McGill University
School of Computer Science

Introduction to Software Systems

Midterm Exam ANSWERS & GRADING SCHEME

February 23, 2016 – In Class
Instructor: Joseph Vybihal

Student Name: ____________________________

Student ID: ______________________________

Instructions

- No notebooks or textbooks permitted in this exam.
- Language translation dictionaries are permitted.
- You are permitted to write your answers in either English or French.
- Attempt all questions.
- **Parts marks are given for all questions - show your work**.
- All answers must be written on the exam sheet
- Everything is proportionally graded (i.e. if 50% correct then you get 50% of the grade)

Grading

Section	Grade	Your Mark
Section #1 – Short answer questions	30 %	
Section #2 – Bash Programming	35 %	
Section #3 – C Programming	35 %	
	--------	-------------
Total	100 %	

PLAN 25 MINUTES PER QUESTION

YOU CAN WRITE YOUR ANSWERS ON BOTH SIDES OF THE SHEET

RETURN THE EXAMINATION SHEET TO THE INVIGILATOR

Section 1: Short Answer Questions [10 points, 3.3 for each example/argument]

Question 1: Provide three arguments or three examples that show why a student writing a C function that computes the factorial of a positive integer number using recursion should be considered a software system.

This question refers to the factorial function.

Three examples: Compiler converts C to machine language. Run-time stack implements recursion. Integer is an abstract concept implemented by the hardware.

The arguments must be related to compiling into machine language, the implementation of a run-time environment by the OS, and how hardware emulates theoretical values.

Question 2: The UNIX operating system is constructed out of four components: the Kernel, the File System, the Shell, and the Utilities. Below are UNIX subcomponents. Beside each subcomponent is a blank line. Write, in the blank line, the name of the component it belongs with. **[10 points, 1 each]**

Subcomponent	Component	Subcomponent	Component
Command line	shell	cp	utilities
login prompt	kernel	login script	shell
boot script	kernel	session	shell
USB Stick	file system	gcc	utilities
Executing program	kernel	Printer queue	file system

Question 3: (A) Name the components of a Packet, as described in class. (B) Give a single argument as to why a packet needs to be part of a software system. (C) Bob wants to send a secret message to Alice. Alice is connected by wire to Tom and Tom is connected by wire to Bob. Describe how a secret message travels from Bob to Alice. **[10 points]**

(A) From address, to address, message size, message, error correcting codes [3 points, it must have what I listed here as minimum and it must not have wrong elements. Score proportionally]

(B) A packet is a data structure, it needs algorithms and a network to make it useful [3 points, the answer must be similar, not exact, to mine]

(C) Bob must first encrypt his message. Then the encrypted message is placed in a packet. Bob's address is put in the FROM field, Alice's address is placed in the TO field. The packet is sent to Tom's computer. Tom's computer looks as the TO address and realizes it is not for Tom and so passes the packet on to Alice's computer. Alice's computer looks at the TO address and sees that it is for Alice. The message is sent to Alice. Alice decrypts the message. [4 points, the student must provide all the steps I include here, grade proportionally]

Section 2: Bash Programming [35 points]

Write the following Bash script:

Assume the system operator of a server from time to time needs to kill multiple running programs. This system operator wants to write a Bash script with the following signature: ka NAMES

"ka" stands for Kill All the given executing programs.
NAMES is a list of words separated by spaces. Each word is the name of an executing program.

There is no limit to the amount of NAMES provided as arguments. The script returns an error if no NAMES are provided. The script attempts to kill the programs in a twostep process: first it attempts to kill the program using the safe kill operation. It then verifies if the kill was successful. If it was successful it then goes on to the next program to terminate. If it was not successful then it attempts with a forced kill. It does not matter if the forced kill works or not, the script will go on to the next program to terminate. When it exhausts all the programs it ends by displaying all the running processes (for current user, simple output), and then stops.

I will not provide a complete solution here because students will have written there answers in many ways, but their answer must have the following things to receive full points:

- +2 Must have a valid shebang for Bash
- +5 Echo an error if no names, if ["$#" = "0"]
- +5 $@ to save all the names … or as array x=($@)
- +5 Set \`ps | grep $aName\` will set positional variables with output to get PID – at $1
- +4 Kill PID to do soft kill
- +5 X = 'ps | grep $aName\` if X is empty then the kill worked
- +4 Kill -9 PID to do hard kill
- +5 Loop and update variables, shift and set $@ … or index through array ${x[index]}
- -5 bad bash syntax
- Grade proportionally within each point range (eg. Shebang is worth 2, grade proportionally 0-2)

Section 3: C Programming [35 points]

A global integer array of size 100 named buffer. A function with the signature `void add(int n)`. A main function that receives a positive integer number m greater than zero and less than 100 from argv. The main program validates that m agrees with the range and is a valid integer number. If m is not valid the program terminates with an error message. Otherwise, the main method proceeds to increment from 1 to m in a loop. During each iteration the programs asks the user for an integer number n. The program assumes the user inputs valid integer numbers for n without needing to validate. This number is passed as an argument to the function `void add(int n)`, which inserts the number into the buffer array in sorted order (lowest to highest). Make sure to write a properly formatted program. You don't really need library function, other than I/O functions, but if you want to use some then only use library functions we covered during class for C.

- +2 Declaration of global array
- +2 int main (int argc, char * argv[])
- +2 Checking correct number of arguments passed from command line
- +10 Convert argv[1] to int and assign to m
- +3 Check if m is between 1 and 99 inclusive
- +3 Loop from 1 to m
- +3 Ask for integer number n
- +3 Call add function with n
  - +5 The value n is insert sorted into the array immediately
  - +2 Returns to main for the next value to insert sort
- -5 incorrect C syntax
- -5 program not formatted properly
- Grade proportionally within the point range