Program:

This program starts with a root process which then creates n delegator nodes, that each create n worker nodes (where n is a number given by the user). The other parameters taken at the start of execution are:
Lower Bound(-l lowerbound), Upper Bound(-u upperbound), work distribution (-e for equal distribution|-r for random distribution), number of delegator nodes under root/worker nodes under each delegator (-n number)

At the end of its execution, all the primes have been stored in the array primes. We sort this array using quicksort. When reading the time taken by each worker node, we keep updating the max_time, min_time, and the sum_time. Then we divide sum_time by the total number of times received to get avg_time. In the end, the program outputs the relevant statistics, with the sorted primes as asked in the assignment.

Fork():

The first set of fork()s to create the delegators are called inside the for loop from the root. Then, by checking if the fork() returns a 0, we can go inside the delegator's part of the code.

Inside the delegators, we have another for loop, in every iteration of whose we fork() again to create the child processes. We thus create n delegator processes and n^2 worker processes.

exec():

We call execlp for each worker node. We decide which program to execute depending on the type variable. We call it with parameters for upper bound, lower bound, batch, and pid of root.

Pipes:

Pipes are used for 6 different purposes in this program. First, we have an array of pipes to write information from root to delegators,
and then each delegator uses another array of pipes to write information to the workers. In the other direction, we use one pipe to write the primes from the delegators to the root, and every delegator, in turn, has one pipe where the workers under it write primes to it.

Moreover, there is another pipe to write the times of the workers from each worker to its respective delegator, and from each delegator to the root.

Read, and Write: While writing to the pipe is straightforward, we use an infinite loop to perform read() from all the workers in the delegators, and all the delegators in the root. This infinite while() loop breaks when read() returns 0. We know that this will only happen when all the file descriptors that could write to it are closed.

Signals:

Every worker node sends either a SIGUSR1 signal or a SIGUSR2 signal depending on its batch.
We send the signal through kill() which takes two parameters: pid of process to whom signal needs to be sent, and the signal which needs to be sent.
We get the pid of root by getting in the parent program and then passing it as a parameter to the primes programs in execlp. We decide on which signal to send by looking at the batch, which is also a parameter passed when calling execlp().

We need to know that if we get a signal when the signal handler is already handling one, that signal is put into the pending queue. However, if we get even more signals these might get lost. This issue was clear when during the initial runs of the program it was observed that the number of signals received was lesser than the total number of worker nodes.
After checking the implementation of sending the signal in the primes1 and primes2 programs and even using pause() to try and receive all signals, we reach the conclusion that the signals are being sent, but are just getting lost. We can try mending this by using sigaction() instead, but this has not been done in the project.

Timing:
Primes1 and Primes2:
Primes1 and Primes2 have been used as given in the appendix. We incorporate the timing functions too. We send the times back to the delegators from the worker, and to the root from the worker through pipes as discussed before. We use long double to get the times.

We need to note, that unless the range over which we need to primes is big, we might get 0s as max,min and avg times.