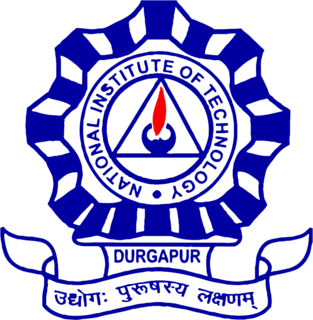
# Assignment **3** Of

**Network & Distributed System Lab (CS2051)**

**Masters of Technology in Computer Science And Engineering**

**submitted to**

**Dr Suvrojit Das Associate Professor Dept. of CSE**

****

**National Institute of Technology, Durgapur**

submitted by Arghya Bandyopadhyay RollNo. 20CS4103

29 June 2021

## **Objective:Write a program to implement total ordering of events using Lamport's Logical Clock. Use only two events: that of Sending and Receiving messages**

## **Solution:**

## // Implementation of Lamport's Logical Clock

## #include <stdio.h>

## #include <stdlib.h>

## #include <unistd.h>

## #include <string.h>

## #include <sys/socket.h>

## #include <arpa/inet.h>

## #define BUFFER\_SIZE 32

## #define PROCESS\_A\_PORT 5000

## #define PROCESS\_B\_PORT 6000

## #define PROCESS\_C\_PORT 7000

## /\* defining a macro to get max value out of two values \*/

## #define MAX(X, Y) (((X) > (Y)) ? (X) : (Y))

## struct message {

## int logical\_clock;

## char msg\_content[BUFFER\_SIZE];

## };

## int err\_handler(int n, char \* error) {

## if(n < 0) {

## perror(error);

## exit(EXIT\_FAILURE);

## }

## return n;

## };

## int send\_message(int sock, struct message \* msg, size\_t size, int flag,

## struct sockaddr\_in \* recvr, socklen\_t addrsize) {

## 

## msg->logical\_clock += 1;

## /\*send the UDP packet\*/

## return sendto(sock, msg, size, flag, recvr, addrsize);

## };

## int recv\_message(int sock, struct message \* rcvd\_msg, size\_t size, struct message \* selfmsg) {

## selfmsg->logical\_clock += 1;

## /\*receive the UDP packet\*/

## int recvd\_bytes = recvfrom(sock, rcvd\_msg, size, 0, NULL, NULL);

## 

## selfmsg->logical\_clock = MAX(selfmsg->logical\_clock, rcvd\_msg->logical\_clock + 1);

## return recvd\_bytes;

## }

## void processA(int sock) {

## struct message mesg\_a = {0, "A:Hello World"}; /\*initialising the message\*/

## /\*Initial vlaue of logical\_clock for each process is zero.\*/

## printf("\nInitial logical clock value of process A: %d\n", mesg\_a.logical\_clock);

## struct message recvd\_from\_b;

## struct sockaddr\_in procB\_address;

## procB\_address.sin\_family = AF\_INET;

## procB\_address.sin\_port = htons(PROCESS\_B\_PORT);

## procB\_address.sin\_addr.s\_addr = INADDR\_ANY;

## socklen\_t size\_addr = sizeof(struct sockaddr\_in);

## printf("\nEvent 1 of Process A: Send msg to process B\n");

## /\*Sending the message to process B\*/

## err\_handler(send\_message(sock, &mesg\_a, sizeof(mesg\_a), 0,

## (struct sockaddr\_in \*) &procB\_address, size\_addr), "SendError");

## printf("Logical clock value of Process A after event 1: %d\n", mesg\_a.logical\_clock);

## printf("\nEvent 2 of Process A: receive msg from process B\n");

## /\* Waiting for a message to be received \*/

## printf("Waiting to receive message...\n");

## err\_handler(recv\_message(sock, &recvd\_from\_b, sizeof(struct message), &mesg\_a),

## "ReceivingError");

## printf("Logical clock value of Process A after event 2: %d\n", mesg\_a.logical\_clock);

## };

## void processB(int sock) {

## struct message mesg\_b = {0, "B:Hello World World"}; /\*initialising the message\*/

## /\*Initial vlaue of logical\_clock for each process is zero.\*/

## printf("\nInitial logical clock value of process B: %d\n", mesg\_b.logical\_clock);

## struct message recvd\_msg;

## struct sockaddr\_in recvr\_address;

## /\*setting up address for any other process running in the same net\*/

## recvr\_address.sin\_family = AF\_INET;

## recvr\_address.sin\_addr.s\_addr = INADDR\_ANY;

## /\*Waiting for the message from process A and C\*/

## /\* loop runs for 2 times for two time waiting for receive by process B\*/

## for(int i = 0; i < 2; i++) {

## printf("\nEvent %d of Process B: recv message\n", i + 1);

## printf("Waiting to receive message...\n");

## err\_handler(recv\_message(sock, &recvd\_msg, sizeof(struct message), &mesg\_b),

## "ReceivingError");

## printf("Logical clock value of Process B after event %d: %d\n",

## i + 1, mesg\_b.logical\_clock);

## }

## /\* Now process B sends message to process A\*/

## /\* setting port address for process A \*/

## recvr\_address.sin\_port = htons(PROCESS\_A\_PORT);

## printf("\nEvent 3 of Process B: Send msg to process A\n");

## err\_handler(send\_message(sock, &mesg\_b, sizeof(mesg\_b), 0,

## (struct sockaddr\_in \*) &recvr\_address, sizeof(recvr\_address)), "SendingError");

## printf("Logical clock value of Process B after event 3: %d\n", mesg\_b.logical\_clock);

## /\* Now process B sends message to process C\*/

## /\* setting port address for process C \*/

## recvr\_address.sin\_port = htons(PROCESS\_C\_PORT);

## printf("\nEvent 4 of Process B: Send msg to process C\n");

## err\_handler(send\_message(sock, &mesg\_b, sizeof(mesg\_b), 0,

## (struct sockaddr\_in \*) &recvr\_address, sizeof(recvr\_address)), "SendingError");

## printf("Logical clock value of Process B after event 4: %d\n", mesg\_b.logical\_clock);

## };

## void processC(int sock) {

## /\*

## Sends out a message to process B.

## Receives a message from process B.

## \*/

## struct message mesg\_c = {0, "C:Hello World"}; /\*initialising the message\*/

## /\*Initial vlaue of logical\_clock for each process is zero.\*/

## printf("\nInitial logical clock value of process C: %d\n", mesg\_c.logical\_clock);

## struct message recvd\_from\_b;

## struct sockaddr\_in procB\_address;

## procB\_address.sin\_family = AF\_INET;

## procB\_address.sin\_port = htons(PROCESS\_B\_PORT);

## procB\_address.sin\_addr.s\_addr = INADDR\_ANY;

## socklen\_t size\_addr = sizeof(struct sockaddr\_in);

## printf("\nEvent 1 of Process C: Send msg to process B\n");

## /\*Sending the message to process B\*/

## err\_handler(send\_message(sock, &mesg\_c, sizeof(mesg\_c), 0,

## (struct sockaddr\_in \*) &procB\_address, size\_addr), "SendError");

## printf("Logical clock value of Process C after event 1: %d\n", mesg\_c.logical\_clock);

## printf("\nEvent 2 of Process C: receive msg from process B\n");

## /\* Waiting for a message to be received \*/

## printf("Waiting to receive message...\n");

## err\_handler(recv\_message(sock, &recvd\_from\_b, sizeof(struct message), &mesg\_c),

## "ReceivingError");

## printf("Logical clock value of Process C after event 2: %d\n", mesg\_c.logical\_clock);

## }

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

## short process\_id;

## /\* to provide different ports to the 3 processes\*/

## int process\_port[3] = {PROCESS\_A\_PORT, PROCESS\_B\_PORT, PROCESS\_C\_PORT};

## /\* the program requires an argument giving process id on comand-line.

## Process id could have any value from set {1, 2, 3}. We are simulating the

## program on 3 processes. If the argument is not given then instruct and exit. \*/

## if(argc != 2) {

## printf("\nUsage: ./process -p<1|2|3>\n\n");

## exit(EXIT\_FAILURE);

## }

## int sock\_fd; /\*file descriptor of socket\*/

## int port; /\* contains the assigned port to the current process \*/

## /\* address of the current process.\*/

## struct sockaddr\_in process\_address;

## socklen\_t size\_of\_addr = sizeof(struct sockaddr\_in);

## /\* zero out process\_address \*/

## memset(&process\_address, 0, size\_of\_addr);

## /\*ensure process\_id argument is in proper format\*/

## if(strlen(argv[1]) != 3) {

## printf("\nUsage: ./lamportclock -p<1|2|3>\n\n");

## exit(EXIT\_FAILURE);

## }

## process\_id = atoi(argv[1] + 2); /\*get 3rd character from the argument as process\_id\*/

## /\* if process\_id value is zero, then atoi() could not convert the given

## argument to a valid integer. \*/

## if(process\_id == 0 || process\_id > 3) {

## printf("\nInvalid Argument.");

## exit(EXIT\_FAILURE);

## }

## port = process\_port[process\_id - 1]; /\*selecting one port from the array for current process\*/

## /\*Create a UDP socket\*/

## sock\_fd = err\_handler(socket(AF\_INET, SOCK\_DGRAM, 0), "SocketCreationError");

## printf("\nSocket created.\n");

## /\*setting up the process address\*/

## process\_address.sin\_family = AF\_INET;

## process\_address.sin\_port = htons(port);

## process\_address.sin\_addr.s\_addr = INADDR\_ANY;

## /\* We will bind the current process to above set-up address \*/

## err\_handler(bind(sock\_fd, (struct sockaddr\_in \*) &process\_address, size\_of\_addr), "BindError");

## /\*

## According to process\_id we will invoke respective functions.

## Only one of these functions will run, for any instance of this program.

## \*/

## (process\_id == 1) ? processA(sock\_fd) :

## ((process\_id == 2) ? processB(sock\_fd) : processC(sock\_fd));

## return 0;

## }

## 

## 