Lamportのアルゴリズムを意識した全順序マルチキャスト通信

Lamportのアルゴリズムを意識してtickというタイムスタンプを用意した。これにより、マルチキャスト通信を用いて分散しているクライアント間で全順序性が保たれている。実行例を見ると、execute Taskが実行されるとき、実行順序が0->1で保たれていることが分かる。

実行方法

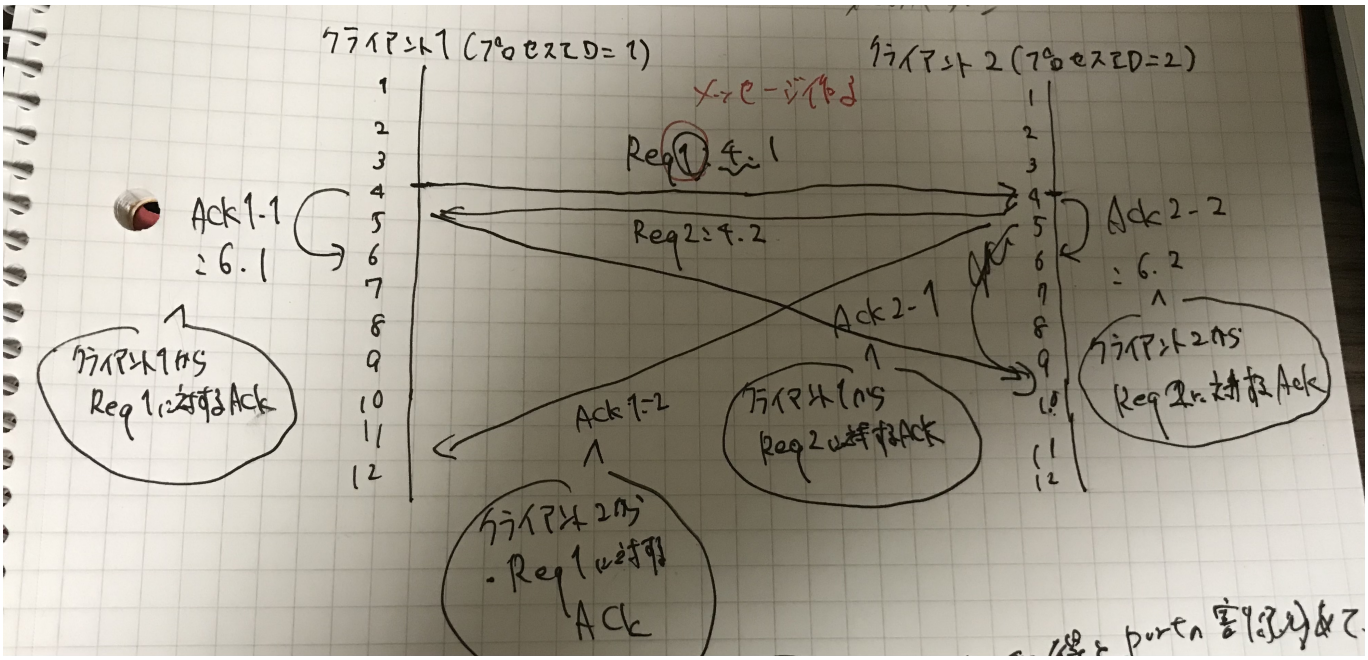
```
make build
make run
```

実行例

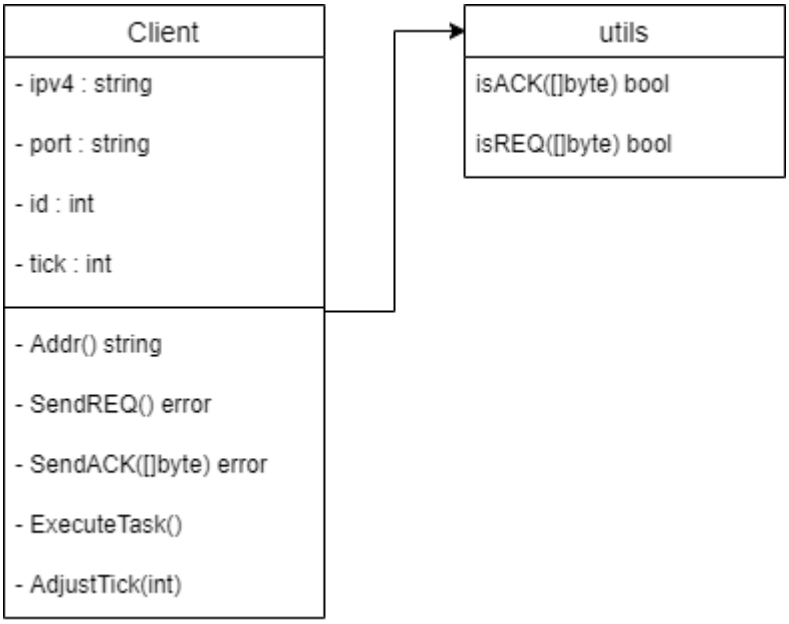
```
$ make run
./lamport 0 &
echo "wait 1sec"; sleep 1;
wait 1sec
[ID0] Begin on 20:14:21
./lamport 1 &
$ [ID1] Begin on 20:14:22
[ID0](t=2) ACK0-0:1.0
[ID0](t=2) Execute Task -> 0
[ID1](t=2) ACK0-1:1.1
[ID0](t=4) REQ0:0.0
[ID0](t=5) ACK1-0:4.0
[ID1](t=3) REQ1:2.1
[ID1](t=4) ACK1-1:3.1
[ID1](t=5) Execute Task -> 1
[ID0](t=6) REQ0:5.0
[ID0](t=7) ACK0-0:6.0
[ID0](t=7) Execute Task -> 0
[ID1](t=7) ACK0-1:6.1
[ID1](t=8) REQ1:7.1
[ID1](t=9) ACK1-1:8.1
[ID1](t=9) Execute Task -> 1
[ID0](t=9) ACK1-0:8.0
[ID0](t=10) REQ0:9.0
[ID1](t=11) ACK0-1:10.1
[ID0] End on 20:14:30
[ID1](t=12) REQ1:11.1
[ID1] End on 20:14:31
```

実装方針

- 全てのクライアント間で全順序性を持たせるために`tick`という変数を全てのクライアントで持たせる
- `tick`は任意の操作をするたびに1加算される
- 操作をしたい場合、ネットワークの他のクライアントに対してマルチキャストでリクエストを送り、その際に`tick`を同期する
- これにより、全てのクライアントで処理の実行前に必ず順序性が保たれる


クライアント1 (プロセスID=1)
クライアント2 (プロセスID=2)
メッセージ作る
Req1:t=1
Req2:t=2
Ack1-1
:6.1
Ack2-2
:6.2
Ack2-1
Ack1-2
クライアント1から
Req1に対するAck
クライアント1から
Req2に対するACK
クライアント2から
Req2に対するAck
クライアント2から
Req1に対する
Ack

構成


Client
- ipv4 : string
- port : string
- id : int
- tick : int
- Addr() string
- SendREQ() error
- SendACK([]byte) error
- ExecuteTask()
- AdjustTick(int)
utils
isACK([]byte) bool
isREQ([]byte) bool