


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. Write your name and section clearly in the specified space.

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 send a 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 and no error in code

```
#define BUFFER_SIZE 25
#define READ_END 0
#define WRITE_END 1
```

```
int main(void) {
    char msg[BUFFER_SIZE] = "Greetings";
    int size; pid_t pid;
    intfd[2];

    pipe(fd);
```

```
pid = fork();

if (pid > 0) {

    close(fd[READ_END]);

    write(fd[WRITE_END], msg, strlen(msg)+1);

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

    close(fd[WRITE_END]);

} else {
    close(fd[WRITE_END]);

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

    close(fd[READ_END]);

}

return 0;
}
```

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

- Hard-disk drives, main memory, cache, registers
- Registers, main memory, hard-disk drives, cache
- Registers, cache, main memory, hard-disk drives
- Cache, registers, main memory, hard-disk drives

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

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

| | |
|--|--|
| d) OS can be defined as a <ol style="list-style-type: none"> Resource Allocator Control Program User Program Both a and b | e) How is modularity added to the Linux kernel that is typically a monolithic kernel: <ol style="list-style-type: none"> By adding more system calls Through loadable modules By adopting micro-kernel approach in Linux Both a and b |
| f) In the five-state model why would a process move from Running to Ready state? <ol style="list-style-type: none"> The process has terminated The process needs to perform an I/O operation The process' time quantum has expired The process needs to execute a system call | g) A process stack does not contain: <ol style="list-style-type: none"> Function parameters Local variables Return addresses 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 _____

Waiting time for P_1 _____ Waiting time for P_2 _____

Waiting time for P_3 _____ Waiting time for P_4 _____

Average waiting time _____

GANTT CHART BELOW: (Draw a neat one!)

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: _____

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.

```
#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;
}
```