Problem 1
Consider a system that has NO virtual memory (i.e., memory addresses refer to real (physical) memory) and every process and the OS kernel are limited to 2 MB of contiguous memory space. This system does have hardware support for memory protection in the form of a special table that contains the following information for each 64 KB partition of memory: owner PID (Process ID), write-ok. The nth entry in this table corresponds to the nth 64 KB partition and also implicitly define the address bounds of the partition. All user processes have PIDs that are greater than or equal to 1024. If write-ok is 0, a write into this area by the owner will cause a hardware interrupt which is handled by the OS. A user process can always read its own memory space but not the memory space of other processes. The OS kernel can read from and write to any memory partition. The OS kernel can be viewed as being a special process with a PID of 0. All processes below 1024 have write permission to the entire memory.

Describe an implementation of fork for this system by listing and briefly describing the steps in the implementation required to construct an executable image of the parent. Your description should include a description of the main data structures employed.

Problem 2
Write a C or C++ program in which a parent process forks off two child processes which each compute a prime number. The command line argument should contain two arguments that indicate which prime number to compute (e.g. 1024th). The parent should wait for both child processes to complete and then display on stdout the message DONE. Each child process should display on stdout the following:

a) Process ID,
b) Parent Process ID,
c) Which prime number (e.g., 1024), and
d) The prime number itself.

The CS422 Web page contains source code for computing the n-th prime number. Note that it uses C++ commenting syntax but uses no other C++ syntax.

Submit the source code, sample output, and a short explanation of why the sample output demonstrates that your program is running properly.
Problem 3
Extend the above program by adding parent process code that displays the resource usage obtained from `getrusage(3C)` and client code that sleeps for $N$ msec (see `usleep(3C)`) before computing the prime number. The parent should display the following information in tabular form for each child:

a) Process id
b) Parent process id
c) User time
d) System time
e) Elapsed time (wall clock time between when `main` started and when it ended.
f) $N = 100$ msec, 500 msec, 2500 msec.

**EXTRA CREDIT:** For the system in Problem 1, describe an implementation of a messaging facility for communicating between two user processes. For example, a process should be able to say:

```c
send (X, buffer);
```

to send a message to the process with PID $X$. The process with PID $X$ should be able to say:

```c
recv (Y, buffer);
```

to receive a message from the process with PID $Y$. 