Author: Jenna Wolf

Program: JennaCFile

Class: C and the Posix environment

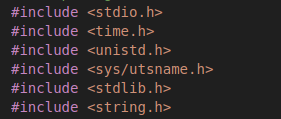
Assignment: Project 3 Linux C Program

Version: 04.02.24

Revision: 4

How To Use JennaCFile program

Purpose: The purpose of this program is to fit the guidelines for the project 3 Linux system c program assignment. This program works to access and display system information to the user. It also uses error checking to make sure proper inputs are made and display error messages when information retrieval fails. There is also an array that holds user data both inside and outside the program. There is also a bash script that can be ran and edited by the user.

**Packages**:

**#include <stdio.h>** - this package allows a c program to be made and ran. It allows inputs and outputs to be made in a c program.

**#include <time.h>** - allows time structure, which allows the time and date to be displayed

**#include <unistd.h>** - the package allows many different functions used throughout the program, such as the sleep function and the gethostname function.

**#include <sys/utsname.h>** - this package allows the program to find out system information, such as the kernel version and the release information

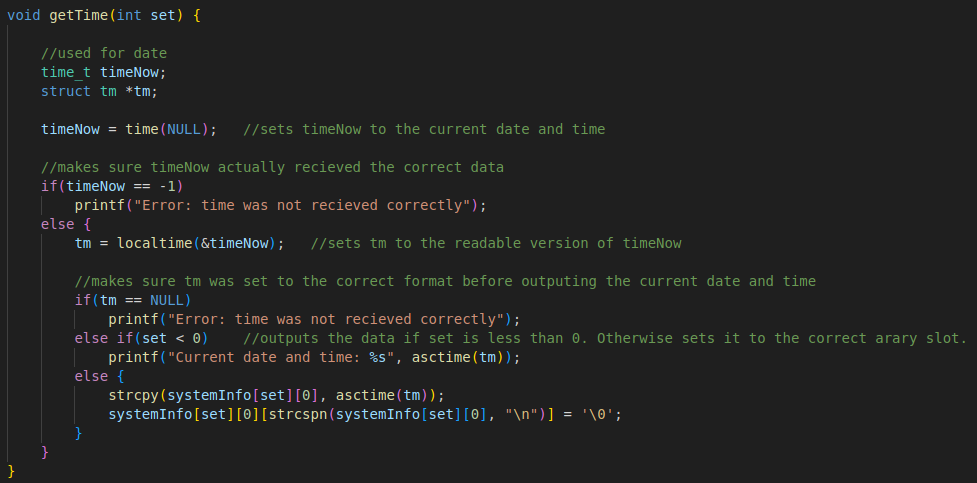
**#include <stdlib.h>** - This package allows for memory allocation, conversion, and other useful commands to the program. It is essential for working with environment variables.

**#include <string.h>** - This package allows for different forms of string manipulation in the program. It is the best way to do anything with strings in a c program.

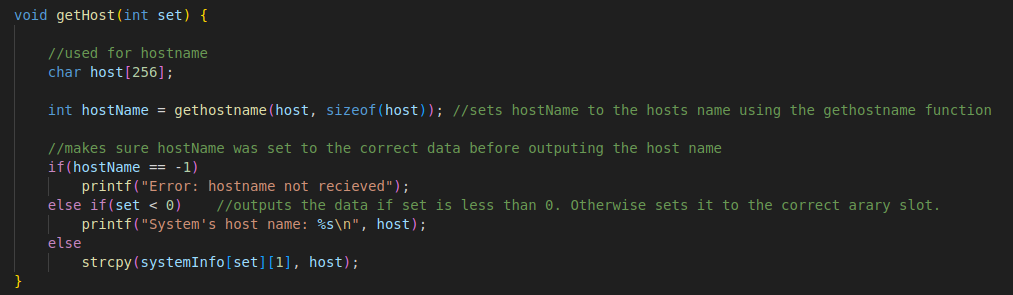
**Variables**

systemInfo is a three dimensional global character array that is used to hold the savable system information that can be gotten using this program. There are 5 slots for the user to use, and it saves all information attainable from this program so the user can see if later.

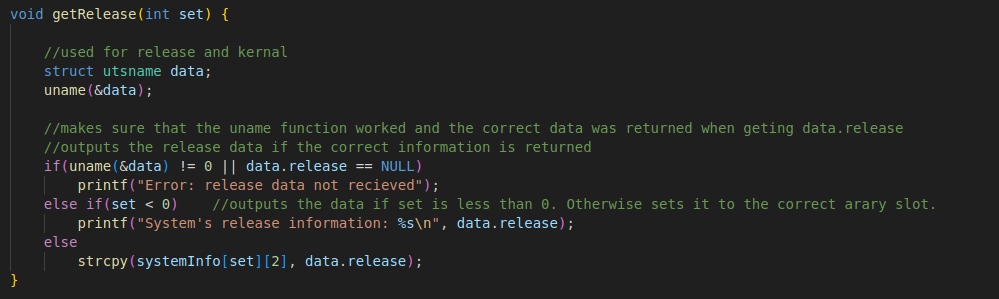
**Functions:**

The first function in the program is the getTime function (lines 29-53). This is a void function that has the set parameter, which is used for the location of the slot being accessed in the systemInfo array. The timeNow variable and the tm variable are the variables used to hold and find the current date and time. The time\_t and struct tm are used for time-related operations. TimeNow holds the current date and time, while tm structures it into a easily readable format.

This is the code for displaying the date and time (lines 38-50). The first piece of code (38) sets timeNow to the current time and date using the time function. We then reach an if statement (lines 38-52) which checks to make sure that timeNow was actually set and did not return an error. If it did not error out, then we go to the line assigns tm using the localtime function (line 41). This function takes the timeNow data and sets it to a readable format. We then reach our last if statement (line 44-51), which makes sure that tm was not set to Null. This is just making sure that tm was actually set to something before it is output. If this is the case, then we check to see weather or not set is greater than one. If it was not, then the time and date is output to the screen. If it was set, then we set the slot in systemInfo that corresponds with the set variable to the current date and time. We also get rid of the end line varaible (\n) at the end of the line (line 50).

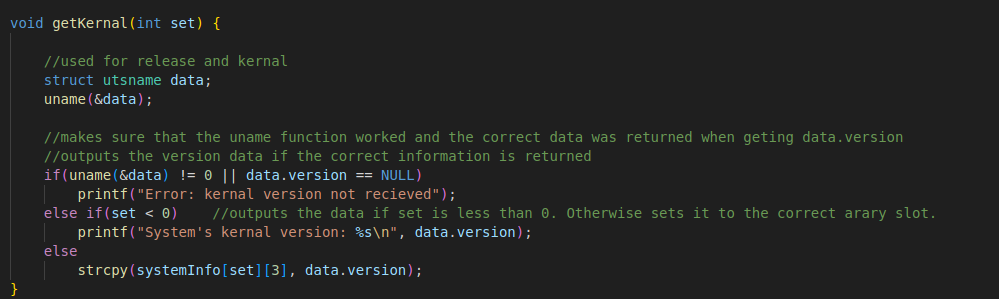
The second function in the program is the getHost function (lines 55-69). This is a void function that has the set parameter, which is used for the location of the slot being accessed in the systemInfo array. The host variable created here (line 58) is a character array of size 256. This is used to hold the hostname like a string. When the gethostname function is called, this variable is set to the host name.

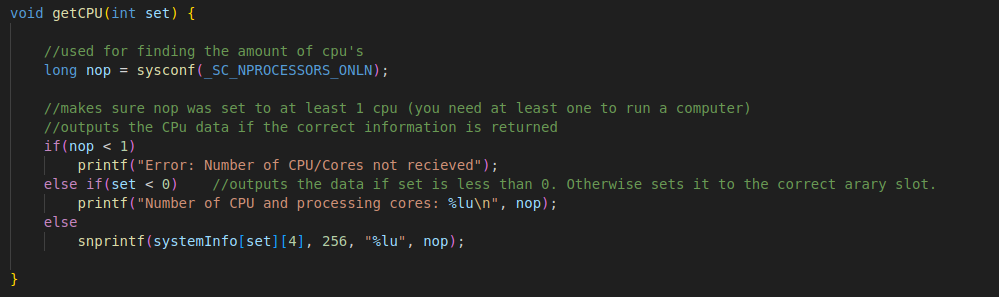
This function starts by creating a hostName variable (line 60) that uses the gethostname function. The parameters for this function uses the host character array along with the size of this variable to get and hold the host name. We then hit a if statement (lines 63-68) that make sure that hostName was actually set and did not return a error value. If this is passed, then we check to see weather or not set is greater than one. If it was not, then the host name is output to the screen. If it was set, then we set the slot in systemInfo that corresponds with the set variable to the host name.



The third function in the program is the getRelease function (lines 71-85). This is a void function that has the set parameter, which is used for the location of the slot being accessed in the systemInfo array. The data variable (line 74) is used to find and hold system information (the release information in this case). Struct utsname is used to access system information, and the uname function (line 75) actually sets to data variable to that system information.

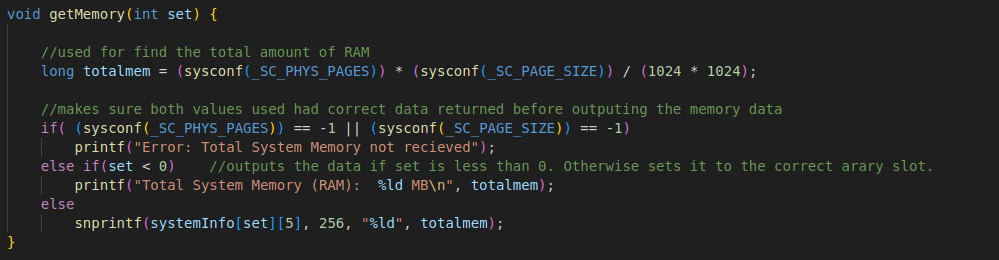
The chunk of code after this is simply made up of an if statement (lines 79-84) that makes sure of two things. It makes sure that the uname function with the data parameter is set to 0, as a 0 return means that the function was successful. It also makes sure that the data.release value is not returning a null value. If both of these conditions were passed, then we check to see weather or not set is greater than one. If it was not, then the release information is output to the screen. If it was set, then we set the slot in systemInfo that corresponds with the set variable to the release information.

The fourth function in the program is the getKernal function (lines 87-101). This is a void function that has the set parameter, which is used for the location of the slot being accessed in the systemInfo array. This function works the exact same way as the getRelease function, except the information being saved and output in the Kernal information. Everything else works the exact same, though.

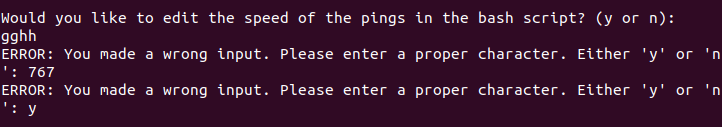
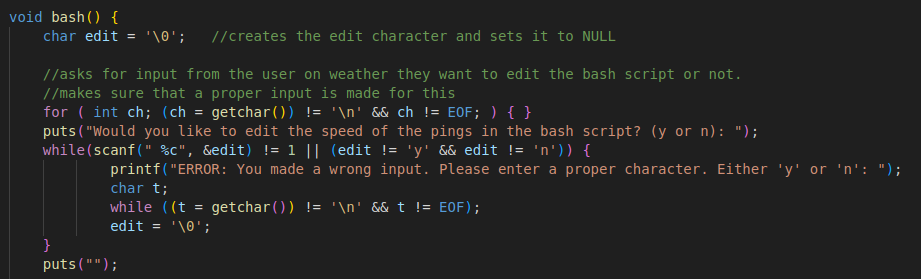


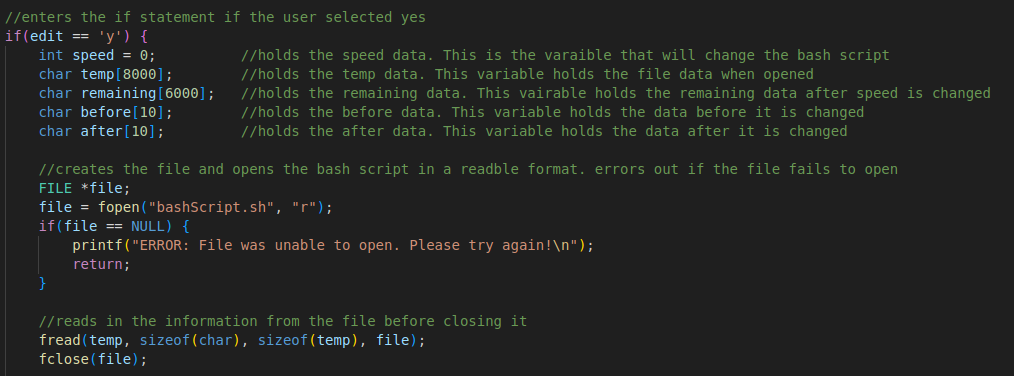
The fifth function in the program is the getCPU function (lines 103-117). This is a void function that has the set parameter, which is used for the location of the slot being accessed in the systemInfo array. The nop variable (line 106) is used to hold the number of processors/cores. The long type is used here because the value returned may need more memory space than an integer can hold. Sysconf is a function used to get system information. In this case, it is setting nop to the current number of cpus online using the parameter \_SC\_NPROCESSORS\_ONLN.

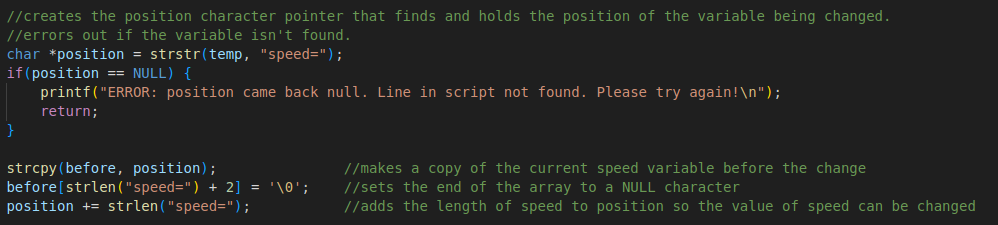
This part of the code is simply made up of an if statement (lines 110-115). This if statement makes sure that nop (number of processors) is at least one, since a computer needs at least one cpu to work. If this is passed, then we check to see weather or not set is greater than one. If it was not, then the number of cpu’s is output to the screen. If it was set, then we set the slot in systemInfo that corresponds with the set variable to the number of cpus.

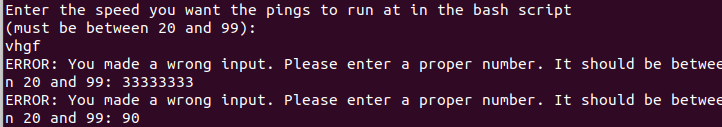
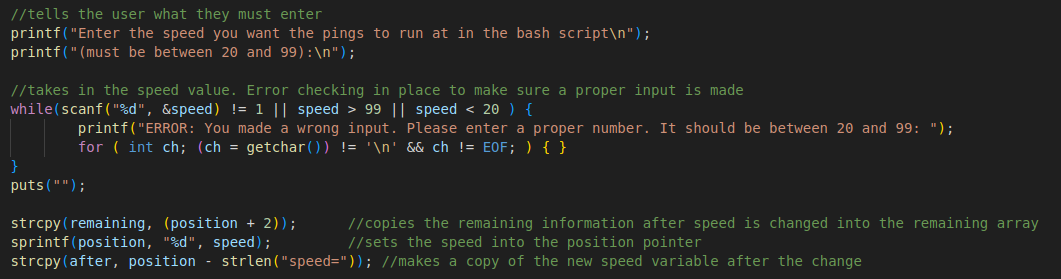
The sixth function in the program is the getMemory function (lines 119-131). This is a void function that has the set parameter, which is used for the location of the slot being accessed in the systemInfo array. The totalmem varaible (line 122) is used to hold the total amount of memory (RAM). This is a long type since this is a larger number, and we are again using the sysconf function since it has access to system information. We use the parameters \_SC\_PHYS\_PAGES and \_SC\_PAGE\_SIZE because when the amounts they return are multiplied together, they give us the total amount of physical memory on the system. PHYS\_PAGES gets the total number of physical memory pages on the system, and PAGE\_SIZE gets the size of a memory page in bytes. It is then divided by 1024\*1024 as this is the amount of bytes in a megabyte, which is the value we are storing memory as.

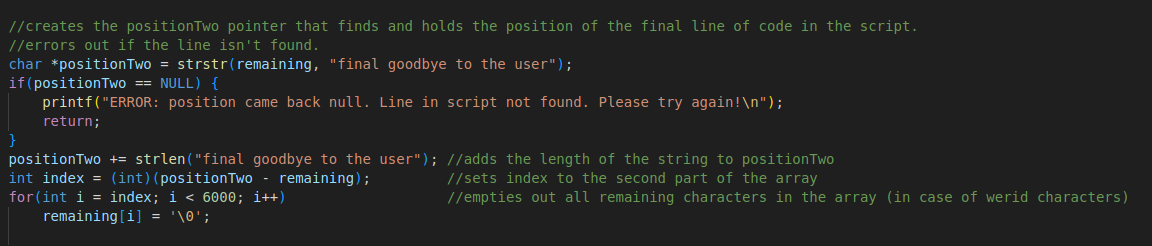
This chunk of code is simply made up of an if statement (lines 125-130) that makes sure of two things (again). It makes sure that the sysconf commands for both pages and page size do not return a negative one (meaning they don’t return an error value). If this condition is passed, then the total memory is output in a string. If this is passed, then we check to see weather or not set is greater than one. If it was not, then the memory is output to the screen. If it was set, then we set the slot in systemInfo that corresponds with the set variable to the memory.

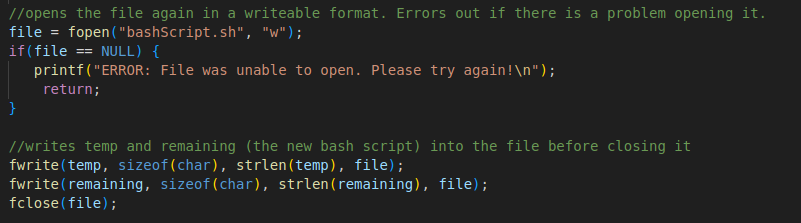
The sixth function in the program is the bash function (line 133-243). This is a void function with no parameters, and this is the function used to both edit and run the bash script. The first section of the code (line 134-147) starts the function by creating the edit variable and setting it to NULL. The input buffer is then cleared (line 140) and a message is output to the user on what they need to do (line 141). There is then a loop that takes the users input (line 141-146). This loop works to make sure a proper character (either y or n) is input. If it is not, then it clears the input buffer and asks for a proper input from the user. After the loop, a empty line is put on the screen (line 147)

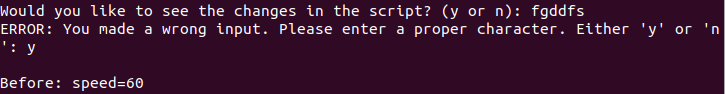
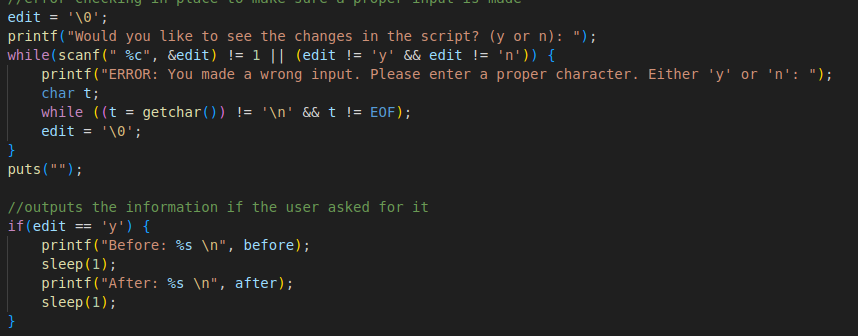
The next part of this function is an if statement (line 150-240) that is entered if the user enters y and chooses to edit the bash script. The first thing it does is set up 5 variables (line 151-155). four of these are arrays, and each of these variables are used to either hold file information or hold the edit the user wishes to make. The program then creates the file variable and opens the bash script in a readable format, erroring out if the file fails to open (line 158-163). If the file opens succesfully, it reads the entire script into the temp array before closing the file (line 166-167).

The next part of this function creates the position character pointer, which points to the spot in the script that the user will edit (line 171), this being the speed variable. If the pointer does not find this spot in the script, then it errors out and returns to the main function (line 172-175). After this, the function copies the current data of speed into the before array using the position pointer (line 177). It then sets the variable after this data to a NULL character, as to stop any weird characters in the array (line 178). The program then adds the length of the string “speed=” to position, as this sets it up to when the number for speed will be entered by the user (line 179).

The next part of the function is the part that takes in the new speed from the user. It first tells the user what they should input (line 182-183). It then takes in the input from the user in a while loop, making sure a proper input is made by the user (line 186-189). If the user gives a wrong input (not a number between 20 and 99), then an error is given and the input is cleared before it takes in a new input from the user. After the while loop is passed, a empty line is put on the screen (line 190). The program then copies the remaining bash script into the remaining array, allowing for the parts of the script after speed to be kept for later (line 192). It then sets the speed into the script based off of the position pointer, setting it so speed equals the new user input (line 193). The program then sets the after array, making it equal the new change in the bash script (line 194).

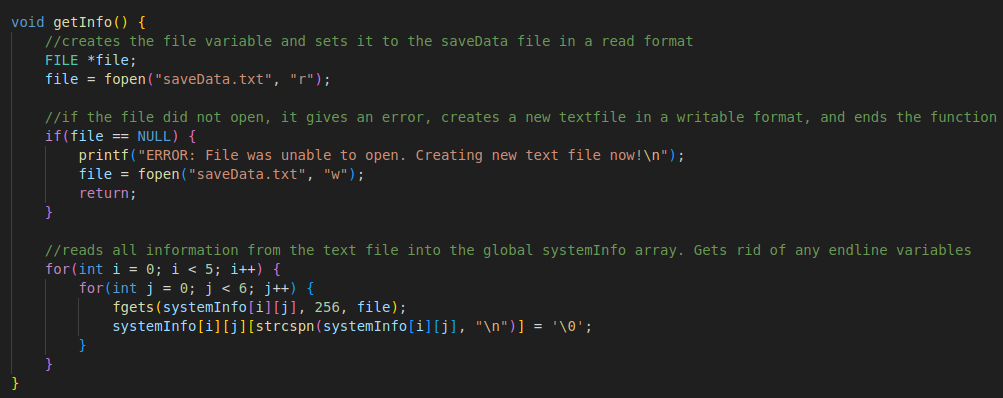
The next part of the function works to clear out any strange characters the array may have. It does this by creating a new pointer that points to the final line in the bash script (line 198). It then makes sure the pointer actually found the correct position; if it did not, it errors out and returns to the main function (line 199-202). If the if statement is passed, it then adds the string “final goodbye to the user”’s length to the positionTwo pointer (line 203). The program then creates the index variable, setting it to positionTwo minus remaining, which finds the second half of the array that needs to be set to NULL (line 204). It then enters a for loop that goes through the bottom of the remaining array, setting all characters to \0 (line 205-206).

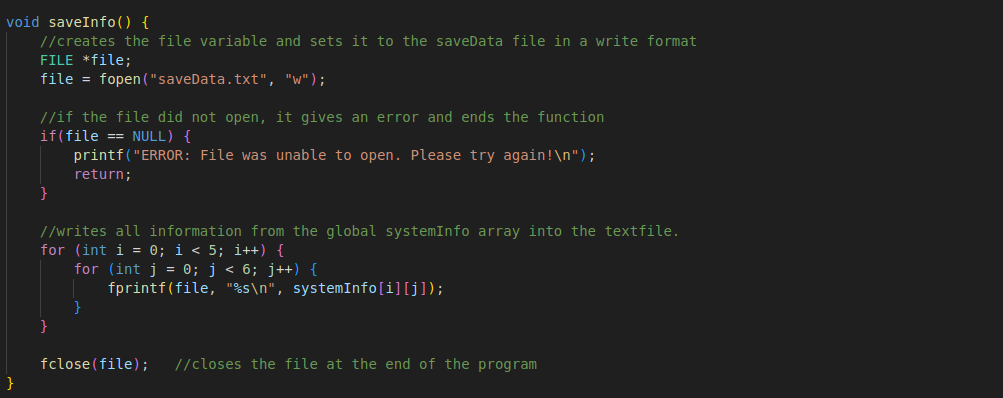
The function then opens the bash script again, this time in a writable format (line 209). It makes sure the bash script opens correctly, erroring out if it did not (line 210-213). If it opened correctly, it first writes the temp array with the new change to the script (line 216) before writing the remaining array to the script, filling in the rest of the original script (line 217). It then closes the file (line 218).

The next part of the script first sets edit back to NULL (line 222). It then asks the user if they would like to view the change that was made to the script (line 223). It enters a while loop to take this input in, making sure the user enters a proper character (line 224-229). It works just like the first loop for this method. We then output a blank line (line 230) before reaching an if statement (line 233-238) that checks to see if edit equals ‘y’. If this is the case, the program outputs the before and after array to the screen.



The last thing this function does is run the bash script through a system call (line 242)

The seventh function in the program is the getInfo function (line 245-264). This is a void function with no parameters, and it works to get saved information from the saveData text file and put it into the systemInfo array. The file variable (line 247) is of type FILE and Is needed to open a file of any type. The first thing this function does is open the saveData text file in a readable format (line 248), that way the program can read information in from the file. There is then an if statement (line 251-255) that makes sure the file was actually open (makes sure file wasn’t set to NULL). If the file was set to NULL, then the program gives the user an error message (line 252), makes a new saveData file for the program to use (does this by opening a writable saveData file, line 253), and returns from the function to the main program (line 254). If this for loop is passed, we then enter two for loops (line 258-263) that go through each slot of the systemInfo array, filling it with data from the text file if there is data there. It also gets rid of any end line variables (line 261). The function then ends.

The eight and final function in the program is the saveInfo function (line 266-285). This is a void function with no parameters, and it works to get save information to the saveData text file from the systemInfo array. The file variable (line 268) is of type FILE and Is needed to open a file of any type. The first thing this function does is open the saveData text file in a writable format (line 269), that way the program can write information in from the file. There is then an if statement (line 272-275) that makes sure the file was actually open (makes sure file wasn’t set to NULL). If the file was set to NULL, then the program gives the user an error message (line 273) and returns from the function to the main program (line 274). We then enter two for loops (line 278-282) that goes through each save slot and each piece of system information, saving it to the saveData text file. The function then ends.

**Code segment:**

****

The beginning of the main segment starts off with two variables. The choice variable, which is the variable that the user inputs to. It is a integer, so it will only take numbers from the user. The program goes based on the input this variable receives. Then, the arr variable, which is the variable that the user inputs which save slot they wish to access. This is used for the systemInfo array, and is also an integer. It can only be set to 1-5.



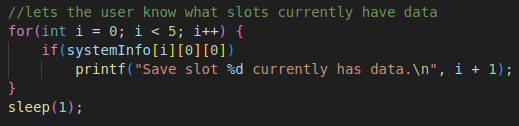


we then reach a function call for the getInfo function (line 291). This calls the getInfo function, which adds all previous information into the systemInfo array. If the saveData text file that it uses does not exist for some reason, we get the error message seen above and a new text file is made.





(line 294-295) The code then says a simple welcome statement to the user. It then sleeps for one second so the screen is not overwhelmed with statements right away.



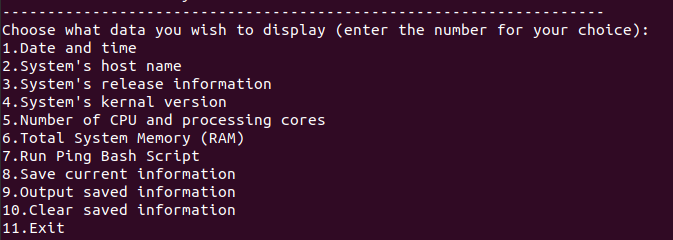
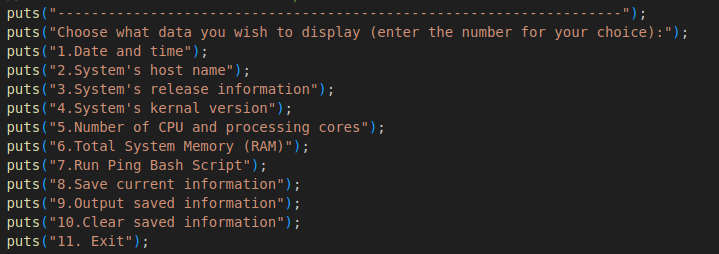


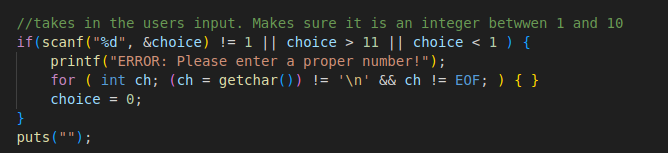
We then enter a short for loop (line 298-302). This while loop checks to see if any slots in the array have data put into them. If it does, it tells the user that said slot has data, that way the user knows which slots already have information.



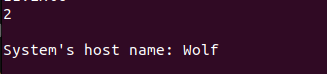


The program then enters a do while loop. This loop starts on line 304 and ends on line 383. This is the main part of the program for the user, as it holds the menu and input areas for the user. The loop exits when 11 is entered for the choice variable.

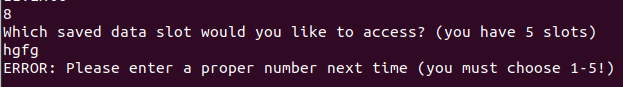
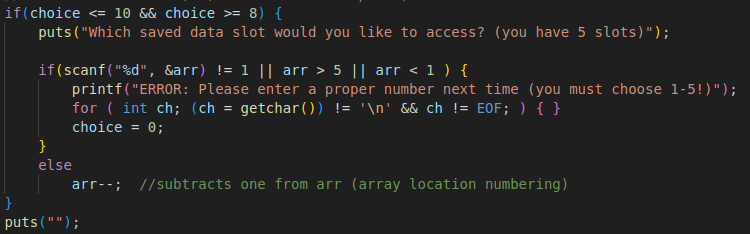
We then reach the menu part of the program (lines 306-318). This is the section that tells the user what to do. It lists the options the user has, along with a number for each option so the user knows what to input. There are 11 options for the user to choose from. This it output every time the code loops.

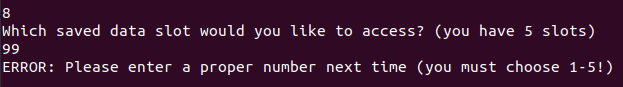


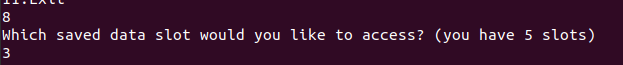




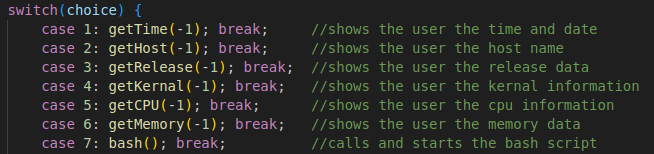
The input section of the code (lines 321-325) is where the user first gets to interact with the code. This is made into an if statement to check what the user is entering, This is used to make sure the user enters an integer, as if they do not, the if statement is entered. The while statement is also entered when an integer less that 1 or greater that 11 is entered. When the while statement is gone through, an error message is sent to the user first (line 322). This simply asks the user to input a proper input. After that, a for statement is gone through (line 323). This for loop is used to consume and clear out input if a character or string was entered. It goes through all the characters until it reaches a EOF or end character. Choice is then set to 0 (line 324) as a way of resting the input.





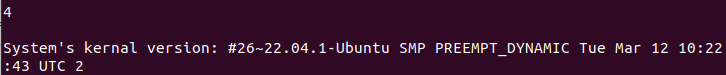


The next part of the code (line 328-337) is only activated if the user enters 8-10 on their first input. If this is the case, the if statement is entered, as the user must pick the slot or the array they wish to deal with. If this is the case, the program first asks the user which slot they would live to access and tells them that there are 5 slots (line 329). We then enter an if statement just like the one for the first input that makes sure a number between 1 and 5 is entered. (line 331). If this is not the case, then it does the same thing as the first while statement for input (gives an error, clears the input, sets choice to 0). Once this while statement is passed it subtracts one from arr after it is entered (line 336). This is to make sure the correct array slot is accessed, as array counting starts from 0 instead of 1.







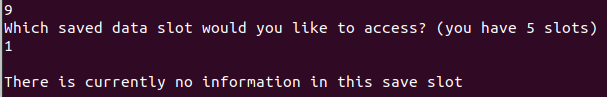
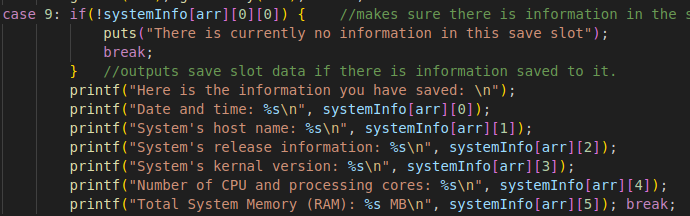


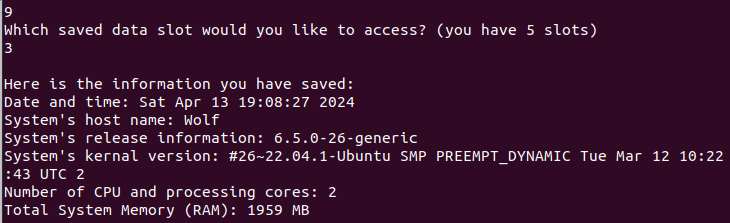


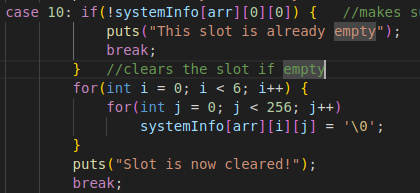
We then reach a switch/case statement (line 341-374). This statement goes off of the value choice is currently set to. The first 7 case statements are all just singular function calls. 1-6 is for the system information, which calls the respective function for the information that needs to be displayed. All of these function get sent -1 as their parameter, as it tells the function to just output the data and not save it to an array slot. If 7 is entered, then the bash function gets called, which enters the bash function, that can change the bash script or run it.

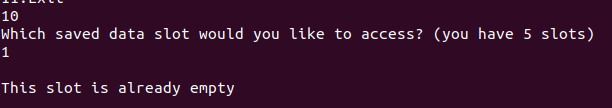


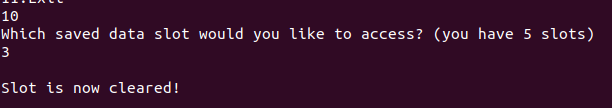
The next case statement is for if 8 is entered as the users choice (line 349-251). This cases statement is what sets the array to system information. It calls each of the system functions used previously, but this time it sends over the arr value, which holds the slot the user wishes to put the information into. The case statement then breaks.



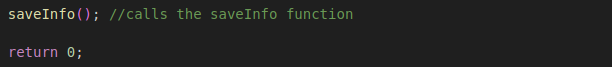
The next part of the case statement activates if the user enters 9 (line 352-362). This works as an output for the slots in the array. The user picks which slot they would like to output. If there is no information in this slot, then the program says that to the user (line 352-355). If there is information, then the program outputs all information from that slot to the screen (line 356-362).

****

****

****

The last case statement is for when the user enters 10 (line 363-372). This section of code is used for if the user wants to clear a slot out. It first checks to see if the slot is already clear, giving a message to the user and exiting the case if it is already clear (line 363-366). If not, then it goes through every slot in the array, making it an empty character (line 367-370), before telling the user the slot is now clear (line 371).

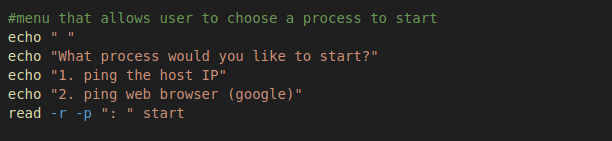
****

If 11 was entered, we then exit the for loop. We then reach the end of the code, which first calls the saveInfo function (line 380), which will save the information from the systemInfo array to a text file. We then end the program using the return 0 command.

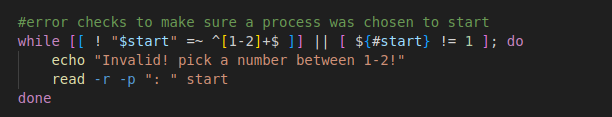
**Bash Script:**

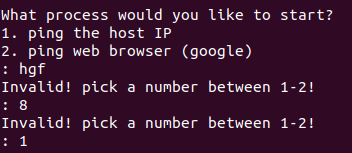
A bash script has been implemented into this program. This bash script starts out with three functions.

**- Add Process Functions**

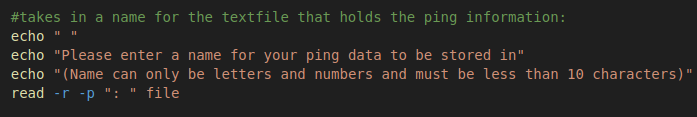
****

The first thing this function does is ask the user which process they would like to start and takes in their input (line29-33). It asks the user if they want to ping the host IP or a web browser (specifically google). It then reads the users input, with formating that prevents backslash and prompts the user

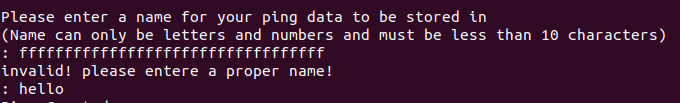
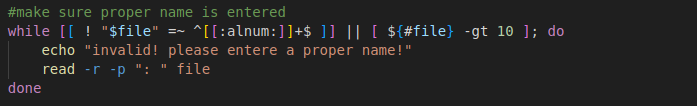


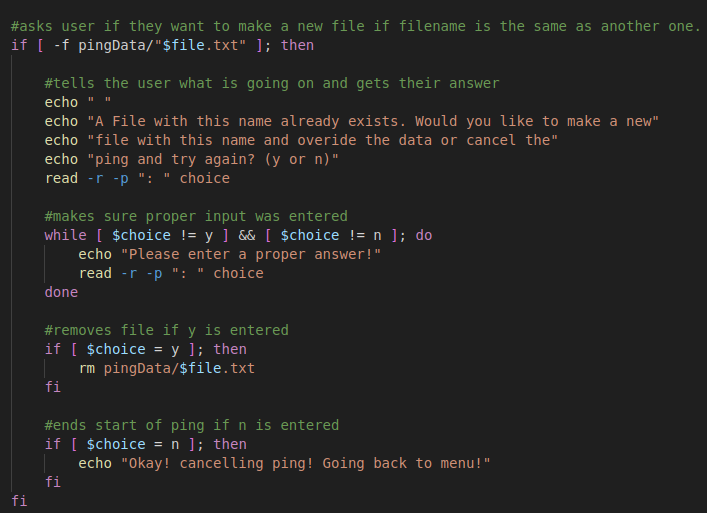


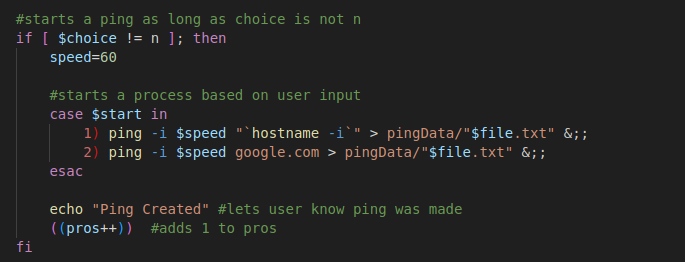
There is then a simple while loop (line 36-39) that makes sure that start (the variable the user entered into) is set to a length of one and only numbers between 1 and 2. If this is not the case, the while loop is entered, and it will ask the user for a proper input until one is input.



The program then asks the user for a name for their ping data to be stored in (line 42-45). This is because the ping data is stored into a text file for the user to access. It clarifies that only letters and numbers can be entered and only a name up to 10 characters can be entered. It takes in input the same way as above.

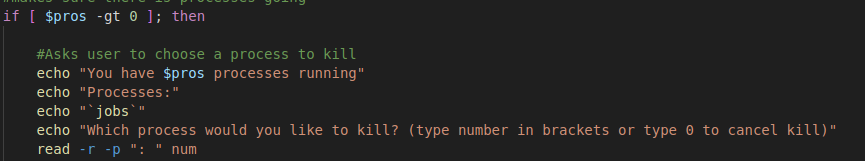
We then reach another while statement (line 48-51) that makes sure only number and letters were entered and that the length of the input is less than 10. If this is not the case, it enters the loop, which will keep going until a proper input is made.

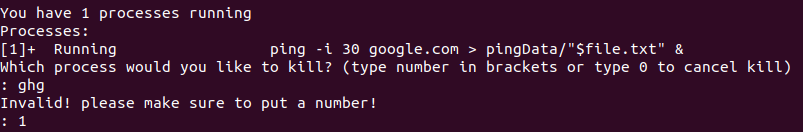
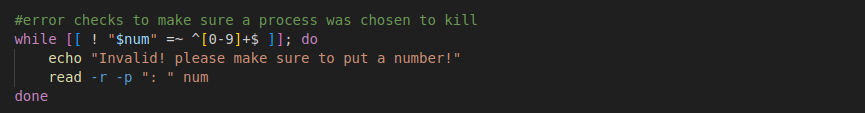
We then hit a lengthy if statement (line 54-78) that checks to see if a file by the name the user entered already exists. If it has, it asks the user if they would like to replace that file and its data or cancel the ping and try again (line 57-61). It takes in the users input, and then we hit a while loop again (line 64-67). This while loop simply makes sure that a proper input was entered (either y or n). If not, the loop keeps going until a proper input was made. We then hit two if statements. The first one checks to see If y was entered (line 70-72). If it was, then the current file with the same name is removed. We then hit an if statement that checks to see if n was entered (line 75-77). If this is the case, the program simply gives a message that lets the user know the ping was canceled.

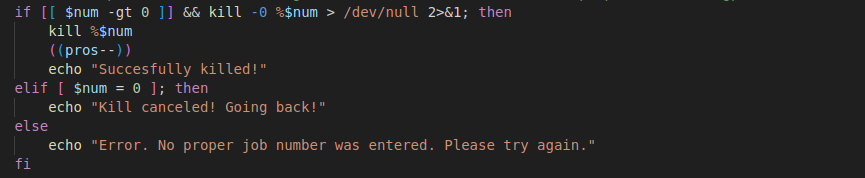


We then reach the end of this function, which is one last if statement (line 81-92). This if statements checks to see if n was entered, as if it was, the ping was canceled, and the if statement will not be entered. If this is not the case, then the if statement is entered, where we first hit a new variable called speed (line 82). This is the variable changed by the bash script, and it is used by the pings for how often a ping will be sent out. a case statement (line 85-88) is then entered. This case statement goes off of what the user entered for their first choice, and starts a ping based off that input, directing it into a text file. It then tells the user that a ping was created, and adds 1 to pros (processes) (line 90-91). The last thing the function does is set choice to y (line 94) and echo out a blank line (line 95).

**- Remove Process Function**

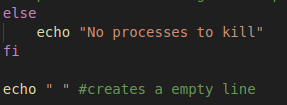
The first thing this function does is check to see if pros is greater than 0 (line 102). This is to make sure there are actually processes going that can be removed. If this is the case, then the if statement is entered. First thing that happens is that the program asks the user which process it would like to stop.(line 105-109) It does this by telling them how many processes are going, showing what processes are going and their job number, and telling them what they need to enter. It then takes in the user input much like the add process function.

We then reach a while loop that makes sure a proper input was made (line 112-115). It makes sure a number was entered into the input, and if it was not, the while loop is entered and goes until a proper input is made.

****

****

We then reach another if statement (line 118-126). This statement first checks to see if the user input was greater than 0 and if the number entered actually corresponds to a process. If this is the case, then it enters the if statement, killing the process (line 119), subtracting one from pros (line 120), and telling the user the kill was succesful. There is then an if else statement, which checks to see if 0 was entered. If this is the case, it tells the user the kill was canceled. If not, then we reach an else statement, which gives an error, telling the user there number entered did not correspond with a job to be killed.

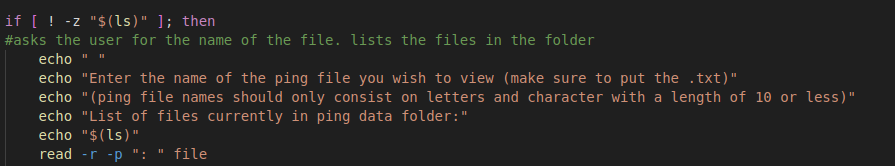
****

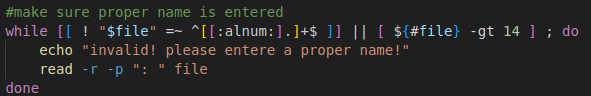
There is an else statement that is entered if pros was at 0 before the program started. If it was at 0, then it tells the user there are no processes to kill (line 130). The function then outputs a blank line before ending.

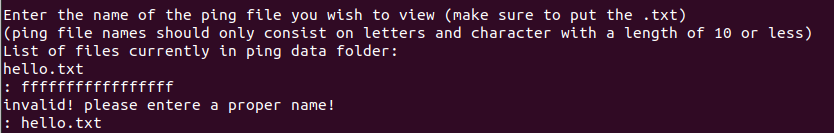
**- View File Function**

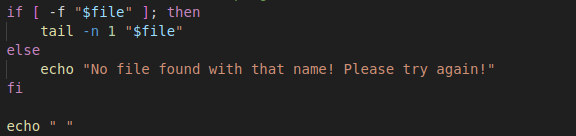
****

The first thing this function does is cd into the pingData file (line 140). It does this so it can view and read from the files in this folder, which are the ping text files.

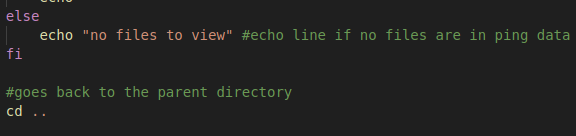
The function then has an if statement that checks to see if there are any files in pingData (line 144-149). This is to make sure there are text files to view. If this is the case, it then asks the user for the name of the file they wish to view. It tells them the requirements for input and shows them the files in pingData before getting the input.



There is then a while loop that checks the users input (line 152-155). It makes sure the user entered only numbers/letters and makes sure they didn’t enter more than 14 character (10 characters + .txt). If this is not the case, the while loop is entered, and it keeps getting input from the user until a proper input is made.

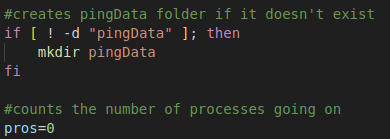
****

We then reach another if statement (line 158-162). This if statement checks to see if a file by the name that the user enters exists. If it does, then it gives the last line entered into the file to the user. If not, then it says that no file was found to the user. There is then a output line.

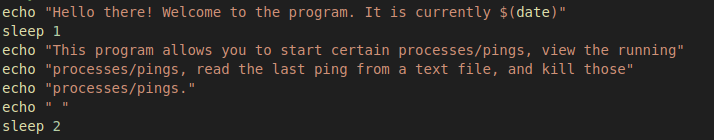


The last part of this function is an else that activates is no files are found in the folder. If this is the case, a line is output telling the user there is no files to view (line 166). The last thing the function does is cd back into the original directory (line 170).

**- Main Code**

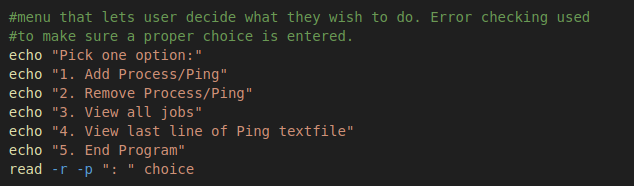


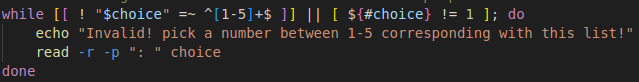
The first thing the program does when it starts is make sure that a ping folder exists (line 174-176). If the pingData folder does not exists, it makes one. The pros variable is also created and set to 0 (line 179).

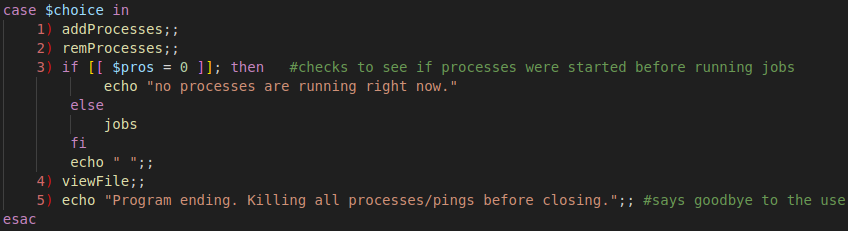
The program then outputs a information to the user (line 182-188). It says hello and tells the user what the program does.

****

We then enter a while loop that takes up most of the rest of the program (line 191-225).

The first thing the while statement does is output a menu to the user about what they can do. (line 195-201) This menu repeats each time the loop goes through. It also takes in the users input to the choice variable.

We then reach another while loop that checks the user input (line 204-207). Much like before, this checks to make sure the users input is between 1 and 5 and and has a length of one. If the while loop is entered, then it repeats and gives an error message until the user enters a proper input.





We then reach a case statement that checks the choice input (line 210-221). This case statement has 5 options: 1 (line 211) calls the addProcesses function, and 2 (line 212) calls the remProcesses function. If choice is 3 (line 213-218), it first checks to see if pros is equal to 0. If this is the case, it tells the user there are no processes running. If this is not the case, then it outputs the jobs command, along with a blank line. If 4 is entered (line 219), the viewFile function is called, and if 5 is entered, it gives a line about the program ending.

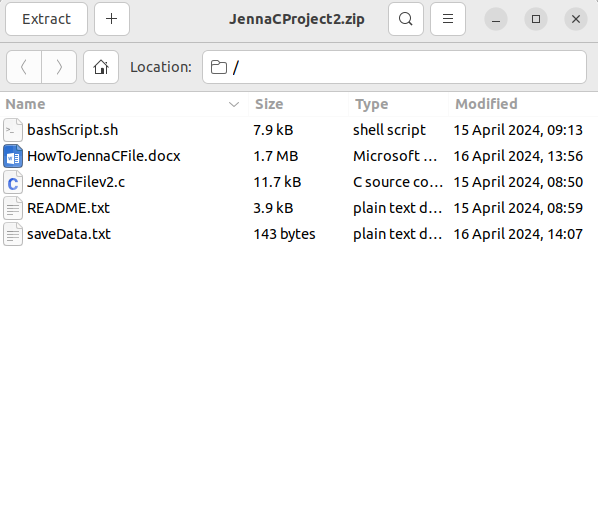


****

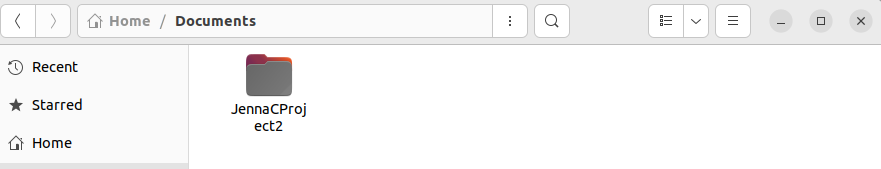
If 5 is entered, we exit the while loop. We then reach the last two lines in the code. The program first kill all processes still happening (line 228), and then says goodbye to the user (line 230). The bash script then ends.

**How to run:**

1. Download the zip file. The zip file should look like the following and have five or six files inside: a C file, a read me, a how to, a bash script, and a saveData text file. You might also have a pingData folder (if not, that is fine).

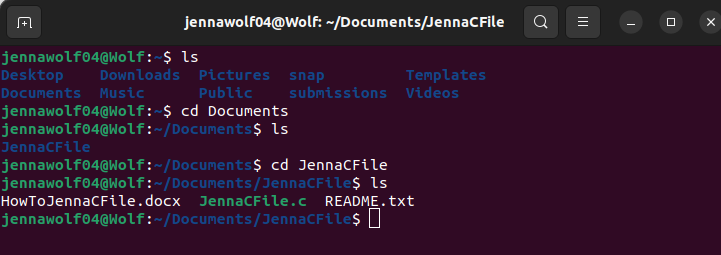


2. Unzip the file into the directory/fouler you wish to run the program from. Please note where you put this file, as you will need it to run the program. Documents is suggested, but not required. You can then delete the zip file after this, as you should have a folder with the name of the zip file (before deleting, enter your new folder to make sure all files are there).

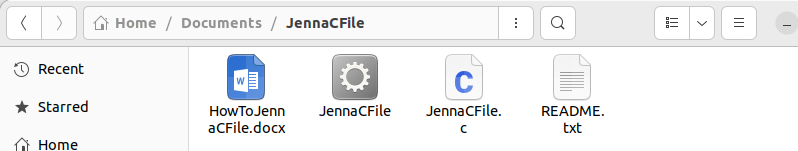


3. open up your terminal.

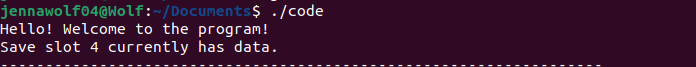
4. use the cd and ls commands to enter the directory/folder with the program. Cd with a folder/directory name will place you in the folder/directory you pick. Ls tells you where you can go from your current directory. When you are in the correct directory, you should be able to see the runnable c program JennaCFilev2.c . It should look like the picture below.



5. You then need to mod your program to work. Simply type in the command “ gcc JennaCFilev2.c -o code “ to do so. When you hit enter on this command, you should not get an error on screen, and if you check your folder, you will see a new file in there. This is the runnable version of the program.



6. Finally, type in “ ./code “ to run the program, or whatever name you chose for running your code. From here, you can follow the prompts on screen to use the program. Keep in mind that only number inputs will work with this program.



7. If you are running the program and the bash script is not working/is saying permission denied, you need to end the program and type in “chmod +x bashScript.sh”. This will add your permission to the bash script so you can run it!