Problem 1. (20 points) [CS&G 2.3] There are two dominant models for how parent and children processes relate to each other in a shared address space. In the heavyweight so-called process model, when a process creates another process, the child gets a private copy of the parent's image; that is, if the parent had allocated a variable \( x \), then the child also finds a variable \( x \) in its address space, and the variable is initialized to the value that the parent had for \( x \) when it created the child. However, any modifications that either process makes subsequently are to its own copy of \( x \) and are not visible to the other process. In the lightweight threads model, the child process or thread gets a pointer to the parent's image, so that it and the parent now see the same storage location for \( x \). All data that any process or thread allocates is shared in this model, except the data on a procedure's stack.

(a) Consider the problem of a process having to reference its process identifier \( \text{pid} \) in various parts of a program, in different routines called (in a call chain) by the routine at which the process begins execution. How would you implement this in the first model? In the second? Do you need private data per process, or could you do this with all data being globally shared?

(b) A program written in the process model may rely on the fact that a child process gets its own private copies of the parents' data structures. What changes would you make to port the program to the threads model for data structures that are (i) only read by processes after creation of the child and (ii) are both read and written?

Problem 2. (30 points) Suppose we have \( m \) memory modules and wish to store an \( m \times m \) array. The simplest way to store the array would be to store one column in each memory module (at left, below). But this means that if all elements of a column needed to be accessed in succession, a maximum amount of memory conflicts would take place. Instead, it is possible to start each row in a different memory module, so that successive elements of a column can be accessed without conflict (at right, below).

The distance (in words) between the addresses of two consecutive elements of the same column is called the **column stride**. In the matrix above right, for example, the column stride is \( m+1 \). (The
blank elements—one blank element after the $m$th element of each row of the matrix—are necessary to assure a constant column stride.)

It is proposed that we store our matrix with a column stride of $s$, where $\gcd(s, m) = 1$, so that $m$ successive accesses along a column vector will be directed to $m$ different modules. (Of course, it is easy to see that our row stride should be 1, since $\gcd(m, 1) = 1$.)

This property can be generalized as follows:

For any stride $s$, $m$ successive accesses with stride $s$ will be directed to $\frac{m}{\gcd(s, m)}$ different memories.

Prove this general property.

**Problem 3.** (25 points) For each of the four different kinds of cache organizations, display one sequence of block references for which the organization outperforms the other three organizations (i.e., one sequence for which direct mapping performs the best of all organizations, one sequence for which fully associative is the best of all, etc.; four sequences altogether). To make your answers short, assume—

- there are four block frames in the cache;
- the set or sector size (where applicable) is 2 block frames;
- in all cases, LRU replacement is used.

None of your examples should require more than about ten block references.

**Problem 4.** (10 points) The average memory access time (AMAT) is defined as

$$AMAT = \text{hit\_time} + \text{miss\_rate} \times \text{miss\_penalty}$$

(a) Find the AMAT of a machine 100 MHz machine, with a miss penalty of 20 cycles, a hit time of 2 cycles, and a miss rate of 5%.

(b) Suppose doubling the size of the cache decrease the miss rate to 3%, but causes the hit time to increases to 3 cycles and the miss penalty to increase to 21 cycles. What is the AMAT of the new machine?

**Problem 5.** (15 points) Assume a computer with the following cache characteristics:

- 20% of all operations are writes.
- Hit ratio is 80%.
- 25% of all pages in cache are modified.
- Memory access (both read and write) takes 16 cycles.
- There can be only one memory access at a time

Compute CPI and compare the performance of—

(a) a write-back cache with allocate
(b) a write-through cache with no allocate
(c) and (d): same as (a) and (b) but with a write buffer