteLockfile, 101	Process.c, 92
DoesLockfileExist, 102	Process.h, 110
HelpCommand, 103	size
IndicateRereadDone, 103	map, 8
IndicateRereadNeeded, 104	src/server/Build.c, 11, 22
Initialize, 104	src/server/Build.h, 24, 36
is_stopping, 112	src/server/Data.c, 36, 38
IsRereadNeeded, 105	src/server/Data.h, 38, 41
LOCKFILE, 100	src/server/Files.c, 41, 46
Process, 106	src/server/Files.h, 47, 53
ResetCommand, 107	src/server/main.c, 54, 55
RunCommand, 108	src/server/map.c, 56, 64
nuncommanu, 100	510/5e1ve1/111ap.c, 50, 54

124 INDEX

```
src/server/map.h, 67, 74
src/server/memShare.c, 75, 77
src/server/memShare.h, 78, 81
src/server/Process.c, 82, 95
src/server/Process.h, 99, 113
src/server/util.c, 113, 116
src/server/util.h, 116, 119
STATIC_USER_CUMULATIVE_FILE
     Files.h, 49
STATIC_USER_DATA_FILE
     Files.h, 49
StopCommand
     Process.c, 93
     Process.h, 111
Student, 9
    active, 9
     age, 10
     fullName, 10
    gpa, 10
    lastLogin, 10
    loginDuration, 10
    userID, 10
student_map
     Process.c, 94
     Process.h, 112
students
     Build.c, 21
     Build.h, 35
TerminateExistingServer
     Process.c, 93
     Process.h, 111
UpdateFromWho
     Build.c, 19
     Build.h, 33
userID
     Student, 10
util.c
     RandomFlag, 114
     RandomFloat, 114
     RandomInteger, 115
util.h
     RandomFlag, 116
     RandomFloat, 117
     RandomInteger, 118
WriteStudentArrayToFile
     Files.c, 45
     Files.h, 53
WriteStudentsToMemory
     Build.c, 20
     Build.h, 34
```