**UNIX Programming**

**Lab Manual**

**PROGRAM-1**

**Check POSIX runtime limits**

**Write a C/C++ POSIX compliant program to check the following limits:**

**1. Number of clock ticks**

**2. Max number of child processes**

**3. Max path length**

**4. Max number of characters in a file name**

**5. Max number of open files/processes**

#include<stdio.h>

#include<unistd.h>

#include<limits.h>

//#define \_POSIX\_SOURCE

//#define \_POSIX\_C\_SOURCE 199309L

int main()

{

printf("Runtime values\n");

printf("The max number of clock ticks : %ld\n",sysconf(\_SC\_CLK\_TCK));

printf("The max runtime child processes : %ld\n",sysconf(\_SC\_CHILD\_MAX));

printf("The max runtime path length :%ld\n",pathconf("usp1.cpp",\_PC\_PATH\_MAX));

printf("The max characters in a file name :%ld\n",pathconf("usp1.cpp",\_PC\_NAME\_MAX));

printf("The max number of opened files : %ld\n",sysconf(\_SC\_OPEN\_MAX));

return 0;

}

**2a. Copy of a file using system calls.**

#include <stdio.h>

#include <stdlib.h> // For exit()

int main()

{

FILE \*fptr1, \*fptr2;

char filename[100], c;

printf("Enter the filename to open for reading \n");

scanf("%s", filename);

// Open one file for reading

fptr1 = fopen(filename, "r");

if (fptr1 == NULL)

{

printf("Cannot open file %s \n", filename);

exit(0);

}

printf("Enter the filename to open for writing \n");

scanf("%s", filename);

// Open another file for writing

fptr2 = fopen(filename, "w");

if (fptr2 == NULL)

{

printf("Cannot open file %s \n", filename);

exit(0);

}

// Read contents from file

c = fgetc(fptr1);

while (c != EOF)

{

fputc(c, fptr2);

c = fgetc(fptr1);

}

printf("\nContents copied to %s\n", filename);

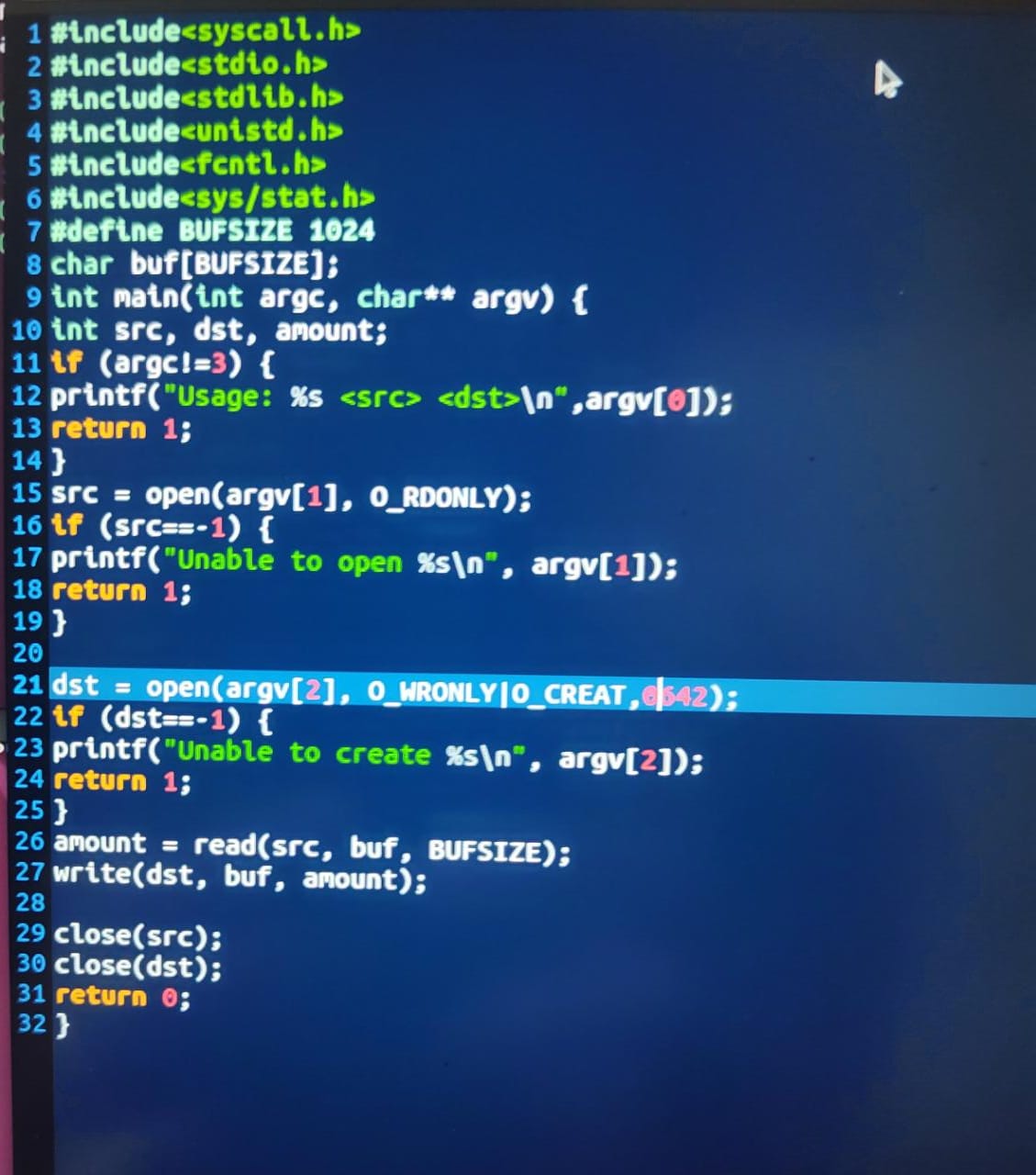
fclose(fptr1);

fclose(fptr2);

return 0;

}

2a - easier program



**2b Output the contents of its Environment list.**

#include<stdio.h>

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

{

int i;

char \*\*ptr;

extern char \*\*environ;

for( ptr = environ; \*ptr != 0; ptr++ ) /\*echo all env strings\*/

printf("%s\n", \*ptr);

return 0;

}

**3a.Emulate the UNIX ln command**

#include<stdio.h>

#include<sys/types.h>

#include<unistd.h>

#include<string.h>

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

{

if(argc < 3 || argc > 4 || (argc == 4 && strcmp(argv[1],"-s")))

{

printf("Usage: ./a.out [-s] <org\_file> <new\_link>\n");

return 1;

}

if(argc == 4)

{

if((symlink(argv[2], argv[3])) == -1)

printf("Cannot create symbolic link\n") ;

else printf("Symbolic link created\n") ;

}

else

{

if((link(argv[1], argv[2])) == -1)

printf("Cannot create hard link\n") ;

else

printf("Hard link created\n") ;

}

return 0;

}

**3b. Create a child from parent process using fork() and counter counts till 5 in both processes and display.**

#include<unistd.h>

#include<stdio.h>

#include<stdlib.h>

#include <sys/types.h>

#include <sys/wait.h>

int main()

{

for(int i=0;i<5;i++) // loop will run n times (n=5)

{

if(fork() == 0)

{

printf("[son] pid %d from [parent] pid %d\n",getpid(),getppid());

exit(0);

}

}

for(int i=0;i<5;i++) // loop will run n times (n=5)

wait(NULL);

}

**4. Write a C program that illustrates 2 processes communicating using shared memory.**

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <unistd.h>

#include <string.h>

#include<stdio.h>

#include <errno.h>

int main(void) {

pid\_t pid;

int \*shared; /\* pointer to the shm \*/

int shmid;

shmid = shmget(IPC\_PRIVATE, sizeof(int), IPC\_CREAT | 0666);

printf("Shared Memory ID=%u",shmid);

if (fork() == 0) { /\* Child \*/

/\* Attach to shared memory and print the pointer \*/

shared = shmat(shmid, (void \*) 0, 0);

printf("Child pointer %d\n", \*shared);

\*shared=1;

printf("Child value=%d\n", \*shared);

sleep(2);

printf("Child value=%d\n", \*shared);

} else { /\* Parent \*/

/\* Attach to shared memory and print the pointer \*/

shared = shmat(shmid, (void \*) 0, 0);

printf("Parent pointer %d\n", \*shared);

printf("Parent value=%d\n", \*shared);

sleep(1);

\*shared=42;

printf("Parent value=%d\n", \*shared);

sleep(5);

shmctl(shmid, IPC\_RMID, 0);

}

}

**PROGRAM-5**

**5. Write a C program that implements producer –consumer system with two processes**

**using semaphores.**

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <semaphore.h>

#define BUFFER\_SIZE 5

int buffer[BUFFER\_SIZE];

int in = 0, out = 0;

int producerPrompt = 0;

sem\_t mutex, empty, full;

void \*producer(void \*arg) {

int item;

for (int i = 0; i < BUFFER\_SIZE; i++) {

printf("Enter item to produce: ");

scanf("%d", &item);

sem\_wait(&empty); // Wait for an empty slot in the buffer

sem\_wait(&mutex); // Obtain exclusive access to the buffer

// Lock a semaphore -> decreases the value

buffer[in] = item;

printf("Producer produced item: %d\n", item);

in = (in + 1) % BUFFER\_SIZE;

sem\_post(&mutex); // Release exclusive access to the buffer

sem\_post(&full); // Signal that a new item is available

//unlock a semaphore -> increases the value

producerPrompt = 1; // Signal that producer has prompted

while (producerPrompt) {

// Wait until consumer consumes the item

}

}

pthread\_exit(NULL);

}

void \*consumer(void \*arg) {

int item;

for (int i = 0; i < BUFFER\_SIZE; i++) {

sem\_wait(&full); // Wait for an item to be available

sem\_wait(&mutex); // Obtain exclusive access to the buffer

item = buffer[out];

printf("Consumer consumed item: %d\n", item);

out = (out + 1) % BUFFER\_SIZE;

sem\_post(&mutex); // Release exclusive access to the buffer

sem\_post(&empty); // Signal that an empty slot is available

producerPrompt = 0; // Signal that consumer has consumed the item

}

pthread\_exit(NULL);

}

int main() {

pthread\_t producerThread, consumerThread;

// Initialize semaphores

sem\_init(&mutex, 0, 1); //int sem\_init (sem\_t \*sem, int pshared, unsigned int value)

sem\_init(&empty, 0, BUFFER\_SIZE);

sem\_init(&full, 0, 0);

// Create producer and consumer threads

pthread\_create(&producerThread, NULL, producer, NULL);

pthread\_create(&consumerThread, NULL, consumer, NULL);

// Wait for threads to finish

pthread\_join(producerThread, NULL);

pthread\_join(consumerThread, NULL);

// Destroy semaphores

sem\_destroy(&mutex);

sem\_destroy(&empty);

sem\_destroy(&full);

return 0;

}

**PROGRAM 5: Another version - GFG**

#include <stdio.h>

#include <stdlib.h>

int mutex = 1;

int full = 0;

int empty = 10, x = 0;

void producer()

{

--mutex;

++full;

--empty;

x++;

printf("\nProducer produces"

"item %d",

x);

++mutex;

}

void consumer()

{

--mutex;

--full;

++empty;

printf("\nConsumer consumes "

"item %d",

x);

x--;

++mutex;

}

int main()

{

int n, i;

printf("\n1. Press 1 for Producer"

"\n2. Press 2 for Consumer"

"\n3. Press 3 for Exit");

#pragma omp critical

for (i = 1; i > 0; i++) {

printf("\nEnter your choice:");

scanf("%d", &n);

switch (n) {

case 1:

if ((mutex == 1)

&& (empty != 0)) {

producer();

}

else {

printf("Buffer is full!");

}

break;

case 2:

if ((mutex == 1)

&& (full != 0)) {

consumer();

}

else {

printf("Buffer is empty!");

}

break;

case 3:

exit(0);

break;

}

}

}

**6. Demonstrate round robin scheduling algorithm and calculate average waiting time and average turn around time**

#include<stdio.h>

int main()

{

int i, limit, total = 0, x, counter = 0, time\_quantum;

int wait\_time = 0, turnaround\_time = 0, arrival\_time[10], burst\_time[10], temp[10];

float average\_wait\_time, average\_turnaround\_time;

printf("\nEnter Total Number of Processes:t=");

scanf("%d", &limit);

x = limit;

for(i = 0; i < limit; i++)

{

printf("\nEnter Details of Process[%d]\n", i + 1);

printf("Arrival Time:t=");

scanf("%d", &arrival\_time[i]);printf("\nBurst Time:t");

scanf("%d", &burst\_time[i]);

temp[i] = burst\_time[i];

}

printf("\nEnter Time Quantum:t=");

scanf("%d", &time\_quantum);

printf("\nProcess ID\t\tBurst Time\t Turnaround Time\t Waiting Time\n");

for(total = 0, i = 0; x != 0;)

{

if(temp[i] <= time\_quantum && temp[i] > 0)

{

total = total + temp[i];

temp[i] = 0;

counter = 1;

}

else if(temp[i] > 0)

{

temp[i] = temp[i] - time\_quantum;

total = total + time\_quantum;

}

if(temp[i] == 0 && counter == 1)

{

x--;

printf("\nProcess[%d]\t\t%d\t\t %d\t\t\t %d", i + 1, burst\_time[i], total - arrival\_time[i],

total - arrival\_time[i] - burst\_time[i]);

wait\_time = wait\_time + total - arrival\_time[i] - burst\_time[i];

turnaround\_time = turnaround\_time + total - arrival\_time[i];

counter = 0;

}

if(i == limit - 1)

{

i = 0;

}

else if(arrival\_time[i + 1] <= total)

{

i++;

}

else{

i = 0;

}

}

average\_wait\_time = wait\_time \* 1.0 / limit;

average\_turnaround\_time = turnaround\_time \* 1.0 / limit;

printf("\n\nAverage Waiting Time:t=%f", average\_wait\_time);

printf("\nAvg Turnaround Time:t=%f\n", average\_turnaround\_time);return 0;

}

**7.Implement Priority based scheduling algorithm and calculate average waiting time and turn around time**

#include<stdio.h>

int main()

{

int bt[20],p[20],wt[20],tat[20],pr[20],i,j,n,total=0,pos,temp,avg\_wt,avg\_tat;

printf("Enter Total Number of Process:");

scanf("%d",&n);

printf("\nEnter Burst Time and Priority\n");

for(i=0;i<n;i++)

{

printf("\nP[%d]\n",i+1);

printf("Burst Time:");

scanf("%d",&bt[i]);

printf("Priority:");

scanf("%d",&pr[i]);

p[i]=i+1;

}

for(i=0;i<n;i++)

{

pos=i;

for(j=i+1;j<n;j++)

{

if(pr[j]<pr[pos])

pos=j;

}temp=pr[i];

pr[i]=pr[pos];

pr[pos]=temp;

temp=bt[i];

bt[i]=bt[pos];

bt[pos]=temp;

temp=p[i];

p[i]=p[pos];

p[pos]=temp;

}

wt[0]=0;

for(i=1;i<n;i++)

{

wt[i]=0;

for(j=0;j<i;j++)

wt[i]+=bt[j];

total+=wt[i];

}

avg\_wt=total/n;

total=0;

printf("\nProcess\t Burst Time \tWaiting Time\tTurnaround Time");

for(i=0;i<n;i++)

{ tat[i]=bt[i]+wt[i]; //calculate turnaround time

total+=tat[i];

printf("\nP[%d]\t\t %d\t\t %d\t\t\t%d",p[i],bt[i],wt[i],tat[i]);

}

avg\_tat=total/n; //average turnaround time

printf("\n\nAverage Waiting Time=%d",avg\_wt);

printf("\n\nAverage Waiting Time=%d",avg\_wt);

printf("\nAverage Turnaround Time=%d\n",avg\_tat);

return 0;

}

**8.Act as sender to send data in message queues and receiver that reads data from message queue**

**Sender**

#include <stdio.h>

#include <sys/ipc.h>

#include <sys/msg.h>

#define MAX 10

struct mesg\_buffer {

long mesg\_type;

char mesg\_text[100];

} message;

int main()

{

key\_t key;

int msgid;

key = ftok("progfile", 65);

msgid = msgget(key, 0666 | IPC\_CREAT);

message.mesg\_type = 1;

printf("Write Data : ");

fgets(message.mesg\_text,MAX,stdin);

msgsnd(msgid, &message, sizeof(message), 0);

printf("Data sent is : %s \n", message.mesg\_text);

return 0;

}

**Receiver**

#include <stdio.h>

#include <sys/ipc.h>

#include <sys/msg.h>

struct mesg\_buffer {

long mesg\_type;

char mesg\_text[100];

} message;

int main()

{

key\_t key;

int msgid;

key = ftok("progfile", 65);

msgid = msgget(key, 0666 | IPC\_CREAT);

msgrcv(msgid, &message, sizeof(message), 1, 0);

printf("Data Received is : %s \n", message.mesg\_text);

msgctl(msgid, IPC\_RMID, NULL);

return 0;

}

**9.Write a program where parent writes message to a pipe and child reads message from the pipe**

**Parent**

#include<stdio.h>

#include<unistd.h>

#include<fcntl.h>

#include<stdlib.h>

#include <sys/stat.h>

#define MAXSIZE 10

#define FIFO\_NAME "myfifo"

int main()

{

int fifoid;

int fd, n;

char \*w;

system("clear");

w=(char \*)malloc(sizeof(char)\*MAXSIZE);

int open\_mode=O\_WRONLY;

fifoid=mkfifo(FIFO\_NAME, 0755);

if(fifoid==-1)

{

printf("\nError: Named pipe cannot be Created\n");

exit(0);

}

if( (fd=open(FIFO\_NAME, open\_mode)) < 0 )

{

printf("\nError: Named pipe cannot be opened\n");

exit(0);

}

while(1)

{

printf("\nProducer :");

fflush(stdin);

read(0, w, MAXSIZE);

n=write(fd, w, MAXSIZE);

if(n > 0)

printf("\nProducer sent: %s", w);

}

}

**Child**

#include<stdio.h>

#include<unistd.h>

#include<fcntl.h>

#include<stdlib.h>

#include <sys/stat.h>

#define MAXSIZE 10

#define FIFO\_NAME "myfifo"

int main()

{

int fifoid;

int fd, n;

char \*r;

system("clear");

r=(char \*)malloc(sizeof(char)\*MAXSIZE);

int open\_mode=O\_RDONLY;

if( (fd=open(FIFO\_NAME, open\_mode)) < 0 )

{

printf("\nError: Named pipe cannot be opened\n");

exit(0);

}

while(1)

{

n=read(fd, r, MAXSIZE);

if(n > 0)

printf("\nConsumer read: %s", r);

}

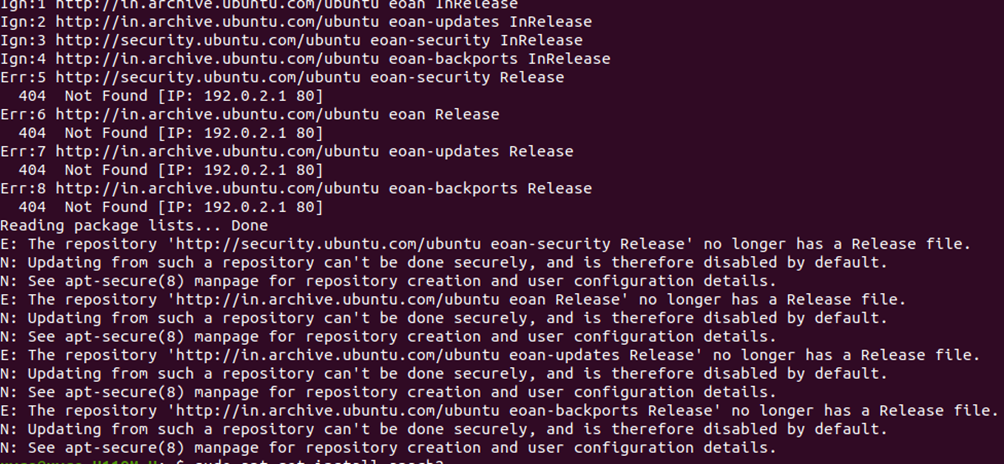
}

**10.Demonstrate setting up a simple Web Server and Host Website on Your Own Linux Computer**

**Installing apache2 and php-7.3**

Note: The commands should be executed in the **terminal**.

Step 1: run the command ‘sudo apt update’ without quotes.



If you get the above error:

go to this website: <https://www.digitalocean.com/community/questions/unable-to-apt-update-my-ubuntu-19-04>

and run the commands given in the first answer.

The commands are:

1. sudo sed -i -re 's/([a-z]{2}\.)?archive.ubuntu.com|security.ubuntu.com/old-releases.ubuntu.com/g' /etc/apt/sources.list

2. sudo apt-get update && sudo apt-get dist-upgrade

It will install the legacy packages required.

Incase you do not get the error in the above picture, continue from step 2.

Step 2: Installing apache2

Run the commands:

1. sudo apt update

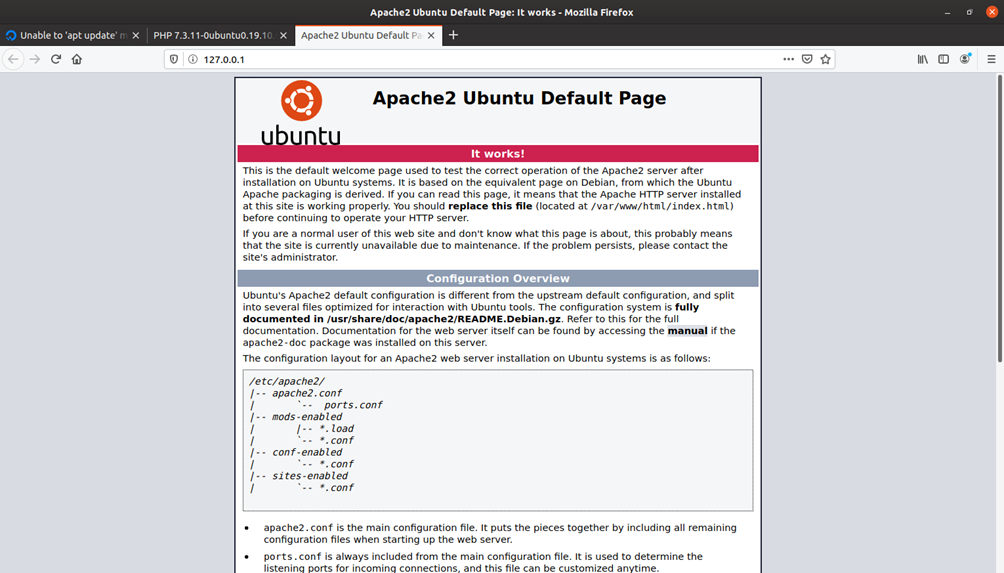
2. sudo apt-get install apache2

To check if apache2 has been installed properly:

1. Execute the command –> sudo service apache2 restart

2. Open a browser

2. Type 127.0.0.1 in the address bar on the window.



Your screen should be as the above image.

Step 3: Install MySQL server

Execute the command: sudo apt-get install mysql-server

Step 4: Install PHP

1. Run the command –> php -v

2. If the php is installed it will return the version number.

|  |  |
| --- | --- |
|  |  |

3. If not, it will return an error and at the end will give a command to install the required file.

4. Copy, paste and run the command and give ‘y’ without quotes when prompted.

5. Kindly note down the version number of PHP.

Step 5: Install necessary files to to connect PHP to Apache2 and MySQL

1. Run the command > sudo apt-get install php libapache2-mod-php php-mysql -y

2. After installation make sure apache2 and mysql are running.

3. To check and enable that execute the following:

sudo systemctl start apache2

sudo systemctl enable apache2

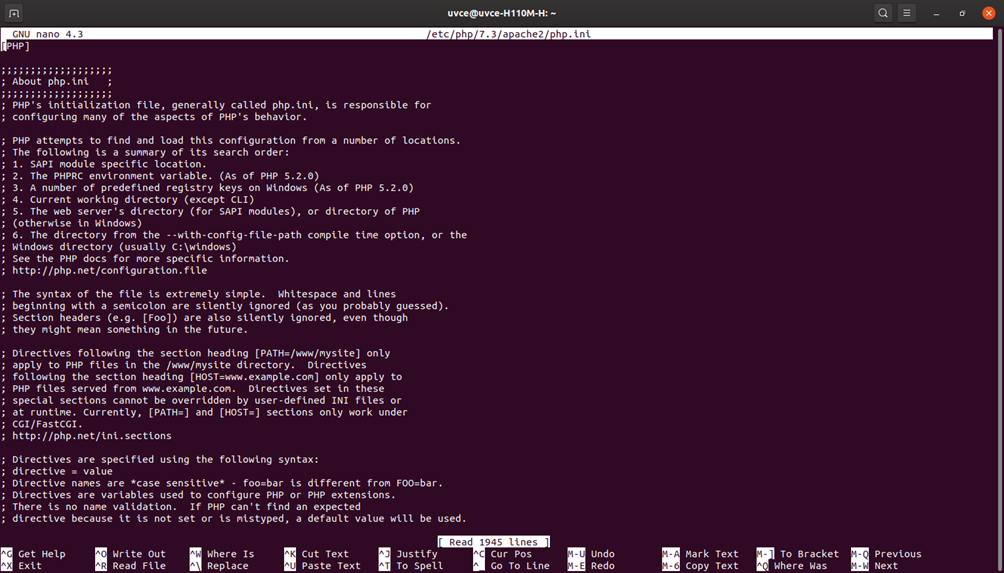
sudo systemctl start mysql

sudo systemctl enable mysql

4. Run the command **sudo nano *etc/*php/ <php\_version\_number>/apache2/php.ini**

For example: If your PHP version number is 7.3, the above command will be:

**sudo nano etc/php/7.3/apache2/php.ini**



5. The command will return a file something like above. This just ensures that you have installed PHP correctly.

6. Run the command -> **sudo nano *var/*www/html/info.php**

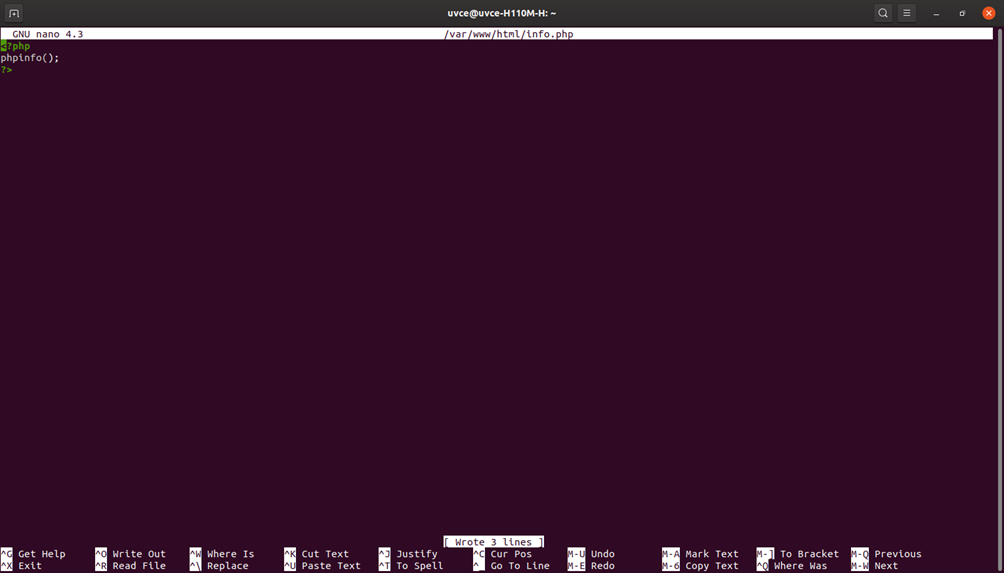
This will create a new file named info.php. Type the following code in the file:

**<?php**

**phpinfo();**

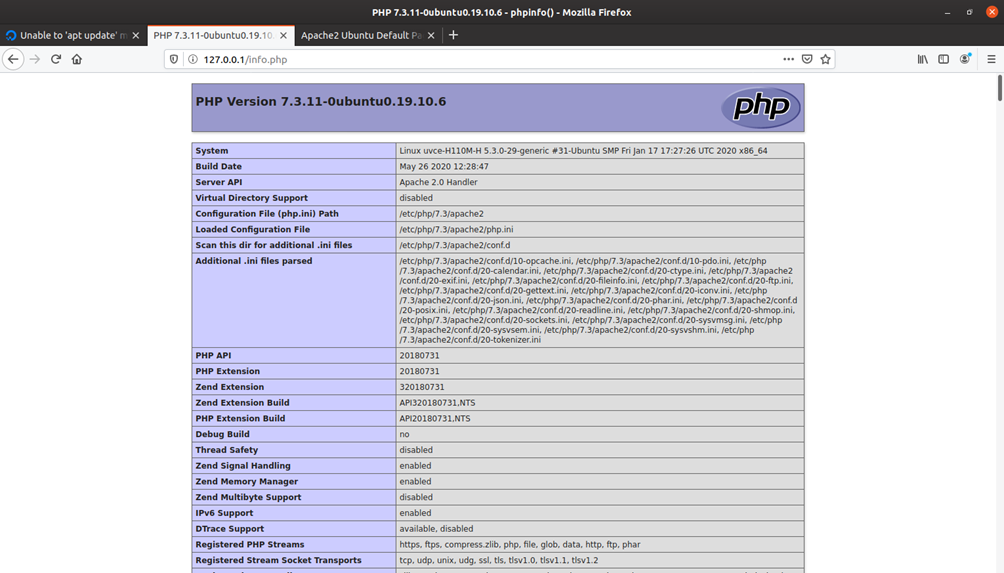
**?>**

Press Ctrl+S and Ctrl+X to save and exit the file.



Step 6: Check if PHP is working on Apache2

Go to web browser and type the URL -> 1**27.0.0.1/info.php**



You should get a window like below which shows the details about PHP.

**Congratulations. You have successfully installed the required modules for Question 10.**

**11(a). Create two threads using pthread, where both thread count until 100 and joins later.**

#include<stdio.h>

#include<unistd.h>

#include<pthread.h>

#include<stdlib.h>

void\* myturn(void \*arg)

{

for(int i=1;i<=10;i++)

{

sleep(1);

printf("process 1: i=%d\n",i);

}

return NULL;

}

void yourturn()

{

for(int i=1;i<=10;i++)

{

sleep(2);

printf("process 2: j=%d\n",i);

}

}

int main()

{

pthread\_t newthread;

pthread\_create(&newthread,NULL,myturn,NULL);

yourturn();

pthread\_join(newthread,NULL);

return 0;

}

**11b Create two threads using pthreads. Here, main thread creates 5 other threads for 5**

**times and each new thread print “Hello World” message with its thread number.**

**Question is Unclear so decide which code to refer (This one or manual )**

#include <stdio.h>

#include <pthread.h>

#define NUM\_THREADS 5

void \*myNewThread(void \*vargp) {

pthread\_t tid = pthread\_self();

printf("Hello World from thread %ld\n", tid);

pthread\_exit(NULL);

}

void \*myThreadFun(void \*vargp) {

int i;

pthread\_t tid[NUM\_THREADS];

for (i = 0; i < NUM\_THREADS; i++) {

for(int j =0;j<5;j++)

pthread\_create(&tid[i], NULL, myNewThread, NULL);

}

for (i = 0; i < NUM\_THREADS; i++) {

pthread\_join(tid[i], NULL);

}

pthread\_exit(NULL);

}

int main() {

pthread\_t t1,t2;

pthread\_create(&t1, NULL, myThreadFun, NULL);

// pthread\_create(&t2, NULL, myThreadFun, NULL);

pthread\_join(t1, NULL);

//pthread\_join(t2, NULL);

return 0;

}

**12 Using Socket APIs establish communication between remote and local processes.**

**Server**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <string.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#define PORT 8000

#define BUFFER\_SIZE 1024

int main()

{

int server\_fd, new\_socket;

struct sockaddr\_in address;

int addrlen = sizeof(address);

char buffer[BUFFER\_SIZE];

// Create socket

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

if (server\_fd < 0)

{

perror("socket failed");

exit(EXIT\_FAILURE);

}

// Set address parameters

address.sin\_family = AF\_INET;

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

address.sin\_port = htons(PORT);

// Bind socket to specified address and port

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

{

perror("bind failed");

exit(EXIT\_FAILURE);

}

// Listen for incoming connections

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

{

perror("listen failed");

exit(EXIT\_FAILURE);

}

// Accept incoming connection

new\_socket = accept(server\_fd, (struct sockaddr \*)&address, (socklen\_t \*)&addrlen);

if (new\_socket < 0)

{

perror("accept failed");

exit(EXIT\_FAILURE);

}

// Read client message into buffer

read(new\_socket, buffer, BUFFER\_SIZE);

printf("Client message: %s\n", buffer);

// Send response to client

const char \*response = "Hello from server";

write(new\_socket, response, strlen(response));

// Close sockets

close(new\_socket);

close(server\_fd);

return 0;

}

**Client**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <string.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#define PORT 8000

#define BUFFER\_SIZE 1024

int main()

{

int sock;

struct sockaddr\_in serv\_addr;

char buffer[BUFFER\_SIZE];

// Create socket

sock = socket(AF\_INET, SOCK\_STREAM, 0);

if (sock < 0)

{

perror("socket failed");

exit(EXIT\_FAILURE);

}

// Set server address parameters

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

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

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

// Connect to server

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

{

perror("connect failed");

exit(EXIT\_FAILURE);

}

// Get message from user for server

printf("Enter a message for the server: ");

fgets(buffer, BUFFER\_SIZE, stdin);

// Send message to server

write(sock, buffer, strlen(buffer));

// Clear buffer

memset(buffer, 0, BUFFER\_SIZE);

// Read server response into buffer

read(sock, buffer, BUFFER\_SIZE);

printf("Server response: %s\n", buffer);

// Close socket

close(sock);

return 0;

}