AIN SHAMS UNIVERSITY FACULTY OF ENGINEERING Computer Engineering Department/CHEP		
September, 20th, 2020	Course Code: CSE345/CSE347	*Time: 2 hours*

Real Time and Embedded System Design; Final Exam

The Exam Consists of 4 Questions, in 4 pages (ONE BLANK PAGE)	*Total Marks: 40% Marks*	1/4

Student Name: Marks:

Examined ILOs

1. Illustrate the technical background of embedded systems
2. Demonstrate knowledge of the different architectures of embedded systems
3. Demonstrate understanding of the embedded system design methodologies
4. ~~Outline embedded system design kits (done in course project)~~
5. Demonstrate understanding of design skills with embedded systems
6. Program embedded systems
7. ~~Demonstrate understanding of embedded systems interfacing (done in course project)~~

Question 1

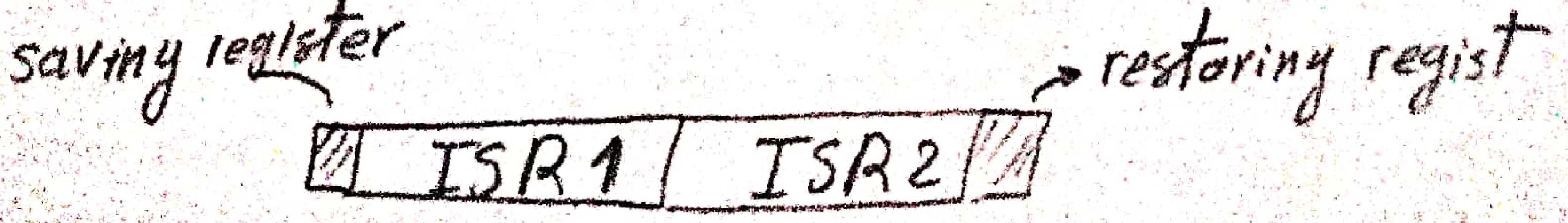
A) What is the most difficult challenge in Embedded System Design?
B) What is the difference between a Binary Semaphore and a Mutex object?
C) What is the difference between foreground and background task?
D) What is the technique called tail-chaining in Interrupts? What does it provide?

A) The most difficult challenge is to operate multiple task on one cpu

B) semaphore: must be given before being take if not it will be block
mutex : it can be taken normally if it not given

C) foreground : where schedular lies and it used to control & organize the entry of tasks to cpu
Background : where the task & its defination lies

D) Tail chaining : happen when theres no need to save mult as getting in reg = getting out register

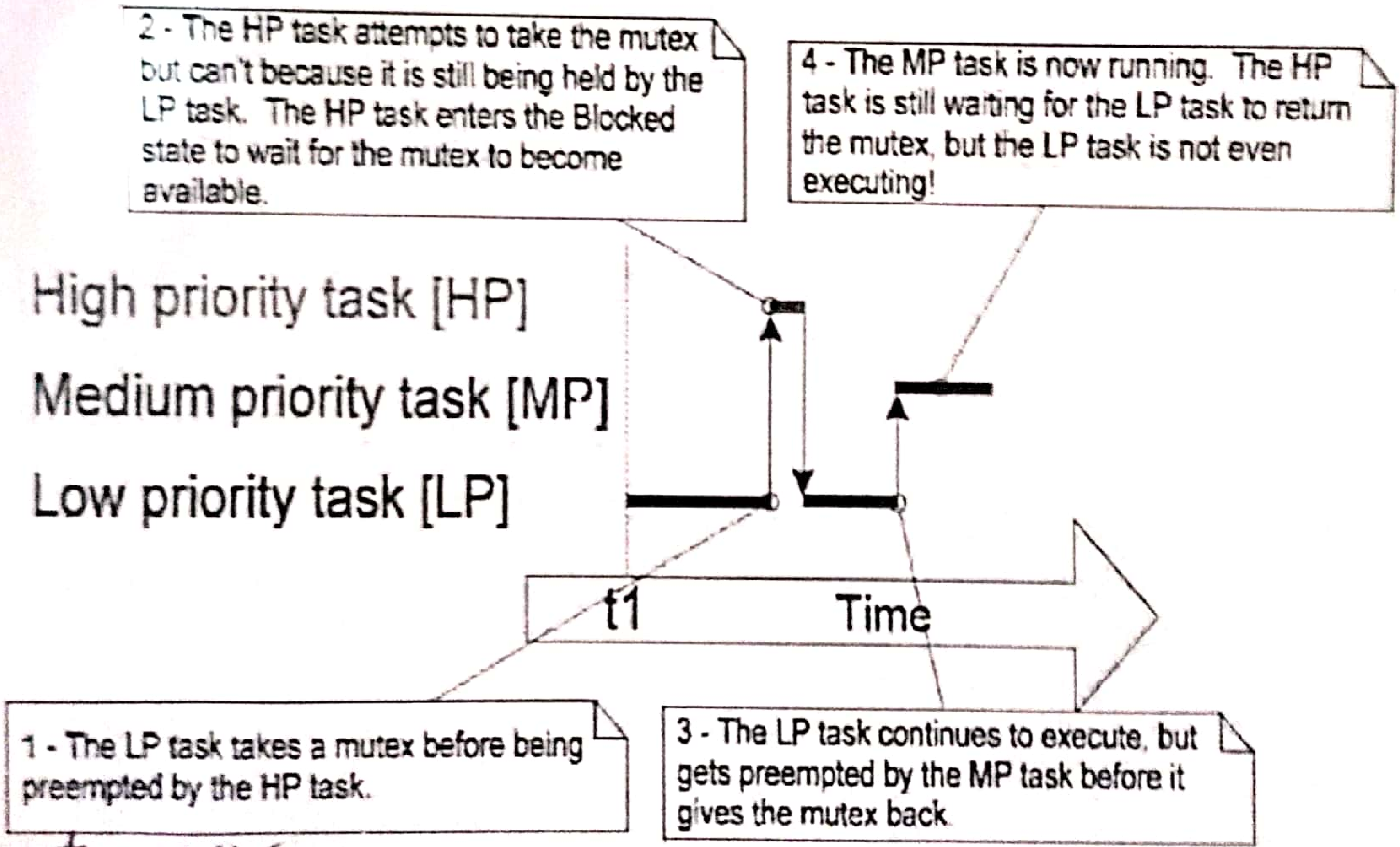


Scanned with CamScanner

AIN SHAMS UNIVERSITY **FACULTY OF ENGINEERING** **Computer Engineering Department/CHEP**		
September, 20th, 2020	Course Code: CSE345/CSE347	***Time: 2 hours***
Real Time and Embedded System Design; Final Exam		
The Exam Consists of 4 Questions, in 4 pages (ONE BLANK PAGE)	***Total Marks: 40% Marks***	**2/4**

Question 2

Write a FreeRTOS based application that could achieve this descriptive timing diagram:



2 - The HP task attempts to take the mutex but can't because it is still being held by the LP task. The HP task enters the Blocked state to wait for the mutex to become available.

4 - The MP task is now running. The HP task is still waiting for the LP task to return the mutex, but the LP task is not even executing!

1 - The LP task takes a mutex before being preempted by the HP task.

3 - The LP task continues to execute, but gets preempted by the MP task before it gives the mutex back

```
int main (){
        xmutex = xsemaphorecreatemutex();
        vtaskcreate(LP_task, "LP", null, 1, null);
         vtaskstartschedular();
         }
         void LP_task(){
                for( ; ; ){
                    xsemaphoretake(xmutex);
                    vtaskcreate(HP_task, "Hp", null, 3, null);
                    for(i=0; i<1000000; i++);
                    vtaskcreate(MP_task, "MP", null, 2, null);
                    xsemaphore Give(xmutex);
                    }
```

```
void MP_task(){
    for(;;){
        }
    }
```

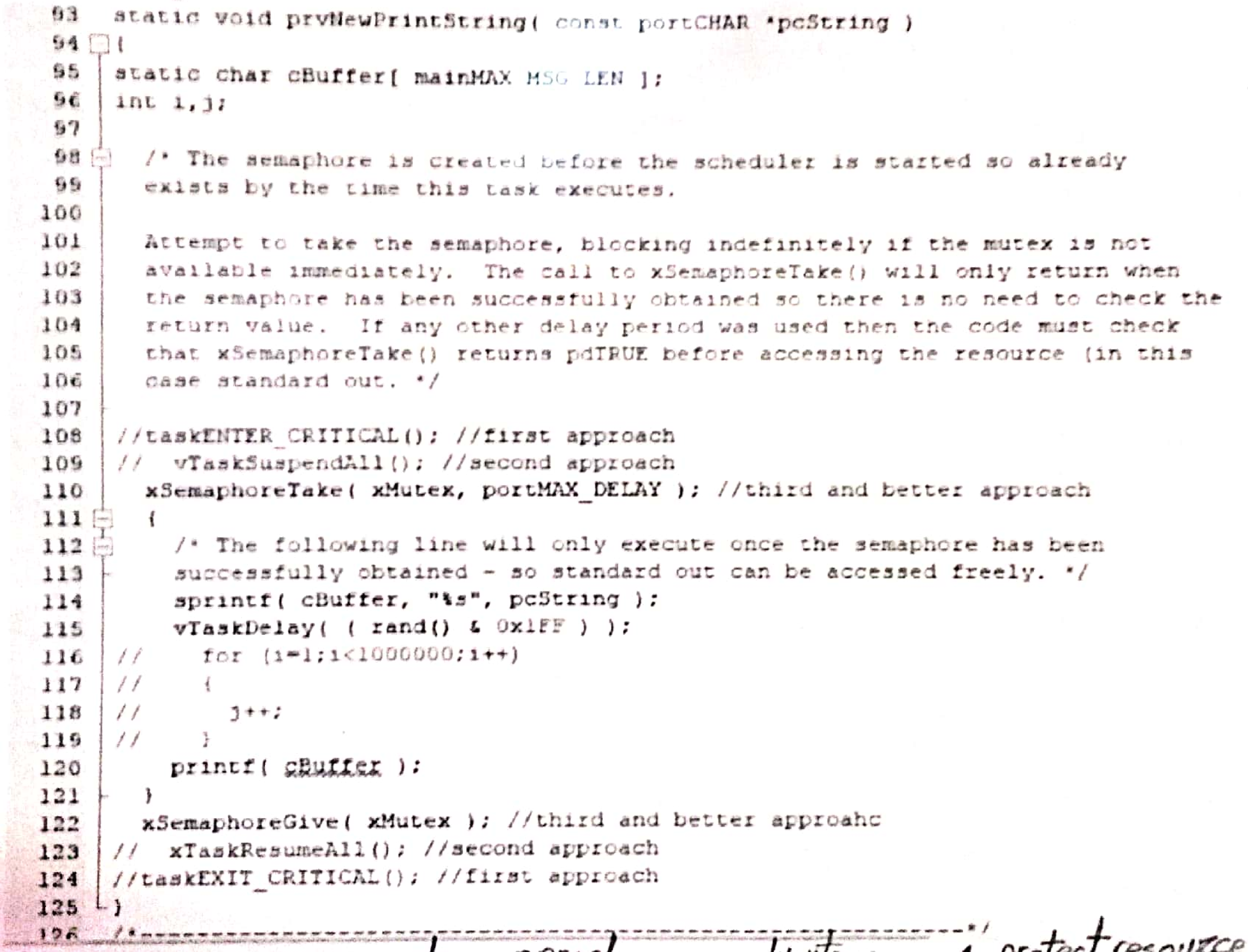
Prof. Omer Alkelany

```
void HP_task(){
 for(;;){
   xsemphore mutex(xmutex,portmaxdl);
   }
}
```

Scanned with CamScanner

Question 3 (10 Marks)

The figure below is a snap shot showing different approaches of implementing the critical section. Compare the three approaches, in terms of their advantages and disadvantages.



```
static void prvNewPrintString( const portCHAR *pcString )
{
static char cBuffer[ mainMAX_MSG_LEN ];
int i,j;

    /* The semaphore is created before the scheduler is started so already
    exists by the time this task executes.

    Attempt to take the semaphore, blocking indefinitely if the mutex is not
    available immediately.  The call to xSemaphoreTake() will only return when
    the semaphore has been successfully obtained so there is no need to check the
    return value.  If any other delay period was used then the code must check
    that xSemaphoreTake() returns pdTRUE before accessing the resource (in this
    case standard out. */

//taskENTER_CRITICAL(); //first approach
//   vTaskSuspendAll(); //second approach
    xSemaphoreTake( xMutex, portMAX_DELAY ); //third and better approach
    {
        /* The following line will only execute once the semaphore has been
        successfully obtained - so standard out can be accessed freely. */
        sprintf( cBuffer, "%s", pcString );
        vTaskDelay( ( rand() & 0x1FF ) );
//      for (i=1;i<1000000;i++)
//      {
//          j++;
//      }
        printf( cBuffer );
    }
    xSemaphoreGive( xMutex ); //third and better approahc
//   xTaskResumeAll(); //second approach
//taskEXIT_CRITICAL(); //first approach
}
/*-----------------------------------------------------------*/
```

Mutex approch → advantage → 1- protect resource
→ 2. doesnt suspend the schedula
→ disad - deadlock can happen if The High critical task want to execute

Critical → adv → No task or interupt can premept
→ disadvantge → all interupt is being stopped

suspend approch → advantage → No task pemeption could interupt during resource manage
→ disadvantge → schedular stop

Scanned with CamScanner

AIN SHAMS UNIVERSITY
FACULTY OF ENGINEERING
Computer Engineering Department/CHEP

September, 20th, 2020	Course Code: CSE345/CSE347	Time: 2 hours

Real Time and Embedded System Design; Final Exam

The Exam Consists of 4 Questions, in 4 pages (ONE BLANK PAGE)	Total Marks: 40% Marks	4/4

Question 4

Explain what can possibly go wrong when this **pseudo code** is executed. How can it be changed to avoid such problem?

```
data_type buffer[N];
int count = 0;
void TaskA() {
  int i;  → int i = 0;
  while( 1 ) {
    produce(&data);
    while( count == N );/*loop*/
    buffer[i] = data;
    i = (i + 1) % N;
    count = count + 1;
  }
}
void TaskB() {
  int i;  → int i=0
  while( 1 ) {
    while( count == 0 );/*loop*/
    data = buffer[i];
    i = (i + 1) % N;
    count = count - 1;
    consume(&data);
  }
}
void main() {
  Task_create (TaskA);
  Task_creat(TaskB);
}
```

Prof. Omer Alkelany

Scanned with CamScanner