1. List and explain the two (2) primary benefits of parallel processing.

* **Increased Reliability:** Failing processors can push their workload to other processors to take over the processes. This requires careful designing to ensure the remaining processor(s) don’t become overloaded with the now extra work.
* **Faster Processing:** Jobs can be divided up among the multiple processors to increase processing speed.

1. Answer the following questions about the three (3) levels of multiprocessing:
   1. What is the name of each level?

1. **Job Level**

2. **Process Level**

3. **Thread Level**

* 1. Explain how processor assignment differs between each level.

1. **Job Level:** A job has its’ own processor and threads and processes get the same processor.

2. **Process Level:** This level allows unrelated processes to given to available processors.

3. **Thread Level:** Threads can now be assigned to available processors regardless of job.

* 1. Explain the synchronization required in each level.

1. **Job Level:** No explicit sync required when jobs are assigned to a processor.

2. **Process Level:** Some sync is required.

3. **Thread Level:** More precise sync is required to track all processes and threads.

1. List the advantages and disadvantages of the three (3) multiprocessing configurations:
   1. Master/Slave

Very simple implementation.

Bad Reliability: If the master processor fails, the entire system fails.

If the master processor is busy, free slaves must wait for more work to be assigned.

Increased Interrupts as slaves will occasionally need operating system intervention.

* 1. Loosely Coupled

Not prone to system wide failure as even if a processor fails others can work independently.

Tables are necessary to indicate where each job is located for processors as jobs stick with the same processor.

* 1. Symmetric

More reliable

Resources used more effectively

Balances loads well

Degrades gracefully

Difficult to Implement: Processes must be synchronized well to avoid deadlocks.

1. For each scenario, assume each block of code is executed by **different** processors (**P1** or **P2**) that are working **concurrently**without synchronization. Furthermore, each block of code refers to non-shared variables (*a* or *b*) and a shared variable (*x*). Finally, if P1 and P2 execute their respective blocks of code only **once**, at any **speed**, determine all of the possible values of the shared variable **x** (assume x is initialized to 0).

Scenario 1

Processor P1 Processor P2

for(a = 1; a <=3; a++) for(b = 1; b <=3; b++)

x = x + 1; x = x + 1;

The only possible value would be 6. It doesn’t matter which processor runs first

Or when because x is only incremented and there is no code that resets it here.

Scenario 2  
Processor P1 Processor P2

x = 0; x = 0;

for(a = 1; a <=3; a++) for(b = 1; b <=3; b++)

x = x + 1; x = x + 1;

The first possible value would be 3 because it’s possible that one of the processors

Executes their code after the other finishes, which would reinitialize x to 0 and

Loop it back to 3. IF, however, both processors start at the same time, the second

Possible value would be six. X would be initialized to 0 in both codes so it won’t be

reset again.

1. Answer the following questions about the **critical region**
   1. What is the critical region?

Important parts of programs which must be treated as units as integrity can be ruined if not.

* 1. What is the primary difference between the code within the critical region and the rest of a program?
  2. Describe how each of the following locking mechanisms can be used to control access to the critical region
     1. Test-and-Set

A single bit can be either a 0 for free or 1 for busy. The subprogram for test and set will give either a free or busy condition code based on this bit to either allow or block a process to touch the critical region. This can cause starvation if too many processes are waiting to enter the critical region.

* + 1. WAIT and SIGNAL

Modification of Test-and-set which adds two news operations: WAIT and SIGNAL. WAIT is placed onto a process encountering a busy code. SIGNAL is activated when a process exits the critical region and a queue then selects a process with WAIT on it to enter the critical region.

* + 1. Semaphores

A non-negative integer variable is used as a flag. Two operations V and P, increase and decrease the variable respectively with the following formulas:

V(s): s: = s + 1

P(s): If s > 0 then s: = s – 1

If s = 0, the critical region is busy and processes calling on the test must wait.

* 1. What would happen if access to the critical region was not controlled using one of the above locking mechanisms?

Integrity of the code would be in jeopardy, and jobs would constantly be starving waiting on critical resources.

1. Answer the following questions about **producers** and **consumers**
   1. What are the two (2) **extremes** of a producer/consumer system?

Adding to bins that are already full, and consumers trying to collect resources from an empty bin.

* 1. What is likely to happen if there are more producers than consumers?

Producers will create a surplus of resources and cause a buffer overflow.

* 1. What is likely to happen if there are more consumers than producers?

Consumers will use up more resources than the producers can create, causing starvation.

* 1. Can a **deadlock** occur if **only** one (1) producer and one (1) consumer exist in the system?

No, it’s a matter of FIFO. Producer A can simply wait until the consumer uses up the next resource.

1. Answer the following questions about **readers**and **writers**
   1. Is it possible for **multiple** readers to access the **same** file simulataneously?

Yes, multiple readers can access the same file as they are not altering it.

* 1. Is it possible for **multiple** writers to access the **same** file simulataneously?

No, this would cause an integrity issue if there was a writer or reader on the same file with a writer.

* 1. Is it possible for **multiple** writers to access **different** files simulataneously?

Yes, as these writers aren’t interfering with each other’s files.

* 1. Why is it important to implement **mutual exclusion** in a reader/writer system?

Because writers are modifying existing data in the database and data integrity would be impossible to maintain otherwise.

* 1. Is it possible for **starvation** to occur in a reader/writer system?

Yes: Writers can be waiting on readers to finish reading and readers can be waiting on a writer to finish writing.

1. What are the four (4) **categories** of parallelism?

* **Data Level Parallelism**
* **Instruction Level Parallelism**
* **Explicit Parallelism**
* **Implicit Parallelism**

1. Rewrite the following equations to take advantage of concurrent processing using the fewest **COBEGIN** and **COEND**blocks possible
   1. (X \* (Y \* Z \* W \* R) + M + N + P)

COBEGIN

T1 = Y \* Z

T2 = W \* R

T3 = M + N

COEND

(X \* (T1 \* T2) + T3 + P)

COBEGIN

T4 = (T1 \* T2)

T5 = T3 + P

COEND

(X \* T4 + T5)

T6 = X \* T4

T7 = T6 + T5

END

* 1. ((J + K \* L \* M \* N) \* I)

COBEGIN

T1 = K \* L

T2 = M \* N

COEND

((J + T1 \* T2) \* I)

COBEGIN

T3 = T1 \* T2

T4 = J + T3

T5 = T4 \* I

COEND

1. Use the code below to answer the following questions.

for(int x = 0; x < 20; x++)

z(x) = a(x) + b(x);

* 1. How many **identical operations** need to be performed?

Only one.

* 1. On how many **different objects** must the identical operation(s) be performed?

20 different objects: z(0 – 19)

* 1. If a system contains one (1) processor, how many **steps** are required to perform the code above?

Each processor can handle a single step, so 20 steps.

* 1. If a system contains five (5) parallel processors, how many **steps** are required to perform the code above?

4 steps.

* 1. If a system contains ten (10) parallel processors, how many **steps** are required to perform the code above?

2 steps.