UNIVERSITY OF LIMERICK
OLLSCOIL LUIMNIGH

FACULTY OF SCIENCE & ENGINEERING

DEPARTMENT OF ELECTRONIC AND COMPUTER ENGINEERING

MODULE CODE:	**ET4725**
MODULE TITLE:	**Operating Systems 1**
SEMESTER:	**Semester 1 - 2015/16**
DURATION OF EXAM:	**2.5 Hours**
LECTURER:	**Dr. D. Heffernan**

IMPORTANT INSTRUCTIONS TO CANDIDATES:

- **Answer any THREE questions**
- **This exam represents 70% of the full module assessment**
- **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) Answer the following: **6 marks**

- How many **sectors** (512 bytes) are on a **1TByte** disk drive? How many **4kByte blocks** are on a **1TByte** disk drive??
- What is the **size in bytes** for each of the following terms:

  1EBytes (Exa), 1MBytes (Mega), 1GBytes (Giga), 1TBytes (Tera)

b) **7 marks**

For the Microsoft **FAT** file system:

(i) Draw a simple **FAT table** diagram to show an example of how clusters are mapped to a small **3-cluster** file. In the table indicate '*free clusters*' and a '*bad cluster*'.

(ii) If a **FAT-16** file system uses a cluster size of **16kBytes**, what is the maximum **volume size** for this system?

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.

d) **10 marks**

A UNIX file system is implemented using disk blocks of size **1kByte** (1024), and **32-bit** size block pointer 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. Clearly showing your calculations, what is the maximum **file size** for such a file system, and what is the maximum **file system** size?

Q2

33 marks

a)

9 marks

Table Q2 shows the result of a **ps au** command, using the bash shell.

Write a **single line** of bash commands to do the following:

For all the processes belonging to the user **joe2016**, list the **names** (i.e. COMMAND) of these processes along with their respective **%CPU** utilisation, in a file called **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
joe2016	23727	0.0	0.0	6136	1424	pts/1	Ss	09:34	0:00	-bash
joe2016	23818	60.8	0.0	4220	968	pts/1	R	09:49	0:17	/bin/bash ./busy_loop

Table Q2

b)

11 marks

Write a **bash** shell **script program** to find the largest file in your home directory. If the largest file is greater than **2,000,000** bytes in size, then report the file size to the user. (For the **ls – l** command, assume that file size is listed in column **5**.)

c)

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 **70% of the disk space** in use, then send a warning message to the user, to advise that the disk is more than 70% full.

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

Filesystem	Size	Used	Avail	Use%	Mounted on
/dev/sd3a	63G	13G	41G	22%	/

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

Q3 **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 5kBytes to 5MBytes.
- Write a function called **file_copy()** to do the following:
  - The function is called with a file name parameter (positional parameter $1)
  - Copy (**cp**) a file (represented by $1) to any file name
  - Calculate the size of the file that is copied using the **wc** command
  - Measure the elapsed time in **milliseconds** for the file copy operation (use **date** command to read time)
  - Calculate the data transfer **rate** for the file copy operation (i.e. file size/elapsed time)
  - Print a summary output to show: file name, file size, copy time, transfer rate.
- For each file named in the array, call the **file_copy()** function.

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

Q4 **33 marks**

Write a UNIX-like shell command that will copy one file to another. This command will be called **kp** and it will be a simplified version of the well-known **cp** command.

The **kp** command will have the following features:

- The command format will be: kp [options] <source file> <destination file>
- The **kp** program will simply open the **source file** and create the **destination file**, and will copy the contents of the source file to the destination file.
- The **kp** command will parse the command line for options (assume an option is signified by the ' –' character). If there is an option, then print "You have selected an option", otherwise you do not need to further process any option.
- The **kp** program does **not** need to include any **error checking**.

Your program is to be written in the **C** programming language.

*No error checking is required in your program. You do not need to list all of the C library include files, i.e. the .h files. A skeleton program and a system call list are provided in the **Addendum A** to this paper..*

ADDENDUM A: System calls

Some selected system calls

(ver 22/October/15)

PROCESS CONTROL

System call	Brief description
execl ()	A process runs another executable file by replacing its own code, data, and stack with the new executable program Format: int **execl** (char* path, char* arg0, …, char* argn, NULL)
exit ()	The exit operation terminates a process. The parent must acknowledge this. Format: int **exit** (int status)
fork ()	Duplicates a process. Format: int **fork** ()
getpid ()	Returns the process' ID number, i.e. the PID. Format: int **getpid** ()
getppid ()	Returns the process' parent ID number, i.e. the PPID. Format: int **getppid** ()
nice ()	Changes a process' priority. Format: int **nice** (int delta)
wait ()	A process waits for a child process to terminate. Format: **wait** [pid]

FILES AND PIPES

System call	Brief description
kill()	Send a signal to a process Format: int **kill** (pid_t pid, int signum)
signal()	Registers a new signal handler function for the signal signum Format: typedef void (*sighandler_t)(int); sighandler_t **signal** (int signum, sighandler_t handler)
mkfifo()	Creates a named pipe Format: int **mkfifo** (const char *path, mode_t mode)
pipe ()	Creates an unnamed pipe Format: int **pipe** (int fd [])
open()	Opens a file Format: int **open** (char* filename, int mode [, int permissions]) Format using a file descriptor: int **open** (int fd)
close ()	Closes a file Format: int **close** (int fd)
read ()	Reads bytes from a file to a buffer Format: int **read** (int fd, char* buf, int count)
write ()	Writes bytes from a buffer to a file Format: int **write** (int fd, char* buf, int count)
unlink ()	Removes a file (or pipe) Format: int **unlink** (const char* filename)

File flags

Flag	Description
O_RDONLY	Read only access.
O_WRONLY	Write only access.
O_RDWR	Read and write access.
O_APPEND	Position file pointer at the end of file before writing.
O_CREAT	If file does not exist, then create the file.
O_EXCL	If file already exists and O_CREAT is set, open () fails.
O_NONBLOCK	Used for named pipes only.
O_TRUNC	If file exist, then truncate it to zero length.

File **permissions** value is an octal representation.

Permissions	Binary representation	Octal representation
rwx r- - r- -	111,100,100	0744
rw- - - - - - -	110,000,000	0600

File status

The **fstat**() is a C library call has the following format:

int fstat(int *fd*, **struct stat** **buf*);

The function returns 0 on success and -1 on failure.

The information on the file is contained is a **stat** structure that has the following fields – not an inclusive set of fields:

```
struct stat {
   ino_t    st_ino;       // inode number
   mode_t   st_mode;      // protection
   uid_t    st_uid;       // user ID of owner
   gid_t    st_gid;       // group ID of owner
   off_t    st_size;      // total size, in bytes
   etc...                 // all fields not shown
   etc...
};
```

TIME and RESOURCES

System call	Brief description
time()	Gets the current time, i.e. the number of seconds from the UNIX epoch. Type **time_t** is a **long int** Format: time_t **time** (time_t*)
localtime()	Converts a **time_t** value to a **tm structure** as local time. Format: struct tm * **localtime** (const time_t*)
asctime()	Convert the **tm structure** to a printable string representation. Format: char ***asctime** (const struct tm *)
ctime()	Converts a **time_t** value to printable string representation. Format: char ***ctime** (const time_t*)
gettimeofday()	Gets number of seconds and microseconds since the UNIX epoc. Format: int **gettimeofday** (struct timeval *tv, struct timezone *tz)
getrusage()	Reads a process's resource statistics from the kernel. Format: int **getrusage** (int who, struct rusage *usage)

The **struct timeval** lists the number of seconds and **microseconds** since the UNIX epoch (the first second on 1st January 1970):

```
struct timeval {
   time_t        tv_sec;   // seconds (long int)
   suseconds_t   tv_usec;  // microseconds (long int)
};
```

For **getrusage()** the resource usages are returned in a **struct rusage** structure, shown here to emphasise the CPU time entries:

```
struct rusage {
   struct timeval ru_utime;   // user CPU time used
   struct timeval ru_stime;   // system (i.e. kernel) CPU time used
   /* other entries etc... */
};
```

Example call to getrusage() is:

```
struct rusage ru;
getrusage(RUSAGE_SELF, &ru);
```

SIGNALS – some basic signal calls

kill()	int **kill** (pid_t pid, int signo);
signal()	#include <sys/signal.h> int (* **signal** (signo, function)) int signo; void (* function)(int);
pause()	int **pause** (void);

MUTEXES

Purpose	Function format
Initialise the mutex	int **pthread_mutex_init** (pthread_mutex_t *mut,

	const pthread_mutexattr_t *attr); or pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
Lock the mutex	int **pthread_mutex_lock** (pthread_mutex_t *mut);
Unlock the mutex	int **pthread_mutex_unlock** (pthread_mutex_t *mut);
Acquire lock, else EBUSY is returned.	int **pthread_mutex_trylock** (pthread_mutex_t *mut);
Deallocates for this mutex.	int **pthread_mutex_destroy** (pthread_mutex_t *mut);

THREADS

Function call	Purpose
pthread_create()	Creates a new thread and associates a function with that thread
pthread_join()	After creating a thread the calling process must wait for that thread to exit. It does this using the pthread_join() call
pthread_exit()	A thread will exit (i.e. finish) using the pthread_exit() call.
Note, use **#include <pthread.h>** to include the thread linrary	

IP SOCKETS

System call	Brief description
socket()	Creates a new socket of a defined type Format: int **socket**(int domain, int type, int protocol);
bind()	Associates socket with socket address structure. Format: int **bind** (int sockfd, const struct sockaddr *my_addr, socklen_t addrlen);
listen()	Server side causes TCP socket to go to listening state Format: int **listen**(int sockfd, int max);
connect()	Client side attempts to establish a new connection Format: int **connect**(int sockfd, const struct sockaddr *serv_addr, socklen_t addrlen);
accept()	Server side - accepts incoming client attempt Format: int **accept**(int sockfd, struct sockaddr *cli_addr, socklen_t *addrlen);

Example code fragment to show socket address structure

```
struct sockaddr_in myAddr ;    // Declare myAddr as a socket address structure

myAddr.sin_family = AF_INET;            // The IPv4 domain
myAddr.sin_port = htons (5000);         // htons() means 'host to network short'.
myAddr.sin_addr.s_addr = INADDR_ANY ;   // System will select IP address
```

Another approach to assigning the IP address is as follows:

```
inet_pton (AF_INET, "188.40.16.174", &myAddr.sin_addr); // inet_pton() is explained below
```

SEMAPHORES

*For **semaphore** call examples – see the **skeleton.c** program below.*

Skeleton program example

```
/*****************************************************************
Program: skeleton.c
A simple program example, showing a single thread and a semaphore.
ANSI escape characters are use to clear the screen and to set the cursor position.
D. Heffernan        16/March/2007    mod: 18/4/2008  mod: 14/11/2013
*****************************************************************/

#include <stdio.h>
#include <semaphore.h>
#include <stdlib.h>

sem_t my_mutex ; // Declare a semaphore called my_mutex
void *my_function (); // Function - separate thread will call this functions.

// MAIN HERE
int main () {

sem_init (& my_mutex, 0, 1);  // Initialise the semaphore to 1
pthread_t my_thread; // Declare thread
int rc1;

printf ("\033[2J"); // use ANSI escape code to clear the console screen

// Create my_thread
if ( (rc1 = pthread_create (& my_thread, NULL, & my_function, NULL )))
{printf ("Error in creating %d\n", rc1);}

// The 'pthread_join' to wait until a thread is properly created
pthread_join ( my_thread, NULL) ;

sem_destroy (&my_mutex) ;   // destroy the semaphore after use
 return 0;  // exit the main function
}

// THREAD  ... This is my_thread, uses my_function ()

void * my_function()
{
  sem_wait (& my_mutex) ;    // do a semaphore wait on my_mutex
  printf ("\033[%d;%dH", 12 , 20); // set cursor position (row 12 and column 20)
  printf ("The is test in writing to the console\n"); // write something
  printf ("\033[%d;%dH", 24 , 0) ;  // set cursor to the end of screen
  sem_post (& my_mutex) ;      // do a semaphore post (i.e. signal) on my_mutex

  pthread_exit (NULL) ;  // exit the thread
}
```

MEMORY

Useful memory operations as part of the C **string.h** library

Function	Brief description
memset()	Copies character **c** to first **n** characters of memory area that is pointed to by **str**. Format: void ***memset**(void *str, int c, size_t n); *.. returns pointer to memory area str*
memcpy()	Copies **n** bytes from memory area **src** to memory area **dest** Format: void ***memcpy**(void *dest, const void *src, size_t n); *.. returns pointer to memory area dest*

Explictly allocating memory

Function	Brief description
malloc()	Allocates memory for requested number of bytes and returns a pointer to it Format: void ***malloc**(size_t size) Example: str = (char *) malloc(15); // 15 bytes allocated, pointed to by str
free()	Deallocates the block of memory pointed at by ptr. Format: void **free** (void *ptr); Example: free(ptr);

Shared memory API calls: sample C program that uses a shared memory

```
int main()
 {
int mfd;
char *shared_msg;
long p_size;

// get page size on system
p_size = sysconf(_SC_PAGESIZE);

//create 'my_object' – file descriptor is mfd
mfd = shm_open("my_object", O_CREAT | O_RDWR, S_IRWXU | S_IRWXG);

//set 'my_object' size to 1 page long
ftruncate(mfd, p_size) ;

// map 'my_object' to memory - returns pointer
shared_msg = mmap(NULL,1*p_size, PROT_READ | PROT_WRITE, MAP_SHARED, mfd, 0);

memcpy(shared_msg, "University of Limerick", 30); //put something in the shared memory
printf ("Shared memory content: %s\n", shared_msg);
munmap(shared_msg, p_size); // unmap the assigned memory from this process
shm_unlink("my_object");  // remove 'my_object' name from /dev/shm/

close(mfd); // always close file descriptors when finished

return 0 ;
}
```

ADDENDUM B: 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:
# Not intended use of return
#! /bin/bash

# product is declared as a function and defined
product () {
(( product_var = $1 * $2 ))
return $product_var   # The product_var is returned
}

# 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
```