Complex logic mock task

This task prompts you to perform time-based aggregation of two csv files.

Files description

Market data

File `market_data.csv` contains market prices for a set of instruments over given period.

- `symbol` - 6 letters that describe the symbol of instrument.
- `timestamp` - unix time of the price snapshot (in seconds).
- `price` - floating point number representing the price of instrument.

For instance:
`BTCUSD 1717616970 65000` means that price for BTC is 65000 USD at 05.06.2024 19:49 Unix time.

You can assume that price for instrument does not change until new one for this instrument is received.

Note: prices here don't correspond to real, they are synthetic.

Both files are sorted in ascending order by `timestamp` field.

User data

File `user_data.csv` contains actions of users with their balances during some period of time.

- `user_id` - unique identifier of the user.
- `currency` - currency of this particular transaction.
- `timestamp` - timestamp of this particular transaction (in seconds).

- `delta` - transaction amount. This is how much user's balance changed at the time.

Both files are sorted in ascending order by `timestamp` field.

Task

Your task is to develop application that will create files `bars-1h.csv, bars-1d.csv, bars-30d.csv` that contains aggregated information per separate timeranges, that will be called "bars".
One bar lasts some period of time (1h/1d/30d).

Bars only start at timestamps divisible with their period in unix timestamps.

So, for bar with period `p`, timestamp `t` will belong in bar covering `[t//p*p, (t//p*p)+p-1]` period.

For instance, if first data entry has timestamp `1717616970` and bar has period 1h (`3600` seconds), this timestamp will be included in bar with range `[1717614000, 1717617599]`.

Each bar should contain the following aggregated data for each user:

- `user_id` - unique identifier of a user
- `minimum_balance` - minimum balance of a user during specified bar, **reported in USD using prices during transactions time**
- `maximum_balance` - maximum balance of a user during specified bar, **reported in USD using prices during transactions time**
- `average_balance` - average balance of a user during specified bar, **reported in USD using prices during transactions time**
- `start_timestamp` - bar start timestamp

For simplicity let's assume only one user for the next explanation. Multiple users case is similar - you need to create separate bars for each user.

Formal explanation

Let's say that user has accounts for every of currency that exist, and at start time all balances are zero.

- Let's call user's balance for currency `cur` at time `t` - `balance[cur][t]`.

- Transaction `<currency> <amount> <t>` means `balance[currency][t] = balance[currency][t-1] + amount`. Other balances remain the same: `balance[other][t] = balance[other][t-1]`.
- Let's call price for `BTCUSD` at time `t` - `price["BTC"][t]`

Then, formal definitions are:
`s` - bar start time, `p` - bar period.

$$minbalance = min_{t\in[s,s+p-1]} \sum_{cur} balance_{cur,t} \cdot price_{cur,t}$$

$$maxbalance = max_{t\in[s,s+p-1]} \sum_{cur} balance_{cur,t} \cdot price_{cur,t}$$

$$avgbalance = \frac{\sum_{t=s}^{s+p-1} \sum_{cur} balance_{cur,t} \cdot price_{cur,t}}{p}$$

All these quantities need to be rounded to exactly 4 digits after coma using regular mathematical rounding

You can order bars per user in any way you want, the only requirement is that for each timestamp from `[user_data[0].timestamp, user_data[-1].timestamp]` range and for each user there exists exactly one bar.

Address the example provided (`small/bars-*.csv`) for output format details. It contains (seemingly) correct bars for the small split. Note that you should access this file mainly for output format, I don't guarantee 100% correctness of these bars.

Implementation details

We provide three different splits of data ordered by size - `small`, `medium` and `large`. Testing will be done separately on each split.

All splits share the similar generation process and differ by number of entries and number of users.

You can use any programming language for the solution, if you will be able to specify running instructions.
Solution can expect files `user_data.csv` and `market_data.csv` to be located in the same directory as the application itself.

Your solution should create files `bars-1h.csv, bars-1d.csv, bars-30d.csv` in the same directory as well.

Your solution will be evaluated on windows device with 16 threads 11800H Intel CPU and 32 Gb of RAM.

For implementation aim for the following:

- Results should be accurate, but small numerical issues will be ignored.
- Be as fast as possible.
- Overall app architecture will be also taken into account.

Solution time will be measured alike to `time` linux command.
Good luck!