Summing numbers

Create a multi-process program that reads numbers from stdin, processes and sums them in child processes and writes sums to files.

The program is called with two arguments `./calc n m`, where `n` is an integer from the range $[3, 10]$ and defines number of child processes and m is an integer from the range $[1, 500]$ and defines length of pauses in milliseconds when sending signals to child processes.

When the program is started its main process creates n child processes and starts reading lines from stdin in a loop. Each line should be either *exit* word or an integer k from the range $[2, 20]$. When a number was read, the process sends it to all of its child processes in the following form: `SIGUSR1` signal is sent exactly k times with m ms pause between each signal; then `SIGUSR2` signal is sent; after that the process waits for the next line on stdin. When *exit* was read the main process sends SIGTERM to all its child processes, stops reading lines and once all of its children have finished the main process ends. If any other string was given as an input, the process prints an error message and waits for next line.

Child processes. Upon its creation each child process is assigned with a number. 1 for the first child process created, 2 for the second one, etc. At first the process reads a single int from a file named `state.XX` where XX is a process number (not PID) and sets its internal sum counter to the value. If the file does not exists or it's shorter than `sizeof(int)` bytes, 0 is used instead. The process prints its counter value. Once the counter is initialized, the process waits for signals. It counts received `SIGUSR1` and once `SIGUSR2` is received it prints the received number to stdout, preceded by its PID, multiplies it by a random int from the range $[-2, 2]$ and adds the result to its sum counter, prints the new counter value and starts counting signals again. When the `SIGTERM` signal is received the child process saves its sum counter to the file (the same which was read earlier) and terminates. **Low-level file access functions must be used by child processes.**

When started each child process generates a random integer i from range $[2, 10]$. When waiting for a signal, each child process respects timeout of i seconds. If no signal was received within the timeout period, the child process terminates. (Also when all child processes terminate, the main process will still only exit after the exit command).

Note: A sequence of pseudorandom numbers shall be different in every child process.
Note2: Signal counting shall be as precise as possible.
Stages:

1. 3 points The main process creates n child processes. Each child process sleeps a random amount of time between 100 and 1500ms, prints a message `PID: 12343 I was sleeping for XXXms` and terminates. The main process terminates after all its children.

2. 4 points The child processes always initialize their counters with 0, main process sends read numbers, child processes have all logic except file reading and writing.

3. 3 points Timeout is working

4. 4 points Implementation of reading and writing counter to a file. Implementation of *exit* command.

Upload

Please upload your solution to: `/home2/samba/sobotkap/unix/`

You have to upload each stage immediately after finishing it. You upload to a networkshare via `ssh.mini.pw.edu.pl` server:

```
scp user.etapX.tar.bz2 user@ssh.mini.pw.edu.pl:/home2/samba/sobotkap/unix/
```

Please name your stages files according to the schema: `LOGIN.etapN.tar.bz2(.gz)`

THE STATEMENT

By decree 27/2020 of University Rector you must add the following statement to the uploads:

```
------------------------------------------------------------------------
I declare that this piece of work which is the basis for recognition of
achieving learning outcomes in the OPS course was completed on my own.
[First and last name] [Student record book number (Student ID number)]
------------------------------------------------------------------------
```

Please add it as comment at the beginning of each source file you upload.Replace square brackets with your data.

Stage	1	2	3	4	
Points	3	4	3	4	14