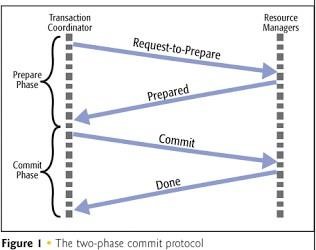
**PROGRAM – 10**

**Aim:** To implement 2-Phase Commit client-server.

**Theory:**

In a local database system, for committing a transaction, the transaction manager has to only convey the decision to commit to the recovery manager. However, in a distributed system, the transaction manager should convey the decision to commit to all the servers in the various sites where the transaction is being executed and uniformly enforce the decision. When processing is complete at each site, it reaches the partially committed transaction state and waits for all other transactions to reach their partially committed states. When it receives the message that all the sites are ready to commit, it starts to commit. In a distributed system, either all sites commit or none of them does.

Distributed two-phase commit reduces the vulnerability of one-phase commit protocols. The steps performed in the two phases are as follows −

**Phase 1: Prepare Phase**

* After each slave has locally completed its transaction, it sends a “DONE” message to the controlling site. When the controlling site has received “DONE” message from all slaves, it sends a “Prepare” message to the slaves.
* The slaves vote on whether they still want to commit or not. If a slave wants to commit, it sends a “Ready” message.
* A slave that does not want to commit sends a “Not Ready” message. This may happen when the slave has conflicting concurrent transactions or there is a timeout.

**Phase 2: Commit/Abort Phase**

* After the controlling site has received “Ready” message from all the slaves –
  + The controlling site sends a “Global Commit” message to the slaves.
  + The slaves apply the transaction and send a “Commit ACK” message to the controlling site.
  + When the controlling site receives “Commit ACK” message from all the slaves, it considers the transaction as committed.
* After the controlling site has received the first “Not Ready” message from any slave –
  + The controlling site sends a “Global Abort” message to the slaves.
  + The slaves abort the transaction and send a “Abort ACK” message to the controlling site.
  + When the controlling site receives “Abort ACK” message from all the slaves, it considers the transaction as aborted.

**Code:**

Server:

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <string.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <arpa/inet.h>

#include <netinet/in.h>

#include <netdb.h>

#include <time.h>

#include <sys/poll.h>

#define MAXLINE 1024

#define MAX\_PROCESS 5

int max(int a, int b)

{

if (a > b)

return a;

else

return b;

}

struct Message

{

char type; // V -> Vote Request, c -> Commit OK, a -> Commit Abort, C -> Global commit, A -> Global abort

};

int create\_connection(int PORT)

{

int sockfd;

struct sockaddr\_in servaddr;

if ((sockfd = socket(AF\_INET, SOCK\_DGRAM, 0)) < 0)

{

perror("socket creation failed");

exit(EXIT\_FAILURE);

}

int optval = 1;

setsockopt(sockfd, SOL\_SOCKET, SO\_REUSEADDR, (const void \*)&optval, sizeof(int));

memset(&servaddr, 0, sizeof(servaddr));

servaddr.sin\_family = AF\_INET; // IPv4

servaddr.sin\_addr.s\_addr = INADDR\_ANY;

servaddr.sin\_port = htons(PORT);

if (bind(sockfd, (const struct sockaddr \*)&servaddr,

sizeof(servaddr)) < 0)

{

perror("bind failed");

exit(EXIT\_FAILURE);

}

return sockfd;

}

void send\_message(int dest\_port, int sockfd, struct Message newMsg)

{

struct sockaddr\_in client\_addr;

memset(&client\_addr, 0, sizeof(client\_addr));

client\_addr.sin\_family = AF\_INET; // IPv4

client\_addr.sin\_addr.s\_addr = INADDR\_ANY;

client\_addr.sin\_port = htons(dest\_port);

sendto(sockfd, (struct Message \*)&newMsg, (1024 + sizeof(newMsg)), 0, (struct sockaddr \*)&client\_addr, sizeof(client\_addr));

}

void \_2pc(int sockfd, int OTHER\_PROCESS\_PORTS[], int NUM\_PROCESSES, int MY\_PORT)

{

printf("Starting the Voting Phase \n");

struct sockaddr\_in recv\_client\_addr[NUM\_PROCESSES], send\_client\_addr;

struct Message \*temp = malloc(sizeof(struct Message));

int len = sizeof(struct sockaddr\_in), n, i, flag = 1;

struct Message newMsg;

newMsg.type = 'V';

for (i = 0; i < NUM\_PROCESSES; i++)

{

send\_message(OTHER\_PROCESS\_PORTS[i], sockfd, newMsg);

}

int commit\_flag = 1;

for (i = 0; i < NUM\_PROCESSES; i++)

{

n = recvfrom(sockfd, temp, sizeof(\*temp), MSG\_WAITALL, (struct sockaddr \*)&recv\_client\_addr[i], &len);

if (temp->type == 'c')

{

commit\_flag = commit\_flag && 1;

}

else if (temp->type == 'a')

{

commit\_flag = commit\_flag && 0;

}

}

printf("Starting the Decision Phase \n");

char final\_decision = (commit\_flag ? 'C' : 'A');

for (i = 0; i < NUM\_PROCESSES; i++)

{

newMsg.type = final\_decision;

send\_message(OTHER\_PROCESS\_PORTS[i], sockfd, newMsg);

}

// return flag;

}

int main(int argc, char \*argv[])

{

int MY\_PORT = atoi(argv[1]);

int NUM\_PROCESSES = atoi(argv[2]);

int OTHER\_PROCESS\_PORTS[MAX\_PROCESS];

int i;

for (i = 0; i < NUM\_PROCESSES; i++)

{

OTHER\_PROCESS\_PORTS[i] = atoi(argv[3 + i]);

}

struct Message \*temp = malloc(sizeof(struct Message));

printf("Initialising the time server at port %d.\n", MY\_PORT);

int sockfd = create\_connection(MY\_PORT);

struct sockaddr\_in recv\_client\_addr, send\_client\_addr;

\_2pc(sockfd, OTHER\_PROCESS\_PORTS, NUM\_PROCESSES, MY\_PORT);

}

Client:

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <string.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <arpa/inet.h>

#include <netinet/in.h>

#include <netdb.h>

#include <time.h>

#define MAXLINE 1024

#define MAX\_PROCESS 5

struct Message

{

char type; // V -> Vote Request, c -> Commit OK, a -> Commit Abort, C -> Global commit, A -> Global abort

};

int create\_connection(int PORT)

{

int sockfd;

struct sockaddr\_in servaddr;

if ((sockfd = socket(AF\_INET, SOCK\_DGRAM, 0)) < 0)

{

perror("socket creation failed");

exit(EXIT\_FAILURE);

}

int optval = 1;

setsockopt(sockfd, SOL\_SOCKET, SO\_REUSEADDR, (const void \*)&optval, sizeof(int));

memset(&servaddr, 0, sizeof(servaddr));

servaddr.sin\_family = AF\_INET; // IPv4

servaddr.sin\_addr.s\_addr = INADDR\_ANY;

servaddr.sin\_port = htons(PORT);

if (bind(sockfd, (const struct sockaddr \*)&servaddr,

sizeof(servaddr)) < 0)

{

perror("bind failed");

exit(EXIT\_FAILURE);

}

return sockfd;

}

void send\_message(int dest\_port, int sockfd, struct Message newMsg)

{

struct sockaddr\_in client\_addr;

memset(&client\_addr, 0, sizeof(client\_addr));

client\_addr.sin\_family = AF\_INET; // IPv4

client\_addr.sin\_addr.s\_addr = INADDR\_ANY;

client\_addr.sin\_port = htons(dest\_port);

sendto(sockfd, (struct Message \*)&newMsg, (1024 + sizeof(newMsg)), 0, (struct sockaddr \*)&client\_addr, sizeof(client\_addr));

}

int main(int argc, char \*argv[])

{

srand(time(NULL));

int MY\_PORT = atoi(argv[1]);

int COORDINATOR\_PORT = atoi(argv[2]);

double random\_number = (double)rand() / (double)((unsigned)RAND\_MAX + 1);

char my\_status = (random\_number >= 0.4 ? 'a' : 'c');

struct Message \*temp = malloc(sizeof(struct Message));

printf("Initialising the client at port %d\n", MY\_PORT);

int sockfd = create\_connection(MY\_PORT);

struct sockaddr\_in recv\_client\_addr, send\_client\_addr;

int len = sizeof(struct sockaddr\_in), n, i, flag = 1;

while (1)

{

n = recvfrom(sockfd, temp, sizeof(\*temp), MSG\_WAITALL, (struct sockaddr \*)&recv\_client\_addr, &len);

struct Message newMsg;

if (temp->type == 'V')

{

printf("COORDINATOR sent a VOTE REQUEST.");

newMsg.type = my\_status;

printf(" Local status : %s\n", (random\_number >= 0.6 ? "abort" : "commit"));

send\_message(COORDINATOR\_PORT, sockfd, newMsg);

}

else if (temp->type == 'C')

{

printf("COORDINATOR sent a GLOBAL COMMIT.\n");

}

else if (temp->type == 'A')

{

printf("COORDINATOR sent a GLOBAL ABORT.\n");

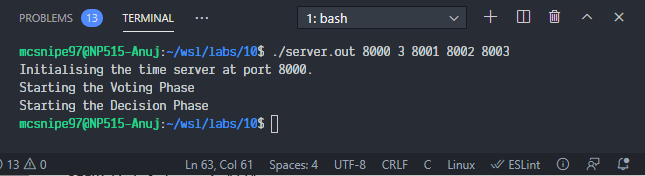
}

}

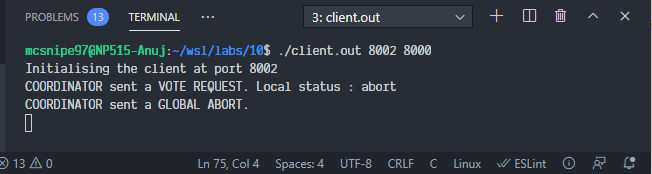
}

**Output:**

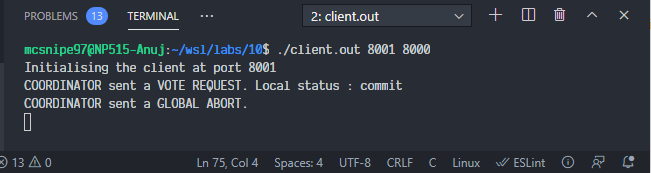
Server:



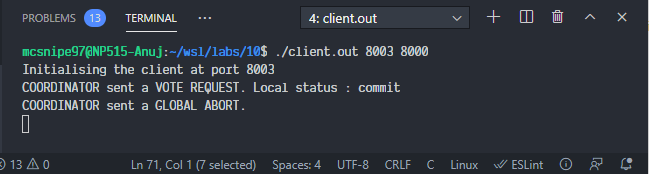
Client1:



Client2:



Client3:



**Conclusion:** We successfully implemented 2-phase commit.