Run the program

1. `make` (That's it). The default recipe expects all the 3 programs to be ready. The recipe for each program is then provided. Since SR and ST both use assembly in intel's syntax, I have used `-masm=intel`.
2. Run `make clean` to clean the builds.

Program logic

1. The `main` executable starts running. It forks 3 processes (`S1`, `ST`, `SR`) and exits itself as it serves no purpose. The orphaned processes get re-parented.
2. The process id of `S1` is obtained and passed to `ST` and `SR`. Note that `execv` is used each time to overwrite existing process image. To run those programs, `getcwd` is used in order to set the correct path.
3. `SR` gets the parent process ID as a command line argv and uses it to send signals. It also registers a SIGALRM handler. The infinite while loop in the program calls `pause()` to await for signals. A timer is set to periodically alarm the process (1sec intervals), this acts as the SIGALRM signal. In the signal handler, the remaining work is done. It generates a random value using RDRAND in inline assembly. The value is a 64 bit / 8byte value. Since it is just a collection of bits, I'd consider them unsigned for this question. Also, as per intel docs, RDRAND could return 0 when it is unable to generate a random number, in such a case it is advised to loop 10 times (refer to docs). When passing the signal, we are allowed to pass EITHER integer or a POINTER. Since a pointer is of 8bytes, I created a pseudo pointer using the RDRAND value and passed it in sigqueue.
4. `ST` gets the parent process ID as a command line argv. It registers a SIGALRM as well as SIGTERM (as specified in question) handler. The infinite while loop in the program calls `pause()` to await for signals. A timer is set to periodically alarm the process (1sec intervals), this acts as the SIGALRM signal. In the signal handler, the remaining work is done. It gets the timestamp using RDTSC in inline assembly. The timestamp is number of cpu clock ticks since boot. As per intel docs, recent systems are enabled with a clock ticker irrespective of the fluctuating clock speed (`constant_tsc`), mine clocked at 2419.200 MHz (obtained from `/proc/cpuinfo`), so I divided the RDTSC value with this speed to obtain number of seconds. Converted that seconds to Hours:Minutes:Seconds format, which is again an 8byte string (assuming hours,minutes,seconds is always 2 digit). This is again passed a fake pointer via sigqueue.
5. `S1` or the first fork also awaits for signals using `pause()`. It also registers a SIGTERM handler using sigaction. This handler is capable of receiving those previously enqueued pointers. The SIGTERM handler parses the passed pointers as a string (a string maybe defined as a collection of bytes terminated by `\0`), matches it with a time stamp regex (hh:mm:ss) and prints the random and time value with appropriate prefix (i.e. `time is hh:mm:ss` and `random value: xyz`).
6. Also, when you run the file, you may observe random values being printed one after the other or time stamp, that's because of kernel scheduling policies and race conditions (due to pause), since both alarm at 1second. That's it, look at source code for implementation.