ECEN 5813 (F'18)

Project - 1

Project Report

Project Details	*Description*
Group Members	**Rushi James Macwan and Poorn Mehta**
Date	**7th Oct 2018**
Project	**PES Project-1 (Linux based Embedded System UI Development)**
Class Name	**Principles of Embedded Software ECEN 5813 (Fall-2018)**
GITHUB Repository Link	*https://github.com/MacRush7/Project_1_R-P*

[PROJECT REPORT]

University of Colorado Boulder

Table of Contents

Project Plan:

As per the file submitted online, the below given image shows our project plan that was submitted online earlier. This project plan shows the planning for difference cases: best, likely, worst and average.

	A	B	C	D	E	F	G	H	I
1	**Milestone**	**Task**	**Best**	**Likely**	**Worst**	**Avg**	**Date**	**Total Number of Hours**	**Hours per Week for PES**
2		Develop and document project plan	1	2	2			70.05	15
3	**Submit project plan**						2018-09-19		
4		Build environment setup	0.25	0.25	1	0.4			
5		Git repository	0.25	0.5	1	0.55			
6		Initial makefile, main() and framework	1	6	16	7			
7		Basic command line prompt and parse	1	4	10	4.6			
8		Help function	0.5	2	6	2.5			
9	**Command prompt with help**		1	3	7	3.4			
10		Allocate	1	3	7	3.4			
11		Free	1	3	7	3.4			
12		Display	1	3	7	3.4			
13		Write	1	3	7	3.4			
14		Invert	1	3	7	3.4			
15		Pattern write	2	6	14	6.8			
16		Pattern verify	1	5	10	5.2			
17	**Testing file generation and linking**		2	7	12	7			
18		Execution time measurement	1	4	10	4.6			
19	**Feature complete**		2	6	12	6.4			
20		Final report	1	4	10	4.6			
21	**Submit final report**						2018-10-21		
22									

The below attached screenshot shows the real values for the time that we took in reality to complete this project.

F12

	A	B	C	D	E	F
1	**Milestone**	**Task**	**Real-time taken**	**Date**	**Total Number of Hours**	**Weeks available**
2		Develop and document project plan	1		37.25	2
3	**Submit project plan**			2018-09-19	**Hours per Week for PES**	
4		Build environment setup	0.5		18.625	
5		Git repository	1			
6		Initial makefile, main() and framework	3			
7		Basic command line prompt and parse	5			
8		Help function	1.5			
9	**Command prompt with help**		2			
10		Allocate	1			
11		Free	0.25			
12		Display	0.25			
13		Write	0.25			
14		Invert	0.5			
15		Pattern write	4			
16		Pattern verify	1			
17	**Testing file generation and linking**		3			
18		Execution time measurement	4			
19	**Feature complete**		5			
20		Final report	4			
21	**Submit final report**			2018-10-03		
22						

Report Questions:

1. How well did your effort estimates match with actual effort? Provide examples of both inaccurate and accurate estimates and discuss any contributing factors to the success or failings of your effort estimates.

   According to our effort estimates, we found that we required pretty much more time than expected in some cases where we had to integrate the whole project work into one refined framework. On the other hand, it took considerably less time in developing individual modules for the project which involved developing individual functions required for the user to allocate memory, free the memory, read/write memory, etc.

   On an overall basis, the inaccurate estimates were truly made in the area of testing and debugging. We could not find sufficient time to test our work in the first place although we could test our work individually (i.e. checking execution of individual functionalities with positive results). While integrating the whole project into one framework, it demanded a lot more time than expected earlier which was due to the underlying fact that we required to modify the internal merge conflicts as well as we had to prioritize individual modules while they worked together. For example, the memory allocation function ("memalloc" in our case) was prioritized and was given the highest priority. The reason behind this was that every other function has been designed to deal with memory (either in the form of storing valuable data or in the form performing operations on the valuable data stored in the memory). So, as long as 'sufficient' memory space has not been allocated, the user will be prompted to reconsider allocating the memory first before performing any other operations.

   The accurate estimates were made in areas where the basic functionalities were developed. We could be on the ideal timeline associated with the "best case" where we completed the functionality development in approximately 15 hours of work with all the background checks and error-alleviation functionalities.

   There were some factors that contribute to both our success/failings and they come from a wide variety of experience and interests within the team. Moreover, the varying workload from other subjects gave us the liberty to spend more time developing a robust, user-friendly and error-free system.

2. Were there any tasks that you forgot to include in your project plan?

   According to our recollection of the project work, we have included all the assigned features and functionalities to the best of our abilities. However, we did modify some of the internal aspects of these functionalities after the primary deadline owing to some conflicts of our existing codes back then with the Linux system. So, we had to work again on some memory allocation details to come over the "segmentation fault" that we came across after mounting the integrated project on Git.

3. What is the largest block of memory you can allocate? Discuss.

   As per the discussions made in the class, for a 64-bit OS, the user can exploit upto 2 GB of memory space without violating any other system functionalities. As such, our code allows the user to best utilize the memory space by requesting the system to allocate to the user a specific number of 32-bit word memory space. Therefore, the user is free to request for allocation of as many 32-bit word memory space as required given that the 2 GB boundary is not reached.

4. What happens if you try to read outside of the allocated block?

   Based on the code that we have developed, if the user tries to read outside of the allocated block then the system will prompt the user to read data from the allocated block. The user is warned to read from within the allocated memory block. However, as per the project assignment, the user is given the liberty to disregard the warning and to proceed further. If the user disregards the warning and proceeds further reading from an un-allocated memory block then the system will read the data for the user which can be any value between 0 and the permissible maximum value based on a 64-bit OS storage capacity.

5. What happens if you try to write outside of the allocated block?

   If the user tries to write outside the allocated block of memory space then the system will prompt the user to perform the writing operation within the allocated block since the memory outside the allocated block may be a read-only memory space. Therefore, to avoid the circumstances where the system crashes simply because the user is performing operations on a read-only memory or a system-access-only memory, the user is bound to utilize only the allocated block. This saves the user from losing important data that is not stored anyway.

   However, based on the project assignment guidelines, the user is given the liberty to disregard the system warning to write within the allocated block of memory. In this case, the system will write data to the user-specified memory location which may or may not be stored based on the machine-dependent memory configurations. Furthermore, the user can still verify if the data is indeed stored at a specified location outside the allocated memory. To verify this, the user is again required to read the data stored at the specified memory location outside the allocated region by disregarding the system warning.

6. Analyze the time it takes to invert memory for different memory block sizes. Is there a linear relationship between time and size?

   Yes, there is a linear relationship between the time and size for the time that the UI takes to invert memory for different memory block sizes. We tried modifying our UI to take different memory block sizes for the process of inverting the data and as a result we found that there is a linear relationship between time and size. Moreover, there is a rise in the time taken to invert the memory with the increase in size of the memory block and the rise in the time taken to invert the memory is approximately constant. The relation would be therefore look like this:

Total time taken = (Size of memory block)*(Time for inverting memory) + Time constant

Here, the "Time for inverting memory" and the "Time constant" are both constants which form the slope and the intercept for the equation of the type: y = mx + c where

y = Total time taken
m = Time for inverting memory
x = size of memory block
c = Time constant

7. What are the limitations of your time measurements?

The limitations of our time measurement are as follows:

- The time measuring feature involved in this UI is itself a function that is called every time by the system whenever time measurements are to be made.
- Since the time measurement is itself a function, it consumes some time to call the function and its corresponding library (.h file).
- There is a point at which the system is unable to provide the user-demanded accuracy in measuring the time-taken by the system to execute a particular function. This timing accuracy is more or less dependent on the type and capacity of the embedded system that uses the UI.
- The time measurement feature involves starting and ending the internal clock to calculate the time taken for any operation performed between its starting and ending intervals. This clock excitation is also matched with the system real-time clock to provide the time taken by calculating the difference between the starting and ending intervals. Therefore, the system takes some time to start and end the clock and it also requires time to calculate the difference to print it on the screen. These internal operations also eat up some time which contributes to the limitations in time measurement for the system.

8. What improvements you can make to the invert function, your program or your build, to make it run faster? How much improvement did you achieve?

We can make the following improvements to achieve a faster inverting function:

- Since, all the allocated memory space has been pre-cleared (written a 0 value to it), we can call the inverting function to simply display a 0x f f f f f f f f value whenever the user asks to invert a specified memory location. As long as the user does not ask to store the inverted value, we can sufficiently reduce our run-time and make the process faster.
- On the contrary, the system can even store the inverted values of all the data that is stored within the allocated memory space. Whenever the user seeks the inverted value, the system only needs to print the values from the database. In turn, this will apparently reduce the time for the invert function while the system calculates the inverted values and stores them when the user is performing a different task at hand.

9. What algorithm did you choose for your pattern generator? How does it generate its pattern? What are its strengths and weaknesses?

   The algorithm that we created for pattern generation can be found by clicking here.

   More information about this algorithm:

   This pattern generator basically takes a seed value and divides the seed value by 10 as long as the end value is in the form of a fraction. Furthermore, this fraction of the seed value is multiplied with a different fraction value and the result is added again to a fraction and is stored in the form of a random number. Once again, it is ensured that this end result is a fraction to avoid an output from the generator that violates the boundary for the value of the allowed pattern numbers. The random number is again converted to a fraction and then is multiplied with the maximum limit that the user specifies for the value of any pattern number. The end output is always within the allowed maximum limits for the pseudo-random number and is a finely refined integer that varies finely between 0 and the maximum allowed limit.

   Strengths:

   - This algorithm generates a refined pseudo-random pattern that has its individual values spread widely across the space between 0 and the user-specified maximum allowable pseudo-random number value.
   - The proposed algorithm can work on a seed value that has no limits. The user can use any desirable seed value irrespective of the output. The seed value can even be a floating point number and does not create any issue with the output of the system.
   - The algorithm binds the input and the output in a very strict fashion such that the output never exceeds the permissible limit even though the seed value can take any form.
   - The constant inclusion of the generated random number into the seed value after every iteration delivers a finer output that is different from the previous value since one pseudo-random number will always be different from other if the seed values are different and since the seed is always replaced by the previously generated random number, the system will never provide two equal values that are placed consecutively. This opens the room for a different pattern generation every time when the loop is executed.

   Weaknesses:

   - With seed values like 1, 10, 100, etc. (in multiples of 10), the pseudo-random number pattern will remain the same. This is because the seed value is initially converted down into a fraction and then is processed with other fractions to get a value which is again converted to a fraction which is multiplied with the maximum permissible value of the pseudo-random numbers as specified by the user to get a value within the boundary. Eventually, this sort of a behaviour often generates

similar patterns for one or more numbers when the only difference between those numbers is multiples of 10.

- Based on the previous point, there will be some seed values for which not all the values in the pseudo-random pattern will appear as unique.
- With this algorithm, the precision of the seed value does not matter at all when the maximum allowable pseudo-random value is low. For example, if the maximum allowable value is 10 then the pattern will have so many numbers in common for so many values. This reduces the refined structure of the pattern since the pattern only contains integers and the difference between 0 and the maximum allowable limit for the pseudo-random numbers decides the clarity of the pseudo-random numbers.

Test case input file:

The test case input file is as entered below on this page:

```
y

help

help memalloc

memalloc 15

y

help memwrite

memwrite 3 3adf1bc9

y

help memread

memread 3

n

help meminv

meminv 3

y

help patterngen

patterngen 5 10 345 1000

y
```

help patternverify

patternverify 5 7 345 1000

y

patternverify 5 8 346 1000

y

memfree

help exit

exit

This file can also be accessed on our Github repository by clicking here.

Syntax Documentation / Appendix Section:

The UI begins by allowing the user to decide between the use of relative addressing and absolute addressing throughout the program. The terms can be understood through the information given as under:

- Relative addressing: If every allocated 32-bit memory block within the allocated memory space is ranged with a value starting from 0 then n such 32-bit memory blocks will provide a relative addressing starting from 0 to n-1.
- Absolute addressing: Ideally, every allocated memory location is uniquely represented by a hex address that is singularly assigned to that particular 32-bit memory block within the allocated memory space.

The user is thus provided the liberty to address every allocated memory address either through relative address in the form of integer values ranging from 0 to n-1 for n allocated 32-bit locations or through absolute addressing by addressing every allocated 32-bit location by its unique hex address assigned to it by the system.

Sr No.	Function Name/Syntax	Description
1.	***Help*** *Syntax:* *help* *help <command>*	The help function is the most basic command of the entire UI. This enables the user to learn more about the UI and the functions that are available to the user to exploit for a certain set of inputs. The help function also throws light upon the syntaxes of the individual functions if the user enters help <command> without the <> symbols where the command can be any function name as given below. For example: To use this function, simply enter: help <memalloc> will explain more on the memory allocation function. The system will provide the following output for executing the given command: `Enter Command: help memalloc` `Memory Allocation:` `Type memalloc and then enter the` `number of memory locations that you want` `to use and have access to.` `Alternatively, type memalloc <number> without <> for value`
2.	***Exit*** *Syntax:* *exit*	The exit command will simply let the user to exit/close the UI.
3.	***Memory allocation***	The memory allocation function allocates the user a user-specified number of 32-bit memory locations from the memory space available

	Syntax: *memalloc <number of 32-bit allocations demanded>*	to the system of the UI. In turn, the UI will also display to the user the allocated locations with their relative addresses, absolute addresses and the existing data in hex at the time of allocation. To use this function, simply enter: memalloc <number of 32-bit allocations demanded> and do not enter the <> symbols.
4.	***Memory Freeing*** *Syntax:* *memfree*	The memory free function frees all the previously user allocated memory space. To use this function, simply enter: memfree.
5.	***Memory write*** *Syntax:* *memwrite <relative/absolute address of the user-specified allocated memory location> <data in hex to be written>*	The memory write function writes the user specified 32-bit data to a user-specified 32-bit memory location previously allocated to the user by the UI from the memory space available to the system of the UI. In turn, the UI will also display to the user the modified data at the allocated memory locations with their relative addresses, absolute addresses and the newly modified data in hex after executing the memwrite instruction. To use this function, simply enter: memwrite <relative/absolute address of the user-specified allocated memory location> <data in hex to be written> and do not enter the <> symbols.
6.	***Memory read*** *Syntax:* *memread <relative/absolute address of the 32-bit location for which the data has to be read>*	The memory read function reads the data from a user specified 32-bit address. In turn, the UI will also display to the user the stored data at the specified allocated memory location with their relative addresses, absolute addresses and the newly modified data in hex after executing the memread instruction. To use this function, simply enter: memread < relative/absolute address of the 32-bit location for which the data has to be read> and do not enter the <> symbols.
7.	***Memory Inversion*** *Syntax:* *meminv <relative/absolute address of the 32-bit location for which the data has to be inverted>*	The memory inversion function inverts the data from a user specified 32-bit address and stores the newly inverted value to that memory location. In turn, the UI will also display to the user the newly inverted and stored data at the specified allocated memory location with their relative addresses, absolute addresses and the newly modified data in hex after executing the meminv instruction. To use this function, simply enter: meminv < relative/absolute address of the 32-bit location for which the data has to be inverted> and do not enter the <> symbols.
8.	***Pseudo-random pattern generation*** *Syntax:* *patterngen <starting relative/absolute*	The pseudo-random pattern generator function generates a series of pseudo-random numbers based on a user-specified seed value. The user decides the max value for every generated number and also specifies the starting address from where the user-specified number (n) of generated numbers need to be stored. In turn, the UI will also display to the user the newly generated pattern and the stored pattern

	address for storing the generated pattern> <number of 32-bit pseudo-random numbers to be generated> <seed value in decimal> <max permissible value for every generated number>	values at the specified allocated memory locations with their relative addresses, absolute addresses and the newly stored data in hex after executing the patterngen instruction. To use this function, simply enter: patterngen <starting relative/absolute address for storing the generated pattern> <number of 32-bit pseudo-random numbers to be generated> <seed value in decimal> <max permissible value for every generated number> and do not enter the <> symbols.
9.	***Pseudo-random pattern verification*** *<u>Syntax:</u>* *patternverify <starting relative/absolute address for testing the generated pattern> <number of 32-bit pseudo-random numbers to be tested> <original seed value entered in decimal> <original max permissible value entered for every generated number>*	The pseudo-random pattern verification function tests a series of pseudo-random numbers previously generated and stored in the allocated memory based on a user-specified original seed value and a user-specified original max value for every user-specified memory location by taking the test start address and the number of 32-bit locations to be tested thereafter for the pattern. In turn, the UI will check the results and will provide a 'test success' output if the pattern matches entirely as per the input constraints. To use this function, simply enter: patternverify <starting relative/absolute address for testing the generated pattern> <number of 32-bit pseudo-random numbers to be tested> <original seed value entered in decimal> <original max permissible value entered for every generated number> and do not enter the <> symbols.
10.	***Memory clear*** *<u>Syntax:</u>* *memclear*	This particular function when called allows the user to clear the data and replace it with 0s at all the locations within the allocated memory space. To use this function, simply enter: memclear.
11.	***(Internal Function) Time calculation***	The internal timing function is not available to the user to execute but the system itself provides time taken for executing any of the functions mentioned from points 3 to 9.
12.	***(Internal Function) Boundary check function***	The UI has an incorporated boundary check functionality that tests every input if it falls within the allowed boundaries. If the user enters a value otherwise that fails to meet the boundary limits then the system warns the user before continuing ahead with the execution. However, in some case, the user is provided the liberty to overrule the warning and disregard its comments in which case the user can proceed towards the end of the function execution.

		Example: If the user tries to read data from a memory location that is outside the allocated memory space to UI then the UI will warn the user before trying to read the data stored at the specified memory location. The boundary check function takes care to provide important warnings as and when required. Furthermore, if the user ignores the warning, the function will provide the data at the given memory location if the UI is allowed by the OS to access the user-specified memory location for read-only purposes. In case of writing to locations beyond allocated memory space, the system warns the user and if the user chooses to overrule the warning then the system tries to write to the specified memory location if it is not a read-only memory location given that the OS allows the UI to access that memory location in the first place.
13.	***(Internal Function) Error-checks***	The UI has an incorporated error-checking functionality that tests the user's inputs if they fall within the system specified format. If the tests fail then the user function is terminated and the user is taken back to main terminal (help terminal). This repeats as long as the user does not provide an acceptable input. Example: If the user provides a false hex data to the system which is 4f2bs then the system provides an error that the entered hex data is invalid since hex numbers do not include s in their format. It only includes 0-9 digits and a-f or A-F alphabets.
14.	***Null-pointer check & pointer clearing***	The UI has an extended capability for null-pointer checks and it also ensures that the previously stored/generated system garbage data at the new allocated memory locations is cleared so that when the user is allocated memory space, it mostly contains a null value to ensure that it is easy for the user to use those memory spaces for different operations without clearing the data at the allocated space every time when an allocation is performed.