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:	12th April 2019	Weight	15
Section:	ALL	Page(s):	3
Exam Type:	Mid 2		

Student : Name:_____ Roll No._____ Section:_____

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

Q1: Choose the correct answer. [10 points]

1. Identify the one which is not an advantage of threads
   a. Is lightweight
   b. Increases efficiency
   c. Allows scalability
   d. **Enables multiprocessing on uniprocessor system**

2. Threads within a process do not share
   a. Data
   b. Files
   c. **Registers**
   d. code

3. It is possible to achieve concurrency with one processor but it is not possible to achieve parallelism with one processor.
   a. **True**
   b. False

4. Which of the following multithreading models is efficient as well as avoids blocking issues
   a. One to many
   b. **Many to many**
   c. Many to one

5. The state where 1 process out of total 4 processes is unable to acquire CPU is known as:
   a. Deadlock
   b. **Starvation**
   c. Priority inversion
   d. Bounded Waiting

6. The `pthread_join(workerthread, NULL)` does the following:
   a. `workerThread` waits for calling thread
   b. **calling thread waits for `workerThread`**
   c. calling and `workerThread` execute independently
   d. Calling and `workerThread` implement mutual exclusion

7. Which of the following is not a condition for synchronization problems solution
   a. **Relative speed**
   b. Mutual Exclusion
   c. Progress
   d. Bounded waiting

8. Peterson solution is considered as the primitive solution to synchronization problems because
   a. It lacks synchronization capabilities
   b. Uses too much memory
   c. **Only provides synchronization between two processes**
   d. Does not allow *progress*

9. Which of the following is not a solution to synchronization problems
   a. Monitors
   b. **Using `block()` and `wait()` operations**
   c. Mutex
   d. Semaphore

10. Which of the following is a not an advantage of semaphores
    a. Solving synchronization problem
    b. Signaling
    c. Resource utilization management
    d. **Efficient hardware utilization**

Q 02: [a – 6 points] Consider the following concurrently executing processes. Synchronize them using semaphores to generate the following infinite string: "yesyesyesyes...". Your solution should use the minimum possible number of semaphores. Please add statements to the code below without modifying any existing statements.

Initialize: semaphore s1 = 1, s2 = 0, s3 = 0;

Process 1	Process 2	Process 3
while(true)	while(true)	while(true)
{	{	{
wait(s1);	wait(s2);	wait(s3);
cout << "y";	cout << "e";	cout << "s";
signal(s2);	signal(s3);	signal(s1);
}	}	}

Q 02: [b – 4 points] Consider the following set of semaphores and their initialization values:

Semaphore car-avail = 0, car-taken=0; car-filled=0;

Assume that the **implementation of semaphores uses waiting queues instead of busy waiting**.

Your job is to keep track of the number of processes waiting in the queue for each semaphore. Given the values above and the statements given in the table below, fill the fields (car-avail, car-taken, car-filled) with integer values such that the number of processes waiting in the waiting queues for each semaphore are demonstrated.

Note: For each column, fill the cells with integers where there is a change in value ONLY. Leave a "-" otherwise

Statements	car-avail	car-taken	car-filled
Wait (car-avail)	-1	0	0
Wait (car-avail)	-2	0	0
Signal(car-avail)	-1	0	0
Signal(car-avail)	0	0	0
Signal(car-taken)	0	1	0
Wait(car-filled)	0	1	-1
Wait(car-taken)	0	0	-1
Wait(car-taken)	0	-1	-1

Q 03 [4+6 points]: There are two code segments given in this question. Part A whose output has to be provided is given in the left box. Part B has to be answered in the right box containing the question.

Question statement Part B:
The following program finds the maximum value in an array in parallel. For simplicity, size of the array and number of threads are chosen as 100 and 4 respectively. Assume that the size of the array is perfectly divisible by

the number of threads. Also, assume that there is no compilation error. The code for #include statements and the init() function is not provided for brevity. The init() function initializes elements of the array to random values between 1 and 1000. The code is executing on a laptop with 8 idle cores.

Main requirements: The program must divide the work between the 4 threads and they must run in parallel.

Your job here is to find and fix a subtle benign bug in the code that is hindering the code from executing efficiently and meeting main requirements mentioned above. Just modify the code clearly in the box below. No need to rewrite. Hard-coded line modifications will be penalized. (6 marks)

(Part A) Provide output at the bottom

```c
#include <pthread.h>
#include <stdio.h>
int value = 0;

void *runner(void *param); /* the thread */

int main(int argc, char *argv[]) {

  int pid;
  pthread_t tid;
  pid = fork();

  if (pid == 0) {

    pthread_create(&tid, NULL, runner, NULL);
    pthread_join(tid, NULL);
    printf("CHI: val = %d \n", value); /*LINE C*/

  } else if (pid > 0) {

    value -= 5;
    wait(NULL);
    printf("PAR: val = %d \n", value); /*LINE P*/

  }
}

void *runner(void *param) {
        value += 5;
        pthread_exit(0);
}
```

Write the output of the code above at **LINES C** and **P** in this box. (4 marks)

Output at LINE C = _____5_____

Output at LINE P = _____-5_____

(Part B)

```c
#define SIZE 100
#define THREADS 4

int retval[THREADS];

int buffer[SIZE];
void* max_find (void*);
void init (int *buffer);

int main() {
    init (buffer);
    pthread_t tid[THREADS];

    for(int i = 0; i < THREADS; i++)
        pthread_create (&tid[i], NULL, max_find, (void*) i);

    for(int i = 0; i < THREADS; i++)
        pthread_join (tid[i], NULL);

    int max_val = retval[0];
    for(int i=1; i < THREADS; i++)
        if(retval[i] > max_val)
            max_val = retval[i];

    printf("The maximum value is %d\n\n", max_val);
    return 0;
}

void* max_find (void* args) {
    int myid = (int) args;
    int start = 0;  //int start = mynum * SIZE/THREADS;
    int end = SIZE;  //int end = start + SIZE/THREADS;
    int localmaxval = buffer[start];

    for(int i = start; i < end; i++)
        if(buffer[i] > localmaxval)
            localmaxval = buffer[i];
    retval[myid] = localmaxval;
}
```