National University of Computer and Emerging Sciences, Lahore Campus

	Course Name:	Operating Systems	Course Code:	CS 205
	Program:	Bachelors in Computer Science	Semester:	Spring 2019
	Duration:	60 minutes	Total Marks:	30
	Paper Date:	26th Feb 2019	Weight	15
	Section:	ALL	Page(s):	3
	Exam Type:	Mid 1		

Student : Name:_______________________________ Roll No.________________ Section:_______

Instruction/Notes: Attempt all questions. Programmable calculators are not allowed.

Q1: Complete the missing code and answer the Multiple Choice Questions. **[10 marks]**

a) Add the missing code.

The following piece of code—after completion—will senda string from the parent process to the child process. The child process, in turn, will calculate the length of the string and send it back to the parent. The parent will then print the length sent back by the child. The output of the code below should be: (4 marks)

```
child read Greetings
size sent by child 9
```

You are required to add missing statements so that the child is able to send the length back to the parent. API for write and read system calls are:

```
ssize_t write(intfd, const void *buf, size_t count);
ssize_t read(intfd, void *buf, size_t count);

//assume all #include statements are there.
#define BUFFER_SIZE 25
#define READ_END       0
#define WRITE_END      1

int main(void) {
        char msg[BUFFER_SIZE] = "Greetings";
        int size;  pid_tpid;
        intfd[2];
        int fd1[2];

        pipe(fd);
        pipe(fd1);
        pid = fork();

        if (pid> 0) {

                close(fd[READ_END]);
                close(fd1[WRITE_END]);

                write(fd[WRITE_END], msg, strlen(msg)+1);
                read(fd1[READ_END], &size, 4);

                printf("size sent by child %d\n", size);

                close(fd[WRITE_END]);
                close(fd1[READ_END]);

        } else {
                close(fd[WRITE_END]);
                close(fd1[READ_END]);

                read(fd[READ_END], msg, BUFFER_SIZE);
                printf("child read %s\n",msg);
                size = strlen(msg);

                write(fd1[WRITE_END], &size, 4);

                close(fd[READ_END]);
                close(fd1[WRITE_END]);

        }
        return 0;
}
```

b) Which of the option arranges the following technologies in the order from fastest to slowest:

a. Hard-disk drives, main memory, cache, registers
b. Registers, main memory, hard-disk drives, cache
c. Registers, cache, main memory, hard-disk drives
d. Cache, registers, main memory, hard-disk drives

c) When two processes communicate through Message-Passing with Zero Capacity buffering, the process sending the message:

a. Gets an error returned to it if the receiver is not ready.
b. Gets blocked if the receiver is not ready.
c. Cannot send a message because the buffer size is zero.
d. Allocates more memory before calling the send() method

d) OS can be defined as a

a. Resource Allocator
b. Control Program
c. User Program
d. Both a and b

e) How is modularity added to the Linux kernel that is typically a monolithic kernel:

a. By adding more system calls
b. Through loadable modules
c. By adopting micro-kernel approach in Linux
d. Both a and b

f) In the five-state model why would a process move from Running to Ready state?

a. The process has terminated
b. The process needs to perform an I/O operation
c. The process' time quantum has expired
d. The process needs to execute a system call

g) A process stack does not contain:

a. Function parameters
b. Local variables
c. Return addresses
d. PID of child process

Q2: CPU Scheduling **[10 marks]**

Suppose four (4) processes given in the table below. You have to execute them using a scheduling algorithm that allows *preemption* and prefers executing the process with the least (minimum) CPU (remaining) bursts. The arrival times and the CPU bursts needed to complete them are also provided. Among the four processes, P_2 needs I/O bursts to complete its execution such that after every 3 CPU bursts (3 time units to be specific), it requires I/O bursts time equivalent to 3 CPU bursts.Keeping in view the following requirement, you are required to find the following:

- Draw the Gantt chart to show how these processes would complete their execution
- Find the waiting time of each process and average waiting time of all the processes

Processes	**CPU Bursts needed**	**Arrival Time**
P_1	*13*	*0*
P_2	*6*	*5*
P_3	*10*	*7*
P_4	*3*	*10*

Answer the following:

What is this algorithm called ________Shortest Remaining Time First____________

Waiting time for P1 _(8-5) + (16-10) = 8__ Waiting time for P2 0 + (13-11) = 2

Waiting time for P3 22-7 = 15 Waiting time for P4 _______0______

Average waiting time _________8+15+2+0 / 4 = 6.25_______

GANTT CHART BELOW:

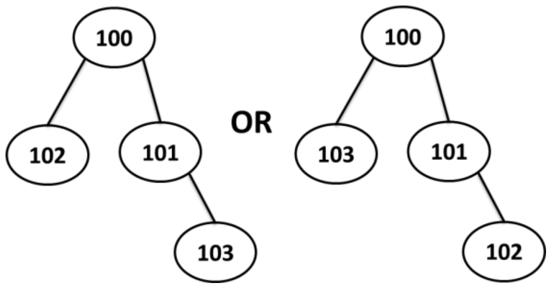
P1	P2	P1	P4	P2	P1	P3
0 5	8	10	13	16	22	32

Q3: Processes **[10 marks]**

Consider the code segment in the right box that creates multiple child processes. Assume a variable NEXT_PROCESS_ID, maintained by the OS, initialized to 100. Each time a new process is created, it gets value of NEXT_PROCESS_ID as its process id. NEXT_PROCESS_ID is then incremented to prepare next id for the next process creation request. There is no compilation or execution error in this code.

1. How many *new* processes are created *(do not count the initial main() process)*?
   Answer: ________4______________

2. Create Process tree by showing each process node with its process id. Tree must clearly show parent child relationship for all the processes.



3. Show output of the code below. Write only one sequence if you feel that multiple sequences can be printed.

Parent process waiting for child termination
Message: Welcome
Child Process called
Message: OS course
Child Process called
Message: OS course
Parent Terminating

Green lines are fixed.
Order of yellow lines can be modified. Note that it is not possible for both "Message: OS course" lines to appear above "Child Process Called" lines.

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

#define BUFFER_SIZE 25

int main () {

  char msg[BUFFER_SIZE] = "Welcome";
  pid_t pid = fork ();

  if (pid > 0) {
    strcpy (msg, "Welcome to OS course");
    printf ("Parent process waiting for child termination \n");
    wait (NULL);
    printf ("Parent Terminating \n");
  }
  else {
    printf ("Message: %s \n", msg);
    pid_t pid1 = fork ();

    strcpy (msg, "OS course");
    pid_t pid2 = fork ();

    if (pid2 == 0) {
              strcpy (msg, "Adv OS course");
              printf ("Child Process called \n");
    }
    else  {
              wait (NULL);
              printf ("Message: %s \n", msg);
    }

    if(pid1 > 0) {
              wait(NULL);
    }
  }

  return 0;
}
```