UNIVERSITY OF LIMERICK

OLLSCOIL LUIMNIGH

FACULTY OF SCIENCE AND ENGINEERING

DEPARTMENT OF ELECTRONIC AND COMPUTER ENGINEERING

MODULE CODE:	**ET4725**
MODULE TITLE:	**Operating Systems 1**
SEMESTER:	**Semester 2 2017/18**
DURATION OF EXAM:	**2.5 Hours**
LECTURER:	**H. Trifonov**

IMPORTANT INSTRUCTIONS TO CANDIDATES:

- **Answer any THREE questions**
- **This exam represents 70% of the full module assessment**
- **Module's percentage allocation of components:**

o Final exam(this exam):	70%
o Laboratory assignments:	20%
o In-class tests during term:	10%

- **All questions are of equal weight**
- **If you answer more than three questions you will be marked on the three best answers only**

Q1 **33 Marks**

a) **15 marks (5 marks each)**

Answer the following in relation to the **UNIX/Linux** operating system:

- ❖ Draw a simple **block diagram of the Linux operating system** and clearly label each block.
- ❖ With the aid of a state diagram show the various states for a process in a multitasking system. Label clearly all state transitions.
- ❖ Draw a diagram for a **round robin** scheduler and state one advantage and one disadvantage for this type of scheduler.

b) **18 marks**

Consider the following Bash script program.

```
#! /bin/bash
# The main code is here
./progB &        # start program progB in the background
# simple loop to simulate some activity
while true; do
        echo " Looping continuously "
        sleep 1
done
wait
exit
```

Modify the above program so that it will include a **signal trap**. The **trap** will do the following:

- ❖ Acts on the receipt of a **SIGINT** signal (i.e. Ctrl C from keyboard)
- ❖ Contains a function called **trap_func()**
- ❖ The **trap_func()** does the following:
  - displays (echoes) a simple message to say what is the **PID** for **progB**
  - sends a **TERM** signal to the running **progB** program
  - properly exits the script program without **orphaning** progB

Q2 **33 Marks**

Write a bash shell script to do the following:

- Make an array of **five** file names.
- Create the five actual files, using **dd**, with various file sizes ranging from 10kBytes to 10MBytes.
- Write a function called **file_copy()** to do the following:
  - ❖ The function is called with a **filename** parameter (positional parameter $1)
  - ❖ Copy (**cp**) the specified file (represented by $1) to any file name
  - ❖ Calculate the size of the file that is copied using **wc** command
  - ❖ Measure the **elapsed time** in **milliseconds** for the file copy operation
  - ❖ Calculate the data transfer **rate** for the file copy operation (i.e. file size/elapsed time)

  - ▪ For each file named in the array, call the **file_copy()** function.
  - ▪ Print a summary output for each coped file to show:
    - o *file name*
    - o *file size*
    - o *copy time*
    - o *transfer rate.*

NOTE – In the Addendum B of this paper there is a list of common bash shell commands.

Q3 **33 Marks**

a) **13 marks**

Write a **bash** shell **script program** to do the following:

- ❖ Find the **largest** file in the **/home/user** directory and save its name and size in variables **f_name** and **f_size**. The output from the command **ls –l** is in ***Table 1*** bellow.
- ❖ If the largest file is greater than **4096** bytes in size, then report the file size and its name to the user.

```
-rw-rw-r--   1  joe2018  joe2018  995   Feb  1   12:22  test1
-rw-rw-r--   1  joe2018  joe2018  1055  Jan  31  14:33  test2
drwxr-xr-x   26 joe2018  joe2018  4096  Mar  3   15.34  Documents
-rw-rw-r--   1  joe2018  joe2018  2596  Jun  22  08:05  test5
drwxr-xr-x   9  joe2018  joe2018  4096  Dec  31  15:12  Downloads
-rw-rw-r--   1  joe2018  joe2018  345   Nov  30  21:33  test3
-rw-rw-r--   1  joe2018  joe2018  170   May  2   15:12  test4
-rw-rw-r--   1  joe2018  joe2018  4870  Sep  20  10:47  test6
```

Table 1

b) **5 marks**

Briefly define the following terms in relation to **computer virtualisation**:

- ❖ Virtual machine
- ❖ Virtual appliance
- ❖ Type 1 hypervisor
- ❖ Type 2 hypervisor
- ❖ KVM

c) **10 marks**

Draw a **block diagram** for a computer system that has the following features, and **clearly** identify each feature on your diagram:

A single physical rack server has a **Type-1** hypervisor installed. There are four **VMs** hosted on this system where one guest runs **Windows 10**, another runs **Fedora 27**, another runs **Lubuntu 17.10** and the other runs **FreeBSD**. Various applications run under each one of the four guest operating systems. A **service/management console** is used in the scheme.

d) **5 marks**

Answer the following in relation to the **block size** (cluster size) for a file system:

- ❖ What is considered to be a **typical** block size?
- ❖ State an advantage for a **large** block size
- ❖ State an advantage for a **small** block size

Q4 33 Marks

a) 10 marks

If a UNIX file system is implemented using **1kByte** disk blocks and a **32-bit** size block addresses. The ***i-node*** holds 12 direct block addresses, one single-indirect block address, one double-indirect block address and one triple-indirect block address.

- ❖ What is the maximum **file size** for such file system?
- ❖ What is the maximum **file system** size?

Show your calculations step-by-step.

b) 13 marks

Write a **bash** shell script program to check the amount of disk space that is available on your disk volume, where your home directory resides. If there is more than **60% of the disk space** in use, then issue a warning message to the user, to advise that the disk is more than 60% full.

Assume that the output from the **df –h** command is as follows:

Filesystem	**Size**	**Used**	**Avail**	**Use%**	**Mounted on**
/dev/sda1	**57G**	**13G**	**44G**	**23%**	/

NOTE: In the Addendum of this paper there is a list of common bash shell commands.

c) 10 marks

In the context of a UNIX style file system, draw a typical UNIX **i-node** structure, labelling each field entry. In your diagram show how the i-node's **pointers** are used to keep track of a file's disk blocks.

ADDENDUM: Commands

Quick Command Reference Chart

The bash shell commands and utilities – a brief summary card (8/Dec/15)

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
```

Bash array example

```
#! /bin/bash

my_array=("black" "brown" "red" "sea blue")

for colour in "${my_array[@]}"; do

   echo "$colour"

done

exit 0
```