**ASSIGNMENT – 2**

**i) There are two nodes A and B. Write a program to determine CPU load of node B from node A.**

// node\_b\_cpu\_load\_server.c

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <sys/sysinfo.h>

#define PORT 12345

#define BUFFER\_SIZE 256

void get\_cpu\_load(char \*buffer) {

struct sysinfo info;

if (sysinfo(&info) == 0) {

snprintf(buffer, BUFFER\_SIZE, "Load average: %f, %f, %f",

info.loads[0] / (float)(1 << SI\_LOAD\_SHIFT),

info.loads[1] / (float)(1 << SI\_LOAD\_SHIFT),

info.loads[2] / (float)(1 << SI\_LOAD\_SHIFT));

} else {

snprintf(buffer, BUFFER\_SIZE, "Failed to get CPU load.");

}

}

int main() {

int server\_fd, new\_socket;

struct sockaddr\_in address;

int addrlen = sizeof(address);

char buffer[BUFFER\_SIZE] = {0};

// Create socket

if ((server\_fd = socket(AF\_INET, SOCK\_STREAM, 0)) == 0) {

perror("Socket creation error");

exit(EXIT\_FAILURE);

}

// Set up the server address structure

address.sin\_family = AF\_INET;

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

address.sin\_port = htons(PORT);

// Bind the socket to the address and port

if (bind(server\_fd, (struct sockaddr \*)&address, sizeof(address)) < 0) {

perror("Bind failed");

close(server\_fd);

exit(EXIT\_FAILURE);

}

// Listen for incoming connections

if (listen(server\_fd, 3) < 0) {

perror("Listen failed");

close(server\_fd);

exit(EXIT\_FAILURE);

}

printf("Waiting for connection...\n");

// Accept a connection

if ((new\_socket = accept(server\_fd, (struct sockaddr \*)&address, (socklen\_t\*)&addrlen)) < 0) {

perror("Accept failed");

close(server\_fd);

exit(EXIT\_FAILURE);

}

// Get CPU load and send it to the client

get\_cpu\_load(buffer);

send(new\_socket, buffer, strlen(buffer), 0);

printf("CPU load sent to the client: %s\n", buffer);

close(new\_socket);

close(server\_fd);

return 0;

}



// node\_a\_cpu\_load\_client.c

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#define PORT 12345

#define BUFFER\_SIZE 256

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

int sock = 0;

struct sockaddr\_in serv\_addr;

char buffer[BUFFER\_SIZE] = {0};

if (argc != 2) {

fprintf(stderr, "Usage: %s <server\_ip>\n", argv[0]);

return EXIT\_FAILURE;

}

// Create socket

if ((sock = socket(AF\_INET, SOCK\_STREAM, 0)) < 0) {

perror("Socket creation error");

return EXIT\_FAILURE;

}

serv\_addr.sin\_family = AF\_INET;

serv\_addr.sin\_port = htons(PORT);

// Convert IP address from text to binary form

if (inet\_pton(AF\_INET, argv[1], &serv\_addr.sin\_addr) <= 0) {

perror("Invalid address/ Address not supported");

close(sock);

return EXIT\_FAILURE;

}

// Connect to the server

if (connect(sock, (struct sockaddr \*)&serv\_addr, sizeof(serv\_addr)) < 0) {

perror("Connection failed");

close(sock);

return EXIT\_FAILURE;

}

// Read the CPU load from the server

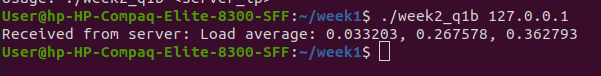
read(sock, buffer, sizeof(buffer));

printf("Received from server: %s\n", buffer);

close(sock);

return 0;

}



**ii) Write a server C program using shared memory and semaphore (server increments counter between sem\_wait() and sem\_post()). Create shared memory using mmap.**

// shared\_memory\_server.c

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/mman.h>

#include <fcntl.h>

#include <sys/stat.h>

#include <semaphore.h>

#include <sys/types.h>

#define SHARED\_MEM\_NAME "/my\_shm"

#define SEM\_NAME "/my\_sem"

#define SHM\_SIZE sizeof(int)

int main() {

int fd;

int \*counter;

sem\_t \*sem;

// Create or open the shared memory

fd = shm\_open(SHARED\_MEM\_NAME, O\_CREAT | O\_RDWR, 0666);

if (fd == -1) {

perror("shm\_open");

exit(EXIT\_FAILURE);

}

ftruncate(fd, SHM\_SIZE);

// Map shared memory

counter = mmap(NULL, SHM\_SIZE, PROT\_READ | PROT\_WRITE, MAP\_SHARED, fd, 0);

if (counter == MAP\_FAILED) {

perror("mmap");

exit(EXIT\_FAILURE);

}

// Initialize the counter

\*counter = 0;

// Create or open semaphore

sem = sem\_open(SEM\_NAME, O\_CREAT, 0666, 1);

if (sem == SEM\_FAILED) {

perror("sem\_open");

exit(EXIT\_FAILURE);

}

// Increment the counter with semaphore protection

while (1) {

sem\_wait(sem);

(\*counter)++;

printf("Counter incremented to %d\n", \*counter);

sem\_post(sem);

sleep(1);

}

// Clean up

sem\_close(sem);

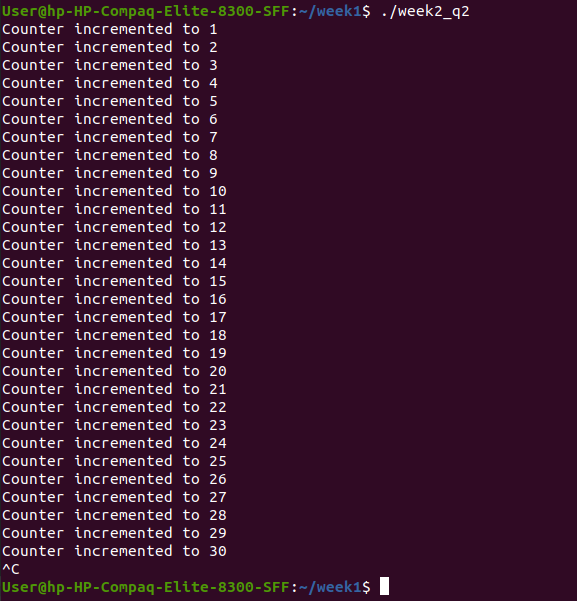
sem\_unlink(SEM\_NAME);

munmap(counter, SHM\_SIZE);

shm\_unlink(SHARED\_MEM\_NAME);

return 0;

}



**iii) Write a client C program that reads counter value between sem\_wait() and sem\_post(). Access shared memory using open().**

// shared\_memory\_client.c

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/mman.h>

#include <fcntl.h>

#include <sys/stat.h>

#include <semaphore.h>

#define SHARED\_MEM\_NAME "/my\_shm"

#define SEM\_NAME "/my\_sem"

#define SHM\_SIZE sizeof(int)

int main() {

int fd;

int \*counter;

sem\_t \*sem;

// Open the shared memory

fd = shm\_open(SHARED\_MEM\_NAME, O\_RDWR, 0666);

if (fd == -1) {

perror("shm\_open");

exit(EXIT\_FAILURE);

}

// Map shared memory

counter = mmap(NULL, SHM\_SIZE, PROT\_READ, MAP\_SHARED, fd, 0);

if (counter == MAP\_FAILED) {

perror("mmap");

exit(EXIT\_FAILURE);

}

// Open semaphore

sem = sem\_open(SEM\_NAME, 0);

if (sem == SEM\_FAILED) {

perror("sem\_open");

exit(EXIT\_FAILURE);

}

// Read the counter value

while (1) {

sem\_wait(sem);

printf("Current counter value: %d\n", \*counter);

sem\_post(sem);

sleep(1);

}

// Clean up

sem\_close(sem);

munmap(counter, SHM\_SIZE);

return 0;

}

