Index number: .....

UNIVERSITY OF RUHUNA

Faculty of Engineering

End-Semester 6 Examination in Engineering: February 2020

Module Number: EE6206

Module Name: Operating System Programming – Part I(MCQ)

[Forty Minutes only]

[This paper consists of 20 MCQs. Answer all questions, each question carries 0.5 marks.
Part I Will be collected at the end of forty minutes.
Write the index number in all pages.
Select only one answer.
Underline or cross the correct answer for each question.]

Q1 Which of the following is not a way of creating an empty text file **Doc1.txt**?

(i) Type `cat > Doc1.txt` in console. Then, press Enter. Type *"Text in file"*. Then, press `Ctrl + D`.
(ii) Type `cat > Doc1.txt` in console. Then, press Enter. Type *"Text in file"*. Then, press `Ctrl + C`.
(iii) Type `touch Doc1.txt` in console.
(iv) Type `cat > Doc1.txt` in console. Then, press Enter. Then, press `Ctrl + D`.

Q2 The correct Linux console command to copy the content of **doc1.txt** to a new **doc2.txt** or an existing **doc2.txt** replacing its content is?

(i) `cat doc1.txt >> doc2.txt`
(ii) `cat doc1.txt doc2.txt`
(iii) `cat > doc1.txt doc2.txt`
(iv) `cat doc1.txt > doc2.txt`

Q3 Which of the following is not one of the tasks performed by **cat** command?

(i) Display the contents of a file
(ii) Change the file permissions.
(iii) Append the content of one file at the end of another file
(iv) Create a file

Q4 Select the statement which will not cause the Operating System to reboot?

(i) `sudo reboot`
(ii) `sudo init 6`
(iii) `sudo init 0`
(iv) `sudo shutdown -r now`

Index number: .....

Q5 True statement regarding the following code segment is?

```
void ascending_swap(double *p1, double *p2);
int main(void)
{
      double values[10];
      for(int i=0;i<10;i++)
      {
            printf("\nEnter number %d: ", (i+1));
            scanf("%lf",&values[i]);
      }
      ascending_swap(values + 1, values +3); //line 10
      return(0);
}
void ascending_swap(double *p1,double *p2) //line 13
{
      if(*p1 > *p2) //line 15
      {
            double temp;
            temp = *p2;
            *p2 = *p1;
            *p1 = temp;
      }
      else{}
}
```

(i) Line 10 is not erroneous. *p1 in line 13, 15 tells that double value of at the address pointed by pointer p1, p1 is a pointer of type double respectively.

(ii) Line 10 is erroneous. *p1 in line 13, 15 tells that p1 is a pointer of type double, double value of at the address pointed by pointer p1 respectively.

(iii) Line 10 is not erroneous. *p1 in line 13, 15 tells that p1 is a pointer of type double, double value of at the address pointed by pointer p1 respectively.

(iv) Line 10 is erroneous. *p1 in line 13, 15 tells that double value of at the address pointed by pointer p1, p1 is a pointer of type double respectively.

Index number: .....

Q6 True statement regarding the following code segment is?

```
union myunion
{
      char char_value;
      int int_value;
      double double_value;
};
int main()
{
      char usr_char[2] = "AB"; int usr_int = 60; double usr_double 12.50;
      union myunion new1;
      new1.char_value = usr_char;
      new1.int_value = usr_int;
      new1.double_value = usr_double;
      printf(" string is %c, integer is %d and double is %lf", new1.char_value,
      new1.int_value, new1.double_value);
      return(0);
}
```

(i) The terminal screen will show **double_value** as 12.50 while other two variables will not be shown as user specified values.

(ii) The terminal screen will show **int_value** as 60 while other two variables will not be shown as user specified values.

(iii) The terminal screen will show **char_value** as AB while other two variables will not be shown as user specified values.

(iv) The terminal screen will show **double_value** as 12.50, **int_value** as 60, **char_value** as AB.

Q7 What is not true about threads?

(i) A thread shares a common address space in virtual memory with other sibling threads of the same process.

(ii) Different areas in stack of a process is allocated to each sibling thread of a process.

(iii) A thread can be terminated either when main thread calls *exit( )* or a sibling thread calls *exit( )*.

(iv) A thread terminates when the thread function returns or *pthread_exit( )* is called.

Index number: .....

Q8 What is false about *fork( )* function?

(i) After calling this function, the new process will have a separate copy of the virtual memory address space for stack, heap and data segments.
(ii) The child process will always execute first after a *fork( )* call.
(iii) *fork( )* function returns the process identifier of the child process inside the parent process.
(iv) It returns -1 on error and new process created will execute program text stored in different address spaces of the virtual memories of the parent and child processes.

Q9 What is true out of the following statements?

(i) When a parent process terminates, all of its child processes are also terminated.
(ii) Swap area is a memory in primary memory.
(iii) A page fault occurs when a virtual memory address is mapped to a memory region in Random Access Memory.
(iv) When a child process is orphaned, the initialization process is assigned as the parent of that child.

Q10 What is true out of the following statements?

(i) The function of a page table is to map virtual memory addresses to pages in primary memory.
(ii) The tendency of a process to refer to adjacent memory addresses of an already accessed process is known as temporal locality of reference.
(iii) All of the virtual memory addresses of a particular process are stored in primary memory at a given instance.
(iv) Pointers to Machine language instructions reside in Heap segment of virtual memory of a process.

Q11 What is true out of the following about virtual memory?

(i) Presence of virtual memory decreases CPU utilization as more processes can be loaded in to the primary memory.
(ii) Stack grows downwards and heap grows upwards.
(iii) *malloc(size1)* function is used allocate size1 bytes of real memory. That memory will have an address in stack of virtual memory of the process which *malloc( )* was called.
(iv) A **static integer x** initialized to 5 inside a function will be having an address inside initialized data segment of the virtual memory.

Index number: .....

Q12 What is true out of the following statements?

(i) Just after a *fork( )* call, the child has content referred by its stack, heap and data segments exactly as the contents(values) refereed by addresses of stack, heap and data segments of the parent process.
(ii) After a child process is created, it cannot change values referred by virtual memory areas independently from the parent process.
(iii) *exit( )* called in a child process will start the waiting of a parent called by *waitpid(-1,NULL,0)* in parent process.
(iv) One pipe can be used for two-way communication between 2 given processes.

Q13 What is true about the following statements?

```
int fd[2];
int x = pipe(fd);
fork( );
```

(i) **fd[0]** is associated with write end of the file.
(ii) **fd[1]** is associated with read end of the file.
(iii) If **x** is zero, a pipe is created.
(iv) Parent process and child process after *fork( )* call will share two file descriptors **fd** with the pipe.

Q14 Select the correct order of the header files that define each of the functions *fork( ), exit( ), pipe( ), printf( ), getpid( )*?

(i) <unistd.h>, <stdlib.h>, <unistd.h>, <stdio.h>, <unistd.h>
(ii) <stdlib.h>, <unistd.h>, <stdio.h>, <unistd.h>, <errno.h>
(iii) <unistd.h>, <sys/wait.h>, <unistd.h>, <stdio.h>, <errno.h>
(iv) <unistd.h>, <stdio.h>, <unistd.h>, <stdlib.h>, <unistd.h>

Q15 **`ret_typ msgsnd(dt1 arg1, dt2 arg2, dt3 arg3, dt4 arg4)`** is a pseudo code showing the definition of *msgsnd( )* function that is used to send a message to a message queue. Here, "**ret_typ**" is the return type of the function. **dti, argi** are the data type and argument name of the i^{th} argument respectively. The correct sequence showing the data types of **ret_typ, dt1, dt2, dt3, dt4** in correct order is?

(i) int, int, void *, long, size_t
(ii) int, int, void *, size_t, int
(iii) int, int, void *, size_t, long
(iv) int, int, size_t, void *, int

Index number: .....

Q16 File descriptors associated by default with a C program for standard output, standard error, keyboard respectively is?

(i) 0, 2, 1
(ii) 2, 1, 0
(iii) 0, 1, 2
(iv) 1, 2, 0

Q17 Which of the following cannot be a value for the arguments **offset, whence** of *lseek( )* function respectively are?

(i) 5, 0
(ii) 4, SEEK_SET
(iii) 1, 2
(iv) 1, 3

Q18 What is true out of the following?

(i) When *sem_unlink( )* is called, the semaphore will be erased from the memory immediately even though there are processes which have opened the semaphores and have **sem_t** type pointers to it.
(ii) If multiple threads are waiting based on a particular condition variable, when that condition variable is signaled, only one thread will reacquire the lock. The other threads have to wait for another signal on the condition.
(iii) A semaphore resembles function of a mutual exclusive lock only.
(iv) Concurrency control using semaphores is required between processes which update process global variables.

Q19 What is false out of the following?

(i) Any process can increment the value of a semaphore.
(ii) Destroying a locked mutex can lead to locks which are never unlocked.
(iii) When a process closes a semaphore, it will remove the semaphore from both the kernel memory and the processes' virtual address space.
(iv) Multiple threads can attempt to lock an already locked mutex, but only one of them will be able to lock the mutex when the already locked mutex is unlocked.

Index number: .....

Q20 What is false out of the following?

(i) Thread synchronization is not mandatory when each thread update its own local variable.
(ii) A concurrency control mechanism is required for hyper-threading applications that update the data segment variable of virtual address space.
(iii) Multiple threads updating a static variable defined inside a function does not cause inconsistent updates.
(iv) A semaphore can be used in shared memory to control simultaneous updates of the shared memory.