1. Write C programs to implement basic UNIX system calls - read(), write(), open(),close(), lseek(), create().

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>

   int main() {
   int fd, ret;
   char buffer[20];

   // Create a new file
   fd = creat("example.txt", 0644);
          if (fd == -1)
          {
          perror("creat");
          exit(EXIT_FAILURE);
          }

   // Write to the file
   ret = write(fd, "Hello, World!", 13);
          if (ret == -1)
          {
          perror("write");
          exit(EXIT_FAILURE);
          }

   // Close the file
   ret = close(fd);
          if (ret == -1)
          {
          perror("close");
           exit(EXIT_FAILURE);
          }
 // Open the file again
   fd = open("example.txt", O_RDWR);
          if (fd == -1)
          {
          perror("open");
          exit(EXIT_FAILURE);
          }
```

```
    // Move the file cursor to the beginning of the file
    ret = lseek(fd, 0, SEEK_SET);
            if (ret == -1)
            {
             perror("lseek");
            exit(EXIT_FAILURE);
             }

    // Read from the file
    ret = read(fd, buffer, 13);
             if (ret == -1)
            {
            perror("read");
            exit(EXIT_FAILURE);
            }
                    buffer[ret] = '\0'; // Null-terminate the string

                    printf("Read from file: %s\n", buffer);

    // Close the file
    ret = close(fd);
            if (ret == -1)
            {
            perror("close");
            exit(EXIT_FAILURE);
            }
    return 0;
}
```

Output:

```
(base) sahyadri@sahyadri:~/OS LAB_IV SEm/OS_Programs$ gcc 1.c
(base) sahyadri@sahyadri:~/OS LAB_IV SEm/OS_Programs$ ./a.out
Read from file: Hello, World!
```

2.Write C programs to implement UNIX Directory API's - opendir, closedir, readdir, mkdir.

```
#include <stdio.h>
#include <stdlib.h>
#include <dirent.h>
#include <sys/stat.h>
#include <errno.h>

        void listDirectory(const char *path) {
        DIR *dir = opendir(path);
                if (dir == NULL) {
      perror("opendir");
      return;
   }

   struct dirent *entry;
   while ((entry = readdir(dir)) != NULL) {
      printf("%s\n", entry->d_name);
   }

   if (closedir(dir) == -1) {
      perror("closedir");
   }
}

void createDirectory(const char *path) {
   if (mkdir(path, 0755) == -1) {
      if (errno == EEXIST) {
         printf("Directory %s already exists.\n", path);
      } else {
         perror("mkdir");
      }
   } else {
      printf("Directory %s created successfully.\n", path);
   }
}

int main() {
   const char *dirPath = "./testdir";

   // Create a directory
```

```
    createDirectory(dirPath);

    // List the contents of the current directory
    printf("Listing current directory contents:\n");
    listDirectory(".");

    // List the contents of the new directory
    printf("\nListing new directory contents:\n");
    listDirectory(dirPath);

    return 0;
}
```

Output:

```
(base) sahyadri@sahyadri:~/OS LAB_IV SEm/OS_Programs$ gcc 2.c
(base) sahyadri@sahyadri:~/OS LAB_IV SEm/OS_Programs$ ./a.out
Directory ./testdir created successfully.
Listing current directory contents:
..
4write.c
3fork.c
4read.c
6.c
os_manual_21scheme_2023.doc
.~lock.Untitled 1.odt#
a.out
3vfork.c
Lab8 programs
5priority.c
5process.c
lab7 programs
2.c
testdir
1.c
Listing new directory contents:
..
.
```

3. Demonstrate the Process creation and Termination using System calls –fork(), vfork(), exit(), return 0.

……

Program:

Fork()

```
#include <stdio.h>
#include <unistd.h>
#include <sys/wait.h>

int main()
{
	pid_t pid;
	int status;
	// fork() system call
	pid = fork();
	if (pid < 0)
	{
		printf("Error: fork() failed.\n");
		return 1;
	}
	else if (pid == 0)
	{
		// child process
		printf("This is the child process with PID: %d\n", getpid());
		printf("Parent process PID: %d\n", getppid());
		// exec() system call
		execlp("/bin/ls", "ls", NULL);
		printf("This should not be printed if exec() is successful.\n");
		return 0;
	}
	else
	{
		// parent process
		printf("This is the parent process with PID: %d\n", getpid());
		printf("Child process PID: %d\n", pid);
		 // wait() system call
		wait(&status);
		printf("Child process exited with status: %d\n", status);
		return 0;
	}
```

```
}
```

Output:

```
sahyadri@sahyadri-desktop:~$ gcc 1.c
sahyadri@sahyadri-desktop:~$ ./a.out
This is the parent process with PID: 5412
Child process PID: 5413
This is the child process with PID: 5413
Parent process PID: 5412
1.c    Desktop    eclipse  p2.c  p.c       student.java
2.c    a.out Documents  fcfs.c   p4    readersem.c         writersem.c
abc.c  deadlock.c Downloads  p1.c     p4.c  student.class
Child process exited with status: 0
```

Description:

In this program, we use three different system calls for process creation and termination:

fork(): This system call is used to create a new process by duplicating the calling process. The child process and the parent process will have different process IDs (PIDs), and they will run independently of each other.

exec(): This system call is used to replace the current process image with a new process image. In the child process, we use execlp() to replace the current process image with the **ls** command, which lists the contents of the current directory.

wait(): This system call is used by the parent process to wait for the child process to exit. The status of the child process is stored in the status variable.

This output shows that the parent process creates a child process using the **fork() system call,** and then **waits for the child process** to **exit using the wait() system call**. The child process **uses the exec() system call to replace its process image** with the **ls** command, which lists the contents of the current directory.

Vfork()

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>

int main()
{
	pid_t pid;
	// Fork a child process using vfork()
	 pid = vfork();
	if (pid == -1)
	{
		// Forking failed
		 perror("vfork");
		 return 1;
	}
	else if (pid == 0)
	{
		// Child process
		printf("Child process: Hello, I'm the child!\n");
		printf("Child process: My PID is %d\n", getpid());
		printf("Child process: My parent's PID is %d\n", getppid());
		// Terminate the child process
		_exit(0);
	}
	else
	{
		// Parent process
		printf("Parent process: Hello, I'm the parent!\n");
		printf("Parent process: My PID is %d\n", getpid());
		printf("Parent process: My child's PID is %d\n", pid);
		// Wait for the child process to terminate
		int status;
		waitpid(pid, &status, 0);
		if (WIFEXITED(status))
		{
			printf("Parent process: Child process terminated normally.\n");
		}
```

```
                else
                {
                        printf("Parent process: Child process terminated abnormally.\n");
                }
        }
        return 0;
}
```

Output:

```
sahyadri@sahyadri-desktop:~$ gcc 1b.c
sahyadri@sahyadri-desktop:~$ ./a.out
Child process: Hello, I'm the child!
Child process: My PID is 10004
Child process: My parent's PID is 10003
Parent process: Hello, I'm the parent!
Parent process: My PID is 10003
Parent process: My child's PID is 10004
Parent process: Child process terminated normally.
```

4. Write C programs to simulate Inter – Process Communication (IPC) techniques: Pipes, Messages Queues, and Shared Memory.

Writer Process (writer.c)

```
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>

int main()
{
int fd;
char * myfifo = "/tmp/myfifo"; /* create the FIFO (named pipe) */
mkfifo(myfifo, 0666);

fd = open(myfifo, O_WRONLY);
write(fd,"Hi", sizeof("Hi")); /* write "Hi" to the FIFO */
close(fd);

unlink(myfifo); /* remove the FIFO */
return 0;
}
```

Reader Process (reader.c)

```
#include <sys/stat.h>
 #include <unistd.h>

 #define MAX_BUF 1024

 int main()
 {
 int fd;
 char *myfifo = "/tmp/myfifo";
 char buf[MAX_BUF];

 /* open, read, and display the message from the FIFO */
 fd = open(myfifo, O_RDONLY);
 read(fd, buf, MAX_BUF);
 printf("Received: %s", buf);
 close(fd);
 return 0;
 }
```

To Run the Program

```
[root@localhost /]# gcc writer.c
[root@localhost /]# ./a.out
```

After this Open New Terminal by pressing shift+ctrl+N or Go to File->Open Terminal

```
[root@localhost /]# gcc reader.c
[root@localhost /]# ./a.out
```

OUTPUT :

```
(base) sahyadri@sahyadri:~/OS LAB_IV SEm/OS_Programs$ gedit writ.c
(base) sahyadri@sahyadri:~/OS LAB_IV SEm/OS_Programs$ gcc writ.c
(base) sahyadri@sahyadri:~/OS LAB_IV SEm/OS_Programs$ ./a.out
(base) sahyadri@sahyadri:~/OS LAB_IV SEm/OS_Programs$
```

```
((base) sahyadri@sahyadri:~/OS LAB_IV SEm/OS_Programs$ gedit read.c
(base) sahyadri@sahyadri:~/OS LAB_IV SEm/OS_Programs$ gcc read.c
(base) sahyadri@sahyadri:~/OS LAB_IV SEm/OS_Programs$ ./a.out
Received: Hello
```

5. Simulate the following CPU scheduling algorithms

1. **FCFS**
2. **SJF**
3. **Priority**
4. **Round Robin.**

……

Program:

```
#include <stdio.h>
void fcfs(int processes[], int n, int burst_time[]) {
int waiting_time[n], turnaround_time[n], total_waiting_time = 0, total_turnaround_time = 0;

waiting_time[0] = 0; // Waiting time for first process is 0

// Calculating waiting time for each process
for (int i = 1; i < n; i++) {
waiting_time[i] = burst_time[i - 1] + waiting_time[i - 1];
total_waiting_time += waiting_time[i];
}

// Calculating turnaround time for each process
for (int i = 0; i < n; i++) {
turnaround_time[i] = burst_time[i] + waiting_time[i];
total_turnaround_time += turnaround_time[i];
}

printf("First-Come, First-Served (FCFS) Scheduling Algorithm\n");
printf("--------------------------------------------------\n");
printf("Process\tBurst Time\tWaiting Time\tTurnaround Time\n");

// Printing process details
for (int i = 0; i < n; i++) {
printf("%d\t%d\t\t%d\t\t%d\n", processes[i], burst_time[i], waiting_time[i],
turnaround_time[i]);
}

printf("Average Waiting Time: %.2f\n", (float)total_waiting_time / n);
printf("Average Turnaround Time: %.2f\n", (float)total_turnaround_time / n);

printf("\n");
}

void sjf(int processes[], int n, int burst_time[]) {
int waiting_time[n], turnaround_time[n], completion_time[n], total_waiting_time = 0,
total_turnaround_time = 0;

for (int i = 0; i < n; i++) {
int shortest_job_index = i;
```

```
// Find the shortest job
for (int j = i + 1; j < n; j++) {
if (burst_time[j] < burst_time[shortest_job_index])
shortest_job_index = j;
}

// Swap the shortest job with the current process
int temp = burst_time[i];
burst_time[i] = burst_time[shortest_job_index];
burst_time[shortest_job_index] = temp;

temp = processes[i];
processes[i] = processes[shortest_job_index];
processes[shortest_job_index] = temp;
}

waiting_time[0] = 0; // Waiting time for first process is 0

// Calculating waiting time for each process
for (int i = 1; i < n; i++) {
waiting_time[i] = burst_time[i - 1] + waiting_time[i - 1];

total_waiting_time += waiting_time[i];
}

// Calculating turnaround time for each process
for (int i = 0; i < n; i++) {
turnaround_time[i] = burst_time[i] + waiting_time[i];
total_turnaround_time += turnaround_time[i];
}

printf("Shortest Job First (SJF) Scheduling Algorithm\n");
printf("----------------------------------------------\n");
printf("Process\tBurst Time\tWaiting Time\tTurnaround Time\n");

// Printing process details
for (int i = 0; i < n; i++) {
printf("%d\t%d\t\t%d\t\t%d\n", processes[i], burst_time[i], waiting_time[i],
turnaround_time[i]);
}

printf("Average Waiting Time: %.2f\n", (float)total_waiting_time / n);
printf("Average Turnaround Time: %.2f\n", (float)total_turnaround_time / n);
printf("\n");
}

void roundRobin(int processes[], int n, int burst_time[], int quantum) {
int remaining_time[n], waiting_time[n], turnaround_time[n], total_waiting_time = 0,
total_turnaround_time = 0;
```

```
// Copying burst time into remaining time array
for (int i = 0; i < n; i++) {
remaining_time[i] = burst_time[i];
}

int time = 0; // Current time

// Run the round robin algorithm
while (1) {
int all_processes_completed = 1;

// Traverse all processes
for (int i = 0; i < n; i++) {
if (remaining_time[i] > 0) {
all_processes_completed = 0; // There is still a pending process

if (remaining_time[i] > quantum) {
time += quantum;
remaining_time[i] -= quantum;
} else {
time += remaining_time[i];
waiting_time[i] = time - burst_time[i];
remaining_time[i] = 0;
}
}
}

if (all_processes_completed) {
break;
}
}

// Calculating turnaround time for each process
for (int i = 0; i < n; i++) {
turnaround_time[i] = burst_time[i] + waiting_time[i];

total_waiting_time += waiting_time[i];
total_turnaround_time += turnaround_time[i];
}

printf("Round Robin Scheduling Algorithm\n");
printf("--------------------------------\n");
printf("Process\tBurst Time\tWaiting Time\tTurnaround Time\n");

// Printing process details
for (int i = 0; i < n; i++) {
printf("%d\t%d\t\t%d\t\t%d\n", processes[i], burst_time[i], waiting_time[i],
turnaround_time[i]);
```

```
}

printf("Average Waiting Time: %.2f\n", (float)total_waiting_time / n);
printf("Average Turnaround Time: %.2f\n", (float)total_turnaround_time / n);
printf("\n");
}

int main() {
int n;
printf("Enter the number of processes: ");
scanf("%d", &n);

int processes[n], burst_time[n];

printf("Enter the burst time for each process:\n");
for (int i = 0; i < n; i++) {
printf("Process %d: ", i + 1);
scanf("%d", &burst_time[i]);
processes[i] = i + 1;

}

int quantum;
printf("Enter the time quantum for Round Robin: ");
scanf("%d", &quantum);

printf("\n");

fcfs(processes, n, burst_time);
sjf(processes, n, burst_time);
roundRobin(processes, n, burst_time, quantum);

return 0;
}
```

OUTPUT:

```
(base) sahyadri@sahyadri:~/OS LAB_IV SEm/OS_Programs$ gcc process.c
(base) sahyadri@sahyadri:~/OS LAB_IV SEm/OS_Programs$ ./a.out
Enter the number of processes: 5
Enter the burst time for each process:
Process 1: 1
Process 2: 5
Process 3: 1
Process 4: 6
Process 5: 3
Enter the time quantum for Round Robin: 2

First-Come, First-Served (FCFS) Scheduling Algorithm
```

Process Burst Time Waiting Time Turnaround Time
1 1 0 1
2 5 1 6
3 1 6 7
4 6 7 13
5 3 13 16
Average Waiting Time: 5.40
Average Turnaround Time: 8.60

Shortest Job First (SJF) Scheduling Algorithm

Process Burst Time Waiting Time Turnaround Time
1 1 0 1
3 1 1 2
5 3 2 5
2 5 5 10
4 6 10 16
Average Waiting Time: 3.60
Average Turnaround Time: 6.80

Round Robin Scheduling Algorithm

Process Burst Time Waiting Time Turnaround Time
1 1 0 1
3 1 1 2
5 3 6 9
2 5 9 14
4 6 10 16
Average Waiting Time: 5.20
Average Turnaround Time: 8.40

Priority algorithms

```
#include<stdio.h>
int main()
{
```

```
int p[20],bt[20],pri[20], wt[20],tat[20],i, k, n, temp;
float wtavg, tatavg;
printf("Enter the number of processes --- ");
scanf("%d",&n);
for(i=0;i<n;i++)
{
        p[i] = i;
        printf("Enter the Burst Time & Priority of Process %d --- ",i);
        scanf("%d %d",&bt[i], &pri[i]);
}
for(i=0;i<n;i++)
        for(k=i+1;k<n;k++)
                if(pri[i] > pri[k])
                {
                        temp=p[i];
                        p[i]=p[k];
                        p[k]=temp;
                        temp=bt[i];
                        bt[i]=bt[k];
                        bt[k]=temp;
                        temp=pri[i];
                        pri[i]=pri[k];
                        pri[k]=temp;
                }
wtavg = wt[0] = 0;
tatavg = tat[0] = bt[0];
for(i=1;i<n;i++)
{
        wt[i] = wt[i-1] + bt[i-1];
        tat[i] = tat[i-1] + bt[i];
        wtavg = wtavg + wt[i];
        tatavg = tatavg + tat[i];
}
printf("\nPROCESS\t\tPRIORITY\tBURST TIME\tWAITING TIME\tTURNAROUND TIME");
for(i=0;i<n;i++)
        printf("\n%d \t\t %d \t\t %d \t\t %d \t\t %d ",p[i],pri[i],bt[i],wt[i],tat[i]);
printf("\nAverage Waiting Time is --- %f",wtavg/n);
printf("\nAverage Turnaround Time is --- %f",tatavg/n);
return 0;
}
```

Output:

Enter the number of processes --- 4

Enter the Burst Time & Priority of Process 0 --- 8 2
Enter the Burst Time & Priority of Process 1 --- 4 3
Enter the Burst Time & Priority of Process 2 --- 9 1
Enter the Burst Time & Priority of Process 3 --- 5 4

PROCESS	PRIORITY	BURST TIME	WAITING TIME	TURNAROUND TIME
2	1	9	0	9
0	2	8	9	17
1	3	4	17	21
3	4	5	21	26

Average Waiting Time is --- 11.750000
Average Turnaround Time is --- 18.250000

Enter the number of processes --- 4
Enter the Burst Time & Priority of Process 0 --- 8 2
Enter the Burst Time & Priority of Process 1 --- 4 3
Enter the Burst Time & Priority of Process 2 --- 9 1
Enter the Burst Time & Priority of Process 3 --- 5 4

PROCESS	PRIORITY	BURST TIME	WAITING TIME	TURNAROUND TIME
2	1	9	0	9
0	2	8	9	17
1	3	4	17	21
3	4	5	21	26

Average Waiting Time is --- 11.750000
Average Turnaround Time is --- 18.250000

6. Demonstrate the following Classical problems of synchronizationa. producer-consumer, b. dining philosopher problems using semaphores.

……

Program:

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <semaphore.h>
#include<unistd.h>

#define BUFFER_SIZE 5
#define NUM_PRODUCERS 2
#define NUM_CONSUMERS 2
#define NUM_PHILOSOPHERS 5

// Producer-Consumer Problem
int buffer[BUFFER_SIZE];
int in = 0;
int out = 0;

sem_t emptySlots;    // Semaphore to track empty slots in the buffer
sem_t filledSlots;   // Semaphore to track filled slots in the buffer
sem_t bufferMutex;   // Semaphore for mutual exclusion access to the buffer
sem_t rwMutex;
sem_t mutex;

void *producer(void *producerId)
{
        int id = *(int *)producerId;
         int item = 0;

        while (1)
        {
                // Produce item
                 item++;

                sem_wait(&emptySlots); // Wait for an empty slot in the buffer
                sem_wait(&bufferMutex); // Obtain exclusive access to the buffer

                // Add item to the buffer
                buffer[in] = item;
                printf("Producer %d produced item %d\n", id, item);
                 in = (in + 1) % BUFFER_SIZE;

                sem_post(&bufferMutex); // Release exclusive access to the buffer
                sem_post(&filledSlots); // Signal that a slot in the buffer is filled
```

```
                // Sleep for a random period
                usleep(rand() % 1000000);
        }
}

void *consumer(void *consumerId)
{
        int id = *(int *)consumerId;
        int item;
        while (1)
        {
                sem_wait(&filledSlots); // Wait for a filled slot in the buffer
                sem_wait(&bufferMutex); // Obtain exclusive access to the buffer
                // Consume item from the buffer
                item = buffer[out];
                printf("Consumer %d consumed item %d\n", id, item);
                out = (out + 1) % BUFFER_SIZE;
                sem_post(&bufferMutex); // Release exclusive access to the buffer
                sem_post(&emptySlots); // Signal that a slot in the buffer is empty
                // Sleep for a random period
                usleep(rand() % 1000000);
        }
}

 // Dining Philosophers Problem
enum { THINKING, HUNGRY, EATING } state[5];
sem_t philMutex;
sem_t philSem[5];


void test(int id)
{
        if (state[id] == HUNGRY && state[(id + 1) % 5] != EATING && state[(id + 4) % 5] != EATING)
        {
                state[id] = EATING;
                sem_post(&philSem[id]);
        }
}

void *philosopher(void *philosopherId)
{
```

```
        int id = *(int *)philosopherId;
        while (1)
        {
                printf("Philosopher %d is thinking\n", id);
                // Start thinking
                usleep(rand() % 1000000);
                 sem_wait(&philMutex); // Obtain exclusive access to state
                state[id] = HUNGRY;
                printf("Philosopher %d is hungry\n", id);
                // Try to acquire the chopsticks
                test(id);
                sem_post(&philMutex); // Release exclusive access to state
                sem_wait(&philSem[id]); // Wait for chopsticks
                printf("Philosopher %d is eating\n", id);
                // Start eating
                usleep(rand() % 1000000);
                sem_post(&philSem[id]); // Put down chopsticks
                printf("Philosopher %d finished eating\n", id);
        }
}

int main()
{
        // Producer-Consumer Problem
        sem_init(&emptySlots, 0, BUFFER_SIZE); // Initialize emptySlots semaphore with buffer size
        sem_init(&filledSlots, 0, 0); // Initialize filledSlots semaphore with 0
        sem_init(&bufferMutex, 0, 1); // Initialize bufferMutex semaphore with 1
        pthread_t producers[NUM_PRODUCERS];
        pthread_t consumers[NUM_CONSUMERS];
        int producerIds[NUM_PRODUCERS];
        int consumerIds[NUM_CONSUMERS];
        for (int i = 0; i < NUM_PRODUCERS; i++)
        {
                producerIds[i] = i + 1;
                pthread_create(&producers[i], NULL, producer, (void *)&producerIds[i]);
        }
        for (int i = 0; i < NUM_CONSUMERS; i++)
        {
                consumerIds[i] = i + 1;
                pthread_create(&consumers[i], NULL, consumer, (void *)&consumerIds[i]);
        }
```

```
    // Dining Philosophers Problem
    sem_init(&philMutex, 0, 1); // Initialize philMutex semaphore with 1
    pthread_t philosophers[NUM_PHILOSOPHERS];
     int philosopherIds[NUM_PHILOSOPHERS];
    for (int i = 0; i < NUM_PHILOSOPHERS; i++)
    {
        philosopherIds[i] = i + 1;
        sem_init(&philSem[i], 0, 0); // Initialize philSemaphores with 0
        pthread_create(&philosophers[i], NULL, philosopher, (void *)&philosopherIds[i]);
    }
    // Wait for threads to finish
    for (int i = 0; i < NUM_PRODUCERS; i++)
    {
        pthread_join(producers[i], NULL);
    }
    for (int i = 0; i < NUM_CONSUMERS; i++)
    {
        pthread_join(consumers[i], NULL);
    }
    for (int i = 0; i < NUM_PHILOSOPHERS; i++)
    {
        pthread_join(philosophers[i], NULL);
    }

    // Cleanup semaphores
    sem_destroy(&emptySlots);
    sem_destroy(&filledSlots);
    sem_destroy(&bufferMutex);
    sem_destroy(&mutex);
    sem_destroy(&rwMutex);
    sem_destroy(&philMutex);
    for (int i = 0; i < NUM_PHILOSOPHERS; i++)
    {
        sem_destroy(&philSem[i]);
    }
    return 0;
}
```

Output:

```
base) sahyadri@sahyadri:~/OS LAB_IV SEm/OS_Programs$ gcc 6.c
(base) sahyadri@sahyadri:~/OS LAB_IV SEm/OS_Programs$ ./a.out
Producer 1 produced item 1
```

```
Philosopher 3 is thinking
Philosopher 1 is thinking
Consumer 1 consumed item 1
Philosopher 4 is thinking
Producer 2 produced item 1
Philosopher 5 is thinking
Philosopher 2 is thinking
Consumer 2 consumed item 1
Producer 2 produced item 2
Producer 1 produced item 2
Producer 1 produced item 3
Consumer 2 consumed item 2
Consumer 1 consumed item 2
Philosopher 1 is hungry
Philosopher 1 is eating
Philosopher 4 is hungry
Philosopher 4 is eating
Philosopher 2 is hungry
Producer 2 produced item 3
Philosopher 5 is hungry
Consumer 2 consumed item 3
Philosopher 3 is hungry
Producer 1 produced item 4
Producer 2 produced item 4
Producer 1 produced item 5
Producer 1 produced item 6
Consumer 1 consumed item 3
Philosopher 4 finished eating
Philosopher 4 is thinking
Consumer 2 consumed item 4
Producer 2 produced item 5
Producer 2 produced item 6
Philosopher 1 finished eating
Philosopher 1 is thinking
Consumer 2 consumed item 4
Producer 1 produced item 7
Consumer 1 consumed item 5
Consumer 2 consumed item 6
Philosopher 4 is hungry
Philosopher 4 is eating
Philosopher 1 is hungry
```

```
Philosopher 1 is eating
Producer 2 produced item 7
Consumer 1 consumed item 5
Producer 2 produced item 8
Producer 1 produced item 8
Philosopher 4 finished eating
Philosopher 4 is thinking
Consumer 2 consumed item 6
Producer 1 produced item 9
Philosopher 1 finished eating
Philosopher 1 is thinking
Philosopher 4 is hungry
Philosopher 4 is eating
Consumer 1 consumed item 7
Producer 1 produced item 10
Philosopher 1 is hungry
Philosopher 1 is eating
Consumer 2 consumed item 7
Producer 2 produced item 9
Philosopher 4 finished eating
Philosopher 4 is thinking
Consumer 1 consumed item 8
Producer 1 produced item 11
Consumer 2 consumed item 8
Producer 2 produced item 10
Philosopher 1 finished eating
Philosopher 1 is thinking
Philosopher 4 is hungry
Philosopher 4 is eating
Consumer 1 consumed item 9
Producer 1 produced item 12
Philosopher 1 is hungry
Philosopher 1 is eating
Consumer 1 consumed item 10
Producer 2 produced item 11
Consumer 2 consumed item 9
Philosopher 4 finished eating
Philosopher 4 is thinking
Producer 2 produced item 12
Consumer 2 consumed item 11
Producer 1 produced item 13
```

```
Philosopher 4 is hungry
Philosopher 4 is eating
Philosopher 1 finished eating
Philosopher 1 is thinking
Philosopher 4 finished eating
Philosopher 4 is thinking
Consumer 1 consumed item 10
Producer 2 produced item 13
Consumer 2 consumed item 12
Producer 1 produced item 14
Philosopher 4 is hungry
Philosopher 4 is eating
Philosopher 1 is hungry
Philosopher 1 is eating
Consumer 1 consumed item 11
Philosopher 1 finished eating
Philosopher 1 is thinking
Consumer 2 consumed item 12
Producer 1 produced item 15
Philosopher 1 is hungry
Philosopher 1 is eating
Producer 2 produced item 14
Consumer 2 consumed item 13
```

7. Demonstrate following page replacement algorithms- a. FIFO, b. LRU and c. OPTIMAL.

……

Program:

FIFO

```
#include<stdio.h>
int fr[3];
int main()
{
        void display();
        int i,j,page[12]={2,3,2,1,5,2,4,5,3,2,5,2};
        int flag1=0,flag2=0,pf=0,frsize=3,top=0;
        for(i=0;i<3;i++)
        {
                fr[i]=-1;
        }
        for(j=0;j<12;j++)
        {
                flag1=0;
                flag2=0;
                for(i=0;i<3;i++)
                {
                        if(fr[i]==page[j])
                        {
                                flag1=1;
                                flag2=1;
                                break;
                        }
                }
                if(flag1==0)
                {
                        for(i=0;i<frsize;i++)
                        {
                                if(fr[i]==-1)
                                {
                                        fr[i]=page[j];
                                        flag2=1;
                                        break;
                                }
                        }
                }

                if(flag2==0)
                {
```

```
                        fr[top]=page[j];
                        top++;
                        pf++;
                        if(top>=frsize)
                                top=0;
                }
                display();
        }
        printf("\nNumber of page faults : %d ",pf+frsize);
        return 0;
}
void display()
{
        int i;
        printf("\n");
        for(i=0;i<3;i++)
                printf("%d\t",fr[i]);
}
```

Output:

```
2       -1      -1
2       3       -1
2       3       -1
2       3       1
5       3       1
5       2       1
5       2       4
5       2       4
3       2       4
3       2       4
3       5       4
3       5       2
Number of page faults : 9
```

LRU

```
#include<stdio.h>
```

```
int fr[3];
int  main()
{
        void display();
        int p[12]={2,3,2,1,5,2,4,5,3,2,5,2},i,j,fs[3];
        int index,k,l,flag1=0,flag2=0,pf=0,frsize=3;
        for(i=0;i<3;i++)
        {
                fr[i]=-1;
        }
        for(j=0;j<12;j++)
        {
                flag1=0,flag2=0;
                for(i=0;i<3;i++)
                {
                        if(fr[i]==p[j])
                        {
                                flag1=1;
                                flag2=1;
                                printf("\tflag %d -%d",flag1,flag2);
                                break;
                        }
                }
                if(flag1==0)
                {
                        for(i=0;i<3;i++)
                        {
                                if(fr[i]==-1)
                                {
                                        fr[i]=p[j];
                                        flag2=1;
                                        break;
                                }
                        }
                }
                if(flag2==0)
                {
                        for(i=0;i<3;i++)
                                fs[i]=0;

                        for(k=j-1,l=1;l<=frsize-1;l++,k--)
                        {
```

```
                        for(i=0;i<3;i++)
                        {
                                if(fr[i]==p[k])
                                fs[i]=1;
                        }
                }
                for(i=0;i<3;i++)
                {
                        if(fs[i]==0)
                        index=i;
                }
                fr[index]=p[j];
                pf++;
        }
        display();
    }
    printf("\n no of page faults :%d",pf+frsize);
    return 0;
}
void display()
{
    int i; printf("\n");
    for(i=0;i<3;i++)
        printf("\t%d",fr[i]);
}
```

Output:

```
2       -1      -1
2       3       -1      flag 1 -1
2       3       -1
2       3       1
2       5       1       flag 1 -1
2       5       1
2       5       4       flag 1 -1
2       5       4
3       5       4
3       5       2       flag 1 -1
3       5       2       flag 1 -1
3       5       2
no of page faults :7
```

OPTIMAL

```
#include<stdio.h>
```

```
int fr[3], n, m;
void display();
int main()
{
        int i,j,page[20],fs[10];
        int max,found=0,lg[3],index,k,l;
        int flag1=0,flag2=0,pf=0;
        float pr;
        printf("Enter length of the reference string: ");
        scanf("%d",&n);
        printf("Enter the reference string: ");
        for(i=0;i<n;i++)
                scanf("%d",&page[i]);
        printf("Enter no of frames: ");
        scanf("%d",&m);
        for(i=0;i<m;i++)
                fr[i]=-1; pf=m;
        for(j=0;j<n;j++)
        {
                flag1=0; flag2=0;
                for(i=0;i<m;i++)
                {
                        if(fr[i]==page[j])
                        {
                                flag1=1; flag2=1;
                                break;
                        }
                }
                if(flag1==0)
                {
                        for(i=0;i<m;i++)
                        {
                                if(fr[i]==-1)
                                {
                                        fr[i]=page[j]; flag2=1;
                                        break;
                                }
                        }
                }

                if(flag2==0)
                {
```

```
                for(i=0;i<m;i++)
                        lg[i]=0;
                for(i=0;i<m;i++)
                {
                        for(k=j+1;k<=n;k++)
                        {
                                if(fr[i]==page[k])
                                {
                                        lg[i]=k-j;
                                        break;
                                }
                        }
                }
                found=0;
                for(i=0;i<m;i++)
                {
                        if(lg[i]==0)
                        {
                                index=i;
                                found = 1;
                                break;
                        }
                }
                if(found==0)
                {
                        max=lg[0]; index=0;
                        for(i=0;i<m;i++)
                        {
                                if(max<lg[i])
                                {
                                        max=lg[i];
                                        index=i;
                                }
                        }
                }
                fr[index]=page[j];
                pf++;
        }
        display();
}
printf("Number of page faults : %d\n", pf);
pr=(float)pf/n*100;
```

```
        printf("Page fault rate = %f \n", pr);
}
void display()
{
        int i;
        for(i=0;i<m;i++)
                printf("%d\t",fr[i]);
        printf("\n");
}
```

Output:

Enter length of the reference string: 12
Enter the reference string: 2 3 2 1 5 2 4 5 3 2 5 2
Enter no of frames: 3

2	-1	-1
2	3	-1
2	3	-1
2	3	1
2	3	5
2	3	5
4	3	5
4	3	5
4	3	5
2	3	5
2	3	5
2	3	5

Number of page faults : 6
Page fault rate = 50.000000

8. Analyze the seek time for the following Disk scheduling algorithms –

1. FCFS,

2. SCAN,
3. LOOK

……

Program:

1.FCFS

```
#include<stdio.h>
#include<stdlib.h>

int main()
{
	int n, i, head, total = 0;
	printf("Enter the number of requests: ");
	scanf("%d", &n);
	int requests[n];
	printf("Enter the request queue:\n");
	for(i = 0; i < n; i++)
	{
		scanf("%d", &requests[i]);
	}
	printf("Enter the initial head position: ");
	scanf("%d", &head);
	printf("Head Movement Order:\n");
	for(i = 0; i < n; i++)
	{
		printf("%d -> ", head);
		total += abs(requests[i] - head);
		head = requests[i];
	}
	printf("%d->", head);
	printf("End\n");
	printf("Total head movements: %d\n", total);
	return 0;
}
```

Output:

Enter the number of requests: 7

Enter the request queue:
93 176 42 148 27 14 18
Enter the initial head position: 55
Head Movement Order:
55 -> 93 -> 176 -> 42 -> 148 -> 27 -> 14 -> 18->End
Total head movements: 499

2.SCAN

```
#include<stdio.h>
```

```
#include<stdlib.h>

int main()
{
	int n, i, head, direction, total = 0,f;
	printf("Enter the number of requests: ");
	scanf("%d", &n);
	int requests[n];
	printf("Enter the request queue:\n");
	for(i = 0; i < n; i++)
	{
		scanf("%d", &requests[i]);
	}
	printf("Enter the initial head position: ");
	scanf("%d", &head);
	printf("Enter the direction (0 for left, 1 for right): ");
	scanf("%d", &direction);
	// Sort the request queue for SCAN algorithm
	for(i = 0; i < n - 1; i++)
	{
		for(int j = 0; j < n - i - 1; j++)
		{
			if(requests[j] > requests[j+1])
			{
				 int temp = requests[j];
				requests[j] = requests[j+1];
				requests[j+1] = temp;
			}
		}
	}
	int current = head;
	if (direction == 1)
	{
		for(i = 0; i < n; i++)
		{
			if (requests[i] >= head)
			{
				break;
			}
		}
	}
	else
```

```
{
        for(i = n - 1; i >= 0; i--)
        {
                if (requests[i] <= head)
                {
                        break;
                }
        }
}
f=i;
printf("Head Movement Order:\n");
if (direction == 1)
{
        for(; i < n; i++)
        {
                printf("%d -> ", current);
                total += abs(requests[i] - current);
                current = requests[i];
        }
        if (current == requests[n-1])
        {
                printf("%d -> ", requests[n-1]);
                total += abs(199-requests[n-1] );
                printf("199 -> ");
                current = 199;
        }
        for(i = f - 1; i >= 0; i--)
        {
                total += abs(requests[i] - current);
                current = requests[i];
                printf("%d -> ", current);
        }
}
else
{
        for(; i >= 0; i--)
        {
                printf("%d -> ", current);
                total += abs(requests[i] - current);
                current = requests[i];
        }
        printf("%d -> ", current);
```

```
            if (current == requests[0])
            {
                total += abs(current - 0);
                printf("0 -> ");
                current = 0;
            }
            for(i = f+1; i < n; i++)
            {
                total += abs(requests[i] - current);
                current = requests[i];
                printf("%d -> ", current);
            }
        }
        printf("End\n");
        printf("Total head movements: %d\n", total);
        return 0;
}
```

Output:

Enter the number of requests: 7
Enter the request queue:
93 176 42 148 27 14 18
Enter the initial head position: 55
Enter the direction (0 for left, 1 for right): 0
Head Movement Order:
55 -> 42 -> 27 -> 18 -> 14 -> 0 -> 93 -> 148 -> 176 -> End
Total head movements: 231

……

sahyadri@sahyadri-desktop:~/Downloads/7-20230807T031350Z-001/7$ ./a.out
Enter the number of requests: 7
Enter the request queue:
93 176 42 148 27 14 18
Enter the initial head position: 55
Enter the direction (0 for left, 1 for right): 1
Head Movement Order:
55 -> 93 -> 148 -> 176 -> 199 -> 42 -> 27 -> 18 -> 14 -> End
Total head movements: 329

3. LOOK

```
#include<stdio.h>
#include<stdlib.h>

int main()
{
	int n, i, head, direction, total = 0,f;
	printf("Enter the number of requests: ");
	scanf("%d", &n);
	int requests[n];
	printf("Enter the request queue:\n");
	for(i = 0; i < n; i++)
	{
		scanf("%d", &requests[i]);
	}
	printf("Enter the initial head position: ");
	scanf("%d", &head);
	printf("Enter the direction (0 for left, 1 for right): ");
	scanf("%d", &direction);
	// Sort the request queue for SCAN algorithm
	for(i = 0; i < n - 1; i++)
	{
		for(int j = 0; j < n - i - 1; j++)
		{
			if(requests[j] > requests[j+1])
			{
				int temp = requests[j];
				requests[j] = requests[j+1];
				requests[j+1] = temp;
			}
		}
	}
	int current = head;
	if (direction == 1)
	{
		for(i = 0; i < n; i++)
		{
			if (requests[i] >= head)
			{
				break;
			}
		}
	}
```

```
else
{
        for(i = n - 1; i >= 0; i--)
        {
                if (requests[i] <= head)
                {
                        break;
                }
        }
}
f=i;
printf("Head Movement Order:\n");
printf("%d -> ", current);
if (direction == 1)
{
        for(; i < n; i++)
        {
                total += abs(requests[i] - current);
                current = requests[i];
                printf("%d -> ", current);
        }
        for(i = f - 1; i >= 0; i--)
        {
                total += abs(requests[i] - current);
                current = requests[i];
                printf("%d -> ", current);
        }
}
else
{
        for(; i >= 0; i--)
        {
                total += abs(requests[i] - current);
                current = requests[i];
                printf("%d -> ", current);
        }
        for(i = f+1; i < n; i++)
        {
                total += abs(requests[i] - current);
                current = requests[i];
                printf("%d -> ", current);
        }
```

```
        }
        printf("End\n");
        printf("Total head movements: %d\n", total);
        return 0;
}
```

Output:

Enter the number of requests: 8
Enter the request queue:
176 79 34 60 92 11 41 114
Enter the initial head position: 50
Enter the direction (0 for left, 1 for right): 1
Head Movement Order:
50 -> 60 -> 79 -> 92 -> 114 -> 176 -> 41 -> 34 -> 11 -> End
Total head movements: 291

……

sahyadri@sahyadri-desktop:~/Downloads//7-20230807T031350Z-001/7$ ./a.out
Enter the number of requests: 8
Enter the request queue:
176 79 34 60 92 11 41 114
Enter the initial head position: 50
Enter the direction (0 for left, 1 for right): 0
Head Movement Order:
50 -> 41 -> 34 -> 11 -> 60 -> 79 -> 92 -> 114 -> 176 -> End
Total head movements: 204

Demonstrate the system assembly and disassembly of computer hardware components

……

Procedure:

Assembly:

Step 1: Gather the necessary tools and components. You'll typically need a screwdriver (usually a Phillips head), the computer case, motherboard, CPU, RAM modules, storage devices (hard drive or SSD), power supply, and any additional components like a graphics card.

Step 2: Prepare the computer case. Remove the side panel(s) to access the internal components. Ensure the case is placed on a stable surface with proper grounding.

Step 3: Install the power supply. Insert the power supply into the designated area at the top or bottom of the case and secure it with screws.

Step 4: Install the motherboard. Align the screw holes on the motherboard with the standoffs in the case. Carefully place the motherboard on top of the standoffs and secure it with screws.

Step 5: Install the CPU (Central Processing Unit). Open the CPU socket on the motherboard and gently insert the CPU chip, aligning the notches. Close the socket lever to secure the CPU in place.

Step 6: Install the CPU cooler. Apply thermal paste to the CPU, then attach the CPU cooler/fan to the designated area on the motherboard. Connect the cooler's power cable to the CPU fan header on the motherboard.

Step 7: Install RAM modules. Open the RAM slots on the motherboard and align the notch on the RAM module with the slot. Insert the RAM module firmly but gently until it clicks into place.

Step 8: Install storage devices. Connect the SATA data and power cables to the storage drives (hard drive or SSD). Mount the drives in the dedicated slots within the case and secure them with screws if necessary.

Step 9: Install additional components. If you have a graphics card or any expansion cards, insert them into the appropriate PCIe slots on the motherboard. Secure them with screws if needed. Connect power cables to the graphics card if required.

Step 10: Connect cables and wires. Connect the necessary cables, such as SATA cables for storage devices, power cables for components, front panel connectors (power button, reset button, LED lights, etc.), USB headers, and audio connectors.

Step 11: Double-check connections. Ensure all connections are secure and properly plugged in.

Disassembly:

Step 1: Power down and unplug the computer.

Step 2: Remove the side panel(s) of the computer case.

Step 3: Disconnect power cables and data cables from all components (motherboard, storage devices, graphics card, etc.).

Step 4: If necessary, remove the expansion cards (graphics card, sound card, etc.) by unscrewing them and gently pulling them out of the PCIe slots.

Step 5: Remove the RAM modules by pressing the side latches and gently pulling them out of the slots.

Step 6: Remove the CPU cooler/fan by disconnecting the power cable and unscrewing it from the motherboard.

Step 7: Carefully detach the CPU chip by releasing the socket lever and lifting the CPU out of the socket.

Step 8: Remove any additional components, such as storage devices, by unscrewing them and disconnecting cables.

Step 9: Unscrew and remove the motherboard from the case.

Step 10: Remove the power supply by unscrewing it from the case.

Demonstrate the OS installation with Multi Booting and Virtual Machine platform, OS Protection and Security – Firewall, Protections, Threat Detections and prevention.

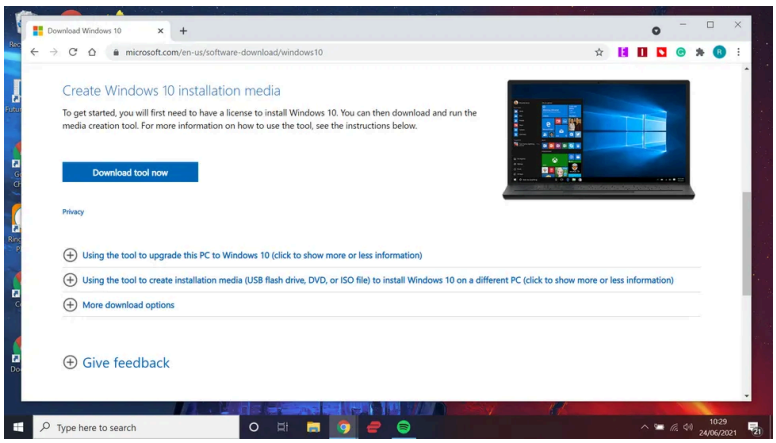
……

How to install Windows 10: Full installation

Before we get into the installation walkthrough, it's worth noting that installing Windows 10 can be a lengthy process. Make sure you have enough time to wait it out.

1. Check your device meets the Windows 10 system requirements. Below you'll find the minimum specs needed to run Windows 10, so check your device is capable:
CPU: 1GHz or faster processor
RAM: 1GB for Windows 10 32-bit or 2GB for Windows 10 64-bit
Storage: 32GB of space or more
GPU: DirectX 9 compatible or later with WDDM 1.0 driver
Display: 800x600 resolution or higher

2. Create USB installation media. Visit Microsoft's Windows 10 download page and select "Download tool now" under the "create Windows 10 installation media" section. Transfer the downloaded installer tool to a USB drive.



3. Run the installer tool. Open the installer tool by clicking on it. Accept Microsoft's terms, and then **select "Create installation media for another PC"** on the "What do you want to do?" page. After selecting which language you want Windows 10 to run in, and which edition you want as well (32-bit or 62-bit), you'll be asked what type of media you want to use.
Installing from a USB drive is definitely the preferred option but you can also install from a CD or ISO file. Once you choose your device, the installer tool will download the required files and put them onto your drive.

4. Use your installation media. Insert your installation media into your device and then **access the computer's BIOS or UEFI**. These are the systems that allow you to control your computer's core hardware.
The process of accessing these systems is unique to each device, but the manufacturer's website should be able to give you a helping hand here. Generally, you'll need to **press the F2, F12 or Delete keys** as your computer boots up.

5. Change your computer's boot order. Once you have access to your computer's BIOS/UEFI you'll need to locate the settings for boot order. You need the Windows 10 installation tool to be higher up on the list than the device's current current boot drive: this is the SSD or HDD that your existing OS is stored on. You should **move the drive with the installer files to the very top of the boot order menu**. Now, when you restart your device the Windows 10 installer should load up first.

6. Restart your device. Save your settings in the BIOS/UEFI and reboot your device.

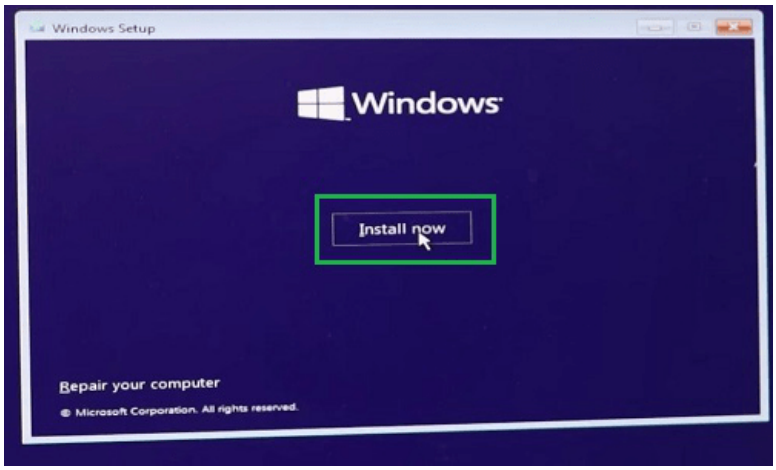
7. Complete the installation. Your device should now load up the Windows 10 installation tool on restart. This will guide you through the rest of the installation process.

Windows 10 Installation Step by Step

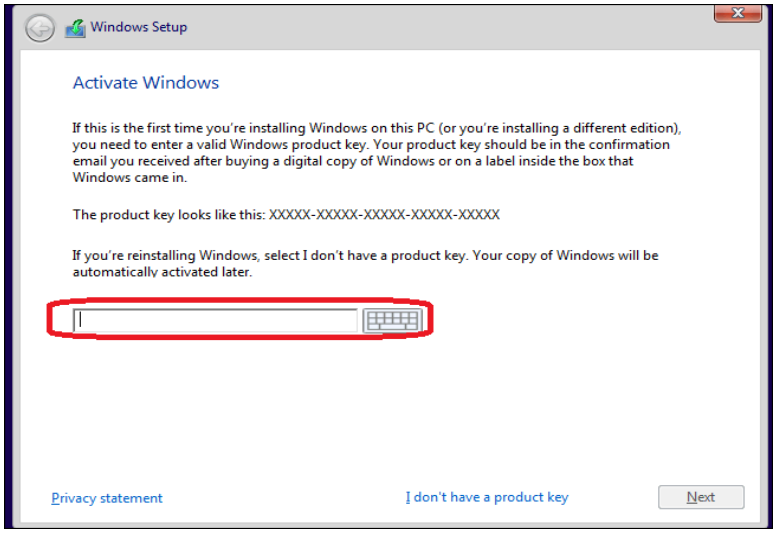
After you have successfully followed the above steps, your system will now boot and is in the process of installing the system. Here is a series of options that we will now go through together:

Select the desired language and region.

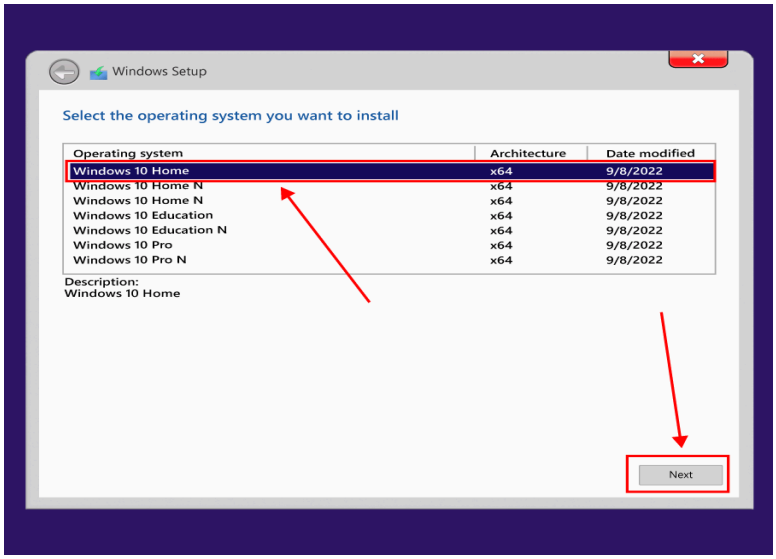
Now click on "Install now".



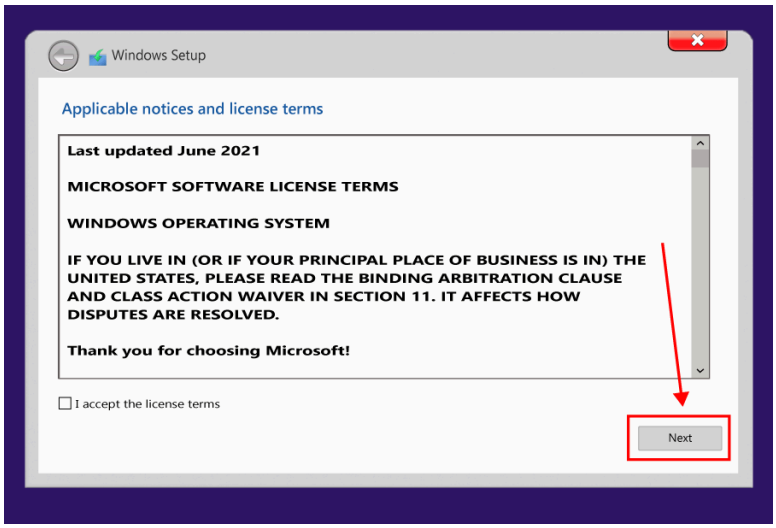
Now you can enter your Windows 10 product key.



On the next page, so many options will be present for Windows 10. Select the **Default** one & click **Next**.

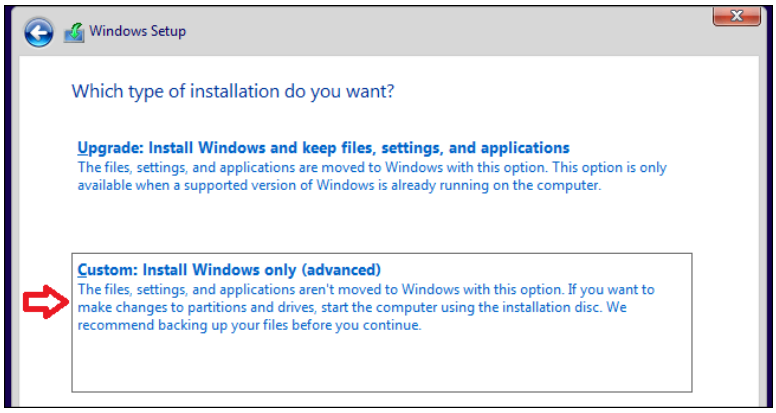


Now accept the license terms and click on “Next”.

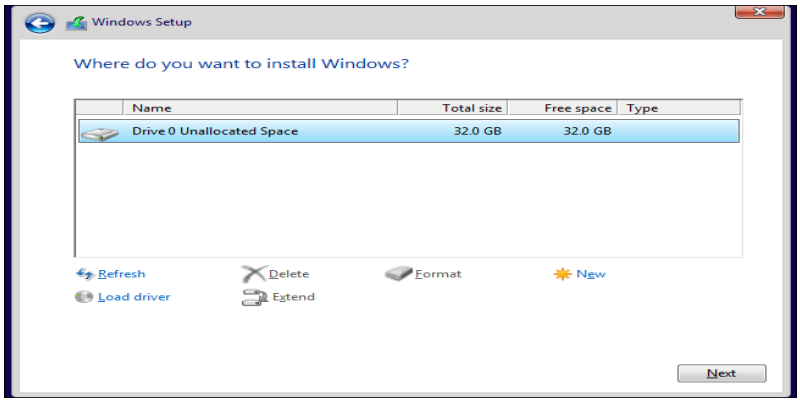


Now you can choose whether you want to upgrade an existing system ("Upgrade: Windows ….") or whether you want a so-called "Clean Install" ("Custom: …"). The latter will delete all data if you install it on an existing partition. Please note that the upgrade only works if the same bit version is selected as for the original system. So it is possible to upgrade from a 32bit system to a Windows 10 system, but not from e.g. from 64bit to 32bit.

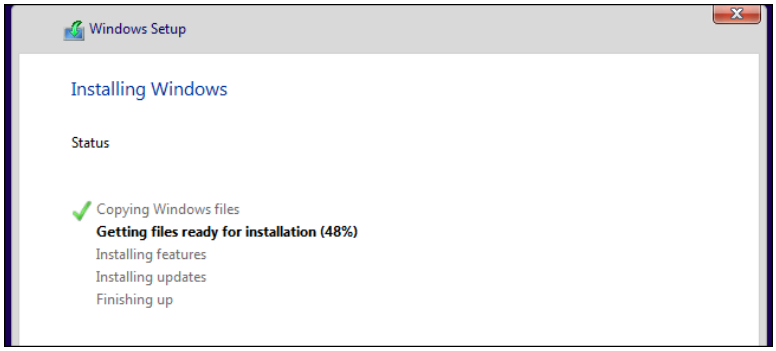
It is only possible to upgrade from Windows 7 to Windows 10, unfortunately older versions such as Windows XP or Vista are no longer supported. In any case, make sure to **backup your data** , as there is always a risk of data loss! In this guide, we opted for a clean reinstallation, ie "user-defined".



Now select the partition on which Windows 10 is to be installed. If necessary, delete or format them beforehand ("delete" or "format" command), you can also create a new partition ("new").



Now Windows is installed. This can take some time.

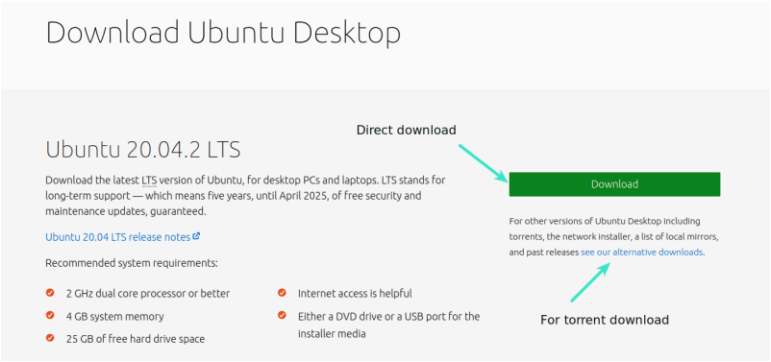


Dual Booting Ubuntu

Step 1: Make a backup of your Windows system [optional]

It is always nice to have a backup of your data, just in case you mess up with the system while dealing with disk partitions.

Step 2: Download Ubuntu (or whichever Linux distribution you are using)



Head over to Ubuntu's website and download the ISO file. The file should be around 2.5 GB in size. If you need to download Ubuntu via torrents, you can click the 'alternative downloads.'

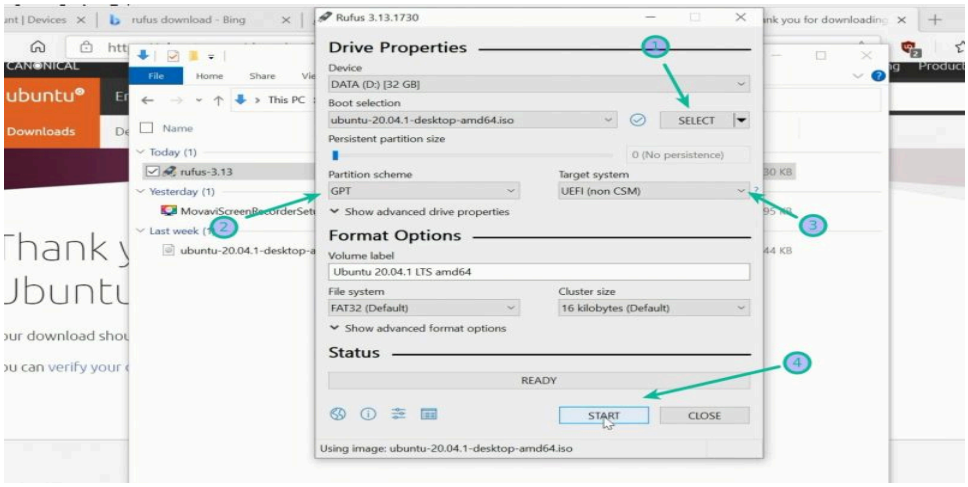
Step 3: Create a live USB/disk of Ubuntu

I presume that you are using Windows to create the live USB. Several free applications allow you to create a live Ubuntu USB. You can use any of these tools. We will go with Rufus.

Download Rufus for free from its website. Download Rufus It will download a .exe file.

Plug in your USB. This device will be formatted so make sure you don't have any important data on this USB disk.

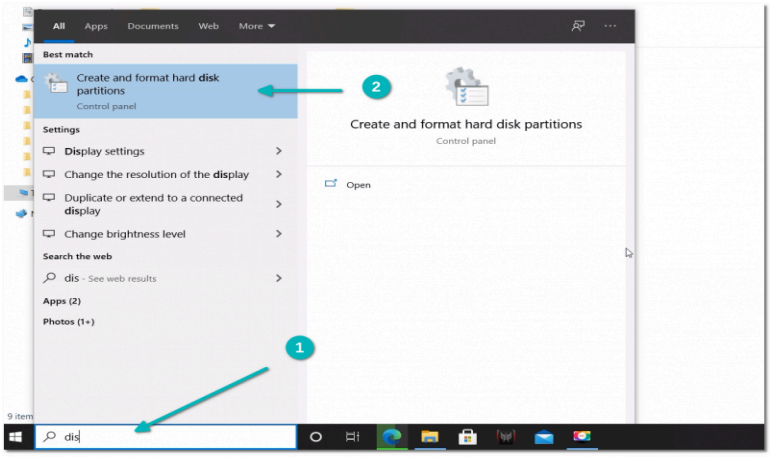
Run the Rufus tool you just downloaded. It automatically identifies the plugged-in USB but double-check it anyway. Now, browse to the location of the downloaded ISO image and ensure that it uses GPT partitioning scheme and UEFI target system.



Hit the start button and wait for the process to complete. Your live Linux USB is ready.

Step 4: Make some free space on your disk for Ubuntu installation

In many systems, while installing Ubuntu, it gives the option to make a disk partition for Ubuntu. However, that is not a surety. This is why making the required free space on the disk would be better before starting the installation procedure. In the Windows menu, search for 'disk partitions' and go to 'Create and format hard disk partitions.'



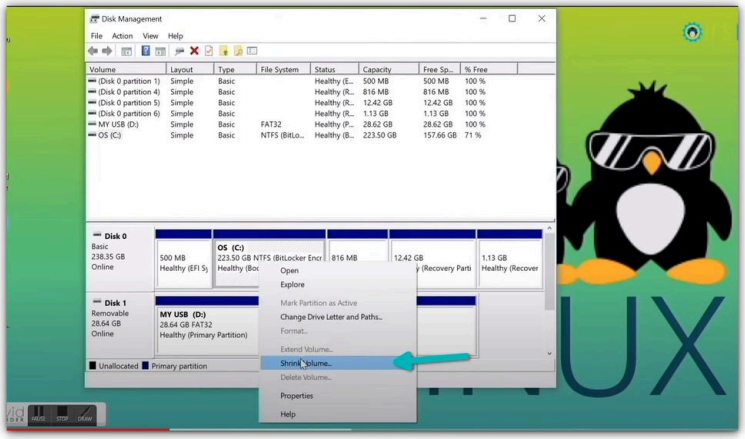
In the Disk Management tool, right-click on the drive which you want to partition and select **shrink volume**. If you have just one partition like this, you need to make some free space out of it for Linux. Use any of them except C drive if you have several partitions of considerable size because it may erase the data. If you have only C drive, shrank the C drive to make some free space for Linux installation.

How much space do you need for Linux in dual boot?

This depends on how much total disk space you have. You may install Ubuntu on 15 or 20 GB but you'll soon start running out of disk space. These days, you should have at least 120 GB of disk. In that case, go for 30-40 GB of disk for Linux. If you have 250 GB disk, allocate 60-80 GB or even more. If you have more disk space, allocate even more free space, if you want.

What if you have D, E or F drives?

This is a common confusion for many people as they think Ubuntu can only be installed on the C drive. That's not true. I had only one C drive, so I shrank it. If you have D, E or F drive, you may shrink one of those drives. You may also choose to delete the D, E or F drive. **NEVER DELETE C DRIVE**.



How much space do you need for Linux in dual boot?

This depends on how much total disk space you have. You may install Ubuntu on 15 or 20 GB but you'll soon start running out of disk space. These days, you should have at least 120 GB of disk. In that case, go for 30-40 GB of disk for Linux. If you have 250 GB disk, allocate 60-80 GB or even more. If you have more disk space, allocate even more free space, if you want.

What if you have D, E or F drives?

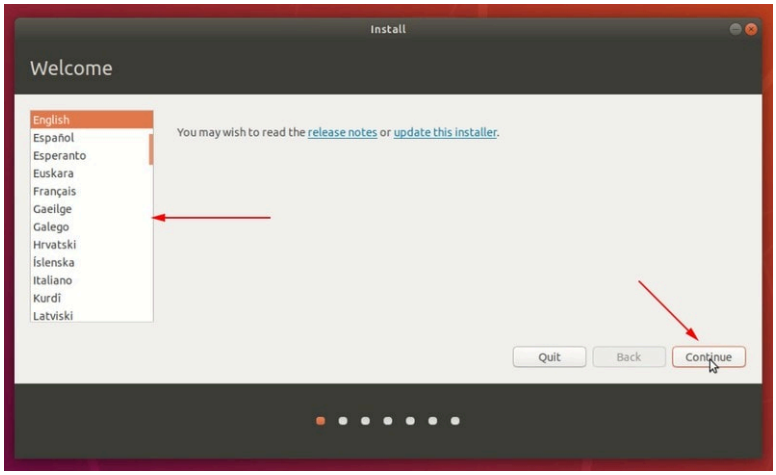
This is a common confusion for many people as they think Ubuntu can only be installed on the C drive. That's not true. I had only one C drive, so I shrank it. If you have D, E or F drive, you may shrink one of those drives. You may also choose to delete the D, E or F drive. **NEVER DELETE C DRIVE**.

Step 5: Boot from live Ubuntu USB

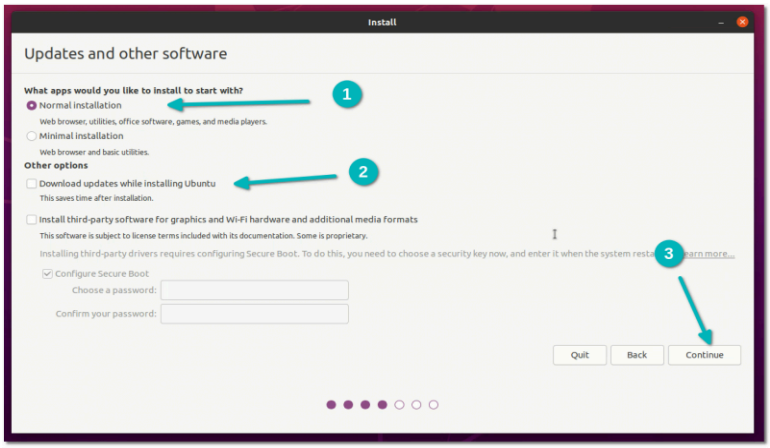
You created a live Ubuntu USB in step 3. Plug it into the system. You can go to the boot settings by pressing F2/F10 or F12 at the system start time and selecting to boot from the USB.

Step 6: Installing Ubuntu along with Windows 10

Start the installation procedure. The first few steps are simple. You choose the language and keyboard layout.



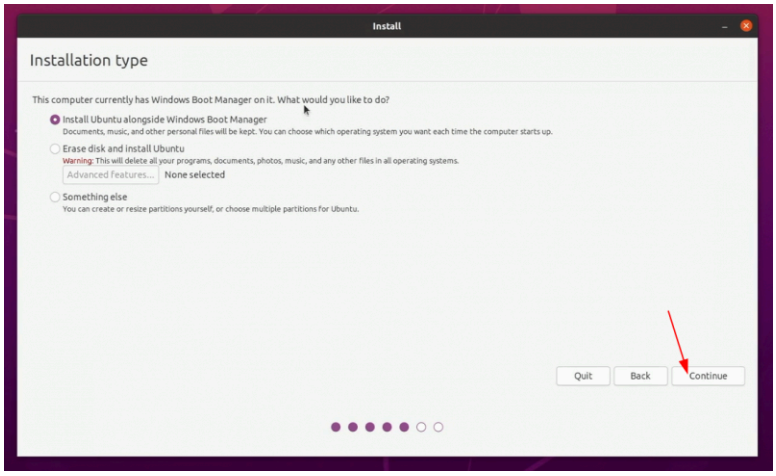
On the next screen, choose Normal installation. No need to download updates or install third-party software just yet. You may do it after the installation completes. Hit continue. It may take some time to go to the next step.



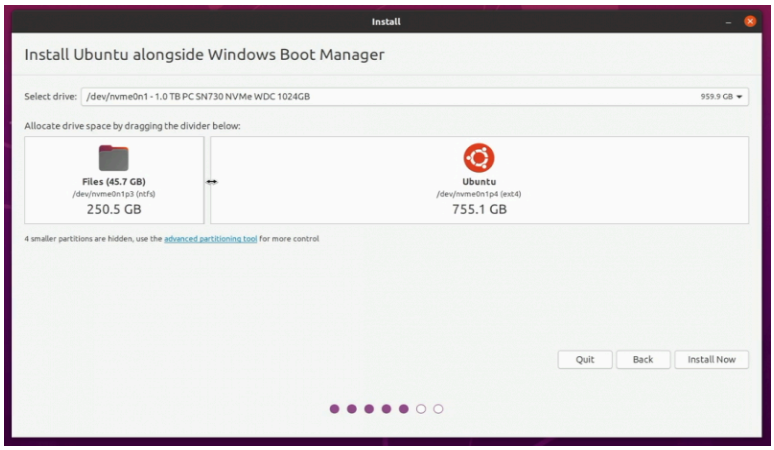
Important: Installation takes two approaches based on what you see on the next screen

Approach 1: You see the "Install Ubuntu alongside Windows Boot Manager"

If you see the "Install Ubuntu alongside Windows Boot Manager" on the **Installation type** screen, you are in luck. You can select this method and hit continue.



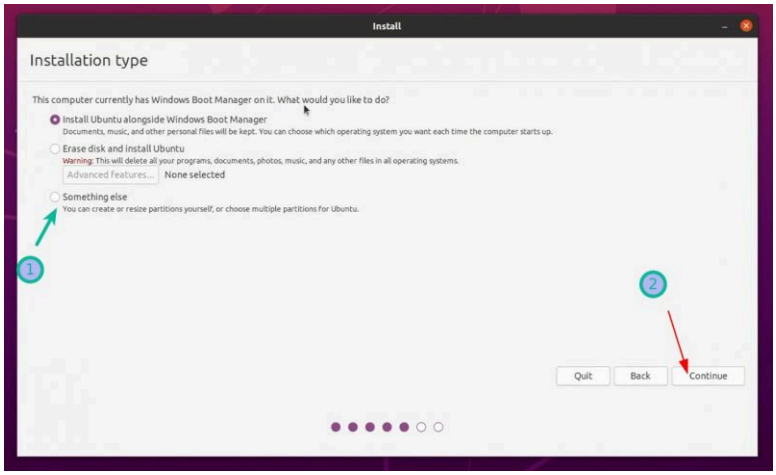
The next screen will give you the option to create a partition for Ubuntu by dragging the divider. You can allocate appropriate disk space to Linux here. Ubuntu will create one partition of the allocated disk space and it will have root with home and a swap file of 2 GB in size under root itself.



Approach 2: You don't see 'Install Ubuntu alongside Windows Boot Manager' option or it is greyed out

But if you are one of the unlucky ones who don't see this option, no need to worry. Things are not that bad for you. You can still install Ubuntu with Windows.

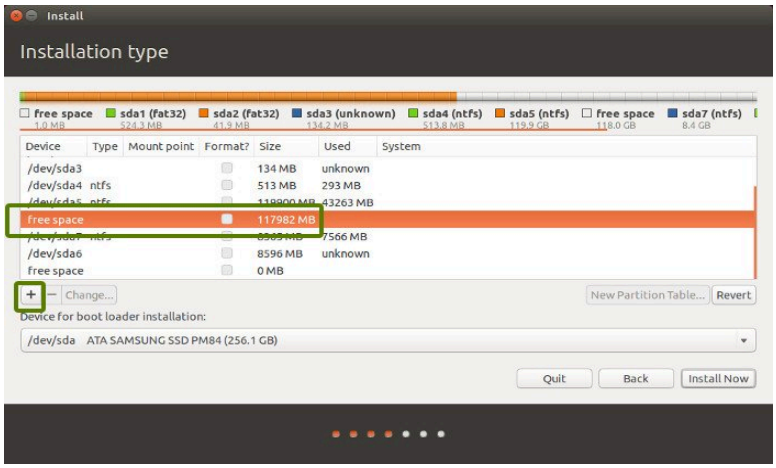
On the Installation type screen, go with **Something Else**.



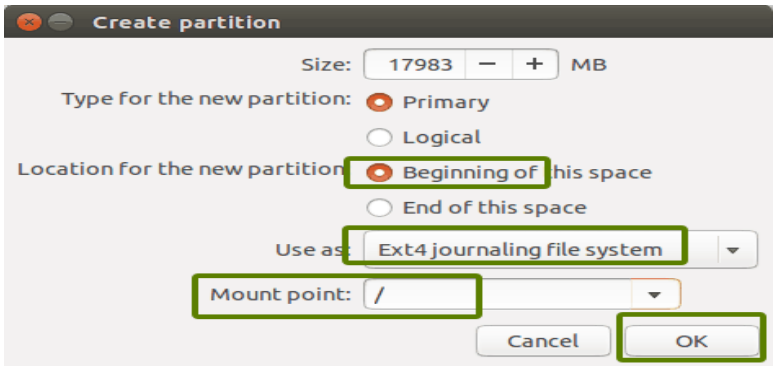
It will take you to the partitioning screen. Remember you had created some free space beforehand?

You may allocate the entire free space to root (swapfile and home will be created automatically under root) or separate root, swap and home partitioning. Both methods are acceptable.

I show the steps for creating root, swap and home partitions separately. But feel free to use a single partition for all of them. Select the free space and click on the + sign.



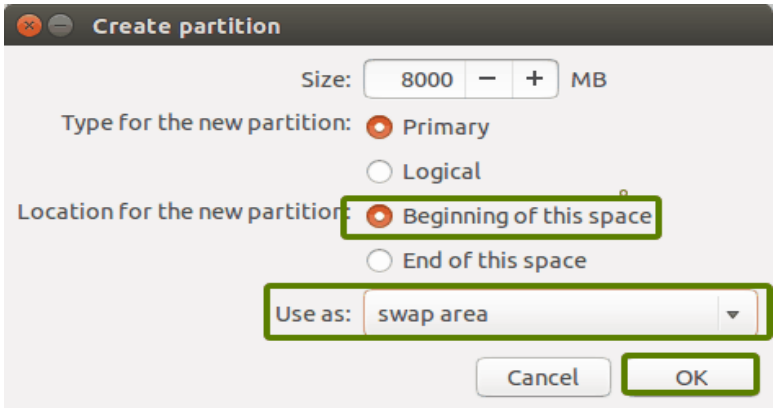
It will provide you with the option to create a Linux partition. You are creating the Root partition. Anything above 25 GB is more than sufficient for it. Choose the size, select Ext 4 as the file type and / (means root) as the mount point.



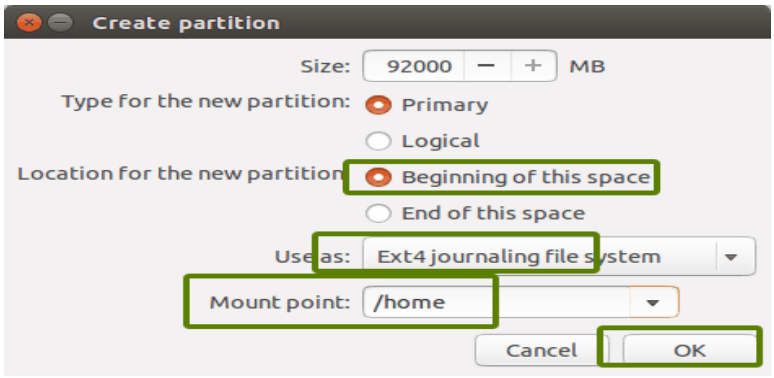
Clicking on OK in the previous step will bring you to the partition screen. Next, create a swap. Like previously, click on the + sign again. This time, use the file type as the Swap area.

If you have 2 GB or less RAM, use swap double the size of RAM. If you have 3-6 GB of RAM, use a swap of the same size as RAM. If you have 8 GB or more RAM, you may use swap half the size of RAM (unless you have plenty of disk space, and you want to use hibernation and in that case, use a swap of at least the same size as RAM).

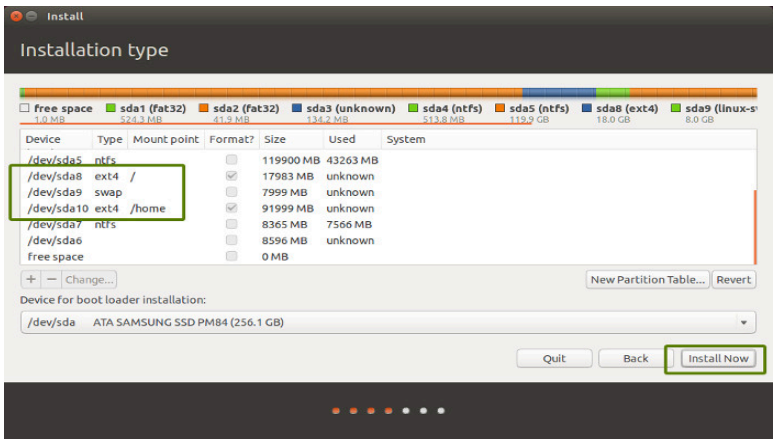
If you feel like you have less swap on your system, don't worry. You can [easily create swapfile and add more swap space to your systems].



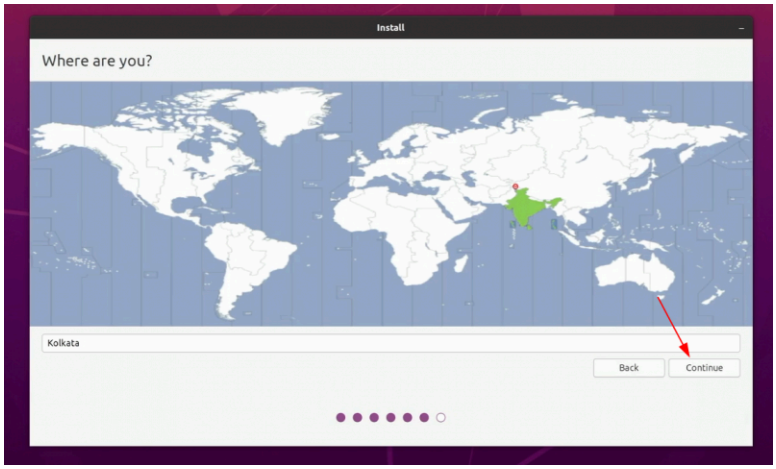
Similarly, create a Home partition. Allocate it maximum space (in fact allocate it the rest of the free space) because this is where you'll save music, pictures and downloaded files.



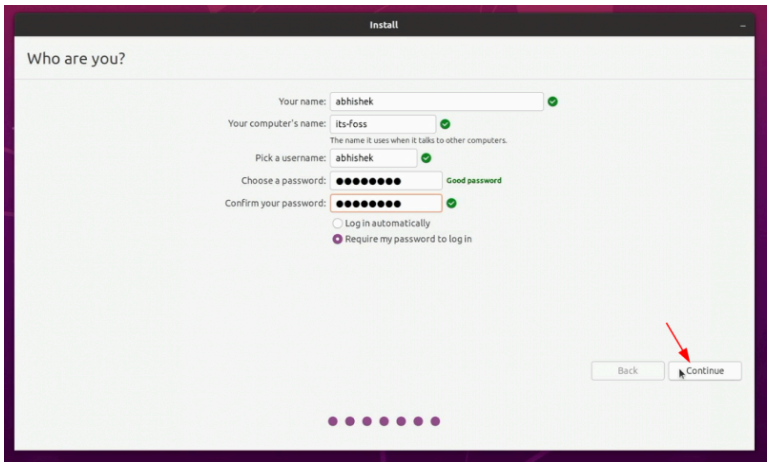
Once you are ready with Root, Swap and Home, click on **Install Now**:



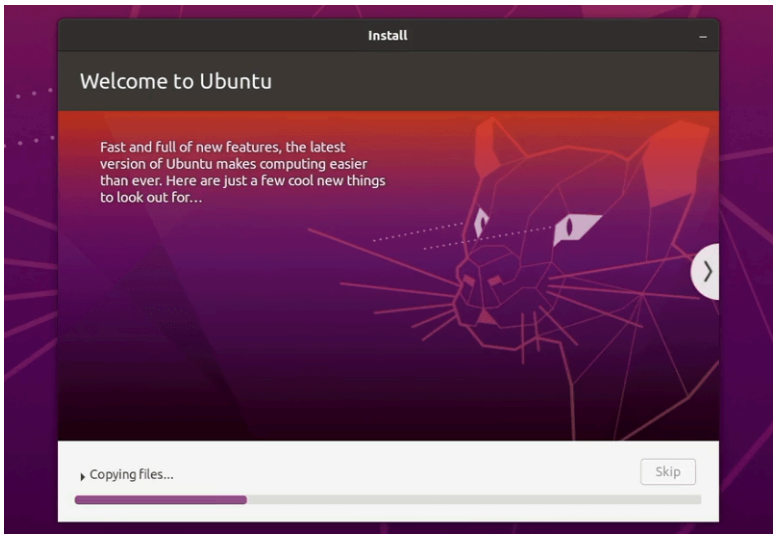
Select a timezone when asked.



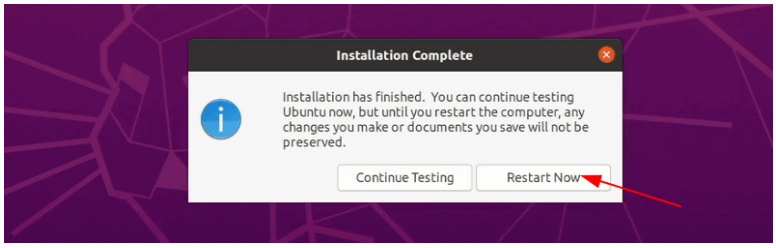
Next, you'll be asked to enter a username, hostname (computer's name) and password.



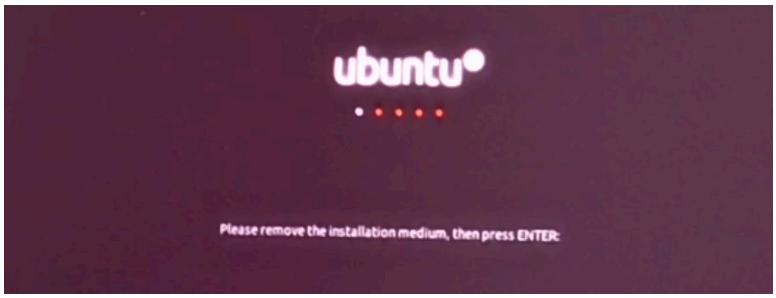
Wait for 8-10 minutes to complete the installation.



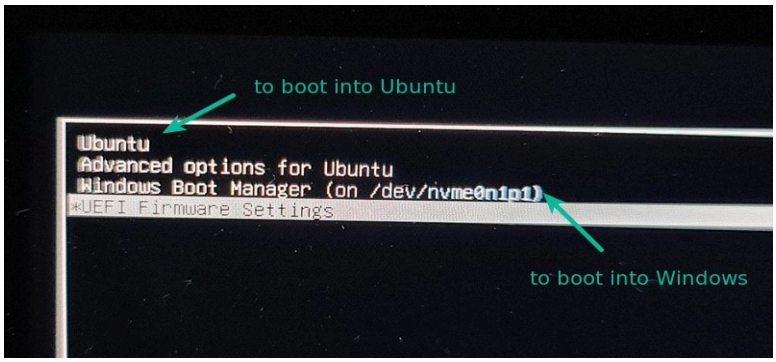
Once the installation finishes, restart the system.



You'll be asked to remove the USB disk. The system reboots after this.



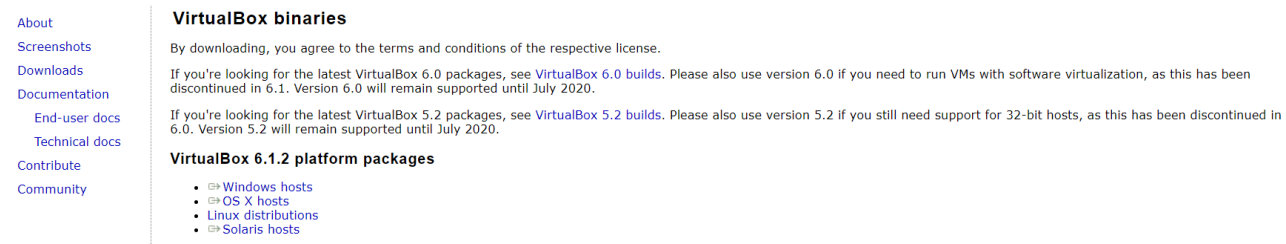
If everything goes smoothly, you should see the grub screen once the system powers on. Here, you can choose Ubuntu to boot into Ubuntu and Windows boot manager to boot into Windows.



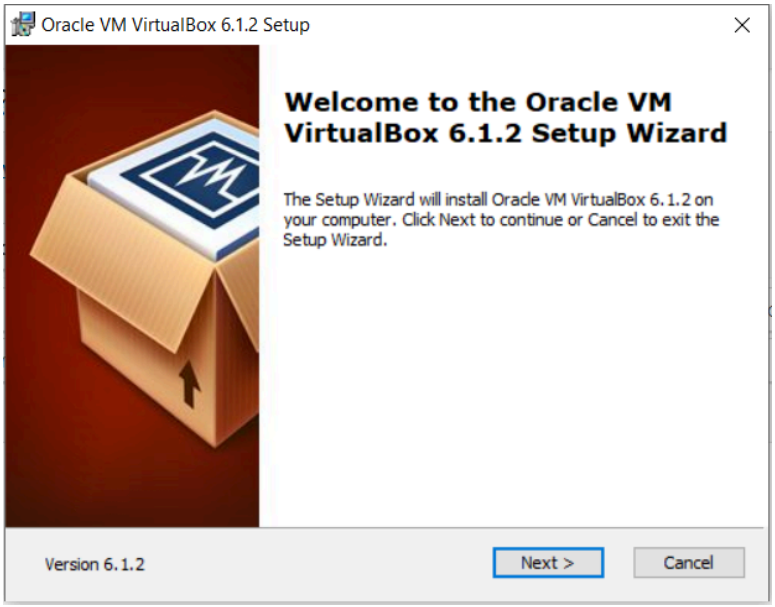
Virtual Machine platform

Downloading and Installing VirtualBox

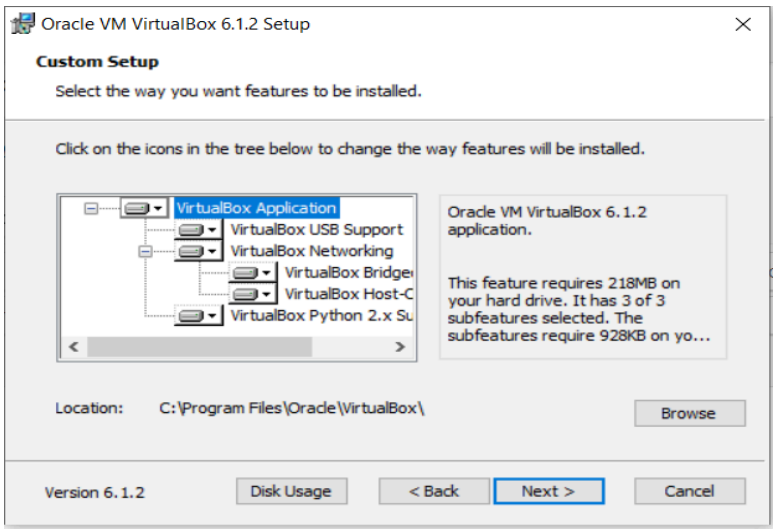
To download VirtualBox, go to the official site **virtualbox.org** and download the latest version for windows.



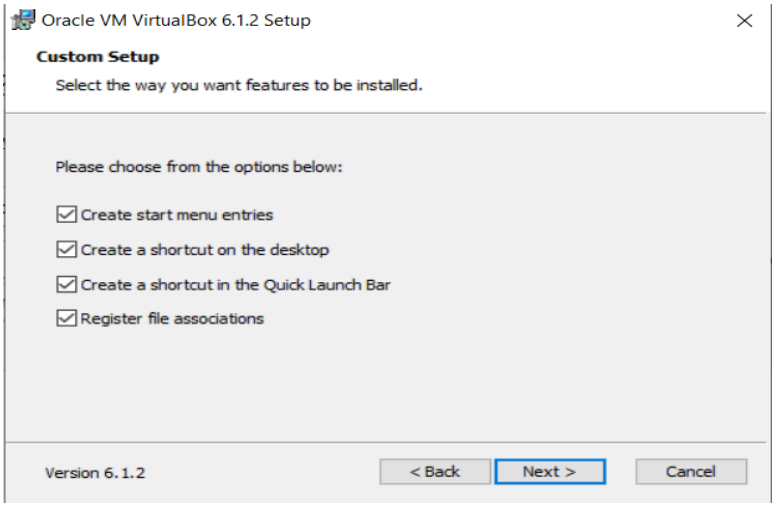
Getting Started:



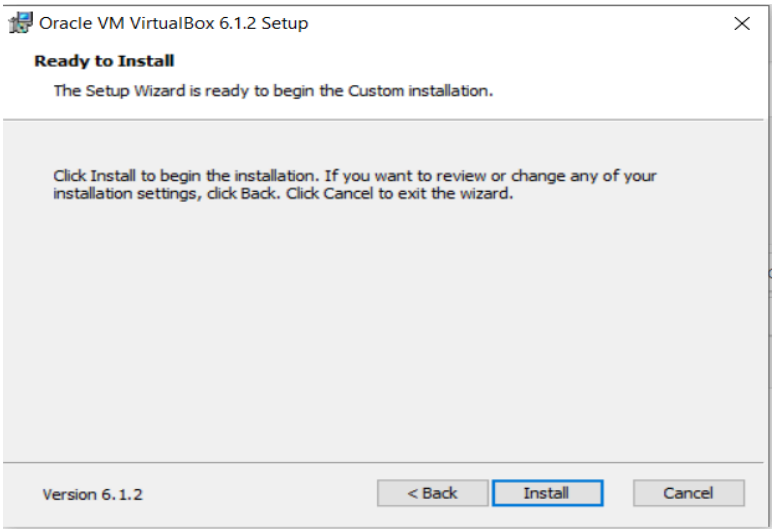
Select Installation Location:



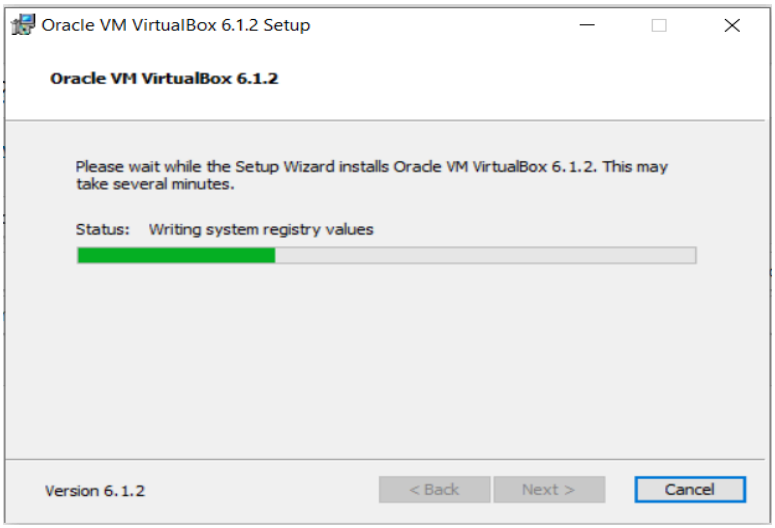
Creating Entries and Shortcuts:



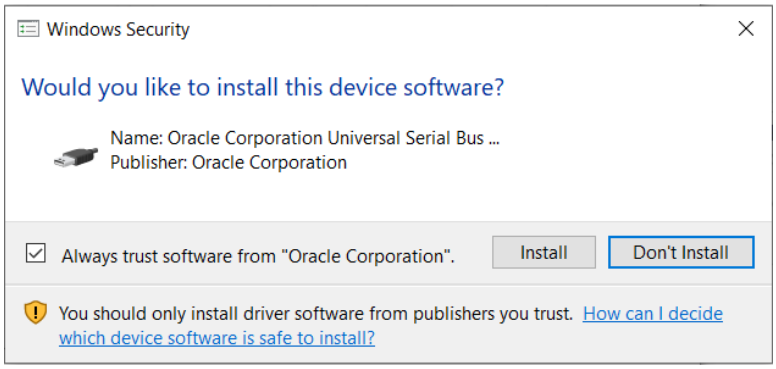
Ready to Install:



Installing Files and packages:

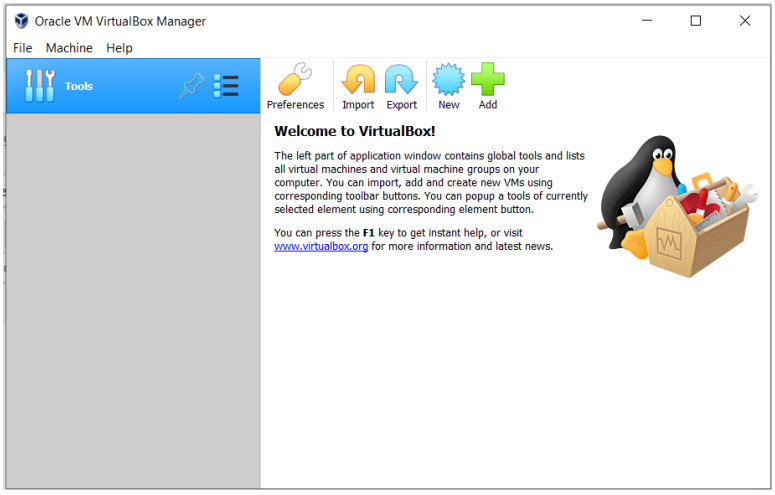


Installing Certificates:



Finished Installation:

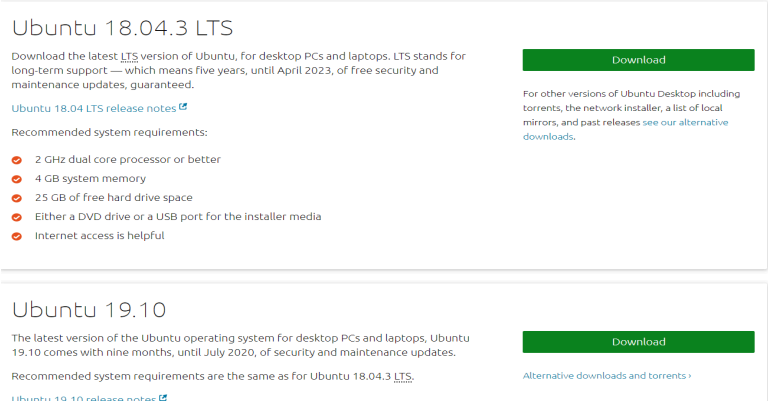
When you will open virtualbox it will look like as shown below:



<u>How to install VirtualBox on Windows?</u>

Downloading and Installing Ubuntu

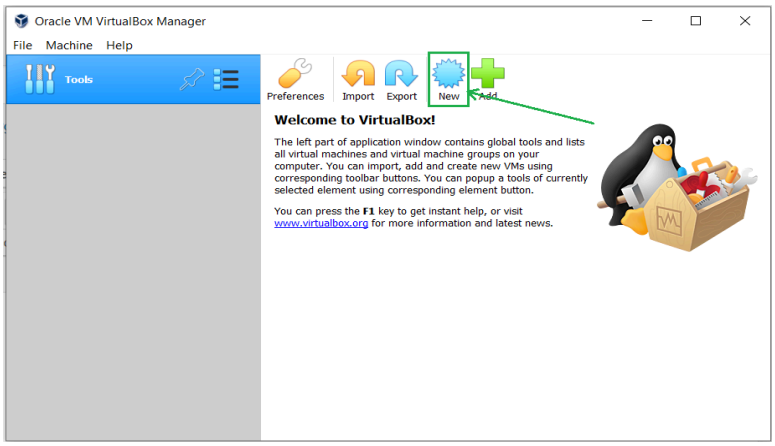
Before, we begin with the installation process, we need to download ISO file for Ubuntu. For that, all the versions of Ubuntu are available on the official site ubuntu.com



After the downloading is over, you can install Ubuntu on VirtualBox with the help of following instructions:

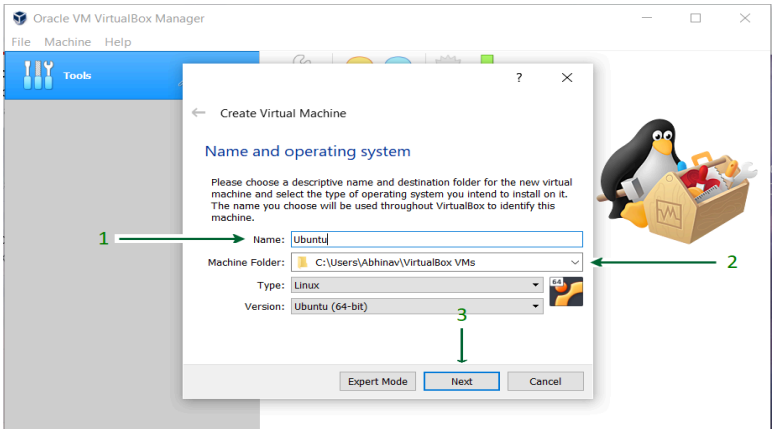
Step 1:

Open **VirtualBox** and click on the **New** button.



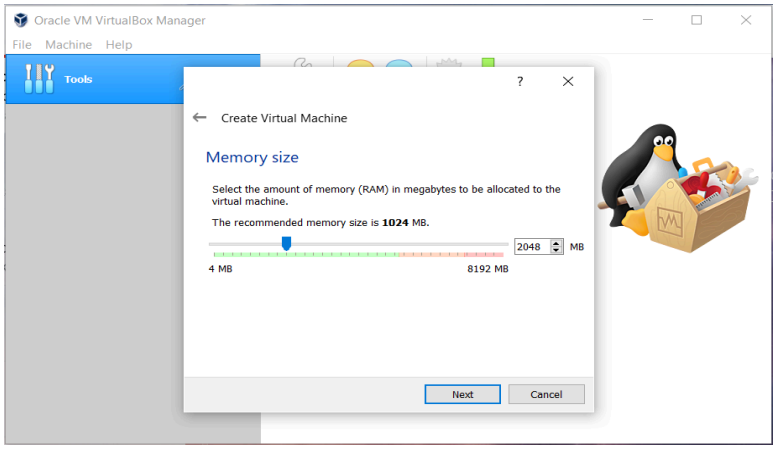
Step 2:

Give a name to your Virtual Machine and select the location for it to install.



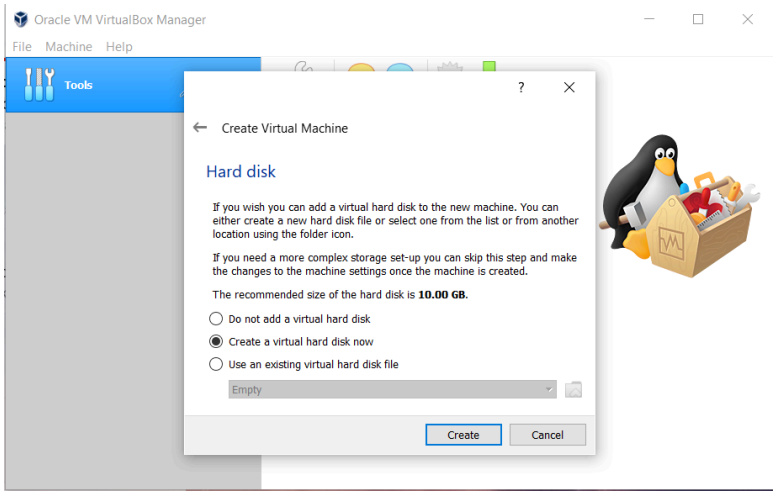
Step 3:

Assign RAM size to your Virtual Machine.



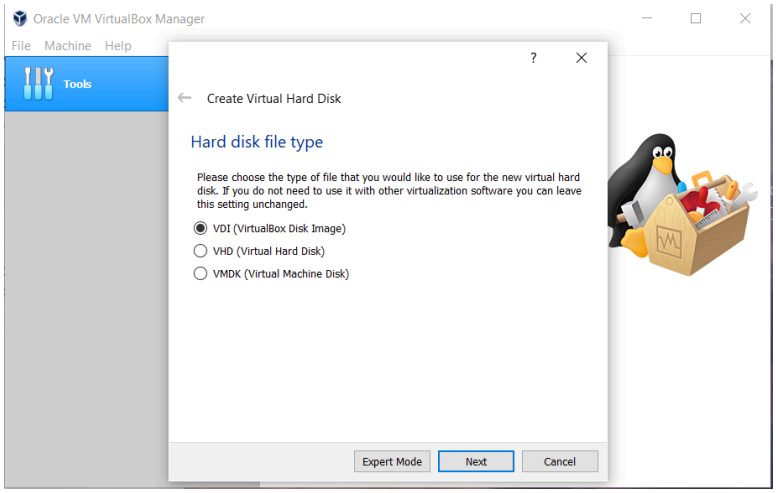
Step 4:

Create a Virtual Hard disk for the machine to store files.



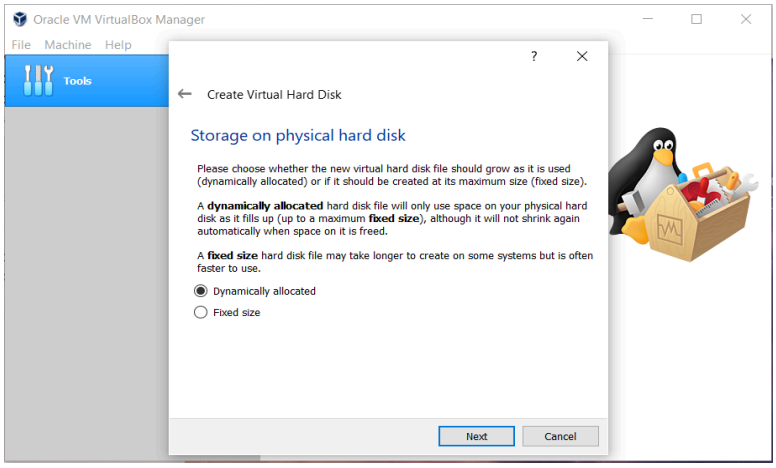
Step 5:

Select the type of Hard disk. Using **VDI** type is recommended.



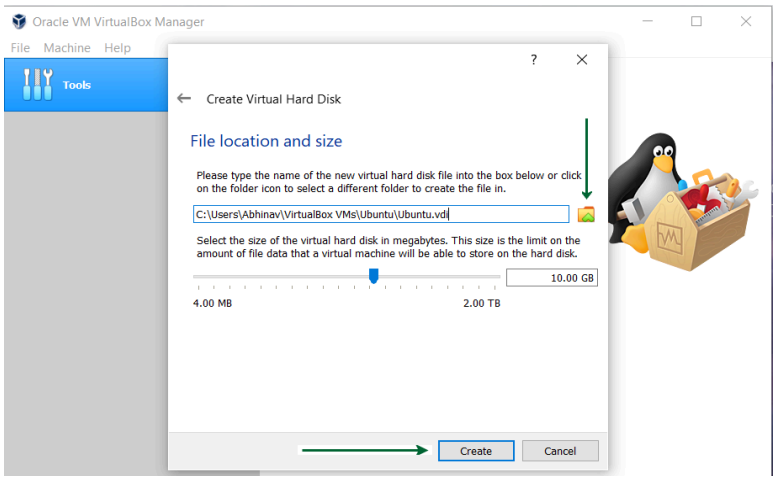
Step 6:

Either of the physical storage type can be selected. Using Dynamically allocated disk is by default recommended.



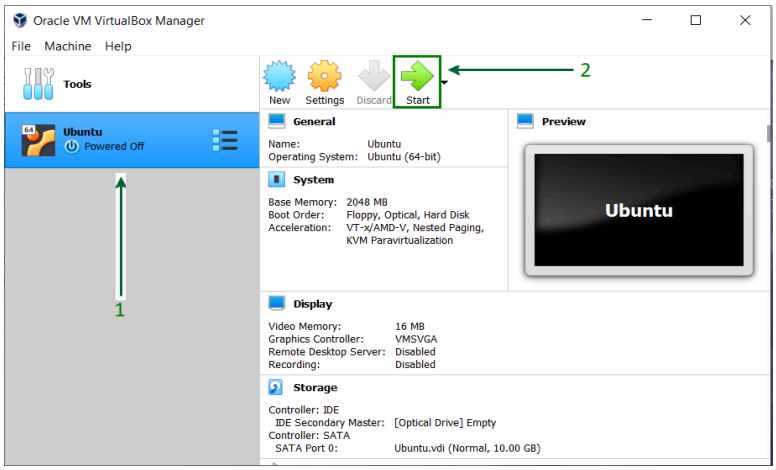
Step 7:

Select disk size and provide the destination folder to install.



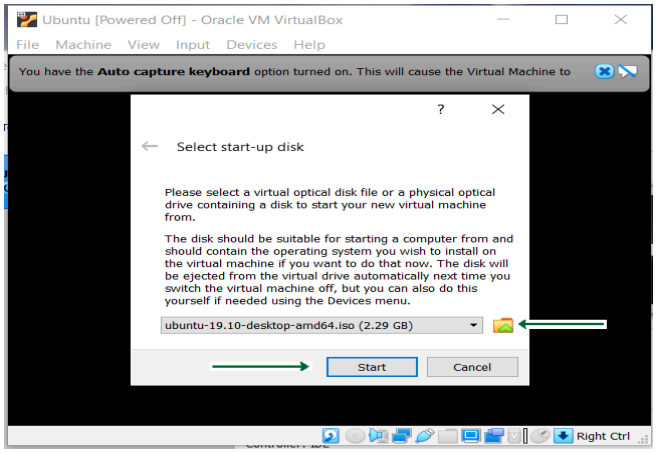
Step 8:

After the Disk creation is done, boot the Virtual Machine and begin installing Ubuntu.



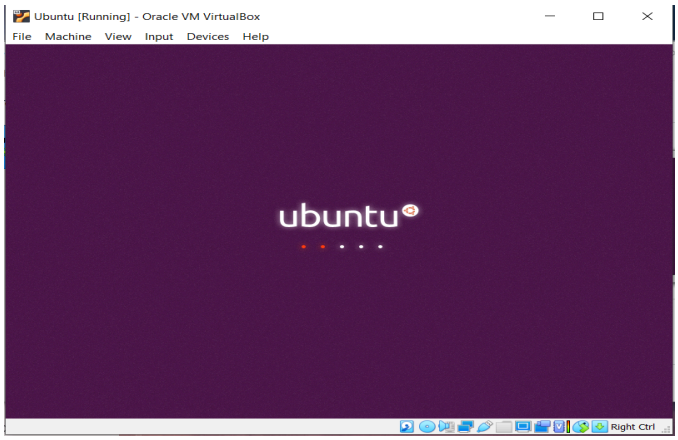
Step 9:

If the installation disk is not automatically detected. Browse the file location and select the ISO file for Ubuntu.



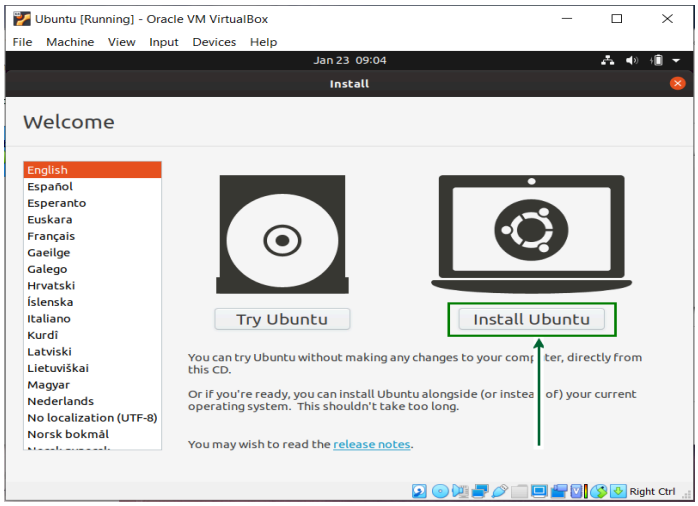
Step 10:

Proceed with the installation file and wait for further options.



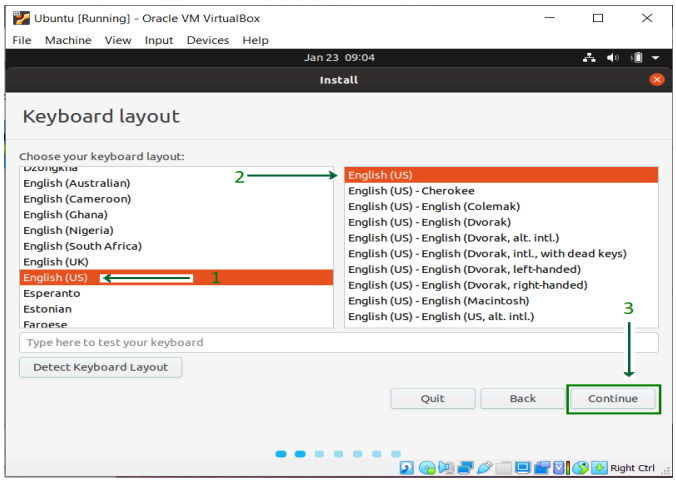
Step 11:

Click on the Install Ubuntu option, this might look different for other Ubuntu versions.



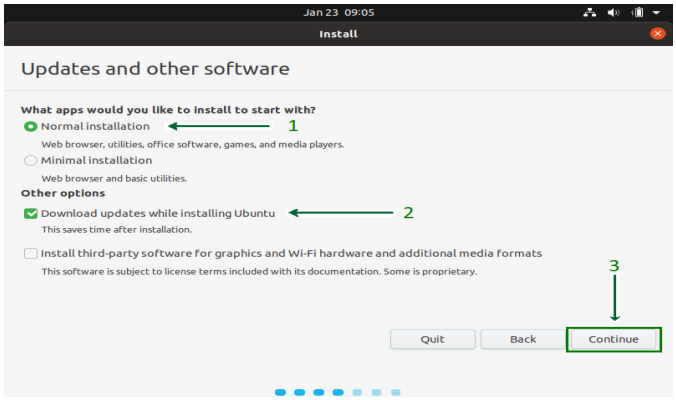
Step 12:

Select Keyboard layout, if the defaults are compatible, just click on the **continue** button and proceed.



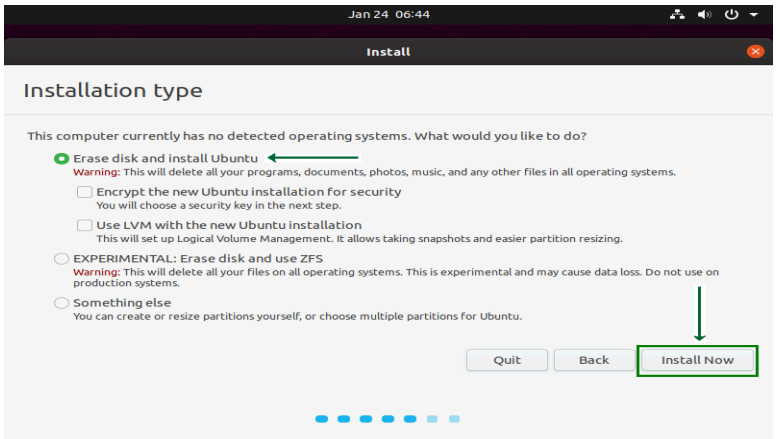
Step 13:

Select installation type. By default, it is set to Normal installation, which is recommended, but it can also be changed to Minimal installation if there is no need for all Ubuntu features.

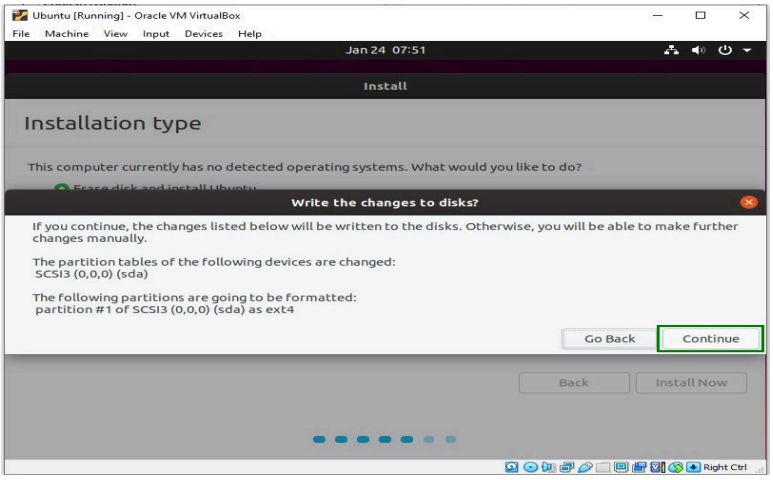


Step 14:

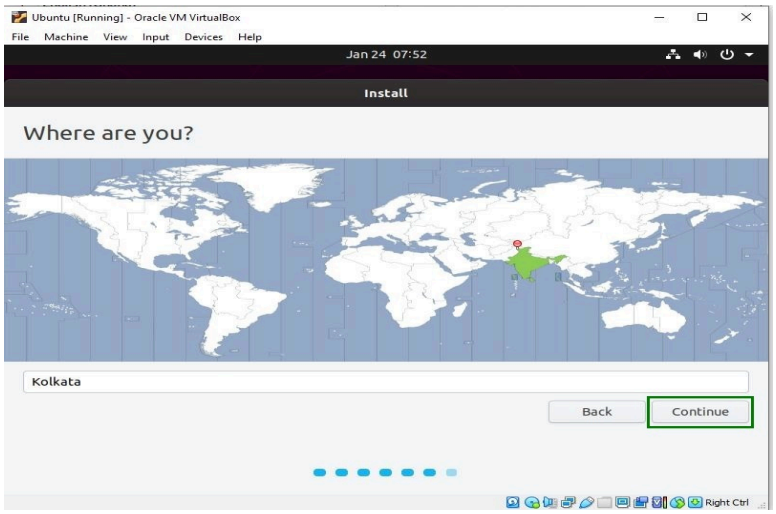
Click on the **Install Now** button and carry on with the installation. Do not get worried with the **Erase disk** option, it will only be effective inside the virtual machine, other system files outside the VirtualBox remain intact.



Step 15: Click on the **continue** button, and proceed with writing changes on the disk.

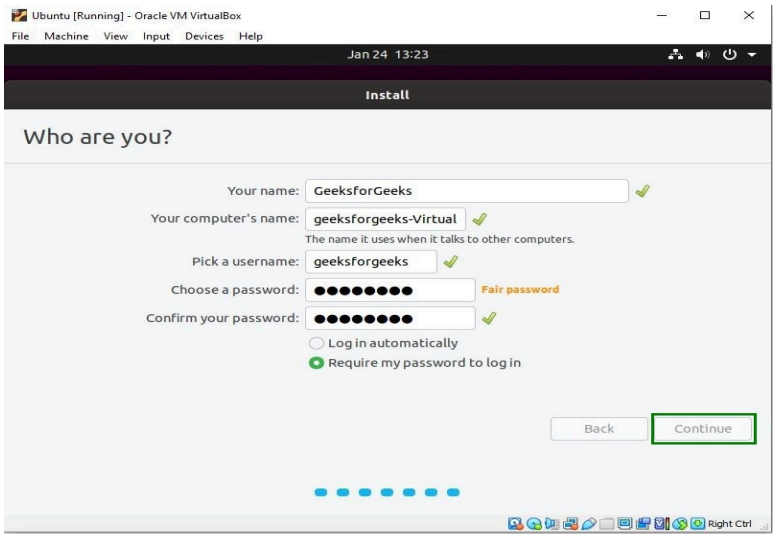


Step 16: Select your location to set the Time Zone.



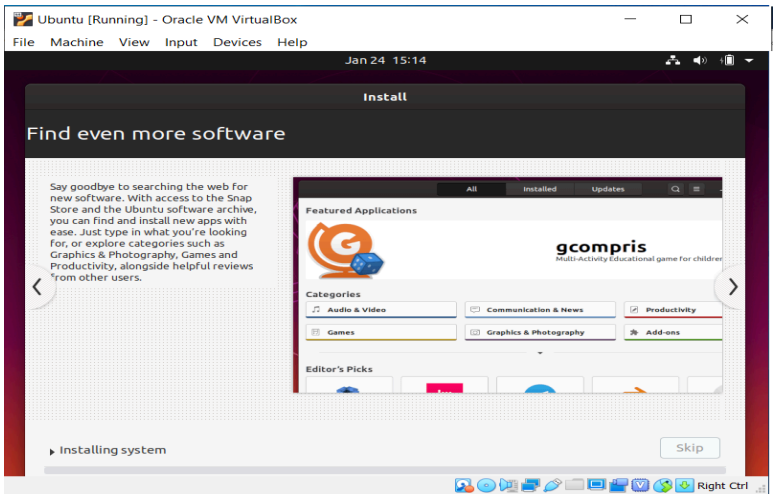
Step 17:

Choose a name for your computer and set a password to secure login info.



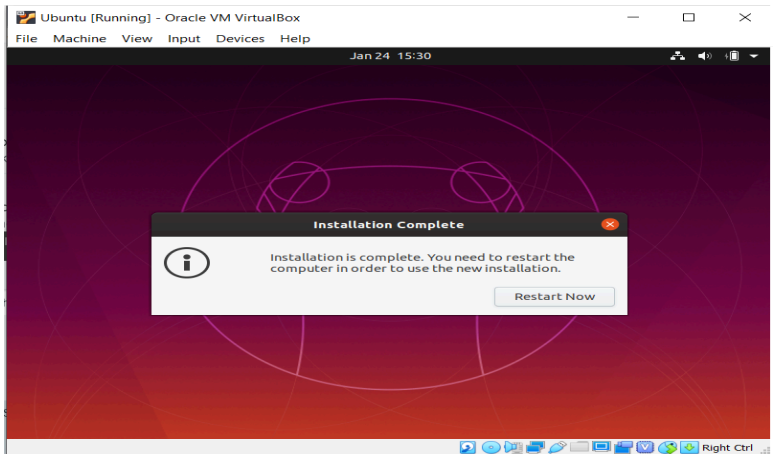
Step 18:

Wait for the installation process to complete.



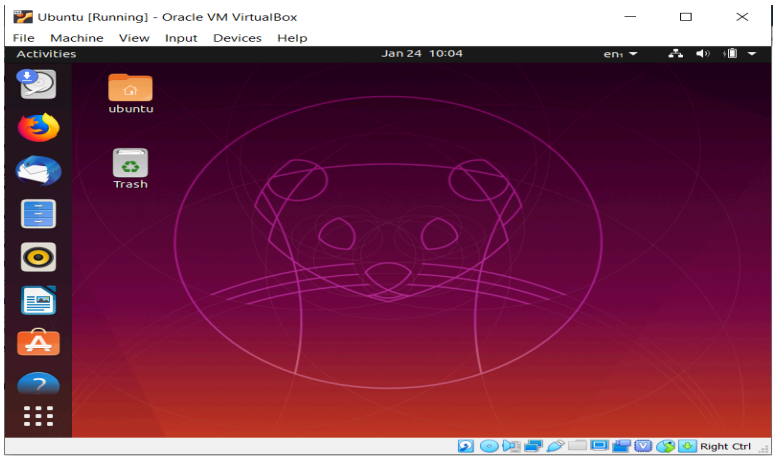
Step 19:

Once the installation process is over, reboot your Virtual Machine.



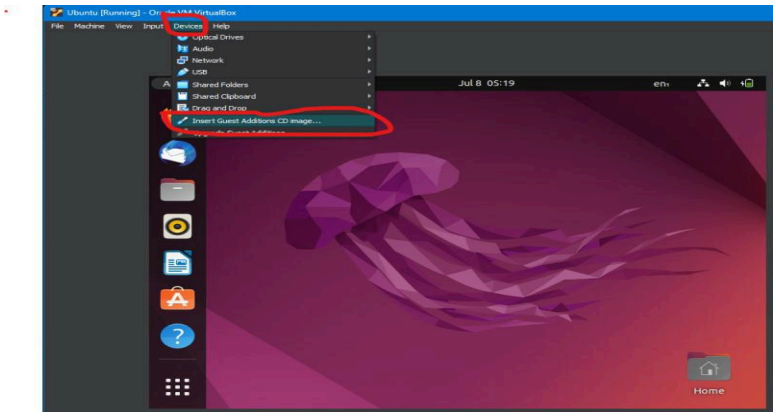
Step 20:

Voila!! You're finished with the installation process. Now you can use Ubuntu along with Windows, without creating a dual boot.

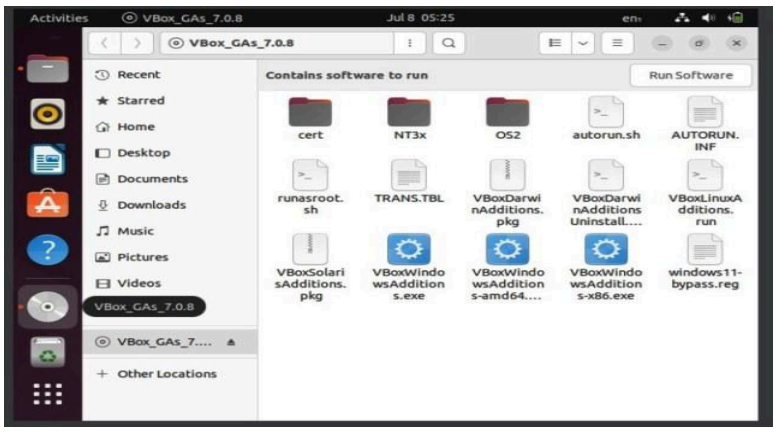


Note: After installing Ubuntu in Virtual Box, you may find that the Operating System is not suitable for full-screen mode. If you try to make it large by clicking on maximize button, the visual part remains the same and the outside becomes white/black.

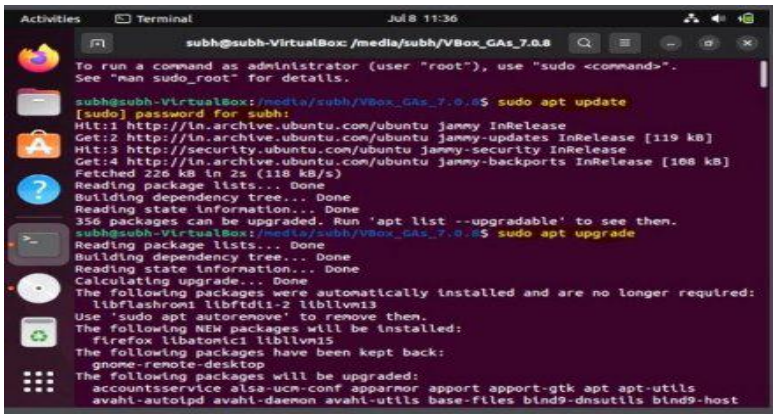
To overcome this and make Ubuntu fullscreen in Virtual Box, click on Devices then on Insert Guest Additions CD image…



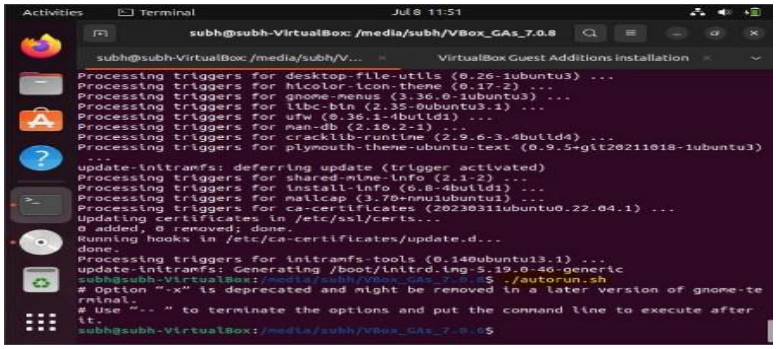
Now you will be able to see a disk icon on the taskbar, click on that and it will open the disk



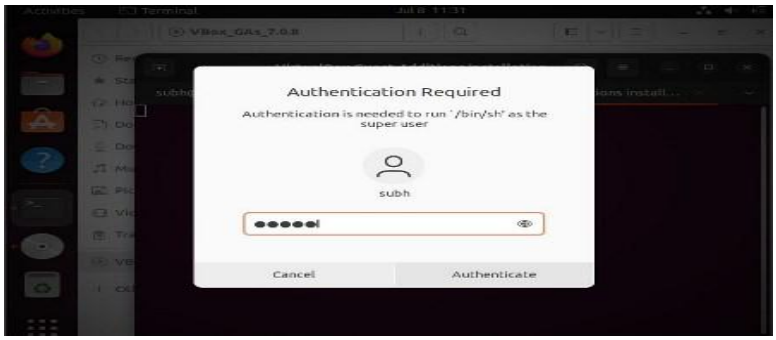
Write right-click inside the folder and select **open in the terminal**, and run the following commands
sudo apt update
sudo apt upgrade
It will ask for the user's password, enter the password and wait till completion.

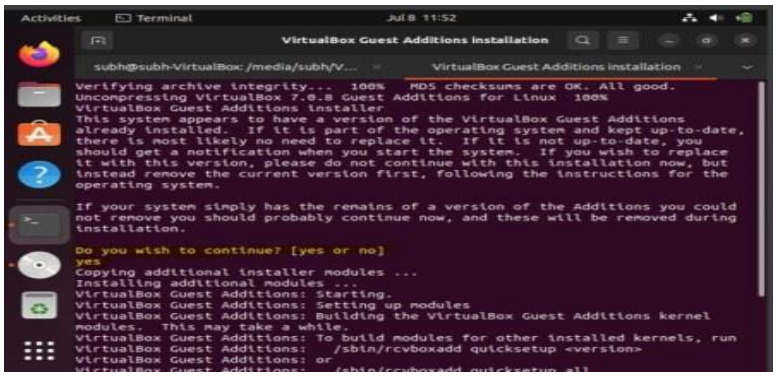


After updating, run the command
./autorun.sh



Now it will ask for the password and after checking the password a new tab will open inside the terminal. Enter **yes** there.





After installation restart Ubuntu and now you can use Ubuntu in full-screen mode.

OS Protection and Security – Firewall, Protections, Threat Detections and prevention.

OS Protection and Security

There are several ways in which an operating system can provide system protection:

User authentication: The operating system requires users to authenticate themselves before accessing the system. Usernames and passwords are commonly used for this purpose.

Access control: The operating system uses access control lists (ACLs) to determine which users or processes have permission to access specific resources or perform specific actions.

Encryption: The operating system can use encryption to protect sensitive data and prevent unauthorized access.

Firewall: A firewall is a software program that monitors and controls incoming and outgoing network traffic based on predefined security rules.

Antivirus software: Antivirus software is used to protect the system from viruses, malware, and other malicious software.

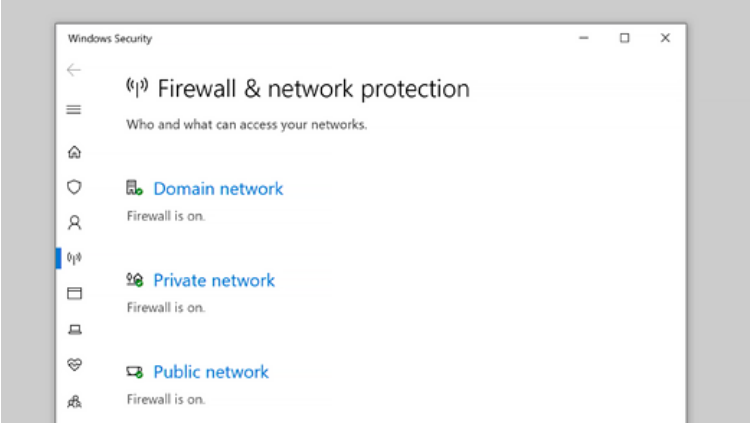
System updates and patches: The operating system must be kept up-to-date with the latest security patches and updates to prevent known vulnerabilities from being exploited.

By implementing these protection mechanisms, the operating system can prevent unauthorized access to the system, protect sensitive data, and ensure the overall security and integrity of the system.

Firewall, Protections

Firewall & network protection in Windows Security lets you view the status of Microsoft Defender Firewall and see what networks your device is connected to. You can turn Microsoft Defender Firewall on or off and access advanced Microsoft Defender Firewall options for the following network types:

- Domain (workplace) networks
- Private (discoverable) networks
- Public (non-discoverable) networks



If you want to change a setting select the network type you want to change it on.

Network settings

When you select one of the three network types you'll get the settings page for it. Here Windows Security will tell you which, if any, networks of that type you're currently connected to. Usually your computer will only be connected to one network at a time.

You'll also find a simple slider for turning the firewall on, or off, for that type of network.

Important: Turning the firewall off may increase the risk to your device or data. We recommend leaving it on unless you absolutely need to turn it off.

Under the **Incoming connections** section you'll find a single checkbox for **Blocks all incoming connections, including those in the list of allowed apps**. Checking this box tells the Microsoft Defender Firewall to ignore the allowed apps list and block everything. Turning this on increases your security, but may cause some apps to stop working.

Also on the Firewall & network protection page:

- **Allow an app through firewall** - If the firewall is blocking an app you really need, you can add an exception for that app, or open a specific port.
- **Network and Internet troubleshooter** - If you're having general network connectivity issues you can use this troubleshooter to try and automatically diagnose and fix them.
- **Firewall notification settings** - Want more notifications when your firewall blocks something?

- **Advanced settings** - If you're knowledgeable about firewall settings this will open the classic Windows Defender Firewall tool which lets you create inbound or outbound rules, connection security rules, and see monitoring logs for the firewall. Most users won't want to dig into it that deeply; adding, changing, or deleting rules incorrectly can cause your system to be more vulnerable or can cause some apps not to work.
- **Restore firewalls to default** - If someone, or something, has made changes to your Windows Firewall settings that is causing things not to work properly you're just two clicks away from resetting the settings back to the way they were when you first got the computer. If your organization has applied any policies to configure the firewall those will be reapplied.

Threat Detections and Prevention

Virus & threat protection in Windows Security helps you scan for threats on your device. You can also run different types of scans, see the results of your previous virus and threat scans, and get the latest protection offered by Microsoft Defender Antivirus.

Under **Current threats** you can:

- See any threats currently found on your device.
- See the last time a scan was run on your device, how long it took, and how many files were scanned.
- Start a new quick scan or go into **scan options** to run a more extensive or custom scan.
- See threats that have been quarantined before they can affect you and anything identified as a threat that you have allowed to run on your device.

Run the scans you want

Even though Windows Security is turned on and scans your device automatically, you can perform an additional scan whenever you want.

- **Quick scan**. Concerned that you may have done something to introduce a suspicious file or virus to your device? Select **Quick scan** (called **Scan now** in previous versions of Windows 10) to immediately check your device for any recent threats. This option is useful when you don't want to spend the time running a full scan on all your files and folders. If Windows Security recommends that you run one of the other types of scans, you'll be notified when the Quick scan is done.
- **Scan options.** Select this link to choose from one of the following advanced scan options:

- **Full scan**. Scans every file and program on your device.

- **Custom scan**. Scans only files and folders that you select.

- **Microsoft Defender Offline scan**. Uses the latest definitions to scan your device for the latest threats. This happens after a restart, without loading Windows, so any persistent malware has a more difficult time hiding or defending itself. Run it when you are concerned that your device has

been exposed to malware or a virus, or if you want to scan your device without being connected to the Internet. This will restart your device, so be sure to save files you may have open.

Manage your Virus & threat protection settings

Use **Virus & threat protection settings** when you want to customize your level of protection, send sample files to Microsoft, exclude trusted files and folders from repeated scanning, or temporarily turn off your protection.

Manage your real-time protection

Want to stop running real-time protection for a short while? You can use the Real-time protection setting to turn it off temporarily; however, real-time protection will turn back on automatically after a short while to resume protecting your device. While real-time protection is off, files you open or download won't be scanned for threats.

Get access to cloud-delivered protection

Provide your device with access to the latest threat definitions and threat behavior detection in the cloud. This setting allows Microsoft Defender to get constantly updated improvements from Microsoft while you're connected to the internet. This will result in more accurately identifying, stopping, and fixing, threats. This setting is turned on by default.

Tamper Protection defends your security settings

Microsoft Defender Antivirus settings can occasionally get changed by malicious, or careless, apps or processes; or sometimes by unaware people. With Tamper Protection turned on, important settings like real-time protection can't be easily or accidentally turned off. This setting is turned on, by default, for consumers.

Protect files from unauthorized access

Use the Controlled folder access setting to manage which folders untrusted apps can make changes to. You can also add additional apps to the trusted list so they can make changes in those folders. This is a powerful tool to make your files safer from ransom ware.

When you turn on Controlled folder access, a lot of the folders you use most often will be protected by default. This means that content in any of these folders cannot be accessed or changed by any unknown or untrusted apps. If you add additional folders, they become protected as well.

Exclude items from virus scans

There may be times when you'll want to exclude specific files, folders, file types, or processes from being scanned, such as if these are trusted items and you are certain you don't need to take time to scan them. In those rare instances you can add an exclusion for them.

Curate your notifications

Windows Security will send notifications about the health and safety of your device. You can turn these notifications on, or off, on the notifications page. In **Virus & threat protection**, under **Virus & threat protection settings**, select **Manage settings**, scroll down to **Notifications** and select **Change notification settings**.

Protect your device with the latest updates

Security intelligence (sometimes referred to as "definitions") is files that contain information about the latest threats that could infect your device. Windows Security uses security intelligence every time a scan is run.

Microsoft automatically downloads the latest intelligence to your device as part of Windows Update, but you can also manually check for it. On the Virus & threat protection page, under **Virus & threat protection updates**, select **Check for updates** to scan for the latest security intelligence.