This lab investigates the synchronization of two threads using `wait`, `poll`, or `intr` driven protocol.

Note that you will **not** be able to run “`timer04 -intr`” on Sun’s Solaris since `sigsuspend()` has (undocumented) problems on this operating system and timing multithreaded programs is never very accurate on Solaris.

1. Get copies of the program `timer04.c` (other versions are old ones). For compatibility across Unix/Linux it is recommended that you use the POSIX thread calls in all new programs. Invocation is by typing

   `timer04 < size > < -wait | -poll | -intr > < cr >`

   This program has a main thread which opens a log file, sets a signal handler, etc. and then starts an auxiliary thread `find_primes()` to find primes. This thread marks primes in `intarr[]` which is passed to it via a pointer in a structure `_th_args` which is itself passed by a pointer. The size is passed the same way (all of this, of course, violates all kinds of cautions we give in beginning programming classes about using untyped things like (void *)). The algorithm is not very efficient and has been known for centuries as the sieve of Erastotesenes. The array entries are initialized to `-1` and then for each `i` between 2 and `size` we try to divide the larger numbers by `i`. If this is possible the number is composite and marked with a 0. When we run into a number which has *not* been marked with a 0 by any of its (prime) predecessors we mark it prime with its own value. If you are interested in better ways to find primes look at the sample program `pseudo.c`.

   The main thread finds spacing between primes but is *synchronized* in different ways depending on whether you specified `wait`, `poll`, or `intr` on the command line. If you specified `wait` it will not do the spacing calculations until `find_primes()` has completed. If you specified `poll` it will keep checking the array for new numbers (composite or prime) and proceed when a new prime has been found. If you specified `intr` it will wait on a signal sent to it by `find_primes()` after every `OPT_INTR_BLOCK` of primes has been found. Then it will go back to sleep. The main thread on completion will print out the average spacing between the primes found and compare it to the theoretical estimate (natural logarithm) `log(size)`. The results in all three case should be the same except for the timing.

**Assignment** Try all three options on the program `timer04` using the same size (choose a number at least as large as 100,000). **Use a text editor to edit the logged results and put in a single file for comparison.** Include these results with your answers below. Answer the following questions and refer to your specific data to justify your answers:

1. What is the *least efficient* option of the three and why?
2. Which option wastes the *least amount* of time and why? Is it impractical in some cases?
3. Which option would you choose for a good combination of flexibility and efficiency and why?