目录

Paxos 算法分析和实现

1 实验目的

掌握 Paxos 算法的原理，理解其证明过程，并完成 Paxos 算法实训。

2 实验内容

在 https://www.educoder.net/加入相关实训后，根据 Paxos 算法流程完成相关核心成员函数设计：

1.根据提示，在编辑器中的 begin-end 间补充代码。

2.需要完成 Proposer.cpp 和 Acceptor.cpp 中 Proposer 和 Acceptor 类的核心成员函数设。

3.当完成后，点击测评按钮，当与预期输出一致时，则算通关。

3 实验方法

3.1 理解Paxos算法原理

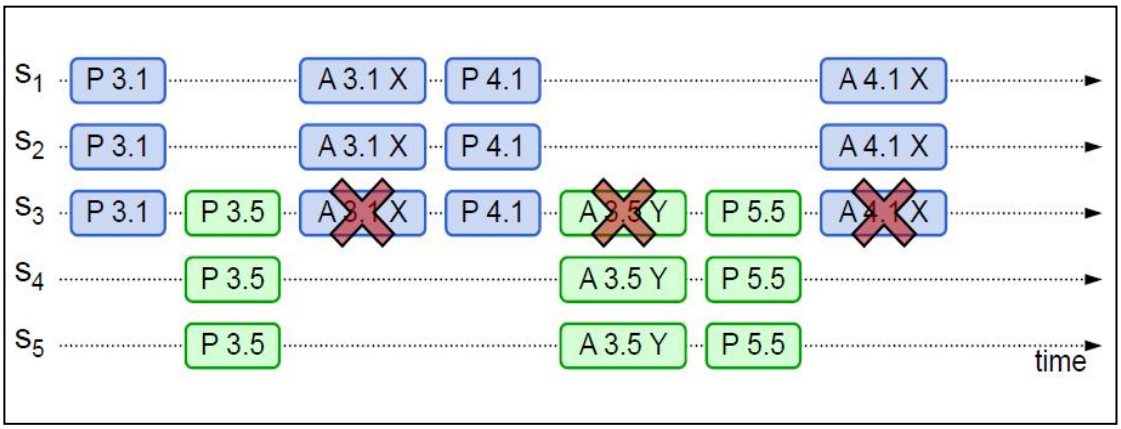
3.1.1 基本特点

Paxos 算法是一种基于消息传递的通讯模型，是为了解决分布式系统中各个节点就某个值（决议）达成一致的问题。不要求可靠的消息传递，可容忍消息丢失、延迟、乱序以及重复。它利用大多数 (Majority) 机制保证了 2F+1 的容错能力，即 2F+1 个节点的系统最多允许 F 个节点同时出现故障。

3.1.2 局限性

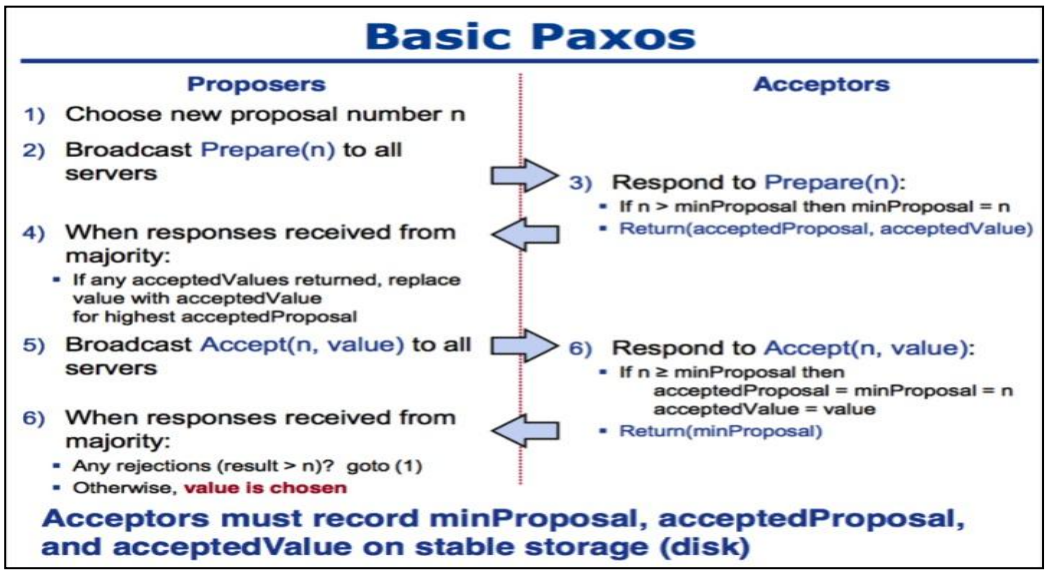
1.不能存在拜占庭将军问题，即各个节点严格遵守算法协议，不存在恶意节点，通迅不能被恶意篡改

2.可能产生“活锁”，即多个提议者轮流提议且互相覆盖，导致无法达成第一阶段议题的一致性表决：



*

3.1.3 算法过程



3.1.4 算法证明

一致性目标：某个决议 Value 被批准(choose)后，该决议 Value 不可修改

通过算法，实现了以下的约束条件：

P1：Acceptor 必须批准它收到的第一个决议。(此条显而易见)

P2：如果一个提案 Proposal(ID, Value)被批准，那么所有被批准通过的提案（编号更高）包含的决议都是 Value

下面来证明 P2：

1.如果一个 Proposal_1(ID_1, Value_1)被批准，则在某一时刻存在 Acceptor 的多数派 C1，其中存储了已经接受的决议为(ID_1, Value_1);

2.因此，如有有 proposer 再提出 Preproposal(ID_2)，获得了通过，则存在多数派 C2 同意此 Preproposal,C1 和 C2 必定有交集,那么 C2 中至少有一个存储的是(ID_1, Value_1)，则这个 preproposal 不可能再提出自己的新值进行投票。

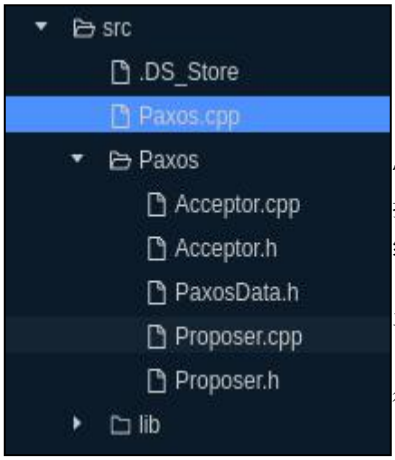
3.根据第 2 条，使用归纳法，可以证明，从第一个被批准的 Proposal(ID_0,VALUE_0)开始，则后面所有能通过第一阶段的 preproposal，其内容的值都是 VALUE_0

4.根据第 3 条可知，如果最终投票过程能正常结束，则能达到一致性，取值为第一个被批准的 VALUE_0。

结论：由于实现了 P1 和 P2，那么如果整个过程能正常结束，至少能使多数派达到一致性。

3.2 完成算法实训

3.2.1 代码结构分析



lib 目录中实现了一些常见的部件，如 thread、log 等

Paxos 目录中有提议者（Proposer）、投票者 (Acceptor)、决议内容(PaxosData)的具体实现

Paxos.cpp 中为每个 Proposer 开启一个线程来模拟分布式一致性达成过程，线程中依次执行：提出预提议(preproposal)->对预提议投票确定议题->对确定的议题投票->得到决策结果（若通过决策则结束线程，否则重新开始整个过程）

总共有 5 个全局的 Propser 和 11 个 Acceptor。则重新发起决议决议号+5(初始为 0-4)，能保证决议号的唯一性和递增性。

如果所有的 Proposer 都能达到一致性，则所有线程都会结束，得到全局的一致性。

3.2.2 完成第一阶段 Preproposal 的代码

Acceptor 收到来自某个 Proposer 的预提议(preproposal)，其提议号为 serialNum

```
bool Acceptor::Propose(unsigned int serialNum, PROPOSAL &lastAcceptValue)
{
    if (0 == serialNum)
        return false;
    //提议不通过
    if (m_maxSerialNum > serialNum)
        return false;
    //接受提议
    //请完善下面逻辑

    /**********Begin**********/
    lastAcceptValue.serialNum = m_lastAcceptValue.serialNum;
    lastAcceptValue.value = m_lastAcceptValue.value;
    m_maxSerialNum = serialNum;
    /**********End**********/

    return true;
}
```

若 m_maxSerialNum 小于 serialNum，则存储 serialNum 作为 m_maxSerialNum，从而作出以下承诺：

1.以后只会处理大于 m_maxSerialNum 的预提议

2. 不会投票赞成 (accept) 任何 serialNum 小于 m_maxSerialNum 的决议。

同时返回已经 accept 的决议。

Proposer 收到 Accptor 对于 Preproposal 的结果

```
bool Proposer::Proposed(bool ok, PROPOSAL &lastAcceptValue)
{
    if (m_proposeFinished)
        return true; //可能是一阶段迟到的回应，直接忽略消息

    if (!ok)
    {
        m_refuseCount++;
        //已有半数拒绝，不需要等待其它acceptor投票了，重新开始Propose阶段
        //使用StartPropose(m_value)重置状态

        //请完善下面逻辑
        /**********Begin**********/
        if (m_refuseCount > m_acceptorCount / 2)
        {
            m_value.serialNum += 5;
            StartPropose(m_value);
            return false;
        }
        /**********End**********/

        //拒绝数不到一半
        return true;
    }
```

统计对于 Preproposal 的投票情况

若当前 Preproposal 阶段已经结束，或者反对票数超过一半，则 Preproposal 失败。

失败，则重新发起 Preposal，其中的决议号+5。

```
    m_okCount++;
    /*
    没有必要检查分支：serialNum为null
    因为serialNum>m_maxAcceptedSerialNum，与serialNum非0互为必要条件
*/
    //如果已经有提议被接受，修改成已被接受的提议
    //请完善下面逻辑
    /*********Begin*********/
    if (lastAcceptValue.serialNum > m_maxAcceptedSerialNum)
    {
        m_maxAcceptedSerialNum = lastAcceptValue.serialNum;
        m_value.value = lastAcceptValue.value;
    }
    /*********End*********/

    //如果自己的提议被接受
    if (m_okCount > m_acceptorCount / 2)
    {
        m_okCount = 0;
        m_proposeFinished = true;
    }
    return true;
}
```

处理所有 Acceptor 回传的提议内容，取其中提议号最大的提议值作为决议值提起正式的议题表决。

如果对于 Preproposal 的赞成票超过一半，则 Preposal 通过，将会进行下一阶段，对决议进行 Accept 投票。

3.2.3 完成第二阶段 Propose(accept)的代码

Acceptor 收到来自某个 Proposer 的第二阶段投票要求

```
bool Acceptor::Accept(PROPOSAL &value)
{
    if (0 == value.serialNum)
        return false;
    //Acceptor又重新答应了其他提议
    //请完善下面逻辑
    /**********Begin**********/
    if (value.serialNum < m_maxSerialNum)
    {
        return false;
    }
    /**********End**********/

    //批准提议通过
    //请完善下面逻辑
    /**********Begin**********/
    m_lastAcceptValue.serialNum = value.serialNum;
    m_lastAcceptValue.value = value.value;
    m_maxSerialNum = value.serialNum;
    /**********End**********/

    return true;
}
```

由于第一阶段的承诺，只会投票赞成决议号大于等于 m_maxSeriaNum 的决议。

若投票赞成此决议，则存储此决议，并且更新 m_maxSeriaNum

Proposer 收到某个 Acceptor 对于决议投票的结果

```
bool Proposer::Accepted(bool ok)
{
    if (!m_proposeFinished)
        return true; //可能是上次第二阶段迟到的回应，直接忽略消息

    if (!ok)
    {
        m_refuseCount++;
        //已有半数拒绝，不需要等待其它acceptor投票了，重新开始Propose阶段
        //使用StartPropose(m_value)重置状态
        //请完善下面逻辑
        /*********Begin*********/
        if (m_refuseCount > m_acceptorCount / 2)
        {
            m_value.serialNum += 5;
            StartPropose(m_value);
            return false;
        }
        /**********End*********/

        return true;
    }

    m_okCount++;
    if (m_okCount > m_acceptorCount / 2)
        m_isAgree = true;

    return true;
}
```

统计所有 Acceptor 对于此决议的投票情况。

若不同意的数量超过 Acceptor 总数的一半，则投票失败，会将决议号+5，此 Proposer 重新开始所有流程。

若同意的数量超过 Acceptor 总数的一半，则投票成功，当前决议者流程成功结束。

4 实验结果

（第一个倒排索引题目也完成了）