Sistemas Distribuídos – Trabalho 3

A linguagem utilizada para a implementação do programa foi a C++.

Primeiramente foram implementadas duas classes a serem usadas como ferramentas. A primeira cria um lock, assim como no segundo trabalho desta disciplina, para garantir a atomicidade das regiões críticas. A segunda implementa o relógio de Lamport local do processo, levando em consideração que o envio de mensagem é evento e, portanto, incrementa o relógio. O recebimento de mensagem compara o relógio de Lamport local e o do evento recebido, incrementa o valor máximo entre os dois e altera o valor do relógio de Lamport local para este valor.

class Lamport

{

atomic<LamportTime> time;

public:

LamportTime getTime()

{

return time;

}

LamportTime sendEvent()

{

return time.fetch\_add(1);

}

LamportTime receiveEvent(LamportTime receivedTime)

{

time = max(time.load(), receivedTime);

return time.fetch\_add(1);

}

};

Foi criada uma rotina que pode ser chamada n vezes para criar n processos. Na chamada da rotina deve-se passar como parâmetro o nome de um arquivo de configuração, o ID e IP do processo em questão, a porta na qual o processo receberá conexões de sockets, o número de eventos a serem gerados por segundo e o número total de eventos.

Por meio do arquivo de configuração, o processo consegue saber quantos processos estão sendo executados além dele e o ID, IP e porta de cada um deles. Exemplo de um arquivo de configuração para 2 processos:

IP ID PORT

127.0.0.1 1 8080

127.0.0.1 2 8081

Primeiramente a porta de conexão do socket do processo é estabelecida, com exceção da do processo de ID 1, que não terá sockets se conectando a ele. Isso acontece porque foi usada a lógica de conectar-se aos processos de ID maior e, sendo o processo de ID 1 o menor, ele apenas fará conexões e receberá nenhuma. Logo após, são feitas todas as conexões de sockets, seguindo a mesma lógica. Com isso, todos os sockets foram criados e os processos estão prontos para trocar mensagens.

// Abre conexão para o processo, com exceção do processo de ID 1

if (processId != "1") {

createServerSocket(portno);

}

//Faz conexão com todos os processos de ID maior

vector<vector<string>> otherProcessesArray = getProcessesArray(processesFileName);

for(int i = 0; i < totalNumberOfprocesses; ++i)

{

string otherProcessIp = otherProcessesArray[i][0];

string otherProcessId = otherProcessesArray[i][1];

int otherProcessPort = stoi(otherProcessesArray[i][2]);

if (processId < otherProcessId) {

createClientSocket(otherProcessIp, otherProcessPort);

}

}

Depois, são criadas 3 threads.

A primeira é responsável por gerar eventos, usando uma função que retorna frases aleatórias. Depois da geração do evento, o relógio de Lamport local é incrementado e a mensagem de ocorrência do evento é enviada a todos os outros processos por meio dos sockets, contendo ID do processo, relógio de Lamport e a frase gerada. Depois, o evento é colocado na fila de espera local.

void localEventsManager(string processesFileName, string processId, int eventsPerSecond, int maxNumberOfEvents)

{

int totalNumberOfEvents = 0;

while(totalNumberOfEvents < maxNumberOfEvents) {

for (int i = 0; i < eventsPerSecond; ++i)

{

string phraseSent = getRandomLineInFile("phrases.txt", 100) + processId;

slock.aquire();

lamport.sendEvent();

string messageSent = "EV\_" + processId + "\_" + to\_string(lamport.getTime()) + "\_" + phraseSent;

addEventInWaitLine(messageSent);

writeInSockets(processId, messageSent);

slock.release();

}

sleep(1);

totalNumberOfEvents += eventsPerSecond;

}

}

Para implementar a fila de espera foram utilizados dois vetores, um armazena as frases conforme são geradas ou recebidas pelo processo, ordenando-as de acordo com o relógio de Lamport e ID do processo de cada uma. O outro vetor armazena, na mesma posição, a quantidade de mensagens de confirmação que cada frase recebeu. Como será visto adiante, a frase só será processada quando apresentar-se no fim do vetor e a quantidade de mensagens de confirmação for igual ao número total de processos.

A segunda thread é responsável por ouvir as mensagens vindas dos sockets, podendo essas serem mensagens de ocorrência de evento ou confirmação. Ao receber a mensagem, o relógio de Lamport local é incrementado. Caso a mensagem seja de ocorrência de evento, esta é adicionada na lista de espera e uma mensagem de confirmação é enviada para todos os outros processos. Caso a mensagem seja de confirmação, o vetor secundário da fila de espera, responsável por armazenar as mensagens de confirmação, é incrementado na posição do evento correspondente.

void externalEventsManager(string processId, int totalNumberOfEvents)

{

while (1) {

vector<string> messagesReceived = readFromSockets(processId);

int numberOfMessagesReceived = messagesReceived.size();

for (int i = 0; i < numberOfMessagesReceived; ++i)

{

string messageReceived = messagesReceived[i];

vector<string> messageReceivedSplited = explode(messageReceived, '\_');

string messageReceivedType = messageReceivedSplited[0];

string processIdReceived = messageReceivedSplited[1];

LamportTime LamportTimeReceived = stoi(messageReceivedSplited[2]);

string phraseReceived = messageReceivedSplited[3];

slock.aquire();

lamport.receiveEvent(LamportTimeReceived);

if (messageReceivedType == "EV") {

addEventInWaitLine(messageReceived);

string messageSent = "OK\_" + processId + "\_" + to\_string(lamport.getTime()) + "\_" + phraseReceived;

writeInSockets(processId, messageSent);

//Manda confirmação para o próprio processo

addOKInEvent(messageSent);

}

if (messageReceivedType == "OK") {

addOKInEvent(messageReceived);

}

slock.release();

}

}

}

A última thread é responsável por, a todo momento, verificar se o evento em primeiro lugar na fila de espera já possui todas as mensagens de confirmação. Se sim, a frase é escrita no arquivo de log correspondente ao processo e depois é deletada da fila de espera.

void writePhraseInLog(string processId, int totalNumberOfprocesses, int totalNumberOfEvents)

{

//Cria arquivo de log

string fileName = "process\_" + processId + "\_log.txt";

ofstream logFile (fileName);

while(1) {

slock.aquire();

int waitLineLenght = eventsWaitLine.size();

if (waitLineLenght != 0) {

int lastElementIndex = waitLineLenght - 1;

if (eventsWaitLineOKReceived[lastElementIndex] == totalNumberOfprocesses) {

//Escreve no arquivo de log

logFile << eventsWaitLine[lastElementIndex] << endl;

eventsAlreadywrittenInLog = eventsAlreadywrittenInLog + 1;

eventsWaitLine.pop\_back();

eventsWaitLineOKReceived.pop\_back();

}

}

slock.release();

}

logFile.close();

}

Abaixo pode ser observado o arquivo de log gerado, de forma idêntica, por dois processos numa mesma máquina, apresentando um total de 10 eventos:

EV\_1\_1\_Vale mais lutar com gente de bem do que triunfar sobre gente ruim.1

EV\_2\_1\_Vale mais lutar com gente de bem do que triunfar sobre gente ruim.2

EV\_1\_4\_Vencer a si próprio é a maior das vitórias.1

EV\_2\_4\_Vencer a si próprio é a maior das vitórias.2

EV\_1\_7\_Quem sabe o que se pode ganhar num dia jamais furta.1

EV\_2\_9\_Quem sabe o que se pode ganhar num dia jamais furta.2

EV\_1\_11\_Se querer é poder, querer é vencer.1

EV\_2\_13\_Se querer é poder, querer é vencer.2

EV\_1\_15\_Só quando aprendemos a perder é que estaremos nos preparando para ganhar.1

EV\_2\_17\_Só quando aprendemos a perder é que estaremos nos preparando para ganhar.2

A fim de teste, o programa foi executado para n = 2,4,8 e k = 100,1000,10000 e lambda = 1,2,10, sendo n o número de processos, lambda o número de frases a ser gerada por segundo e k o número total de frases a serem escritas no arquivo de log.

No entanto, só foi possível o teste para dois processos e os resultados estão expressados no seguinte gráfico:

Pode-se observar um resultado intuitivo. Observa-se que aumentando a taxa x vezes, o tempo decai aproximadamente x vezes. Ao mesmo tempo, aumentando-se o número de eventos x vezes, o tempo aumenta aproximadamente x vezes.

**Código:**

<https://github.com/gabrielalucidi/sistemasdistribuidos2018.1>

**Fontes:**

<http://en.cppreference.com>

<http://www.cplusplus.com>