3410 Final Project Part 1
Group 15
Names: Shaina Stein, Victoria Hennemann, Zuxuan Chen

record.h

Construct_record()
The first thing we do is open up record_fname for reading and writing. This is done with the open() method. The open() method opens a file based on a pathname, in this case, (record_fname), and returns an int that we refer to as the file descriptor. We check for any errors with the file descriptor to make sure it is valid. We use the lseek method to get to the end of the file and write the end of line character, "\0" using the write() method. We also error check here to see if these method calls return -1. The next step to occur is the mmapping of the file into the virtual address space. The starting address is 0, the length of the file is RECORD_FILESIZE, and we give it read or write protection. The file descriptor is the one we created before ("fd") and the offset is 0. The mmap function call returns the address at which the mapping was placed, buffer. Lastly, we call to memset, which copies the blank character, ' ', to the entirety of the buffer.

update_record()
In this method, we take all of the arguments to the function, the weather packet data, and write it into the record.bin file where each value is separated by a comma. To get the total number of chars/current position of entries in the file, we multiply the size of each record length by the number of total records in the file and then by the size of a char. Then, to get to that spot, we create a char* var called index. Next, we call the function sprintf() which sends output to a string pointed to by the variable "index". We take each of the arguments to the update_record function and write them to index, which is basically the end of the file, so that for each new record, we write all of the data on a new line.

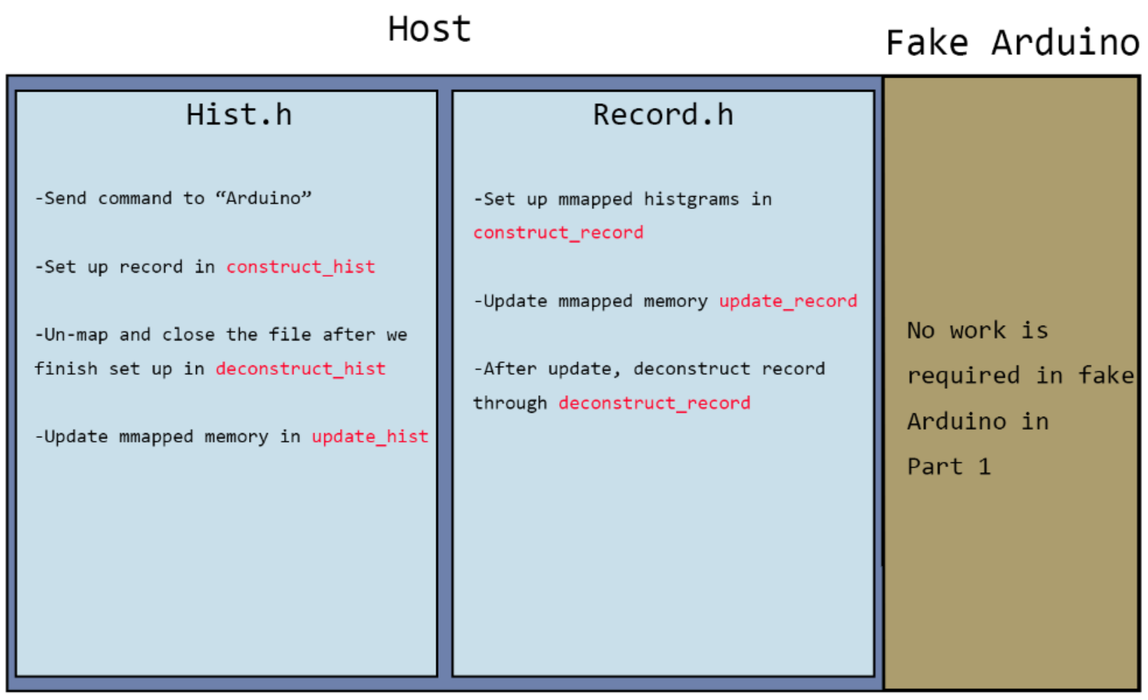
deconstruct_record()
This method unmaps and closes the file. It is basically the same thing as **deconstruct_hist()**. If completed successfully it will return 0, and if it fails it will return -1. After we finish mmapping the record in the construct_record() method, we will call this method, un-map the file and close it.

hist.h

update_hist()
In this method, the hist file is treated as two dimensional array characters. To get the current index of the 2D array, we first multiply time * NBUCKETS. There are 16 buckets along the x axis, and time is the number of rows along the y axis. We then add (value/NBUCKETS) to this value to get it in the correct bucket number. After, we set the temp to the value at hist[index] and increment it by one. Then set the index to the temp which increments the place is the hist file.

construct _hist()
In this method we first open a file for reading and writing with permissions granted to the user. Then we check to see if an invalid input is returned. If it is invalid it will return -1. Then we seek to the end of the file and write a '\0' to extend the file. After this we mmap the proper region of the file into the user's address space.

deconstruct_hist()
This method unmaps and closes the file. If completed successfully it will return 0, and if it fails it will return -1. After we finish mmapping the histogram , we will call this method, un-map the file and close it.


Host
Fake Arduino
Hist.h
-Send command to "Arduino"
-Set up record in construct_hist
-Un-map and close the file after we finish set up in deconstruct_hist
-Update mmapped memory in update_hist
Record.h
-Set up mmapped histgrams in construct_record
-Update mmapped memory update_record
-After update, deconstruct record through deconstruct_record
No work is required in fake Arduino in Part 1