Q1. [1.5 marks x 5 = 7.5 marks].
Write short textual answers.

No marks for making diagrams in part (a), (b) and (c).

Note: No Drawings. Explain all technical terms in your answer to get full marks.

a) What are the Pros and Cons of a Microkernel OS implementation. Explain. [1]

Pros: Modularity. OS functionalities other than kernel itself run as separate process in user mode. Any failure will not impact the kernel. This makes kernel code smaller orders of magnitude thus efficiently of kernel processing. [0.5]

Cons: Microkernel approach suffers from performance overhead due to communication latency when a system call received by kernel need to access functionality that is now running as a user mode process (process management, file sub-system, and others. See diagram). Each access also requires switching between kernel mode and user mode (yet another costly overhead). The software design is modular however, it is much more complex than monolithic. [0.5]

b) Consider three long running processes P1, P2 and P3. For a time, slice of ten microsecond, explain all actions which the scheduler takes to run these processes. [2]

In every ten microseconds, the scheduler performs the following actions:

1. Perform context switch:
    a. Save data of currently running process (e.g. P1) to its PCB.
    b. Put outgoing process into tail of ready queue.
    c. Select a new process for execution using an algorithm from the head of ready queue.
    d. Load data of newly selected processes (e.g. P2) into from PCB.
2. Start executing the newly loaded process (P2 in our case)

[1] for points 1and 2

3. The scheduler also performs context switch when the running process makes a system call and/or kernel receive an Interrupt.
4. Repeat 1 and 2 perpetually. So, processes P1, P2 and P3 run continuously.

[1] for points 3 and 4

c) What action does OS take to manage each type of software interrupts? [1]

OS action for software Interrupts: The OS first saving the current state of the CPU, including registers and program counter, to ensure the interrupted process can resume later. It then identifies the Interrupt type through an interrupt vector table and invokes the corresponding Interrupt handler routine. Finally, the OS restores the state saved and resumes normal execution. This ensures controlled and secure handling of software Interrupts. [0.5]

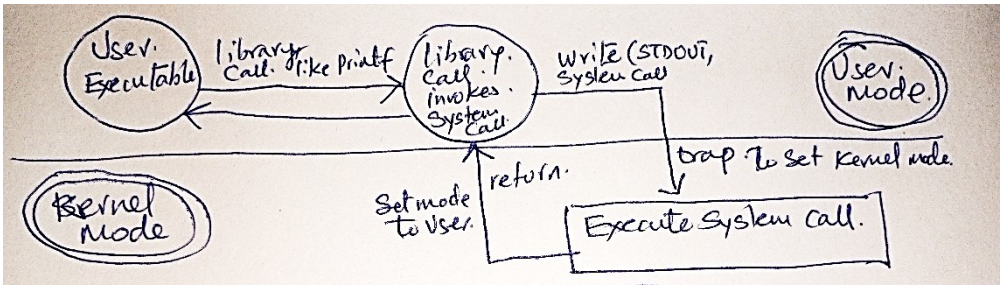
Type of software Interrupts are (for non-technical persons: all means the same software Interrupt): [0.5]

1. INT instruction: User triggered.
2. Traps: OS uses it to perform special functions as part of interrupt processing. For example, change modes: user to kernel and kernel to user.
3. Exceptions: Errors triggered. For example, Divide by Zero.

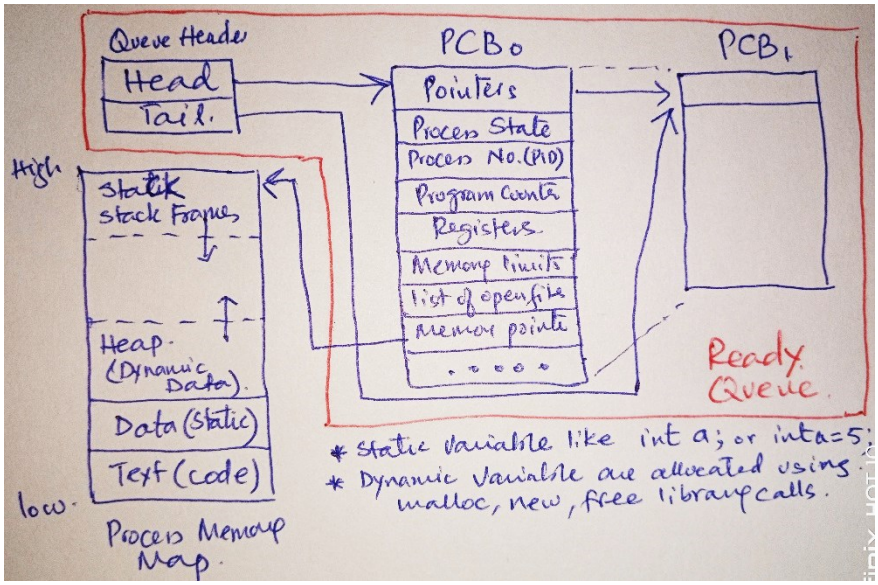
In part (d) and (e), the text provided below as part of the answer is only to explain the solution. No marks for writing any type of text.

Give only labelled diagrams (DO NOT use pencil). [2 + 2] *Note: No grading of textual answers here.*

d) Show how a system call originates from a user executable, how OS completes it and return control back to the same executable. *Note: Deduction of marks for each missing step, label, and errors in flow of execution.*



e) Draw a read queue implemented as a linked list with two processes. Label all elements of one PCB in this list to show your understanding of its contents. Now, draw a labelled process memory map. Show which part of memory map contains process's static and dynamic variables.



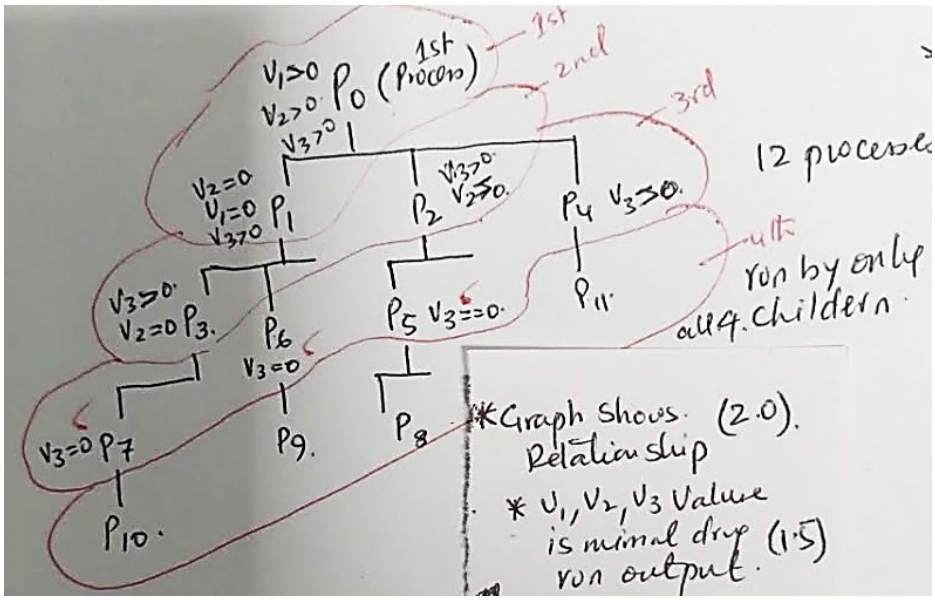
Q2. [3.5 marks + 4 marks = 7.5 marks]

Understanding code semantics

a) Suppose a process executes the code shown in Figure 1. Dry run the execution by showing all variable values and other work. Also, draw the relationship between the resulting processes with brief descriptive hints. [1.5 + 2 = 3.5]

```
v1=fork();
v2=fork();
v3=fork();
if (v3==0)
  fork();
```

Figure 1



Code Implementation

b) Write a C program that creates three processes. Each process prints 5 integers each from an array {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15}. What output will appear on the screen? Explain why it appears in the order you have written it on your answer sheet. [2 + 1 + 0.5 = 4].

Code [2] *Context as per syllabus covered up to Section 3.3 of Chapter # 3 of the textbook.

If student coded logical structure of the program, give partial marks.

No partial marks on individual C statements without the logical structure of code.

[See coding guidelines of Lecture # 1]

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>

void print_val (int lv){
   for (int j = lv; j < lv + 5; j++) {
      printf("%d ", arr[j]);
      }
   printf("\n");
}

int main() { parent process P0
    int arr[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15};
    int start_index = 0;

    pid_t pid = fork();
    if (pid == 0) { // child process P1
       printf("Process : ");
       if (start_index != 0) start_index += 5; // Move to the next 5 elements
       print_val(start_index);
       return 0; // exit P1
    } else if (pid > 0) { // parent process P0
       printf("Process : ");
       if (start_index != 0) start_index += 5;
       print_val(start_index);
       pid_t pid1 = fork();
       if (pid == 0) { // P2 - 2nd child of P0
          printf("Process : ");
          if (start_index != 0) start_index += 5;
          print_val(start_index);
          return 0; // exit P2
          }
       else
          //wait (NULL); // shifted below. No error checking.

    } else
       return 1; // fprintf(stderr, "Fork failed!\n");
       }
    wait(NULL); // Wait for 1st child process to finish.
    wait(NULL); // Wait for 2nd child process to finish.
    return 0;
}
```

Output [1]

1. Process : 1 2 3 4 5
2. Process : 6 7 8 9 10
3. Process : 11 12 13 14 15

Why the above order? [0.5]

The output lines above could be from P0, P1 or P2 as operating system schedules their execution i.e., the order of the output depends on which process gets CPU time first. Output of line 1 could be from P0 as it prints output, spawns P2 and start waiting for termination of children, while P1 and P2 sit on ready queue for their turn for execution.