PROGRAM-9

Aim: Implement entry eventual consistency between processes with mutual exclusive update replicated datastore.

Theory:

Consistency models are used in distributed systems like distributed shared memory systems or distributed data stores (such as a filesystem, databases, optimistic replication systems or web caching). The system is said to support a given model if operations on memory follow specific rules. The data consistency model specifies a contract between programmer and system, wherein the system guarantees that if the programmer follows the rules, memory will be consistent and the results of reading, writing, or updating memory will be predictable. This is different from coherence, which occurs in systems that are cached or cache-less and is consistency of data with respect to all processors. Coherence deals with maintaining a global order in which writes to a single location or single variable are seen by all processors.

Consistency deals with the ordering of operations to multiple locations with respect to all processors. There are two methods to define and categorize consistency models; issue and view.

- Issue: Issue method describes the restrictions that define how a process can issue operations.
- View: View method which defines the order of operations visible to processes.

Entry consistency

- Acquire and release are still used, and the data-store meets the following conditions:
- An acquire access of a synchronization variable is not allowed to perform with respect to a process until all updates to the guarded shared data have been performed with respect to that process.
- Before an exclusive mode access to a synchronization variable by a process is allowed to perform with respect to that process, no other process may hold the synchronization variable, not even in nonexclusive mode.
- After an exclusive mode access to a synchronization variable has been performed, any other process's next nonexclusive mode access to that synchronization variable may not be performed until it has performed with respect to that variable's owner.

Code:

Server:
```
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <time.h>
```

```c
#include <string.h>
// #define MSG_CONFIRM 0

#define TRUE 1
#define FALSE 0
#define ML 1024
#define MPROC 32

typedef struct Resource
{
    int a;
    int b;
    int c;
    int d;
    int e;
} Resource;

void serealize(Resource S, char output[ML])
{
    sprintf(output, "MCON %d\t%d\t%d\t%d\t%d\t", S.a, S.b, S.c, S.d, S.e);
}

Resource unserealize(char input[ML])
{
    char temp[ML];
    int ix = 0, itr = 5;
    Resource S;
    for (itr; input[itr] != '\t'; itr += 1)
        temp[ix++] = input[itr];
    temp[ix] = '\0';
    S.a = atoi(temp);
    ix = 0;

    for (itr = itr + 1; input[itr] != '\t'; itr += 1)
        temp[ix++] = input[itr];
    temp[ix] = '\0';
    S.b = atoi(temp);
    ix = 0;

    for (itr = itr + 1; input[itr] != '\t'; itr += 1)
        temp[ix++] = input[itr];
    temp[ix] = '\0';
    S.c = atoi(temp);
    ix = 0;

    for (itr = itr + 1; input[itr] != '\t'; itr += 1)
        temp[ix++] = input[itr];
    temp[ix] = '\0';
    S.d = atoi(temp);
    ix = 0;

    for (itr = itr + 1; input[itr] != '\t'; itr += 1)
```

```
        temp[ix++] = input[itr];
    temp[ix] = '\0';
    S.e = atoi(temp);
    ix = 0;
    return S;
}

int connect_to_port(int connect_to)
{
    int sock_id;
    int opt = 1;
    struct sockaddr_in server;
    if ((sock_id = socket(AF_INET, SOCK_DGRAM, 0)) < 0)
    {
        perror("unable to create a socket");
        exit(EXIT_FAILURE);
    }
    setsockopt(sock_id, SOL_SOCKET, SO_REUSEADDR, (const void *)&opt,
   sizeof(int));
    memset(&server, 0, sizeof(server));
    server.sin_family = AF_INET;
    server.sin_addr.s_addr = INADDR_ANY;
    server.sin_port = htons(connect_to);

    if (bind(sock_id, (const struct sockaddr *)&server, sizeof(server)) < 0)
    {
        perror("unable to bind to port");
        exit(EXIT_FAILURE);
    }
    return sock_id;
}

void send_to_id(int to, int from, char message[ML])
{
    struct sockaddr_in cl;
    memset(&cl, 0, sizeof(cl));

    cl.sin_family = AF_INET;
    cl.sin_addr.s_addr = INADDR_ANY;
    cl.sin_port = htons(to);

    sendto(from, (const char *)message, strlen(message), MSG_CONFIRM, (const
   struct sockaddr *)&cl, sizeof(cl));
}

void make_consistent(int from, int procs[], int n_procs, Resource S)
{
    char message[ML];
    int i;
    serealize(S, message);
    for (i = 0; i < n_procs; i++)
        send_to_id(procs[i], from, message);
```

```
}

int main(int argc, char *argv[])
{
    int self = atoi(argv[1]);
    int n_procs = atoi(argv[2]);
    int itr, ix = 0;
    int procs[MPROC];
    int key_avail = 1;
    int dest;

    int sock_id, len, n;
    char buffer[ML], msg[ML];
    char flag[256], p_id[256];

    struct sockaddr_in from;

    Resource S = {0, 0, 0, 0, 0};
    for (itr = 0; itr < n_procs; itr++)
        procs[itr] = atoi(argv[3 + itr]);
    printf("Creating node at %d\n", self);
    sock_id = connect_to_port(self);

    while (TRUE)
    {
        memset(&from, 0, sizeof(from));
        n = recvfrom(sock_id, (char *)buffer, ML, MSG_WAITALL, (struct sockaddr
   *)&from, &len);
        buffer[n] = '\0';
        printf("Recieved: %s\n", buffer);

        for (itr = 0; itr < 4; itr++)
            flag[itr] = buffer[itr];
        flag[itr] = '\0';
        printf("Extracted flag %s\n", flag);

        // process asks for key
        if (strcmp(flag, "KEYR") == 0)
        {
            ix = 0;
            for (itr = 5; itr < 9; itr++)
                p_id[ix++] = buffer[itr];
            p_id[ix] = '\0';
            dest = atoi(p_id);
            printf("Extracted dest %d\n", dest);
            if (key_avail)
            {
                send_to_id(dest, sock_id, "PASS");
                key_avail = 0;
            }

            else
            {
                send_to_id(dest, sock_id, "WAIT");
            }
        }
        // process releases key
        else if (strcmp(flag, "DONE") == 0)
        {
            printf("Key released\n");
            S = unserealize(buffer);
            key_avail = 1;
        }
        // process calls for consistency
        else if (strcmp(flag, "MCON") == 0)
        {
            printf("Forcing consistency \n");
            make_consistent(sock_id, procs, n_procs, S);
            for (itr = 5; itr < 9; itr++)
                p_id[5 - itr] = buffer[itr];
            p_id[5 - itr] = '\0';
            dest = atoi(p_id);
            send_to_id(dest, sock_id, "CNOK");
        }
    }
}
```

Client:

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <time.h>
#include <string.h>
// #define MSG_CONFIRM 0

#define TRUE 1
#define FALSE 0
#define ML 1024
#define MPROC 32

typedef struct Resource
{
    int a;
    int b;
    int c;
    int d;
    int e;
} Resource;
```

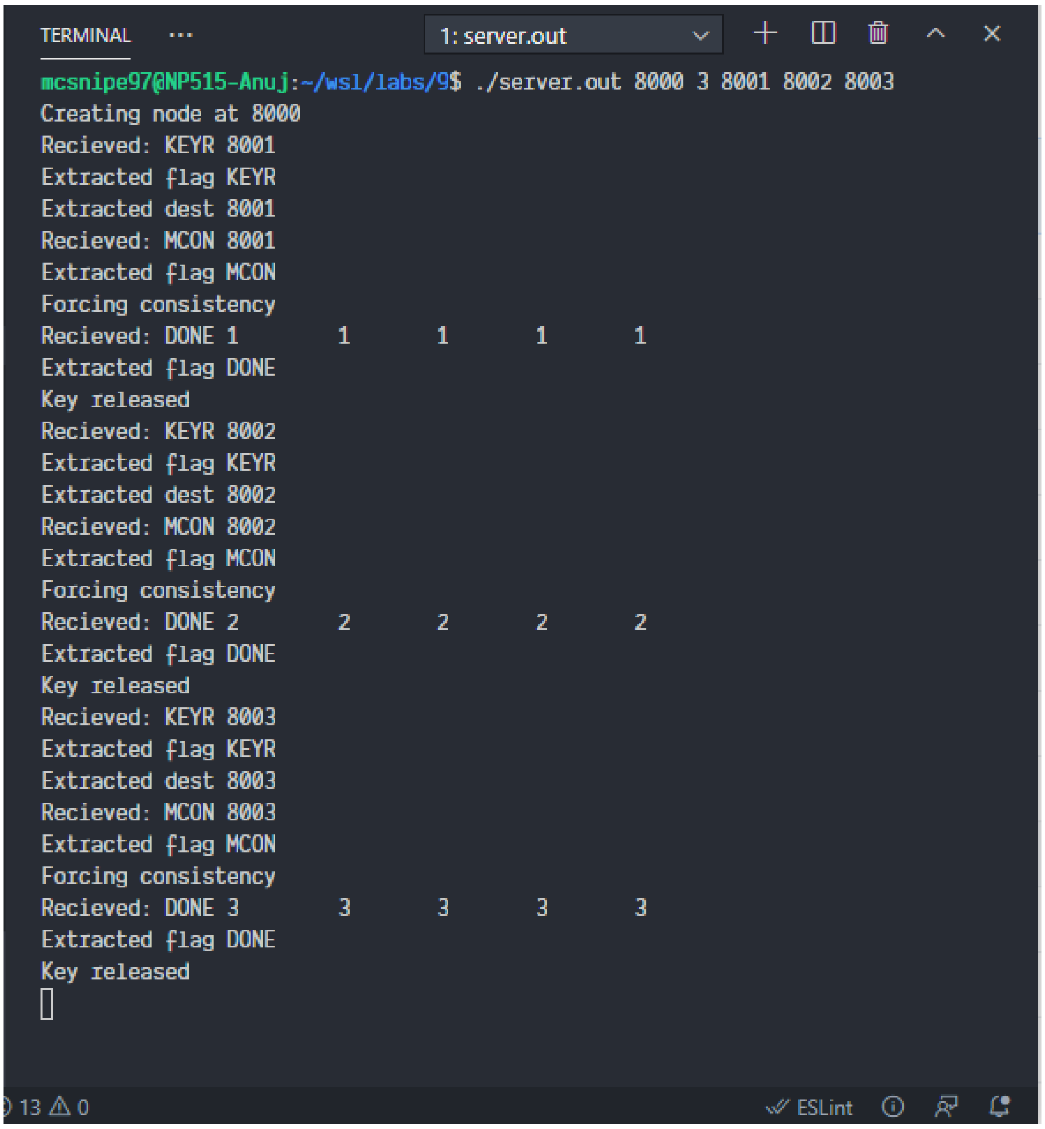
```
void serealize(Resource S, char output[ML])
{
    sprintf(output, "DONE %d\t%d\t%d\t%d\t%d\t", S.a, S.b, S.c, S.d, S.e);
}

Resource unserealize(char input[ML])
{
    char temp[ML];
    int ix = 0, itr = 5;
    Resource S;
    for (itr; input[itr] != '\t'; itr += 1)
    {
        printf("%d %c\n", itr, input[itr]);
        temp[ix++] = input[itr];
    }
    temp[ix] = '\0';
    S.a = atoi(temp);
    ix = 0;
    printf("here\n");
    for (itr = itr + 1; input[itr] != '\t'; itr += 1)
        temp[ix++] = input[itr];
    temp[ix] = '\0';
    S.b = atoi(temp);
    ix = 0;

    for (itr = itr + 1; input[itr] != '\t'; itr += 1)
        temp[ix++] = input[itr];
    temp[ix] = '\0';
    S.c = atoi(temp);
    ix = 0;

    for (itr = itr + 1; input[itr] != '\t'; itr += 1)
        temp[ix++] = input[itr];
    temp[ix] = '\0';
    S.d = atoi(temp);
    ix = 0;

    for (itr = itr + 1; input[itr] != '\t'; itr += 1)
        temp[ix++] = input[itr];
    temp[ix] = '\0';
    S.e = atoi(temp);
    ix = 0;
    return S;
}

int connect_to_port(int connect_to)
{
    int sock_id;
    int opt = 1;
    struct sockaddr_in server;
    if ((sock_id = socket(AF_INET, SOCK_DGRAM, 0)) < 0)
    {
        perror("unable to create a socket");
        exit(EXIT_FAILURE);
    }
    setsockopt(sock_id, SOL_SOCKET, SO_REUSEADDR, (const void *)&opt,
   sizeof(int));
    memset(&server, 0, sizeof(server));
    server.sin_family = AF_INET;
    server.sin_addr.s_addr = INADDR_ANY;
    server.sin_port = htons(connect_to);

    if (bind(sock_id, (const struct sockaddr *)&server, sizeof(server)) < 0)
    {
        perror("unable to bind to port");
        exit(EXIT_FAILURE);
    }
    return sock_id;
}

void send_to_id(int to, int from, char message[ML])
{
    struct sockaddr_in cl;
    memset(&cl, 0, sizeof(cl));

    cl.sin_family = AF_INET;
    cl.sin_addr.s_addr = INADDR_ANY;
    cl.sin_port = htons(to);

    sendto(from, (const char *)message, strlen(message), MSG_CONFIRM, (const
   struct sockaddr *)&cl, sizeof(cl));
}

void request_key(int server, int sock_id, int a)
{
    char msg[256];
    sprintf(msg, "KEYR %d", a);
    send_to_id(server, sock_id, msg);
}

int main(int argc, char *argv[])
{
    int self = atoi(argv[1]);
    int server = atoi(argv[2]);
    int start = atoi(argv[3]);
    int udelay = atoi(argv[4]);
    int itr;
    int dest;
    int key = 0;

    int sock_id, len, n;
    char buffer[ML], msg[ML];
    char flag[256], p_id[256];
```

```
 struct sockaddr_in from;

 Resource S = {0, 0, 0, 0, 0};
 printf("Creating node at %d\n", self);
 sock_id = connect_to_port(self);

 if (start)
 {
     request_key(server, sock_id, self);
 }
 else
 {
     sleep(udelay);
     request_key(server, sock_id, self);
 }

 while (TRUE)
 {
     // sleep(udelay);
     memset(&from, 0, sizeof(from));
     n = recvfrom(sock_id, (char *)buffer, ML, MSG_WAITALL, (struct sockaddr
*)&from, &len);
     buffer[n] = '\0';
     printf("Recieved: %s\n", buffer);

     for (itr = 0; itr < 4; itr++)
         flag[itr] = buffer[itr];
     flag[itr] = '\0';
     printf("Extracted flag %s\n", flag);

     // server denies key
     if (strcmp(flag, "WAIT") == 0)
     {
         sleep(udelay);
         request_key(server, sock_id, self);
     }
     // process releases key
     else if (strcmp(flag, "PASS") == 0)
     {
         printf("Key recieved\n");
         key = 1;
         sprintf(msg, "MCON %d", self);
         send_to_id(server, sock_id, msg);
     }
     // process calls for consistency
     else if (strcmp(flag, "MCON") == 0)
     {
         printf("Pulling data from server before update\n");
         S = unserealize(buffer);
         printf("Pulled file\n %5d, %5d %5d %5d %5d\n", S.a, S.b, S.c, S.d,
S.e);
```

```
        }

        else if (strcmp(flag, "CNOK") == 0 && key)
        {
            printf("Entering critical Seaction\n");
            S.a++;
            S.b++;
            S.c++;
            S.d++;
            S.e++;
            printf("Exiting critical Seaction\n");
            printf("Current file\n %5d, %5d %5d %5d %5d\n", S.a, S.b, S.c, S.d,
    S.e);
            serealize(S, msg);
            send_to_id(server, sock_id, msg);
            exit(EXIT_SUCCESS);
        }
    }
    return 0;
}
```

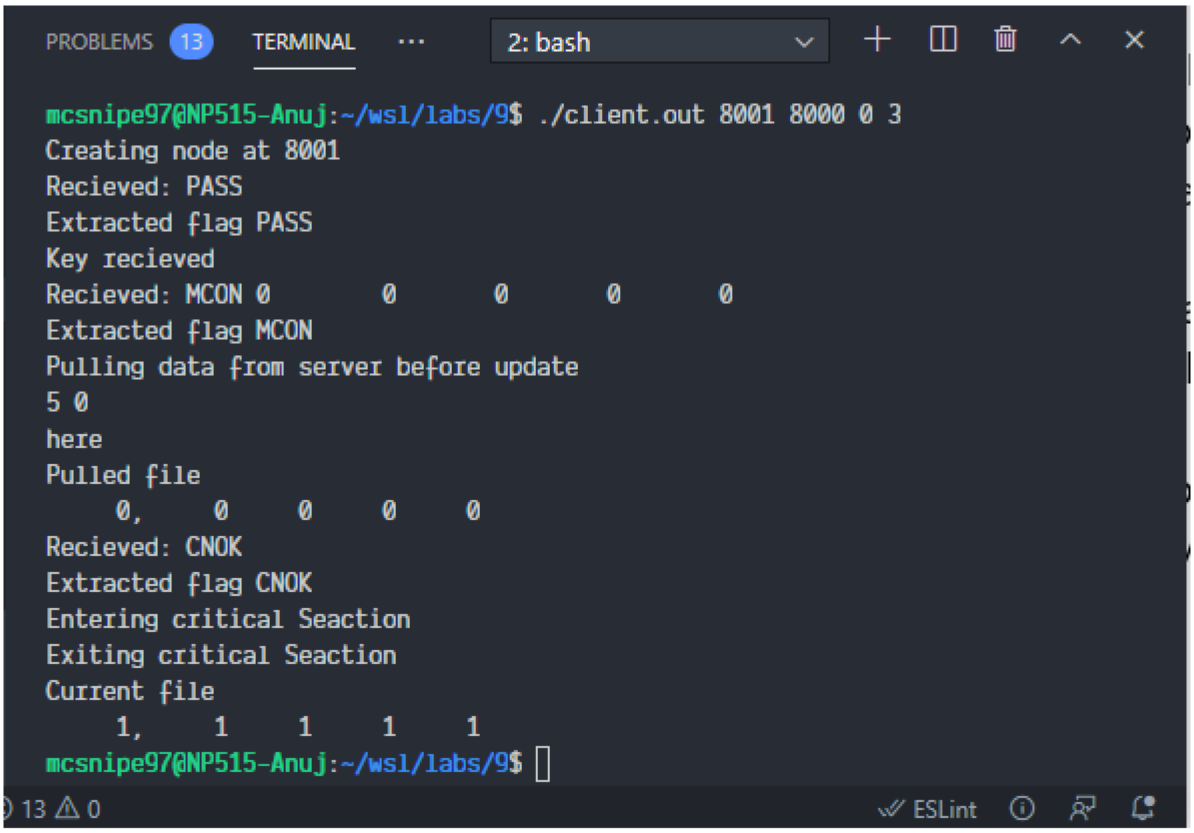
Output:

Server:



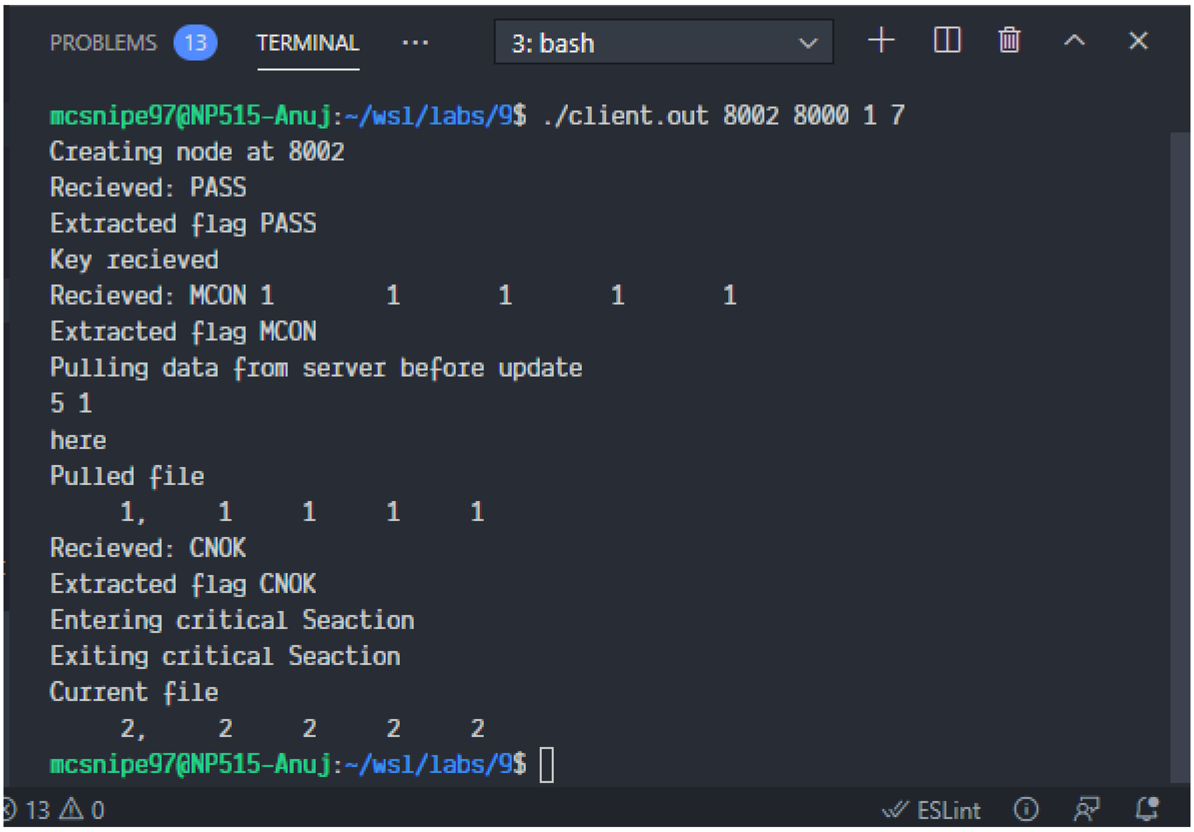
```
mcsnipe97@NP515-Anuj:~/wsl/labs/9$ ./server.out 8000 3 8001 8002 8003
Creating node at 8000
Recieved: KEYR 8001
Extracted flag KEYR
Extracted dest 8001
Recieved: MCON 8001
Extracted flag MCON
Forcing consistency
Recieved: DONE 1        1       1       1       1
Extracted flag DONE
Key released
Recieved: KEYR 8002
Extracted flag KEYR
Extracted dest 8002
Recieved: MCON 8002
Extracted flag MCON
Forcing consistency
Recieved: DONE 2        2       2       2       2
Extracted flag DONE
Key released
Recieved: KEYR 8003
Extracted flag KEYR
Extracted dest 8003
Recieved: MCON 8003
Extracted flag MCON
Forcing consistency
Recieved: DONE 3        3       3       3       3
Extracted flag DONE
Key released
```

Client1:



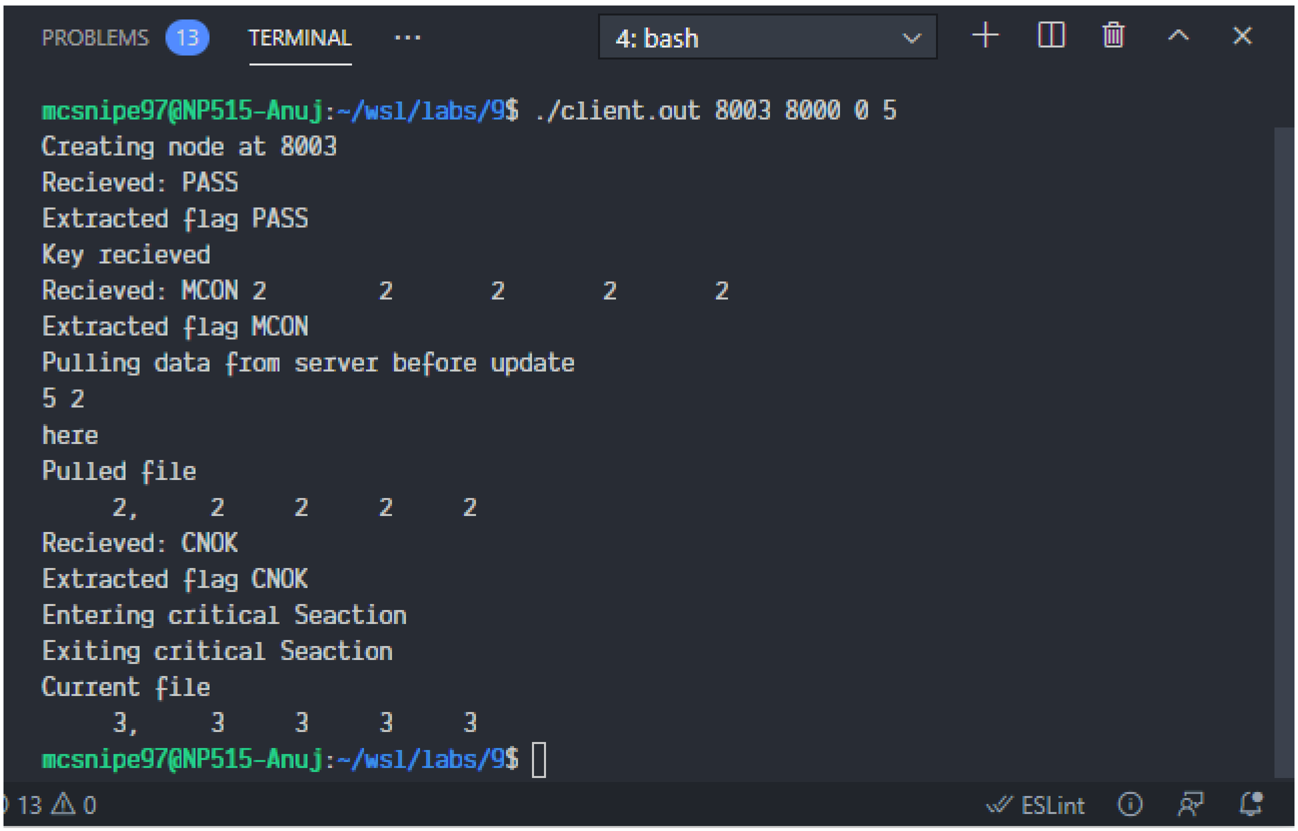
```
mcsnipe97@NP515-Anuj:~/wsl/labs/9$ ./client.out 8001 8000 0 3
Creating node at 8001
Recieved: PASS
Extracted flag PASS
Key recieved
Recieved: MCON 0        0       0       0       0
Extracted flag MCON
Pulling data from server before update
5 0
here
Pulled file
     0,     0     0     0     0
Recieved: CNOK
Extracted flag CNOK
Entering critical Seaction
Exiting critical Seaction
Current file
     1,     1     1     1     1
mcsnipe97@NP515-Anuj:~/wsl/labs/9$
```

Client2:



```
mcsnipe97@NP515-Anuj:~/wsl/labs/9$ ./client.out 8002 8000 1 7
Creating node at 8002
Recieved: PASS
Extracted flag PASS
Key recieved
Recieved: MCON 1        1       1       1       1
Extracted flag MCON
Pulling data from server before update
5 1
here
Pulled file
     1,     1     1     1     1
Recieved: CNOK
Extracted flag CNOK
Entering critical Seaction
Exiting critical Seaction
Current file
     2,     2     2     2     2
mcsnipe97@NP515-Anuj:~/wsl/labs/9$
```

Client3:



```
mcsnipe97@NP515-Anuj:~/wsl/labs/9$ ./client.out 8003 8000 0 5
Creating node at 8003
Recieved: PASS
Extracted flag PASS
Key recieved
Recieved: MCON 2        2       2       2       2
Extracted flag MCON
Pulling data from server before update
5 2
here
Pulled file
     2,     2     2     2     2
Recieved: CNOK
Extracted flag CNOK
Entering critical Seaction
Exiting critical Seaction
Current file
     3,     3     3     3     3
mcsnipe97@NP515-Anuj:~/wsl/labs/9$
```

Conclusion: We successfully implemented entry eventual consistency between processes with mutual exclusive update replicated datastore.