

## Program – 8

**AIM:** To implement Entry consistency between processes with mutual exclusive update replicated datastore.

### Introduction and 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.

## Program – 8

### Code

---

#### Server

```
1  #include <sys/socket.h>
2  #include <netinet/in.h>
3  #include <arpa/inet.h>
4  #include <stdio.h>
5  #include <stdlib.h>
6  #include <unistd.h>
7  #include <errno.h>
8  #include <string.h>
9  #include <sys/types.h>
10 #include <time.h>
11 #include <string.h>
12 #define MSG_CONFIRM 0
13
14 #define TRUE 1
15 #define FALSE 0
16 #define ML 1024
17 #define MPROC 32
18
19 typedef struct Resource
20 {
21     int a;
22     int b;
23     int c;
24     int d;
25     int e;
26 } Resource;
27
28 void serealize(Resource S, char output[ML])
29 {
30     sprintf(output, "MCON %d\t%d\t%d\t%d\t%d\t", S.a, S.b, S.c, S.d,
31 S.e);
32 }
33
34 Resource unserealize(char input[ML])
35 {
36     char temp[ML];
37     int ix = 0, itr = 5;
38     Resource S;
39     for(itr; input[itr] != '\t'; itr +=1)
40         temp[ix++] = input[itr];
41     temp[ix] = '\0';
42     S.a = atoi(temp);
43     ix = 0;
44
45     for(itr = itr + 1; input[itr] != '\t'; itr +=1)
46         temp[ix++] = input[itr];
47     temp[ix] = '\0';
48     S.b = atoi(temp);
49     ix = 0;
50
51     for(itr = itr + 1; input[itr] != '\t'; itr +=1)
52         temp[ix++] = input[itr];
53     temp[ix] = '\0';
```

## Program – 8

```
54     S.c = atoi(temp);
55     ix = 0;
56
57     for(itr = itr + 1; input[itr] != '\t'; itr +=1)
58         temp[ix++] = input[itr];
59     temp[ix] = '\0';
60     S.d = atoi(temp);
61     ix = 0;
62
63     for(itr = itr + 1; input[itr] != '\t'; itr +=1)
64         temp[ix++] = input[itr];
65     temp[ix] = '\0';
66     S.e = atoi(temp);
67     ix = 0;
68     return S;
69 }
70
71 int connect_to_port(int connect_to)
72 {
73     int sock_id;
74     int opt = 1;
75     struct sockaddr_in server;
76     if ((sock_id = socket(AF_INET, SOCK_DGRAM, 0)) < 0)
77     {
78         perror("unable to create a socket");
79         exit(EXIT_FAILURE);
80     }
81     setsockopt(sock_id, SOL_SOCKET, SO_REUSEADDR, (const void
82 *)&opt, sizeof(int));
83     memset(&server, 0, sizeof(server));
84     server.sin_family = AF_INET;
85     server.sin_addr.s_addr = INADDR_ANY;
86     server.sin_port = htons(connect_to);
87
88     if (bind(sock_id, (const struct sockaddr *)&server,
89 sizeof(server)) < 0)
90     {
91         perror("unable to bind to port");
92         exit(EXIT_FAILURE);
93     }
94     return sock_id;
95 }
96
97 void send_to_id(int to, int from, char message[ML])
98 {
99     struct sockaddr_in cl;
100     memset(&cl, 0, sizeof(cl));
101
102     cl.sin_family = AF_INET;
103     cl.sin_addr.s_addr = INADDR_ANY;
104     cl.sin_port = htons(to);
105
106     sendto(
107         from, \
108         (const char *)message, \
109         strlen(message), \
110         MSG_CONFIRM, \
```

## Program – 8

```
111     (const struct sockaddr *)&cl, \  
112     sizeof(cl));  
113 }  
114  
115  
116 void make_consistent(int from, int procs[], int n_procs, Resource S)  
117 {  
118     char message[ML];  
119     int i;  
120     serealize(S, message);  
121     for (i = 0; i < n_procs; i++)  
122         send_to_id(procs[i], from, message);  
123 }  
124  
125 int main(int argc, char* argv[])  
126 {  
127     int self = atoi(argv[1]);  
128     int n_procs = atoi(argv[2]);  
129     int itr, ix = 0;  
130     int procs[MPROC];  
131     int key_avail = 1;  
132     int dest;  
133     int sock_id, len, n;  
134     char buffer[ML], msg[ML];  
135     char flag[256], p_id[256];  
136     struct sockaddr_in from;  
137  
138     Resource S = {0, 0, 0, 0, 0};  
139  
140     for(itr = 0; itr < n_procs; itr++)  
141         procs[itr] = atoi(argv[3 + itr]);  
142  
143     printf("Creating node at %d\n", self);  
144     sock_id = connect_to_port(self);  
145  
146     while(TRUE)  
147     {  
148         memset(&from, 0, sizeof(from));  
149         n = recvfrom(sock_id, (char *)buffer, ML, MSG_WAITALL,  
150 (struct sockaddr *)&from, &len);  
151         buffer[n] = '\0';  
152         printf("Recieved: %s\n", buffer);  
153  
154         for(itr = 0; itr < 4; itr++)  
155             flag[itr] = buffer[itr];  
156         flag[itr] = '\0';  
157         printf("Extracted flag %s\n", flag);  
158         // process asks for key  
159         if (strcmp(flag, "KEYR") == 0)  
160         {  
161             ix = 0;  
162             for (itr = 5; itr < 9; itr++)  
163                 p_id[ix++] = buffer[itr];  
164             p_id[ix] = '\0';  
165             dest = atoi(p_id);  
166             printf("Extracted dest %d\n", dest);  
167             if (key_avail)
```

## Program – 8

```
168         {
169             send_to_id(dest, sock_id, "PASS");
170             key_avail = 0;
171         }
172         else
173         {
174             send_to_id(dest, sock_id, "WAIT");
175         }
176     }
177     // process releases key
178     else if (strcmp(flag, "DONE") == 0)
179     {
180         printf("Key released\n");
181         S = unserealize(buffer);
182         key_avail = 1;
183     }
184     // process calls for consistency
185     else if (strcmp(flag, "MCON") == 0)
186     {
187         printf("Forcing consistency \n");
188         make_consistent(sock_id, procs, n_procs, S);
189         for (itr = 5; itr < 9; itr++)
190             p_id[5-itr] = buffer[itr];
191         p_id[5-itr] = '\0';
192         dest = atoi(p_id);
193         send_to_id(dest, sock_id, "CNOK");
194     }
195 }
196 }
```

## Client

```
1  #include <sys/socket.h>
2  #include <netinet/in.h>
3  #include <arpa/inet.h>
4  #include <stdio.h>
5  #include <stdlib.h>
6  #include <unistd.h>
7  #include <errno.h>
8  #include <string.h>
9  #include <sys/types.h>
10 #include <time.h>
11 #include <string.h>
12 #define MSG_CONFIRM 0
13
14 #define TRUE 1
15 #define FALSE 0
16 #define ML 1024
17 #define MPROC 32
18
19 typedef struct Resource
20 {
21     int a;
22     int b;
23     int c;
24     int d;
```

## Program – 8

```
25     int e;  
26 } Resource;  
27  
28 void serealize(Resource S, char output[ML])  
29 {  
30     sprintf(output, "DONE %d\t%d\t%d\t%d\t%d\t", S.a, S.b, S.c, S.d,  
31 S.e);  
32 }  
33  
34 Resource unserealize(char input[ML])  
35 {  
36     char temp[ML];  
37     int ix = 0, itr = 5;  
38     Resource S;  
39     for(itr; input[itr] != '\t'; itr +=1)  
40     {  
41         printf("%d %c\n", itr, input[itr]);  
42         temp[ix++] = input[itr];  
43     }  
44     temp[ix] = '\0';  
45     S.a = atoi(temp);  
46     ix = 0;  
47     printf("here\n");  
48     for(itr = itr + 1; input[itr] != '\t'; itr +=1)  
49         temp[ix++] = input[itr];  
50     temp[ix] = '\0';  
51     S.b = atoi(temp);  
52     ix = 0;  
53  
54     for(itr = itr + 1; input[itr] != '\t'; itr +=1)  
55         temp[ix++] = input[itr];  
56     temp[ix] = '\0';  
57     S.c = atoi(temp);  
58     ix = 0;  
59  
60     for(itr = itr + 1; input[itr] != '\t'; itr +=1)  
61         temp[ix++] = input[itr];  
62     temp[ix] = '\0';  
63     S.d = atoi(temp);  
64     ix = 0;  
65  
66     for(itr = itr + 1; input[itr] != '\t'; itr +=1)  
67         temp[ix++] = input[itr];  
68     temp[ix] = '\0';  
69     S.e = atoi(temp);  
70     ix = 0;  
71     return S;  
72 }  
73  
74 int connect_to_port(int connect_to)  
75 {  
76     int sock_id;  
77     int opt = 1;  
78     struct sockaddr_in server;  
79     if ((sock_id = socket(AF_INET, SOCK_DGRAM, 0)) < 0)  
80     {  
81         perror("unable to create a socket");
```

## Program – 8

```
82     exit(EXIT_FAILURE);
83 }
84     setsockopt(sock_id, SOL_SOCKET, SO_REUSEADDR, (const void
85 *)&opt, sizeof(int));
86     memset(&server, 0, sizeof(server));
87     server.sin_family = AF_INET;
88     server.sin_addr.s_addr = INADDR_ANY;
89     server.sin_port = htons(connect_to);
90
91     if (bind(sock_id, (const struct sockaddr *)&server,
92 sizeof(server)) < 0)
93     {
94         perror("unable to bind to port");
95         exit(EXIT_FAILURE);
96     }
97     return sock_id;
98 }
99
100 void send_to_id(int to, int from, char message[ML])
101 {
102     struct sockaddr_in cl;
103     memset(&cl, 0, sizeof(cl));
104
105     cl.sin_family = AF_INET;
106     cl.sin_addr.s_addr = INADDR_ANY;
107     cl.sin_port = htons(to);
108
109     sendto(
110         from, \
111         (const char *)message, \
112         strlen(message), \
113         MSG_CONFIRM, \
114         (const struct sockaddr *)&cl, \
115         sizeof(cl));
116 }
117
118 void request_key(int server, int sock_id, int a)
119 {
120     char msg[256];
121     sprintf(msg, "KEYR %d", a);
122     send_to_id(server, sock_id, msg);
123 }
124
125 int main(int argc, char* argv[])
126 {
127     int self = atoi(argv[1]);
128     int server = atoi(argv[2]);
129     int start = atoi(argv[3]);
130     int udelay = atoi(argv[4]);
131     int itr;
132     int dest;
133     int key = 0;
134
135     int sock_id, len, n;
136     char buffer[ML], msg[ML];
137     char flag[256], p_id[256];
138     struct sockaddr_in from;
```

## Program – 8

```
139     Resource S = {0, 0, 0, 0, 0};
140     printf("Creating node at %d\n", self);
141     sock_id = connect_to_port(self);
142     if (start)
143     {
144         request_key(server, sock_id, self);
145     }
146     else
147     {
148         sleep(udelay);
149         request_key(server, sock_id, self);
150     }
151     while(TRUE)
152     {
153         // sleep(udelay);
154         memset(&from, 0, sizeof(from));
155         n = recvfrom(sock_id, (char *)buffer, ML, MSG_WAITALL,
156 (struct sockaddr *)&from, &len);
157         buffer[n] = '\0';
158         printf("Recieved: %s\n", buffer);
159
160         for(itr = 0; itr < 4; itr++)
161             flag[itr] = buffer[itr];
162         flag[itr] = '\0';
163         printf("Extracted flag %s\n", flag);
164
165         // server denies key
166         if (strcmp(flag, "WAIT") == 0)
167         {
168             sleep(udelay);
169             request_key(server, sock_id, self);
170         }
171         // process releases key
172         else if (strcmp(flag, "PASS") == 0)
173         {
174             printf("Key recieved\n");
175             key = 1;
176             sprintf(msg, "MCON %d", self);
177             send_to_id(server, sock_id, msg);
178         }
179         // process calls for consistency
180         else if (strcmp(flag, "MCON") == 0)
181         {
182             printf("Pulling data from server before update\n");
183             S = unserealize(buffer);
184             printf("Pulled file\n %5d, %5d %5d %5d %5d\n", S.a, S.b,
185 S.c, S.d, S.e);
186         }
187         else if (strcmp(flag, "CNOK") == 0 && key)
188         {
189             printf("Entering critical Seaction\n");
190             S.a++;
191             S.b++;
192             S.c++;
193             S.d++;
194             S.e++;
195             printf("Exiting critical Seaction\n");
```



## Program – 8

```
196         printf("Current file\n %5d, %5d %5d %5d %5d\n", S.a,  
197 S.b, S.c, S.d, S.e);  
198         serealize(S, msg);  
199         send_to_id(server, sock_id, msg);  
200         exit(EXIT_SUCCESS);  
201     }  
202 }  
203 return 0;  
204 }
```

## Results and Outputs:

```
DiSLAB — I9s 8000 3 8001 8002 8003 — 71x29  
Anurags-MacBook-Air:DiSLAB jarvis$ ./outs/I9s 8000 3 8001 8002 8003  
Creating node at 8000  
Recieved: KEYR 8002  
Extracted flag KEYR  
Extracted dest 8002  
Recieved: MCON 8002  
Extracted flag MCON  
Forcing consistency  
Recieved: DONE 1      1      1      1      1  
Extracted flag DONE  
Key released  
Recieved: KEYR 8001  
Extracted flag KEYR  
Extracted dest 8001  
Recieved: MCON 8001  
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  
Anurags-MacBook-Air:DiSLAB jarvis$  
  
DiSLAB — -bash — 71x27  
Anurags-MacBook-Air:DiSLAB jarvis$ ./outs/I9c 8001 8000 0 3  
Creating node at 8001  
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  
Anurags-MacBook-Air:DiSLAB jarvis$  
  
DiSLAB — -bash — 71x25  
Anurags-MacBook-Air:DiSLAB jarvis$ ./outs/I9c 8002 8000 1 7  
Creating node at 8002  
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  
Anurags-MacBook-Air:DiSLAB jarvis$  
  
DiSLAB — -bash — 71x27  
Anurags-MacBook-Air:DiSLAB jarvis$ ./outs/I9c 8003 8000 0 5  
Creating node at 8003  
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: 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  
Anurags-MacBook-Air:DiSLAB jarvis$
```

## Findings and Learnings:

1. We successfully implemented Entry Consistency.