CS/240/Project/4

Algorithmic trading depends on being able to make predictions quickly. In this project we will use our order book to make predictions about price behavior. Your program will use separate threads in order to read the data, and to calculate a prediction.

Program operation

Your program will run as two separate threads. The first thread will be your main program, used to make predictions about price behavior. You will launch a second thread that will collect the market data, this thread will also print your predictions to the standard output.

To begin, your program will reuse the command line arguments, `-i` and `-s` from the last project. After `-i` an input file name should be provided. After `-s` a symbol name will be provided. If either argument is missing, your program can give an error message and exit. For this project, all input will be in binary, so there will be no `-b` flag.

Your program should start by initializing any variables and data structures it needs. You then need to call `void init_market(FILE *, char *)`, with a pointer to the input file, and a null terminated string indicating which symbol to deliver. After that, your program needs to launch a thread to execute the `void* market(void*)` function. Both functions are provided as part of `libmarket.a`.

Once your program has launched this thread, it is ready to read market data. To read market data, call `char* get_inputs(unsigned int *)`. This function returns a buffer of binary market data (using the same structure as projects 2 and 3) from the thread running the `market()` function. The unsigned int that you passed to the function will list how many bytes have been returned. The market data will only contain orders for the stock you specified in your call to `init_market()`. Each buffer of market data contains only one T message. It is the last message in each buffer. Your main thread should parse this data, and update the order book. When number of bytes returned is 0, the market data has been exhausted, and your program can exit.

For every trade (T), you will want to make a prediction of where the future price will go. In this project, you will calculate two indicators of the stock behavior, and post recommended actions when the stock seems to have been oversold or overbought. To do this, write a function `struct prediction get_prediction()` that will return the current indicator values and any action recommendations. The struct is defined in `market.h` as follows:

```
enum alert_type { NO_ACTION, SELL, BUY};

struct prediction {
    double SSO;
    double signal_line;
    enum alert_type alert;
};
```

The `get_prediction()` function will be called by the `market` thread each time you use `get_inputs()` to read more data. The `market` thread will print out the results of your predictions.

Predictions

Before making predictions, you should test your threads. To do this, your program should take a command-line flag `-t`. When your program is run with `-t`, your `get_prediction()` function will return the lowest sell

price for the `SSO` field, the highest buy price for the `signal_line` field, and the `alert` variable should be set to `NO_ACTION`. You can use the autograder to check for correct program behavior.

Once you have verified that your program correctly reads from the buffer, you are ready to write your prediction algorithm. In this project we will create a simple Slow Stochastic Oscillator (SSO). For each trade your project should implement the following calculations:

```
t[N] = the previous N trade prices
C = current trade price
L = lowest trade price from C or t
H = highest trade price from C or t
FSO = ((C - L) / (H - L)) * 100
SSO = mean of last M FSO values (include latest calc)
signal_line = mean of last M SSO values (include latest calc)
```

By default, N should be 14 and M should be 3. Implementing this will give you the two numbers you need, the `SSO` and the `signal_line`. To calculate the recommended action, you must implement the following pseudo-code:

```
if ((last SSO > 80) && (current SSO < 80) && (signal_line < current SSO) )
    action = SELL
else if ((last SSO < 20) && (current SSO > 20) && (signal_line > current SSO) )
    action = BUY
else
    action = NO_ACTION
```

Once you have this working, you can verify it with the autograder. Finally, you should modify your code to accept command-line parameters `-n` followed by the number of previous trades to track (N) (we no longer need to write into the iq.txt, here the -n flag has a different meaning from the previous projects 2 and 3), and `-m` followed by the number of values to use to calculate `SSO` and `signal_line` (M).

Libraries

The needed functions will be provided to you as a part of the library `libmarket.a`. You will need to include the file `market.h` in your code to use them. You will need to include the file `pthread.h` to use the pthreads library. You will need to use the flag `-pthread` to link in the pthreads library.

Turning in

Your Makefile must contain at least one target `all` which builds a program named `order_book`. For your submission, pack your files as follows: `tar zcf turnin.tgz Makefile src1.c ...` Do not include `libmarket.a` in your package.

The project is due Monday, April 30th before midnight.

Grading criteria

Your grade will be determined by the autograder. It will be based on the ability to:

- correctly launch a thread, and use the libmarket functions to read data
- correctly implement the Slow Stochastic Oscillator
- correctly allow the range of the oscillator to be changed from the command-line