CS/240/Project/3

High frequency trading requires low memory usage and high speed. In this project we will meet these goals. The behavior of your program in this project should be identical to the behavior of your program from project 2, but in this project your performance will be tested. Further, it can be important for mission critical programs to die gracefully. To do this we will add the ability for your program to catch some signals.

Performance requirements

We will measure speed and memory. Your grade will be determined based on how close your performance is to our reference implementation. Our reference program uses no tricks and no clever optimizations. You can be faster. Your code will be tested on a machine in the lab (escher01-escher25). It will be tested with the binary file `/u/lore_s1/gkrichar/dat10m.bin`. Do not copy the file to your directory, it is 153M in size. For debugging purposes, the files `dat10m.txt`, `dat10m.GOOG.ob.txt` and `dat10m.GOOG.iq.txt` (generated with `-n 1000`) have also been put in the same directory. You may also download these files from `http://dumbo.cs.purdue.edu/dat10m.tar.gz` to your own computer (not the lab machines). We present here our time and memory usage with `dat10m.bin`, the AAPL stock, and `-n 10`. We will test your implementation with other stocks included in `dat10m.bin` as well.

Time

To measure time, use the `/usr/bin/time` command. For this project, only the `user` time is important. On the `pod` and `moore` machines in LWSN, our reference solution runs in about 21 seconds:

```
$ /usr/bin/time ./order_book -b -i /u/lore_s1/gkrichar/dat10m.bin -s AAPL -n 10
20.72user 0.02system 0:21.13elapsed 98%CPU
```

On the `sslab` machines, our reference solution runs in about 33 seconds:

```
$ /usr/bin/time ./order_book -b -i /u/lore_s1/gkrichar/dat10m.bin -s AAPL -n 10
32.60user 0.05system 0:33.17elapsed 98%CPU
```

When measuring the runtime of your program, make sure to run the `/usr/bin/time` command a few times. Final submissions will be tested on the `sslab` machines.

Memory

The reference solution uses 528160 bytes of heap memory. For this project, the `heap peak` as reported by the following command is important:

```
$ env LD_PRELOAD=/lib/libmemusage.so ./order_book -b -i /u/lore_s1/gkrichar/dat10m.bin -s AAPL -n 10
Memory usage summary: heap total: 47305240, heap peak: 528160, stack peak: 1408
         total calls   total memory   failed calls
 malloc|     844709       47305240              0
realloc|          0              0              0  (nomove:0, dec:0, free:0)
 calloc|          0              0              0
   free|     844706       47303536
```

As the data structures are all fixed, your heap usage should be very close to ours.

Signals

You need to modify your program to catch two signals, SIGINT and SIGSEGV. When running your programs on remote systems it can be difficult to know why your program died. Did a sysadmin kill it, or did it seg fault? For this project, add two functions, one to respond to each of these signals.

If your program is killed through a SIGINT signal, your program should print:

```
Program killed after processing 3069 messages for symbol GOOG.
```

Do not include any spaces after the period, and only one newline. Clearly, 3069 should be replaced with the correct number of messages processed, and GOOG should be replaced with the name of the actual symbol you processed. After you print the message, call `exit()`. You can test your signal handler by pressing Ctrl+C while your program is running.

If your program suffers a segmentation fault, your program should print:

```
Program died from a segmentation fault after processing 556 messages for symbol SPY.
```

Again, do not include any spaces after the period, and only one newline. Clearly, 556 should be replaced with the correct number of messages processed, and SPY should be replaced with the name of the actual symbol you processed. After you print the message, call `exit()`. You can test the second case by adding a memory violation to your program to see the this message. Do not turn in the program with that memory violation, we have our own means of testing to see if you catch SIGSEGV.

Turning in

For your submission, pack your files as follows: `tar zcf turnin.tgz Makefile src1.c ...`
The project is due Monday, April 16th before midnight.

Grading criteria

Passing all the autograder tests is required to get a grade for this project, but the score you receive from the autograder will not be used. Your grade will be based on how well you meet or beat the reference solution and whether your program catches SIGINT and SIGSEGV. *Unlike other projects we will not return grades at submission time. Performance will be measured after all projects are returned.*