

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.

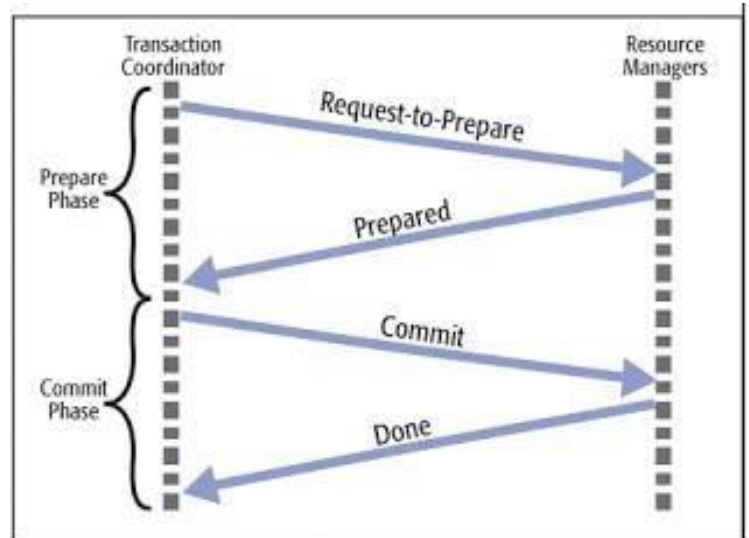


Figure 1 • The two-phase commit protocol

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"));

```

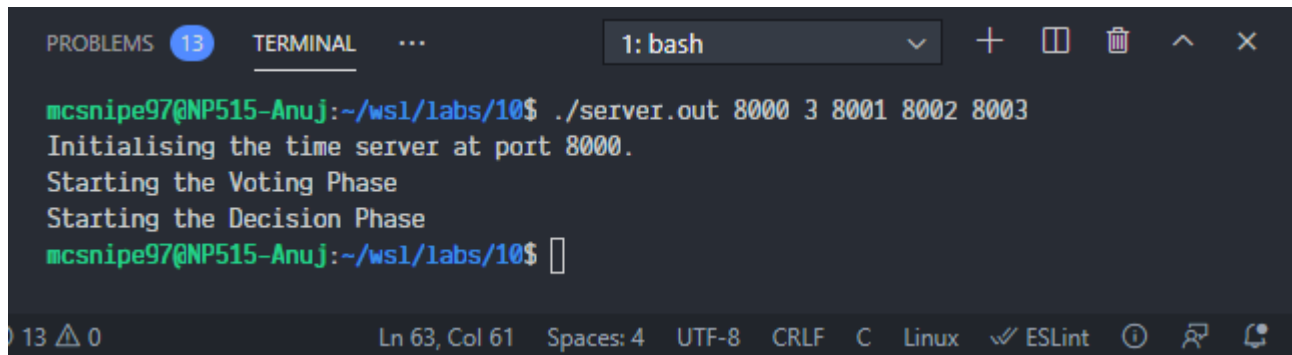
```

        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:

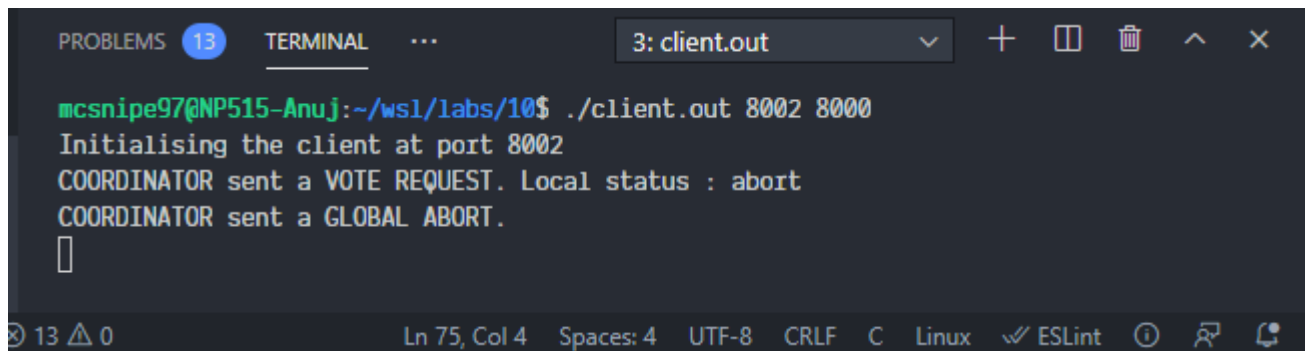


```

PROBLEMS 13 TERMINAL ... 1: bash
mcsnipe97@NP515-Anuj:~/wsl/labs/10$ ./server.out 8000 3 8001 8002 8003
Initialising the time server at port 8000.
Starting the Voting Phase
Starting the Decision Phase
mcsnipe97@NP515-Anuj:~/wsl/labs/10$ 

```

Client1:

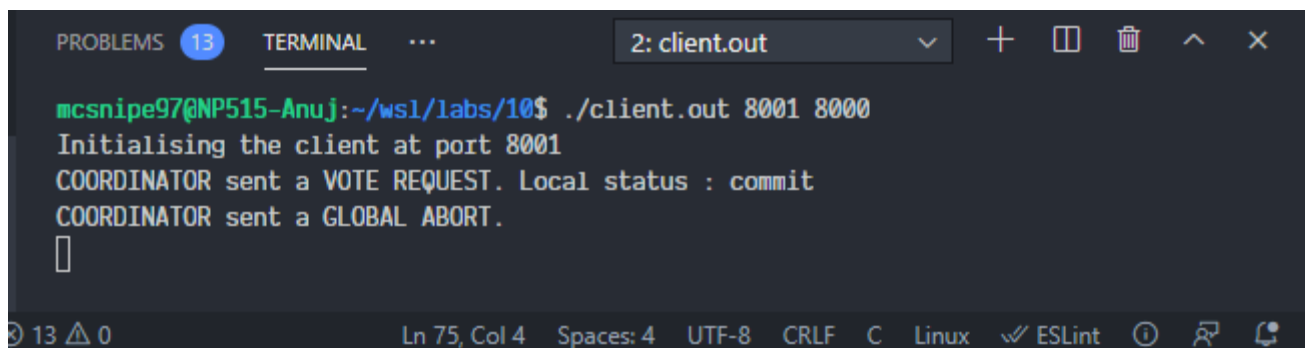


```

PROBLEMS 13 TERMINAL ... 3: client.out
mcsnipe97@NP515-Anuj:~/wsl/labs/10$ ./client.out 8002 8000
Initialising the client at port 8002
COORDINATOR sent a VOTE REQUEST. Local status : abort
COORDINATOR sent a GLOBAL ABORT.

```

Client2:

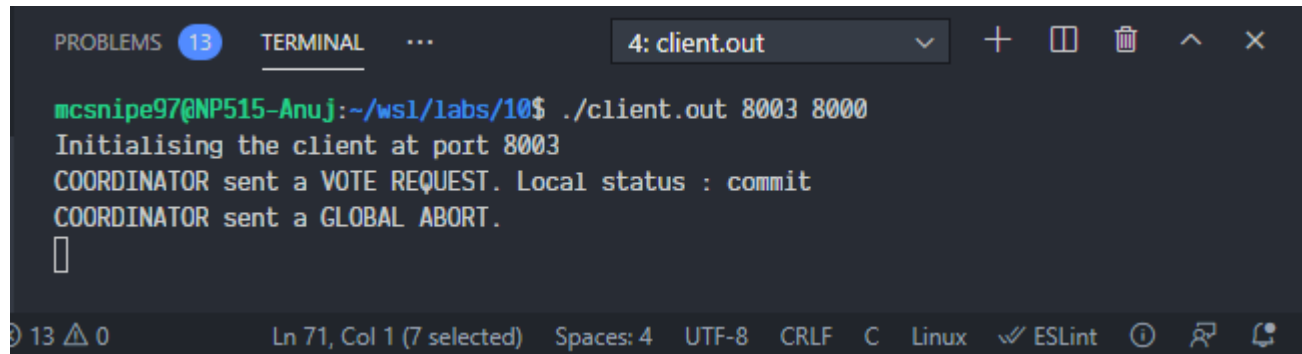


```

PROBLEMS 13 TERMINAL ... 2: client.out
mcsnipe97@NP515-Anuj:~/wsl/labs/10$ ./client.out 8001 8000
Initialising the client at port 8001
COORDINATOR sent a VOTE REQUEST. Local status : commit
COORDINATOR sent a GLOBAL ABORT.

```

Client3:



A screenshot of a terminal window with a dark background. The window title bar shows '4: client.out'. The terminal content shows a user running a command, followed by several lines of output from a program. The output indicates the client is initialized at port 8003 and receives messages from a coordinator.

```
PROBLEMS 13 TERMINAL ... 4: client.out
mcsnipe97@NP515-Anuj:~/wsl/labs/10$ ./client.out 8003 8000
Initialising the client at port 8003
COORDINATOR sent a VOTE REQUEST. Local status : commit
COORDINATOR sent a GLOBAL ABORT.
█
```

At the bottom of the terminal, a status bar shows 'Ln 71, Col 1 (7 selected)' and other editor settings like 'Spaces: 4', 'UTF-8', 'CRLF', 'C', 'Linux', and 'ESLint'.

Conclusion: We successfully implemented 2-phase commit.