BITS PILANI HYDERABAD CAMPUS
OPERATING SYSTEMS (CS F372)
SEM II 2020-2021
MID SEMESTER EXAMINATION
TYPE – OPEN BOOK MAX. MARKS – 60 DURATION – 90 MINS.
NO. OF SECTIONS – 4, NO. OF QUESTIONS – 20

SECTION: 1 – True/False Type Questions (7 x 1 = 7 Marks)

1. Write whether the following statement is **True or False -**
Trap is a hardware generated interrupt.

Ans. False

2. Write whether the following statement is **True or False -**
Each device driver has a local buffer.

Ans. False

3. Write whether the following statement is **True or False -**
The hardware is in kernel mode at boot time.

Ans. True

4. Write whether the following statement is **True or False** -
Trace listings are used during system boot time.

Ans. False

5. Write whether the following statement is **True or False** -
shmget( ) is a system call used for communication among processes.

Ans. True

6. Write whether the following statement is **True or False** -
System-call interface maintains a table indexed according to the number associated with each system call.

Ans. True

7. Write whether the following statement is **True or False** -
A message queue is created using the function msgcreate( ).

Ans. False

8. Write whether the following statement is **True or False** -
Program counter stores the address of the next instruction to be executed.

Ans. True

9. Write whether the following statement is **True or False** -
The signal <Control + C> should be sent to all threads of a process.

Ans. True

10. Write whether the following statement is **True or False** -
Timer expiration is a synchronous signal.

Ans. False

11. Write whether the following statement is **True or False** -
Asynchronous cancellation is not the default cancellation type as per the Pthreads API.

Ans. True

12. Write whether the following statement is **True or False** -
Starvation can never occur in Preemptive Priority scheduling algorithm.

Ans. False

13. Write whether the following statement is **True or False** -
Round Robin scheduling algorithm gives the minimum average waiting time for any set of processes.

Ans. False

14. Write whether the following statement is **True or False** -
The dispatcher gives control of the CPU to the process selected by the short-term scheduler.

Ans. True

15. Write whether the following statement is **True or False -**
In the many-to-one multithreading model, if one thread makes a blocking system call, the entire process blocks.

Ans. True

16. Write whether the following statement is **True or False** -
In the many-to-many multithreading model, if one thread makes a blocking system call, the entire process blocks.

Ans. False

17. Write whether the following statement is **True or False** -
Peterson's solution satisfies bounded waiting requirement.

Ans. True

18. Write whether the following statement is **True or False** -
Peterson's solution does not satisfy progress requirement.

Ans. False

SECTION: 2 – Short Answer Type (9 x 2 = 18 Marks)

1. If $\alpha = 0.4$, $\tau_0 = 25$ milliseconds, $\tau_4 = 30$ milliseconds, $t_4 = 32$ milliseconds, calculate the value of τ_5. Write only the answer. You do not have to show any calculation.

Ans. 30.8 milliseconds (0.5 marks will be deducted if unit is not mentioned)

2. If $\alpha = 0.3$, $\tau_0 = 20$ milliseconds, $\tau_4 = 24$ milliseconds, $t_4 = 22$ milliseconds, calculate the value of τ_5. Write only the answer. You do not have to show any calculation.

Ans. 23.4 milliseconds (0.5 marks will be deducted if unit is not mentioned)

3. What is the full form of BIOS?

Ans. Basic Input Output System

4. What is the full form of POST?

Ans. Power On Self Test

5. Suppose there is file named as input.txt. The file contains 4 lines of text. The contents of the file are shown below.

jack and jill are friends
friends are fun
friends of friends
be right back

Suppose you run the following command at a Linux terminal assuming that you are currently in the directory where input.txt is present.

grep "friends" input.txt | wc -l
What is the output of this command execution? No explanation required.

Ans. 3

6. Suppose there is file named as input.txt. The file contains 4 lines of text. The contents of the file are shown below.

jack and jill are friends
friends are fun
friends of friends
be right back

Suppose you run the following command at a Linux terminal assuming that you are currently in the directory where input.txt is present.

cat input.txt | wc -l
What is the output of this command execution? No explanation required.

Ans. 4

7. Consider the set of processes shown in the Table below along with their arrival times and CPU burst cycles given in milliseconds. The processes are scheduled using the FCFS algorithm. In case of a tie, the process having lower value of ID is scheduled first. Calculate the average waiting time. Write only the answer. You do not have to show any calculation.

Process	Arrival Time	CPU Burst Cycle (milliseconds)
P1	0	5
P2	1	3
P3	1	4
P4	2	6

Ans. 5.25 milliseconds (0.5 marks will be deducted for not writing unit)

8. Consider the set of processes shown in the Table below along with their arrival times and CPU burst cycles given in milliseconds. The processes are scheduled using the FCFS algorithm. Calculate the average waiting time. Write only the answer. In case of a tie, the process having lower value of ID is scheduled first. You do not have to show any calculation.

Process	Arrival Time	CPU Burst Cycle (milliseconds)
P1	0	4
P2	0	6
P3	3	7
P4	4	5

Ans. 6 milliseconds (0.5 marks will be deducted for not writing unit)

9. If the nice value of a process is -10, calculate the priority of the process. Write only the answer. You do not have to show any calculation.

Ans. 10

10. If the nice value of a process is -15, calculate the priority of the process. Write only the answer. You do not have to show any calculation.

Ans. 5

11. If the nice value of a process is -5, calculate the priority of the process. Write only the answer. You do not have to show any calculation.

Ans. 15

12. Suppose process P1 wants to send a message *m* to process P2. P1 and P2 share a mailbox *X*.
P1 uses the following primitive: ***send(X, m)***. What kind of communication has been used here by P1? Write only the answer. No explanation required.

Ans. Indirect Communication

14. Suppose process P1 wants to send a message *m* to process P2.
P1 uses the following primitive: ***send(P2, m)***
What kind of communication has been used here by P1? Write only the answer. No explanation required.

Ans. Direct Communication

15. Suppose a computing environment uses the Round Robin scheduling algorithm. There are 20 processes in the ready queue and the time quantum duration is 10 milliseconds. What is the maximum amount of time for which a process may have to wait until its next time quantum? Assume that no process finishes its CPU burst in one time quantum. Write only the answer. You do not have to show any calculation.

Ans. 190 milliseconds (0.5 marks will be deducted for not writing unit)

16. Suppose a computing environment uses the Round Robin scheduling algorithm. There are 15 processes in the ready queue and the time quantum duration is 20 milliseconds. What is the maximum amount of time for which a process may have to wait until its next time quantum? Assume that no process finishes its CPU burst in one time quantum. Write only the answer. You do not have to show any calculation.

Ans. 280 milliseconds (0.5 marks will be deducted for not writing unit)

17. Consider the solution to the Producer-Consumer problem that we have studied in Synchronization. As you are aware that this solution uses a shared variable counter that is used by both the Producer and the Consumer processes. You may refer to your class notes/book for having a look at the pseudocode. For this problem, assume BUFFER_SIZE = 15. The rest of the producer-consumer code is same as that we had discussed during lecture.
Suppose that there is only one producer process and one consumer process. Both processes are executing concurrently. Currently, counter = 8. At the machine language level, counter++ and counter-- are implemented in the same manner that we have discussed in class. Producer uses register1 and consumer uses register2. Now consider the following sequence of interleaved machine-level instructions shown in the Table below.
What is the final value of the counter variable?
Only write the answer. You do not have to show any calculation or give any explanation.

Time Instant	**Execution**
T_0	consumer executes register2 = counter
T_1	consumer executes register2 = register2 - 1
T_2	producer executes register1 = counter
T_3	consumer executes counter = register2
T_4	producer executes register1 = register1 + 1
T_5	producer executes counter = register1

Ans. 9

18. Consider the solution to the Producer-Consumer problem that we have studied in Synchronization. As you are aware that this solution uses a shared variable counter that is used

by both the Producer and the Consumer processes. You may refer to your class notes/book for having a look at the pseudocode. For this problem, assume BUFFER_SIZE = 20. The rest of the producer-consumer code is same as that we had discussed during lecture.
Suppose that there is only one producer process and one consumer process. Both processes are executing concurrently. Currently, **counter = 15**. At the machine language level, counter++ and counter-- are implemented in the same manner that we have discussed in class. Producer uses register1 and consumer uses register2. Now consider the following sequence of interleaved machine-level instructions shown in the Table below.
What is the final value of the counter variable?
Only write the answer. You do not have to show any calculation or give any explanation.

Time Instant	Execution
T_0	producer executes register1 = counter
T_1	consumer executes register2 = counter
T_2	consumer executes register2 = register2 - 1
T_3	producer executes register1 = register1 + 1
T_4	producer executes counter = register1
T_5	consumer executes counter = register2

Ans. 14

19. How many child processes are created by the following C code snippet? Assume all relevant header files are included and all function calls are executed successfully. Only write the answer. You do not have to give any explanation.

```
int main(){
   for(int i = 1; i <= 9; i++){
      if(i % 3 == 0){
         fork( );
      }
   }
   return 0;
}
```

Ans. 7

20. How many child processes are created by the following C code snippet? Assume all relevant header files are included and all function calls are executed successfully. Only write the answer. You do not have to give any explanation.

```
int main(){
   for(int i = 0; i <= 7; i++){
      if(i % 2 != 0){
         fork( );
       }
   }
   return 0;
}
```

Ans. 15

SECTION: 3 – Descriptive Type Questions

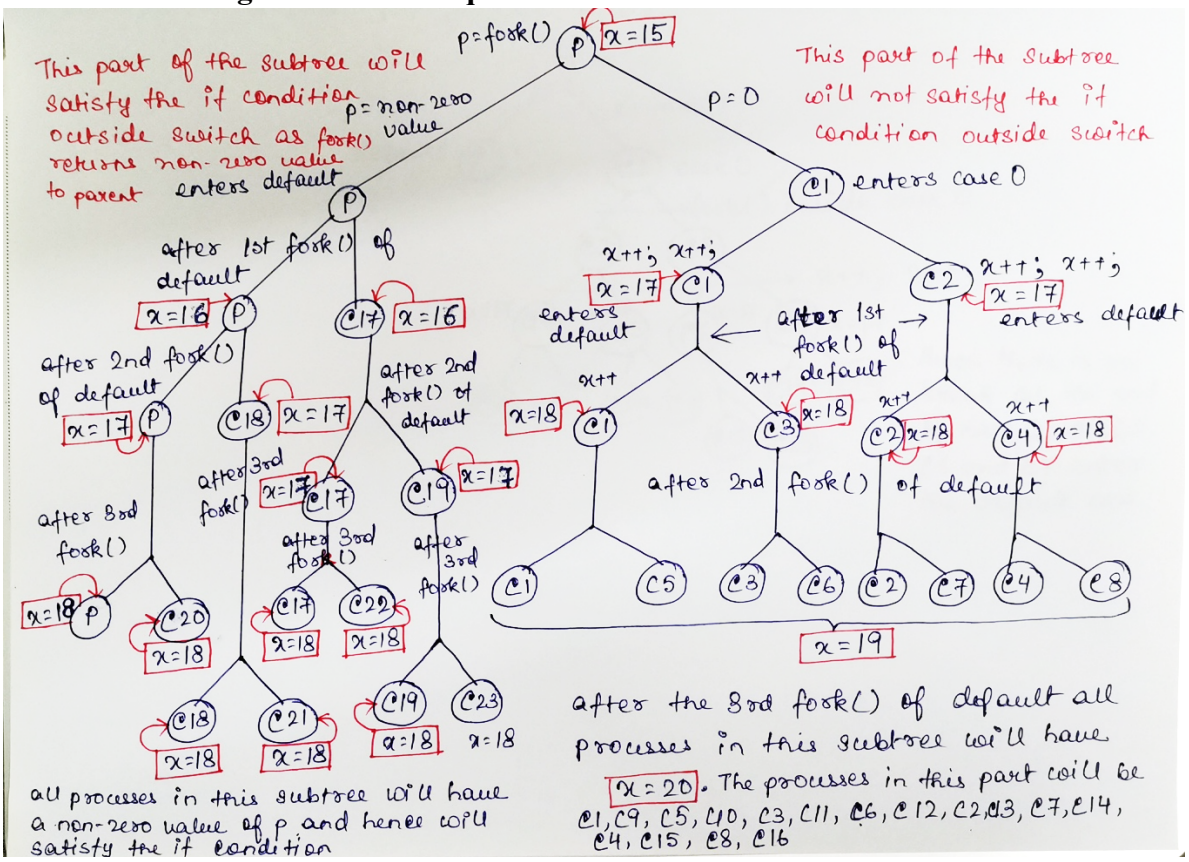
1. What is the output of the following C code snippet? Explain with proper justification the reason behind obtaining the output. If proper justification is not given, only partial marks will be awarded. Also, explain clearly how many child processes are created and why. If you simply write the total number of child processes created, partial marks will be awarded. No diagram is required. Assume all requisite header files are included and all system calls execute successfully. **[2 + 4 = 6 MARKS]**

```
int main()
{
    pid_t p;
    int x = 15;
    p = fork();
    switch(p){
        case 0:
            fork();
            x++;
            x++;
        default:
            fork();  x++;
            fork();  x++;
            fork();  x++;
    }
    if(p) printf("\n%d\n", x);
    return 0;
}
```

Ans. The output is 18 printed 8 times. A total of 23 child processes are created.

Since fork() returns zero to the child process, so the 1st child process will enter case 0. The fork() inside case 0 will create one more child process and the value of x for both child processes will be incremented to 17. As there is no break statement inside case 0, so these 2 child processes will enter the default case. Inside the default case there are 3 more fork() invocations and hence 16 child processes will be created and value of x for these processes will be 20.
fork() returns the non-zero pid of the child process to the parent. Hence, the parent process will enter the default case. The 3 fork() invocations inside default will lead to the creation of 7 more child processes. For these child processes and the parent, the value of x will be 18.
For these child and the parent processes, the if condition outside the switch will be satisfied and hence 18 will be printed 8 times.

Note that this diagram was not required in the exam.



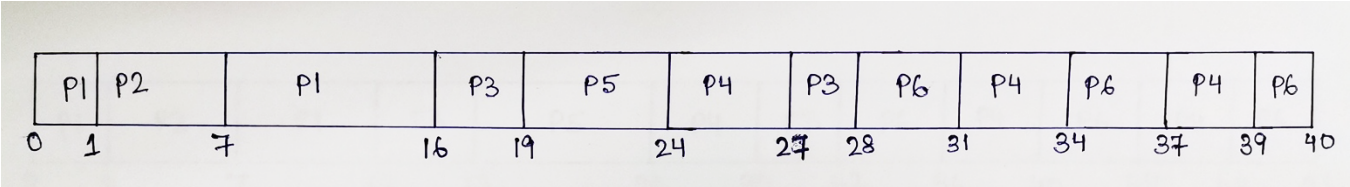
Note – 1 mark for the output and 1 mark for the no. of child processes. For explanation, 2 marks for the output explanation and 2 marks for the explanation for the no. of child processes created.

2. Consider a system implementing multilevel queue scheduling algorithm for scheduling processes. The ready queue on this system is partitioned into two queues, Q1 and Q2. Processes in Q1 are scheduled using SRTF algorithm and the ones in Q2 are scheduled using Round-Robin scheduling algorithm with a time quantum duration of 3 milliseconds. Moreover, processes arriving in Q1 have higher priority than processes arriving in Q2. Consider the six processes - P1, P2, P3, P4, P5 and P6. The associated queue, arrival times and CPU burst cycles (in milliseconds) of the processes are shown below in the Table below. The second column of the table specifies the queue in which each process arrives. Calculate the average waiting time using the concepts of multilevel queue scheduling that have been taught in the lectures. Your answer should contain a proper Gantt Chart and a detailed calculation. You should clearly show the individual waiting time of each of the processes. **DO NOT DRAW 2 SEPARATE GANTT CHARTS. DRAW ONLY A SINGLE GANTT CHART.** For this question, you are allowed to upload files. **[10 MARKS]**

Process	Queue	Arrival Time	CPU Burst cycle (in milliseconds)
P1	Q1	0	10
P2	Q1	1	6
P3	Q2	0	4

Process	Queue	Arrival Time	CPU Burst cycle (in milliseconds)
P4	Q2	2	8
P5	Q1	19	5
P6	Q2	20	7

Ans.



Waiting time of P1 → (0 – 0) + (7 – 1) = 6 milliseconds
Waiting time of P2 → (1 – 1) = 0 milliseconds
Waiting time of P3 → (16 – 0) + (27 – 19) = 24 milliseconds
Waiting time of P4 → (24 – 2) + (31 – 27) + (37 – 34) = 29 milliseconds
Waiting time of P5 → (19 – 19) = 0 milliseconds
Waiting time of P6 → (28 – 20) + (34 – 31) + (39 – 37) = 13 milliseconds

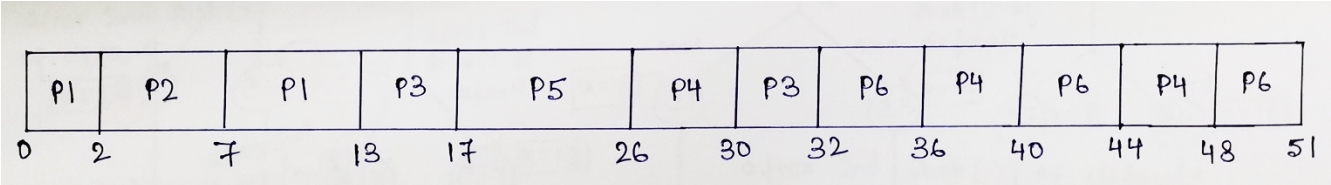
Average waiting time = 72/6 = 12 milliseconds

Note – If unit is not mentioned, 0.5 marks will be deducted. 3 marks for Gantt chart, 1 mark each for individual waiting time and 1 mark for the final answer.

3. Consider a system implementing multilevel queue scheduling algorithm for scheduling processes. The ready queue on this system is partitioned into two queues, Q1 and Q2. Processes in Q1 are scheduled using SRTF algorithm and the ones in Q2 are scheduled using Round-Robin scheduling algorithm with a time quantum duration of 4 milliseconds. Moreover, processes arriving in Q1 have higher priority than processes arriving in Q2. Consider the six processes - P1, P2, P3, P4, P5 and P6. The associated queue, arrival times and CPU burst cycles (in milliseconds) of the processes are shown below in the Table below. The second column of the table specifies the queue in which each process arrives. Calculate the average waiting time using the concepts of multilevel queue scheduling that have been taught in the lectures. Your answer should contain a proper Gantt Chart and a detailed calculation. You should clearly show the individual waiting time of each of the processes. **DO NOT DRAW 2 SEPARATE GANTT CHARTS. DRAW ONLY A SINGLE GANTT CHART.** For this question, you are allowed to upload files. **[10 MARKS]**

Process	Queue	Arrival Time	CPU Burst cycle (in milliseconds)
P1	Q1	0	8
P2	Q1	2	5
P3	Q2	5	6
P4	Q2	6	12
P5	Q1	17	9
P6	Q2	19	11

Ans.



Waiting time of P1 → (0 – 0) + (7 – 2) = 5 milliseconds
Waiting time of P2 → (2 – 2) = 0 milliseconds
Waiting time of P3 → (13 – 5) + (30 – 17) = 21 milliseconds
Waiting time of P4 → (26 – 6) + (36 – 30) + (44 – 40) = 30 milliseconds
Waiting time of P5 → (17 – 17) = 0 milliseconds
Waiting time of P6 → (32 – 19) + (40 – 36) + (48 – 44) = 21 milliseconds

Average waiting time = 77/6 = 12.833 milliseconds

Note – If unit is not mentioned, 0.5 marks will be deducted. 3 marks for Gantt chart, 1 mark each for individual waiting time and 1 mark for the final answer.

SECTION: 4 – Descriptive Type Questions

1. Write a C program to implement the following scenario. **Note that your program should make use of the Pthreads API and should execute on a Linux based operating system. Also, you are not allowed to use any global variable or data structure. All requisite header files have to be included in your program. You should also check if the function calls for pipe and child creation execute successfully or not. [15 MARKS]**

The parent process creates a child process. The parent process also creates an ordinary pipe for communicating with the child process. The parent process takes a multiword string as input from the user and sends it to the child process via the pipe. The individual words of the string are separated by whitespace characters and the last word is not followed by a whitespace character. **Note that you are not allowed to hard-code this string in your program without taking it as user input.** The child process after receiving the multiword string via the pipe, prints it in the reverse order. **Closing the proper ends of the pipe should be done appropriately. Note that the child process prints the entire string in reverse order, not just the individual words.** For eg., If the string is *today is sunny*, the child process prints *ynnus si yadot*. After printing the string, the child process creates two threads. **Note that the same thread attribute object cannot be used for both the threads. Also, thread attributes initialization and thread creation should be done properly.** The child process supplies a value, say x, to each thread and the thread prints from x to $x + 10$ inside a loop. Between two successive iterations of the loop, the thread sleeps for two seconds. For eg., If a thread is supplied with 10, it prints 10 11 12 13 14 15 16 17 18 19 20. **Note that each thread is supplied with a different value. It is not necessary to take these values as inputs from the user. Note that you are not allowed to write two different functions defining the tasks to be carried out by the two threads.** For each of the threads, the child process prints whether the thread has local or global contention scope along with the ID of the thread. The child process also waits for the two threads to terminate. **You cannot make the child process wait using sleep( ).** The parent process waits for the child process to terminate. **You cannot make the parent process wait using sleep( ).** Finally, the parent process deletes a file named as abc.txt using a

single line of code. Assume that abc.txt is present in the same directory where your program is stored. **Note that you are not allowed to open, read or write to abc.txt in your program.**

Ans.

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

#define READ 0
#define WRITE 1

void *runner(void *param){
    int *p = (int *)param;
    printf("\nThe series is: \n");
    for(int i = *p; i <= *p + 10; i++){
        printf(" %d", i);
        sleep(2);
    }
    pthread_exit(0);
}

int main()
{
    pid_t child;
    int fd[2];
    int p = pipe(fd); //pipe between child and parent
    if(p == -1){
        printf("\nFailed to create pipe");
        return 1;
    }

    child = fork();
    if(child < 0){
    printf("\nFailed to create Child");
    return 1;
    }
    else if(child == 0){ //code for child
        char read_msg[1000];
        close(fd[WRITE]);
        read(fd[READ], read_msg, 1000);
        printf("\nChild printing string in reverse\n");
        int length = strlen(read_msg);
        for(int i = length - 1; i >= 0; i--)
            printf("%c", read_msg[i]);

        int arr[2] = {10, 25};
        pthread_t tid[2];
        pthread_attr_t attr[2];
        for(int i = 0; i < 2; i++)
```

```
            pthread_attr_init(&attr[i]);

        for(int i = 0; i < 2; i++)
            pthread_create(&tid[i],&attr[i],runner,&arr[i]);
        int scope;
        for(int i = 0; i < 2; i++){
            pthread_attr_getscope(&attr[i], &scope);
            if(scope == PTHREAD_SCOPE_PROCESS)
                printf("\n%lu PTHREAD_SCOPE_PROCESS\n", tid);
            if(scope == PTHREAD_SCOPE_SYSTEM)
            printf("\n%lu PTHREAD_SCOPE_SYSTEM\n", tid);
        }
        for(int i = 0; i < 2; i++)
            pthread_join(tid[i], NULL);
    }
    else{ //code for parent
        char write_msg[1000];
        printf("\nEnter to parent the multiword string:");
        scanf("%[^\n]%*c", write_msg);
        close(fd[READ]);
        write(fd[WRITE], write_msg, strlen(write_msg)+1);
        close(fd[WRITE]);
        wait(NULL);
        execlp("/bin/rm", "rm", "abc.txt", NULL);
    }
    return 0;
}
```

2. Two processes, P1 and P2, need to access the critical section. The pseudo codes of P1 and P2 are shown below as a solution to the critical section problem. Here, entry1, entry2 and lock are shared boolean variables. All the three variables are initialized to false.

P1 pseudo code

```
while(true){
    entry1 = true;
    while(lock == true && entry2 == true);
    lock = true;
    //CRITICAL SECTION
    entry1 = false;
    lock = false;
}
```

P2 pseudo code

```
while(true){
    entry2 = true;
    while(lock == true && entry1 == true);
    lock = true;
    //CRITICAL SECTION
    entry2 = false;
    lock = false;
}
```

Does this solution to the critical section problem satisfy mutual exclusion? Give proper explanation. **[4 MARKS]**

Ans. This solution does not satisfy mutual exclusion.
If one process is in the critical section, then this solution prevents the other process to enter the critical section. Say P1 is in the critical section. Then, entry1 = true and lock = true. Now, if P2 wants to enter the critical section, P2 will be stuck in the while loop of its entry section code. However, if both P1 and P2 try to simultaneously enter the critical section, this solution fails. Let us assume that both P1 and P2 are executing exactly at the same speed. In that case, P1 sets entry1 to true and checks the while loop condition. P2 also sets entry2 to true and checks the while loop condition. Since lock = false, both P1 and P2 will break out of the while loop and set lock to true and enter the critical section at the same time thus violating the mutual exclusion requirement. Even if P1 and P2 are not executing at the same speed, mutual exclusion will be violated. Say P1 set entry1 = true and P2 sets entry2 = true. Now, say P1 checks the while condition first and breaks out of it since lock is false. Before P1 can set lock = true, if P2 checks the while condition, P2 will find lock = false and will also break out of the while loop. Thus, both processes enter the critical section.

Note – 1 mark for writing yes or no and 3 marks for the explanation.