Sample Questions for EE5012 “SDip in Embedded Systems”

List to date: 12-Mar-19

This set of questions defines the range of material that needs to be reviewed and understood. In an actual exam equivalent questions can be asked that might be structured differently in size and content. This list of questions is intended to indicate the scope of revision that is expected, rather than being the list of actual exam questions.

INTRODUCTION

1) Give a definition for a computer operating system.

2) What are the two main functions of an operating system?

3) What does the term **POSIX** mean is respect to the **UNIX/Linux** operating system and what is its significance?

4) Draw a simple block diagram of a **Linux** operating system and name each block.

5) Briefly state what is meant by a **UNIX shell** and name three well-known UNIX shells.

6) List four well-known **Linux** distributions

7) Provide name, type and short description for the Operating systems running in/on:

- TV sets, Cars, Microwave ovens
- Smart phones, Tablets
- Corporate data centers
- Industrial robots

8) Name the operating system for the following Apple products: iPad; MacBook Pro; iPad; Apple Watch.

9) Draw a diagram to represent Operating system services in a computer system

10) Provide a short definition for the following terms:

- Program
- Process
- Task
- Processor
- Job

11) Provide a quick explanation of the computer system boot sequence.

PROCESSES

1) Draw a diagram to show the **memory sections** occupied by a process. Provide name and short description of each **section**.

2) Draw a typical **PCB** (process control block) for a process in a multitasking environment and briefly state the purpose for each field in your **PCB**.

3) With the aid of a **state diagram** show the various **states** for a process in a multitasking system. Label clearly all **state transitions**.

4) Briefly explain what is a **context switch** and its significance from CPU point of view.

5) List 5 possible actions that can be performed on a process.

6) Assume a system, at a given instant, has one **running** process, three **ready** processes, two **blocked** processes on *blocked queue_x* and one **blocked** process on *blocked queue_z*. Draw the queuing structure at this instant, showing in your diagram how the queued PCBs are linked using the relevant PCB fields

7) In a UNIX/Linux operating system, a process can enter a **zombie** state. Briefly describe what is meant by a **zombie** state and describe what sequence of events can lead a process into such a zombie state.

8) In a UNIX/Linux operating system, briefly describe what is meant by an **orphan** process. What sequence of events might cause a process to be orphaned?

SCHEDULING

1) Briefly state what is meant by the following types of process activity:

- *Processor-bound activity*
- *I/O bound activity*

2) Briefly describe a simple **SJF** (Shortest Job First) scheduler and state its advantages over a **FCFS** (First Come First Served) scheduler.

3) Draw a diagram of a **Round Robin** scheduler and state one advantage and one disadvantage for this type of scheduler.

4) Briefly explain the terms **pre-emptive** and **non-pre-emptive** scheduling. State one advantage and one disadvantage for each.

5) List four **scheduling objectives** when designing a scheduler.

6) Draw a diagram of a three level **static priority system.**

7) Assume a UNIX-like scheduler operates as follows:

- A **base priority** (threshold) for user processes is set to a value of **60** within a priority range **0..99** where 99 is lowest priority.
- A clock interrupts the processor **60** times per second, incrementing the **CPU_count** field for the ***running*** user processes on each interrupt.

- Rescheduling occurs once every second, where the scheduler recalculates the priorities for all *ready* user processes and for the *running* process, as follows:

CPU_count = (CPU_count / 2);

Priority = (CPU_count / 2) + base_priority + nice;

Assume three user processes P1, P2 and P3 are created simultaneously (their priority fields = 60). **Process 3** is given a nice value of **8**. Ignoring any other process activity and ignoring any scheduling or context switching overhead show in a diagram how these three processes are given access to the CPU for the first six seconds of operation. In your diagram show calculations for each process at the one-second intervals.

8) For the MS WINDOWS (WinAPI) scheduler, answer the following:

- Draw a block diagram for the **user level** environment of the Microsoft Windows operating system.

- Draw a block diagram for the **kernel level** environment of the Microsoft Windows operating system.

- What in meant by the ‘WIN-API’ interface?

9) With respect to the Microsoft Windows Win scheduler, please answer the following:

- How many priority levels exist in the scheduler design?
- If a **process** has a nominal priority level of 13 what is the highest **thread level** priority for that process?
- State a typical pre-emption **time quantum** for the Microsoft Windows scheduler in: a) a *Server* configuration and b) in a *Workstation* configuration.

10) Briefly describe what is a **signal** in the context of the UNIX/Linux operating system.

11) Briefly state what is meant by a **thread** in the context of an operating system. Briefly summarise the key differences between a **process** and a **thread**, highlighting any advantages for threads.

12) A UNIX/Linux operating system supports **named pipes** and **unnamed pipes** as interprocess communication mechanisms. Briefly describe each of these pipe types, highlighting the differences between them.

FILE SYSTEM CONCEPTS

VIRTUAL SYSTEMS

UNIX SHELL AND SCRIPTS

See attached addendum for bash command reference chart

1) Write a one line bash command to: find the **largest file** in the home directory and put the result in a file called **longFile** in your **home** directory.

2) Write a one line bash command to: find **how many files** are in the **parent** directory of your home directory and put the answer in a file called **countFile**, in your home directory.

3) Write a one line bash command to: find how many **directory files** are in your home directory, and put the answer in a file called **countDir**, in your **home** directory.

4) Write a one line bash command to: find out how many times the **mkdir** command been used in the past **100 history** lines and list the result in a file called **histCount** in your home directory.

5) Write **one line** bash shell commands solutions for each of the two problems below.

- Find how many **directory files** are in your **home** directory, and assign the answer to a variable called **dirCount**.

- Find out how many times the **mkdir command** been used in the past 100 **history** lines and list the result in a variable called **histCount**.

6) Assume that some variable **x** has the value **25%**. Write a one line bash command to eliminate the **%** character so that the variable can be used as an integer.

7) Write a one-line bash command as follows. The table shows the result of a command, using the bash shell. For all the processes belonging to the user **don2006**, list the **names** of these processes along with their respective **%CPU** utilisation, in a file called **ps_temp1**, in your home directory.

```
USER       PID %CPU %MEM   VSZ  RSS TTY      STAT START   TIME COMMAND
root      3302  0.0  0.0  1700  408 tty1     Ss+  Oct10   0:00 /sbin/mingetty tty1
root      3307  0.0  0.0  2996  408 tty2     Ss+  Oct10   0:00 /sbin/mingetty tty2
don2006  23727  0.0  0.0  6136 1424 pts/1    Ss   09:34   0:00 -bash
don2006  23818 60.8  0.0  4220  968 pts/1    R    09:49   0:17 /bin/bash ./busy_loop
```

8) Write a bash shell script program to check the amount of disk space that is available on your disk, where your home directory resides. If there is more than 90% of the disk space in use, then send a warning message to the user, to advise that the disk is more than 90% full.

Assume the output format for the **df -h** command is as follows:

```
Filesystem        Size  Used   Avail   Use%   Mounted on
/dev/sd3a         56G   13G    44G     22%          /
```

9) Write a Bash shell script program to do the following:

- Display how many processes exist in the system
- List the command name and PID for the busiest process
- Kill the busiest process

Assume the output of a ps -aux command is something like as follows:

```
USER       PID   %CPU  %MEM   VSZ   RSS   TTY     STAT  START    TIME    COMMAND
root       3321   0.0    0.0     1876   408   tty1     Ss+    2007         0:00    /sbin/mingetty tt
root        3340   0.0    0.0     2484   408   tty2     Ss+    2007         0:03    /sbin/mingetty tt
donal     17205   0.0    0.0     4420  1468   pts/2    Ss     08:31        0:25    -bash
joe       19168   0.0    0.0     2928   776   pts/2    R+     09:30        0:00    ps au
```

10) Write a **Bash** shell script program to do the following:

- Create an **array** that contains four unique file names.
- Make four files with these array file names, so that the files have the sizes **1kB**, **1MB**, **100MB** and **1GB**. It does not matter which file has which size.

11) Write a **Bash** shell script program to do the following:

- Write a simple **function** that will do a **floating point number** calculation (you can decide a calculation of your choice)
- Measure the **elapsed time** in **microseconds** that it takes to execute this function
- Print the elapsed time result to the terminal

12) Write a short Bash script program that uses a **trap** to act on the SIGINT signal. The trap is to be written as a **function**. When a SIGINT signal is received the trap will send a simple message to the terminal saying that this process (display the actual process ID for the shell process) is shutting down. The trap then causes the script to exit. *(Alternatively you could be asked to write the code to allow the main program to continue when the trap is finished)*

ADDENDUM: Commands

Quick Command Reference Chart

Command/Util	**Brief description**
awk	Scans a file(s) and performs an action on lines that match a condition. General format: *awk ‘ condition { action } ‘ filename* Example: *awk ' /University/ {print $3,"\t", $11}' myFile*
bc	Arbitary precision calculator Example: *echo "scale=3; (1 + sqrt(5))/2" | bc* …. calculates phi to 3 places
cal	Display a calendar output
cat	Concatenate file to the standard output
cd	Change directory
chmod	Change file access permissions
chown	Change file owner/group
cp	Copy files and subdirectories
cut	Cut columns from a data file Example: *cut –c 49-59 logfile* … extract column defined between characters 49 to 59
dd	Copy a file, converting and formatting Example: *dd if=/dev/zero of=myFile bs=1k count=10* … makes myFile of 10 kiloBytes
date	Display current time, set date etc. Example: *date +%s%N* …time with nanosecond resolution
df	Display disk space information
diff	Compare files line by line to find differences
du	Display disk usage information
echo	Display a line of text
exit	Exit the process e.g.: exit 0 … exits with the code 0
find	Search for files Examples: *find / -type d –print* …find directory files starting at root and display *find . –name “verse”* …find all files, starting at the current directory, with “verse” string at start of name
grep	Scans text files looking for a string match. Examples: *grep “and“ myFile* … search for lines containing “and” *grep “^The” myFile* … search for lines that begin with “The” *grep “floor$” myFile* … search for lines that end with “floor”
head	Display a number of lines at the head of a file
history	Display previous commands
kill	Sends a signal Example: *kill –HUP 43165* … send HUO signal to process 43165
less	Outputs a file to the console, a page at a time
ls	List directory(s) content ls –l long listing to show file details ls –R list subdirectories recursively ls –a list all files, including ones that start with *a* .
mkdir	Make directories
mkfifo	Make a named pipe Example: *mkfifo mypipe*
more	Outputs a file to the console, a page at a time
mv	Move files (effectively means to rename files)
ps	Show process status ps au show all processes, for all users

pwd	Print the name of the current working directory
read	Read user input
rm rm -R	Remove files and/or directories rm –r (or rm –R) will remove files recursively
rmdir	Remove directories (assuming directory is empty).
sed	A stream editor Example: *sed 's/Jack/Jill' filebook* … substitute the string 'Jill' for 'Jack' in file filebook
seq	Generates a sequence of numbers. Examples: seq 1 9 … generates numbers 1 to 9, line by line seq –s "-" 1 9 … default separator can be changed, using the –s option
set	If no options are used, set displays the names and values of all shell variables Examples: set …. shows all shell variables set | grep "USER" … shows shell variables with a specified string
sort	Sort lines in a text file sort –g general numeric sort sort –r reverse result of sort sort -k sort for a key position sort –n sort to string numerical value
tail	Display a number of lines at the end of a file
tee	Diverts a piped input to a second separate output Example: *cat demo_file1 | sort | tee demo_file1_sorted | more*
trap	Defines actions to take upon receipt of a signal or signals Example: *trap ' echo "This is my trap" ' SIGHUP* …. echo some text on receipt of HUP
uniq	Output a file's lines, discarding all but one successive identical lines
wc	Count number of lines, words, bytes etc. in a file wc –l count number of lines wc -c count number of bytes wc -m count number of characters
wait	Wait for child process to exit before finishing. e.g.: wait

Some common built-in shell variables

Variable	**Description**
$?	Exit status of the previous command
$$	Process ID for the shell process
$!	Process ID for the last background command
$0	Name of the shell or shell script
$PPID	Process ID for the parent process
$UID	User ID of the current process
$HOME	The home directory
$SHELL	The shell

Bash function example

```
# Example script program that uses two function parameters.
# The function calculates the product of the # two arguments:
# #! /bin/bash

# product is declared as a function and defined
product ()  {
(( product_var = $1 * $2 ))  # global variable
}

# The main program

product  22  3   # The product function is called, with two arguments
echo  "The answer is: $product_var"
exit
```