Department of

Computer Science & Engineering

LAB MANUAL

OPERATING SYSTEM LAB

B.Tech– IV Semester

KCT College OF ENGG AND TECH.

VILLAGE FATEHGARH

DISTT.SANGRUR

INDEX

EXPERIMENT 1: OPERATING SYSTEM INSTALLATION

The following document guides you step-by-step through the process of installing the operating systems so they are properly configured for boot camp.

The document is divided into 3 parts:

1. Windows XP Installation

Option 1:
If you are going to use a bootable Knoppix CD for the Linux portion, you only need to install Windows XP and follow section 1. For Windows XP you want to perform a full default install of all components. It is critical that you use Windows XP Professional, Windows XP Home Edition will NOT work. You also want to make sure that Service Pack 2 is installed. For Knoppix, please download and boot off of the Knoppix CD prior to coming to class to validate that Knoppix supports your hardware.

Windows XP Installation

It is important to understand that this guide was specifically designed for a lab environment. There are a lot of operating system vulnerabilities that are intentionally left unpatched in these installation steps. This is intentionally done to give you the best results when completing the labs and tutorials in this book. If you are interested, a great reference for building a Windows XP Professional box that is secure enough for a production environment is *Windows XP Security: Step By Step* by SANS

To create a properly configured laptop for the Security Essentials Boot Camp, follow the detailed steps in this document—from the initial setup screen to the final login. This guide was designed for use on a system that doesn't already have a Windows platform installed on it. If your machine does not have a blank hard drive, some of the screens you see at the beginning of the installation may be different from what you see in this chapter. If different screens appear, it is important that you always choose the option to *replace*, or *overwrite*. Do not choose to upgrade. The Windows install should also be placed in the default c:\windows directory.

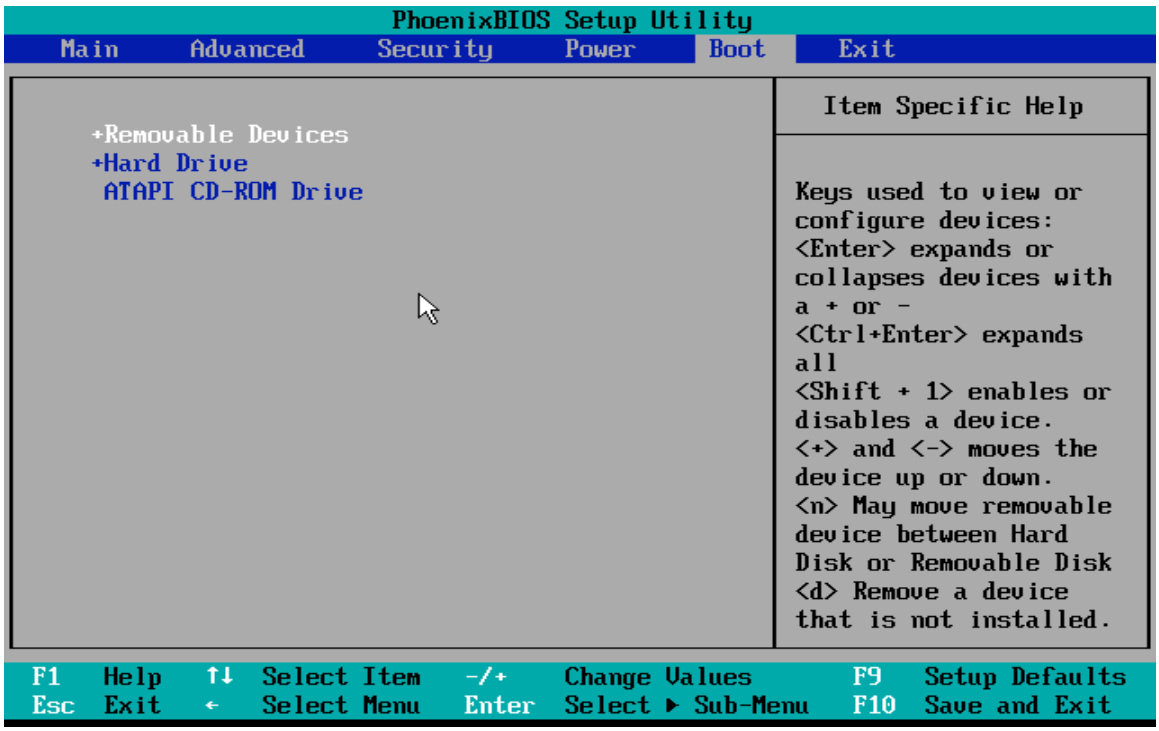
Creating Boot Disks

If your system does not support the capability to boot off of a CD-ROM, you can use the Windows XP boot disk to boot. If you do not have a set of the four disks, you need to use a machine that already has Windows XP Professional installed on it. The following steps show you how to create the four boot disks:

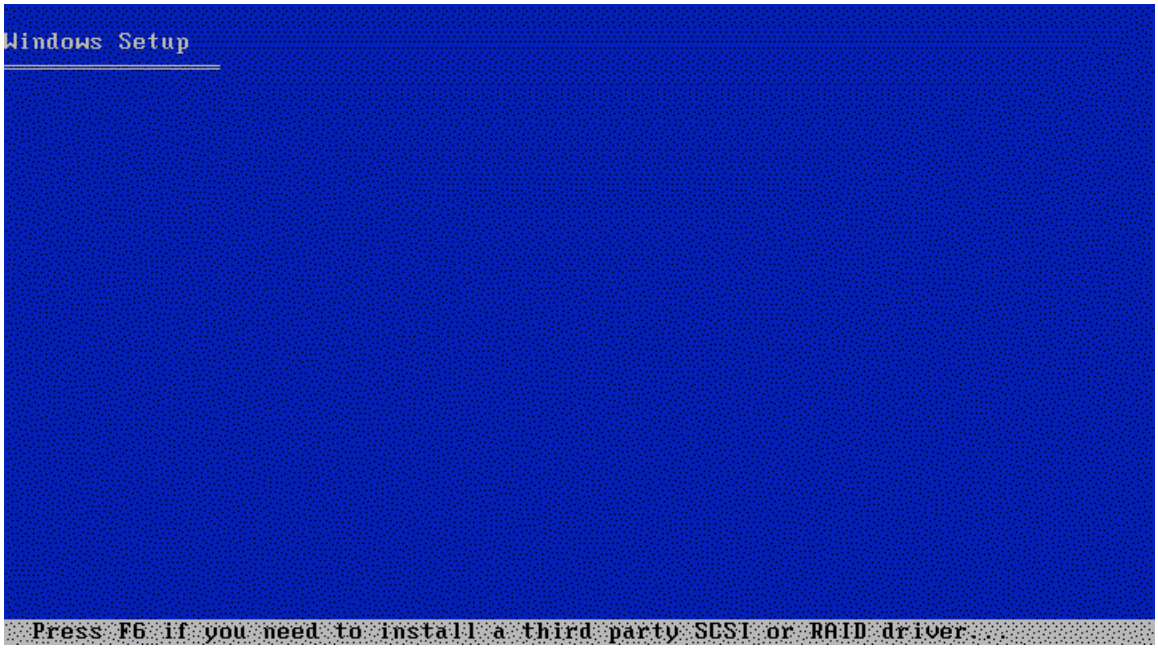
1. Label four blank, formatted, 3.5-inch, 1.44-MB floppy disks as: **Setup Disk One**, **Setup Disk Two**, **Setup Disk Three**, and **Setup Disk Four**.
2. Insert **Setup Disk One** into the floppy disk drive of a Windows or DOS system.
3. Insert the Windows XP CD-ROM into the CD-ROM drive.
4. Click **Start**, and then click **Run**.
5. In the **Open** box, type **D:\bootdisk\makeboot a:** (where **D:** is the drive letter assigned to your CD-ROM drive), and then click **OK**.
6. Follow the screen prompts.
7. After you have completed the screen prompt requests, insert **Setup Disk One** into the floppy disk drive of the lab PC and power the PC on.

Booting from the CD-ROM

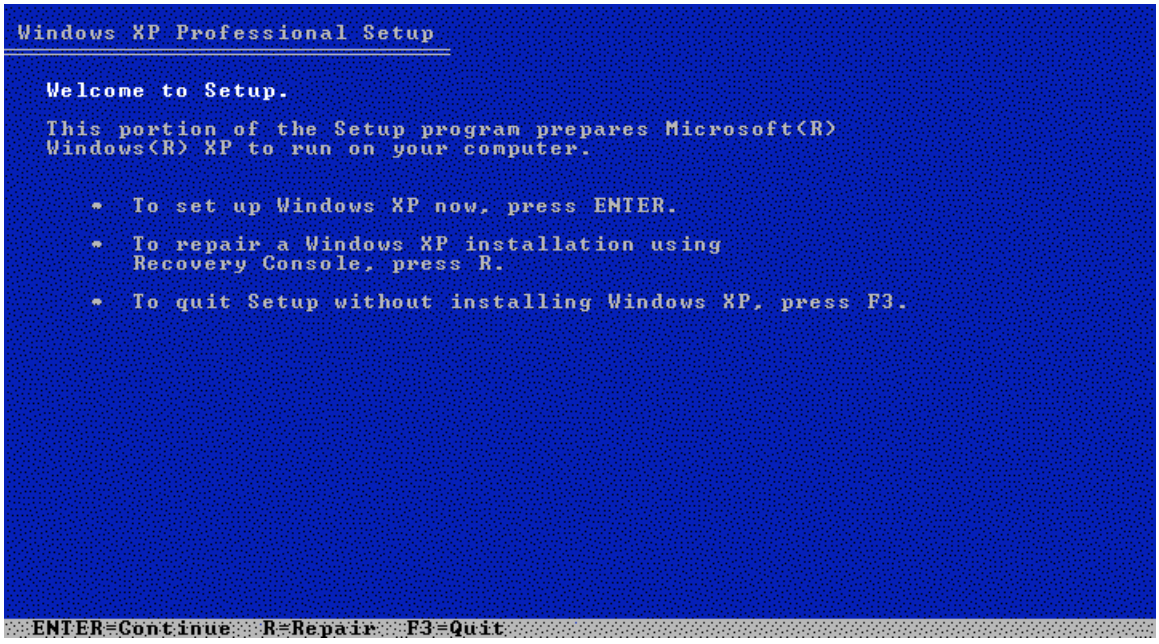
2. If the previous screen does not appear, reboot your machine and open up the BIOS. You need to make the system boot to the CD-ROM first. The following screen is one of several different BIOSes you could have on your system. You need to navigate to a screen that allows you to change the Boot Order. This is where you tell it to boot off of the CD-ROM.



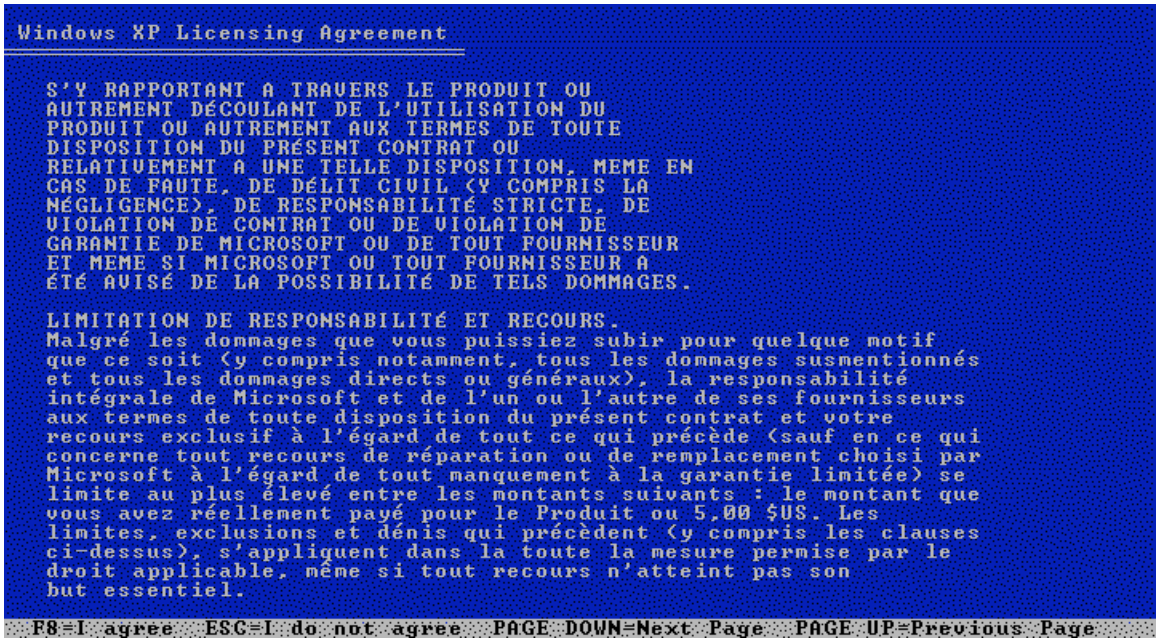
3. Now your system should boot off of the CD-ROM. After a period of time (typically 30-45 seconds), the following screen appears. Because we are doing an initial install, you only need to press **Enter** to continue.

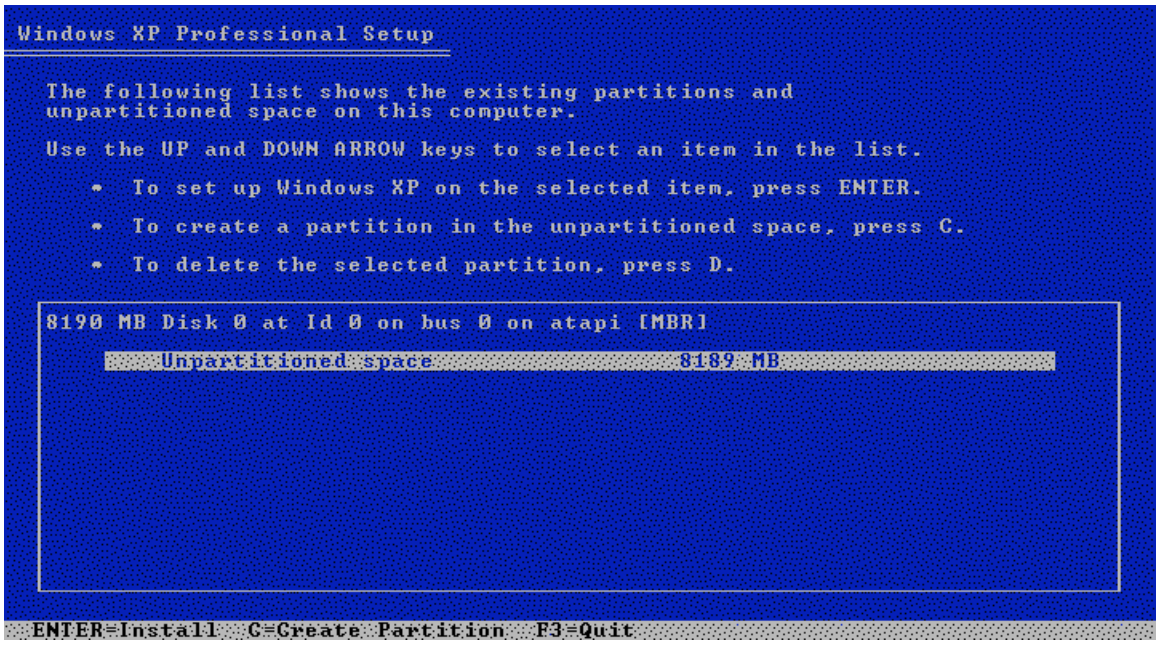


4. Hit **Enter** at the next screen to continue installation.



5. The Microsoft Windows XP Licensing Agreement appears next, as shown in the following screen. It is important that you read and understand this agreement before continuing with the installation. After you have read and agreed to the contents of the license, press **F8** to continue.



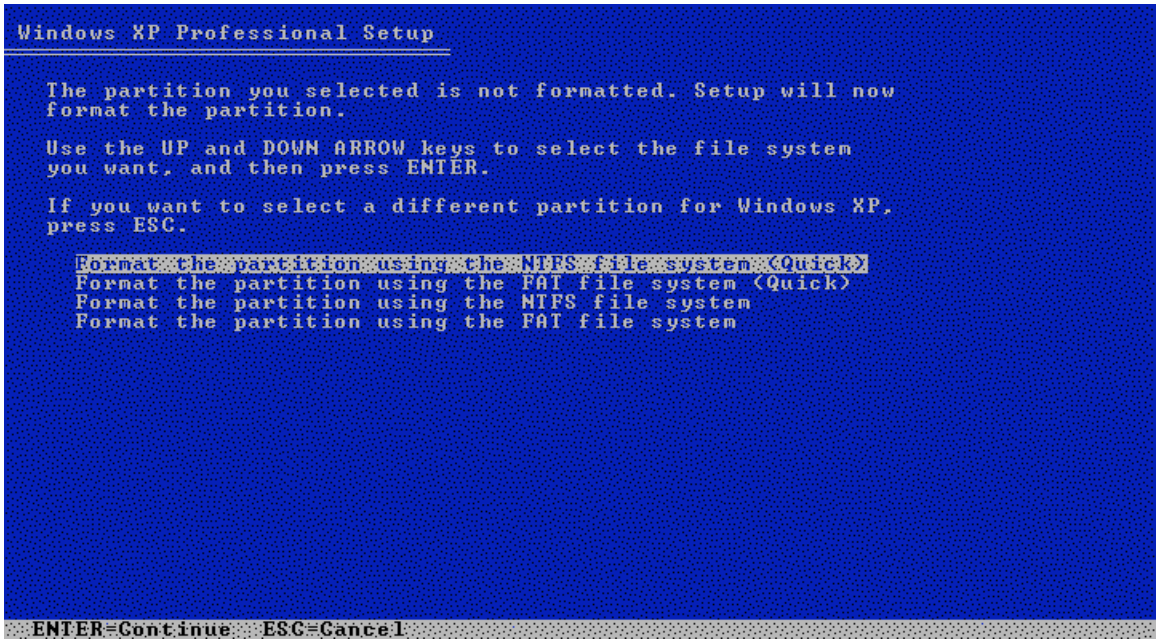


Now create your new partition to be at least 2 Gb. In the provided space type **2047** and press **Enter**

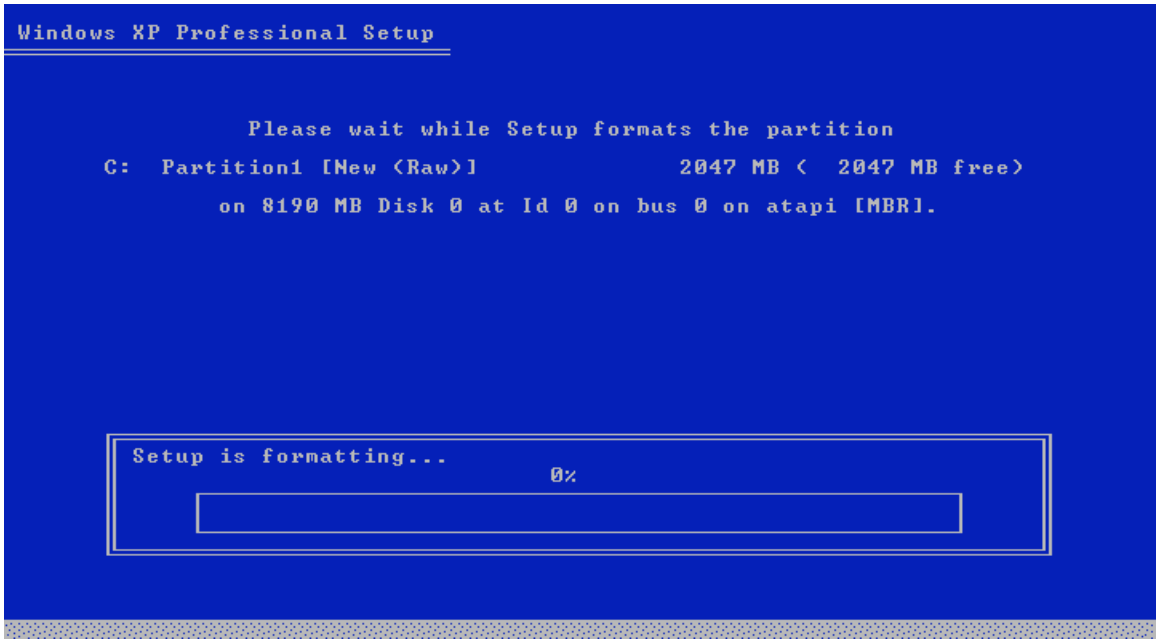
Formatting Drive Partitions

The next step is to format your partition. For security reasons, you should format your partitions using NTFS. NTFS is a Windows partition type that allows you to assign permissions at the folder level. This level of granularity is not the same for FAT partitions. NTFS also allows for lager partition sizes compared to the 2Gb limit that comes with FAT16. The steps for formatting your partition follow:

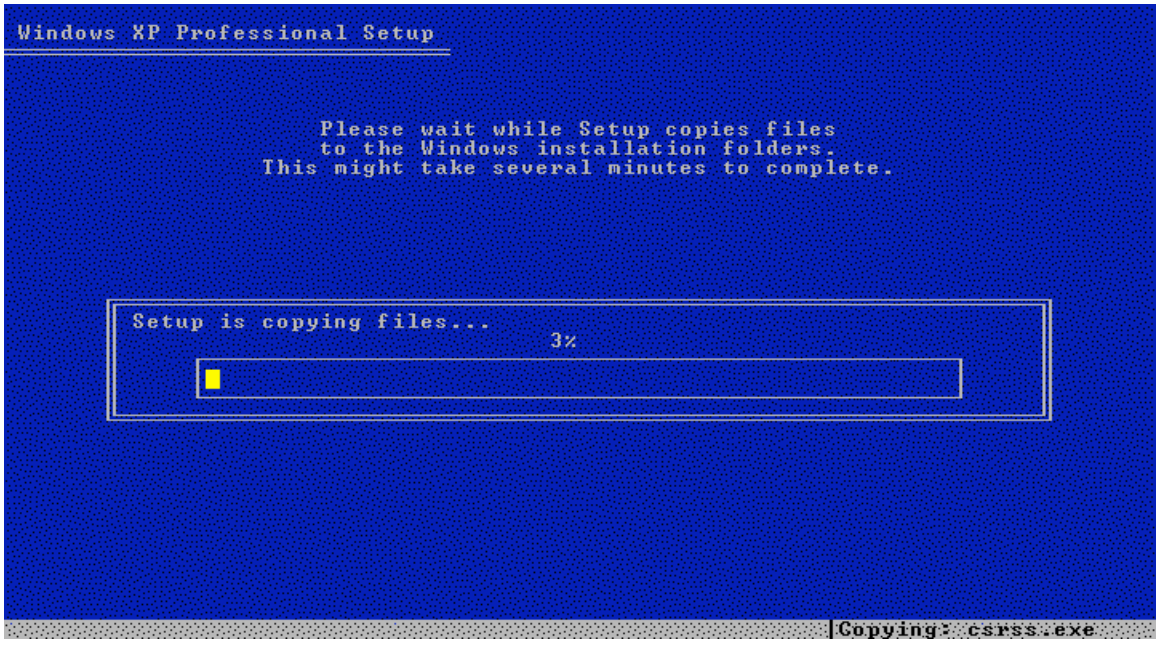
1. Highlight the NTFS <Quick> partition option as shown in the following screen, and press **Enter**.



2. After you press **Enter**, the system formats the partition, as shown in the following screens. Depending on the size of the partition, this step can take from 5 minutes to an hour. This is a great time to refill your caffeine-laced beverage of choice. (You may need it because you have a long way to go.)



Since this will take a while you should just wait while this process continues.



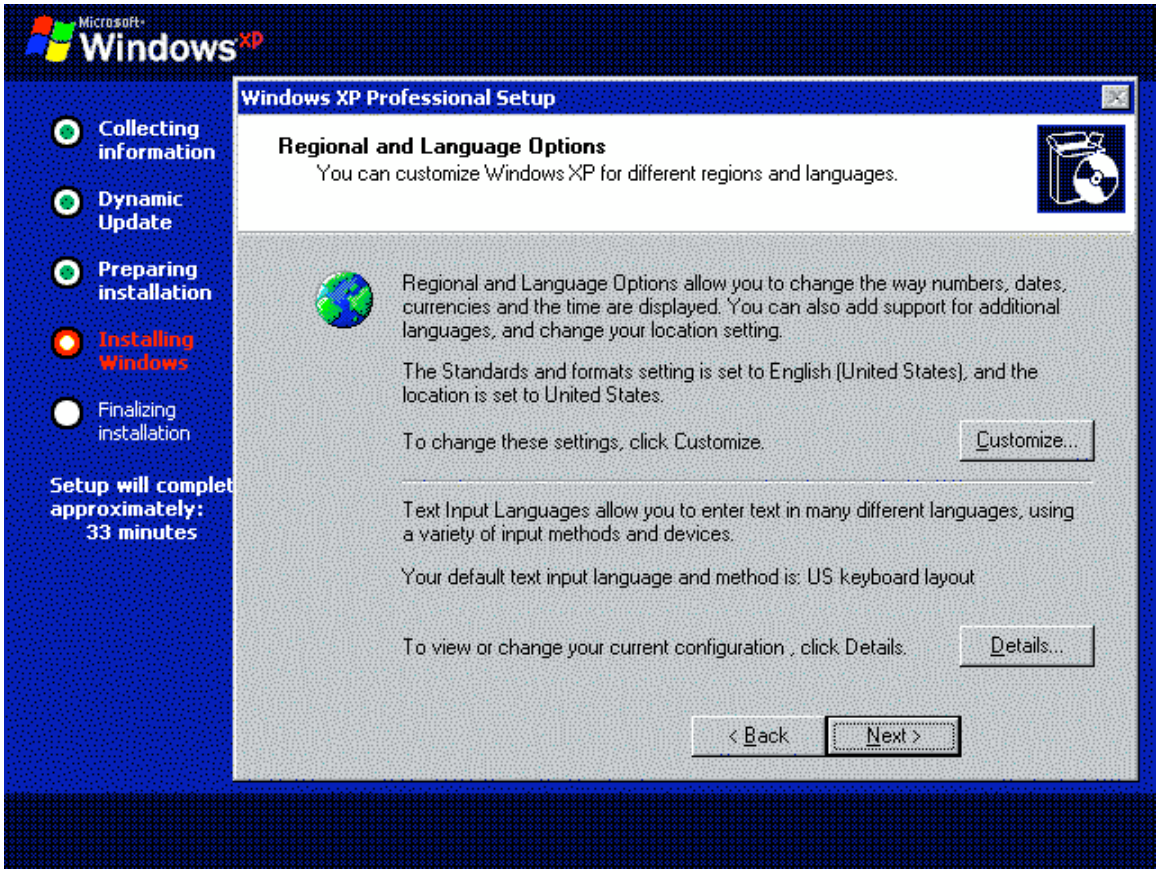
When you return to your machine, you may see one of the following screens. Don't be alarmed. The system has completed the formatting process and has automatically rebooted. After this occurs, you have to answer the remaining install questions.


Microsoft
Windows xp
Professional
Copyright © 1985-2001
Microsoft Corporation
Microsoft
Windows xp
Collecting information
Dynamic Update
Preparing installation
Installing Windows
Finalizing installation
Setup will complete in approximately:
39 minutes
An exciting new look
Windows® XP Professional sports a brand-new visual design that combines a sleek look, clean lines, and appealing colors with a task-oriented design and exceptionally streamlined navigation.
The redesigned Start menu makes it easier to find important information and to access the programs you use most frequently.
By automatically clearing up the notification area of the taskbar and grouping related taskbar items, Windows XP makes it easier to switch between programs and to open, view, or close multiple items at the same time.

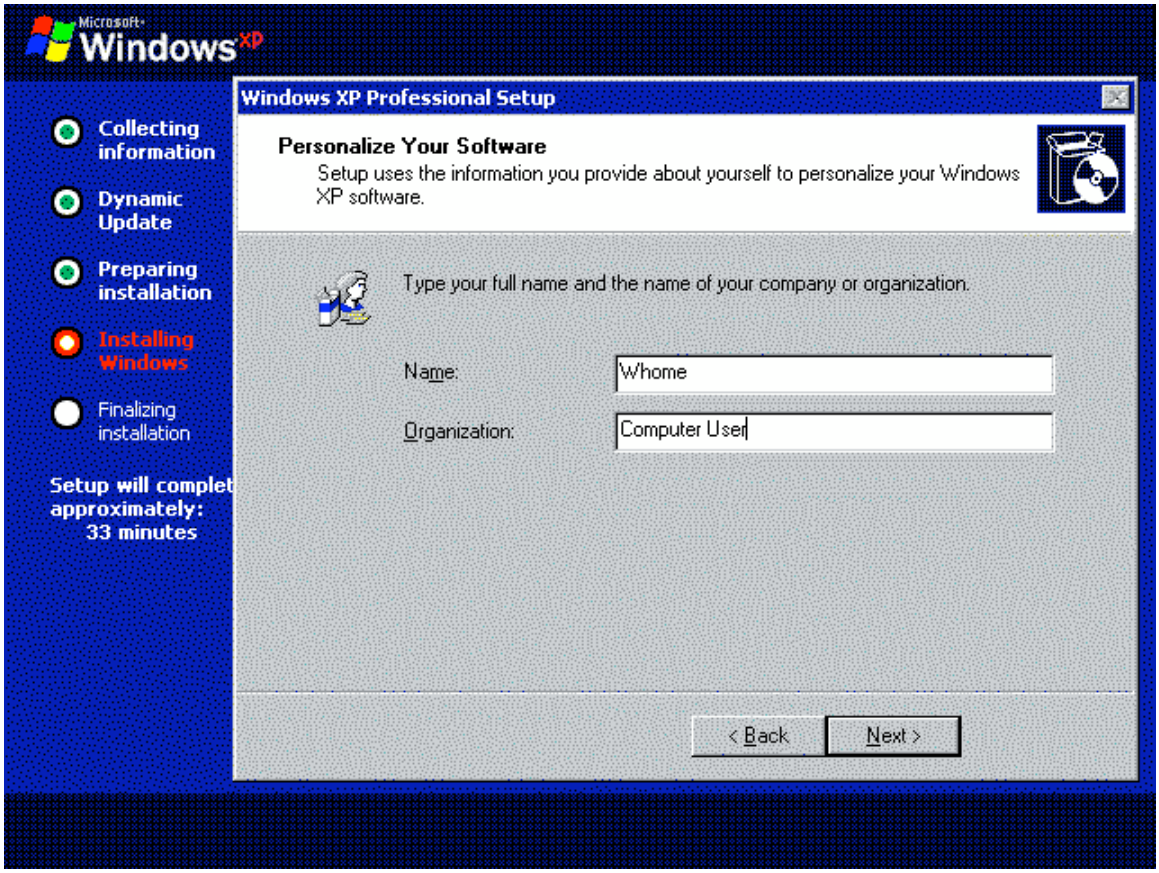
Customizing Your System

Now Windows presents a series of questions, which, when answered, customize your system. The following steps walk you through the process of customizing your system:

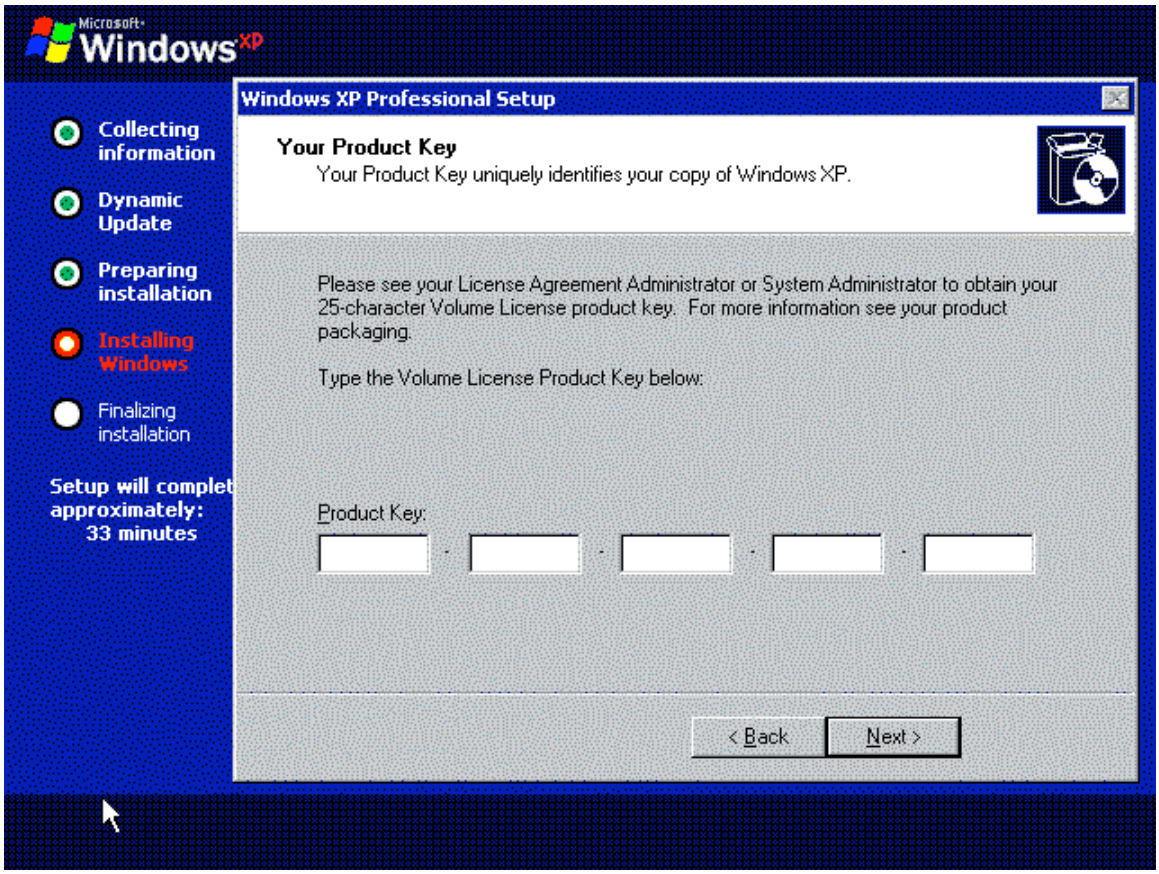
1. Typically, you only need to make changes during the next step (see the following screen) if you are located outside of the United States or if you use a non-standard keyboard. If you are in the United States and you are using a standard QWERTY keyboard, press the **Next** button. If you are located outside of the United States, you should change your locale settings.



2. Enter your name and the organization you work for in the **Name** and **Organization** fields. For the purposes of this course, have some fun making up fictional names. Click the **Next** button when you are done.

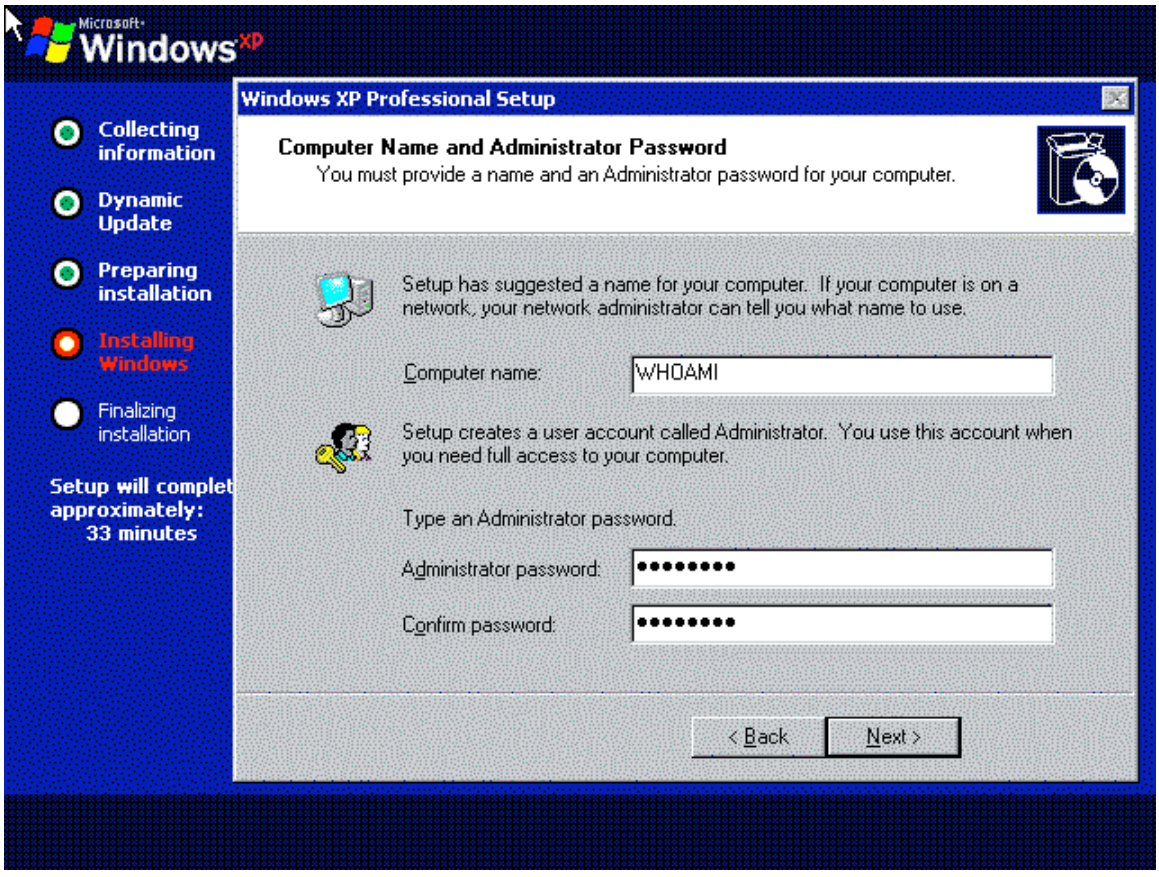


3. In the next screen, enter the Product Key number that came with your software (find it on your CD). If you make a mistake when you enter the key, you receive an **Invalid Key** message and the system gives you another opportunity to enter it. Once you enter in the valid key, press the **Enter** key.



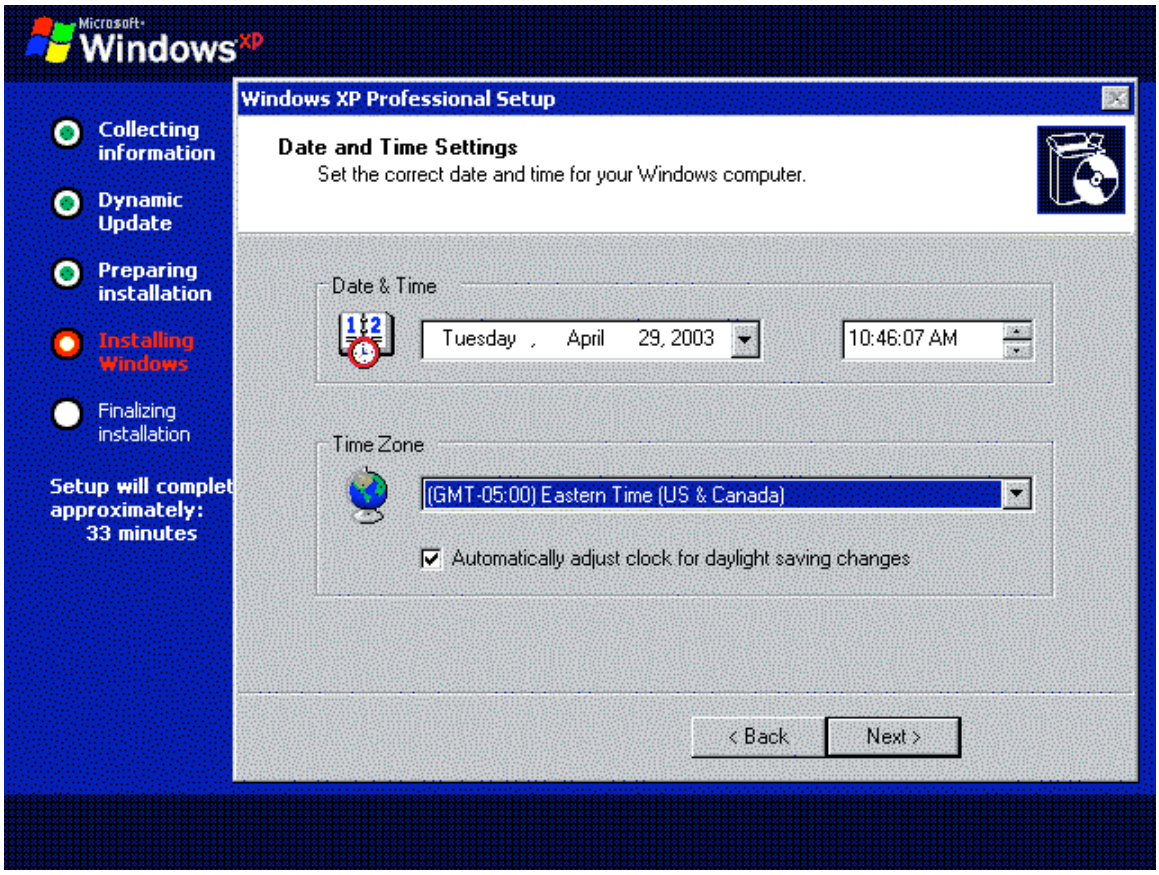
4. Now enter a name in the **Computer name** field to name your computer. If you are part of a corporation's domain, you need to follow your corporation's guidelines for naming systems. For our purposes, name your machine whatever you desire. Then, type in a password in the **Administrator password** field. You also need to confirm the password, as shown in the following screen. Then, click the **Next** button.

   Warning: A common mistake many administrators make at this stage is to leave the **Administrator password** field blank. It is highly advisable that you enter a password that matches your company's password policy for local passwords. You don't want to forget to change the password after you have completed the installation. Also, make sure you remember this password. You will need it to login.

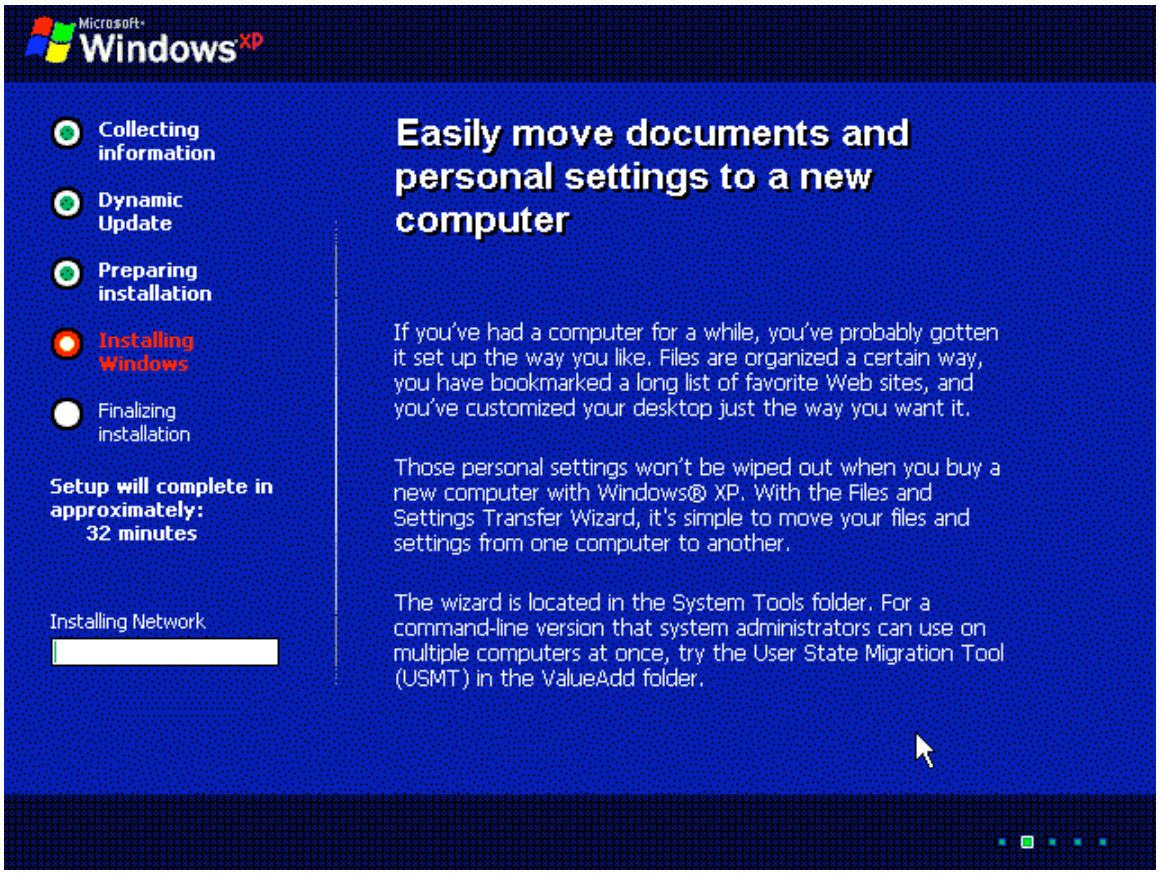


Note: Depending on your configuration, you might receive the Modem Dialing Information Screen. Just cancel out of this or click Next to get to the next screen.

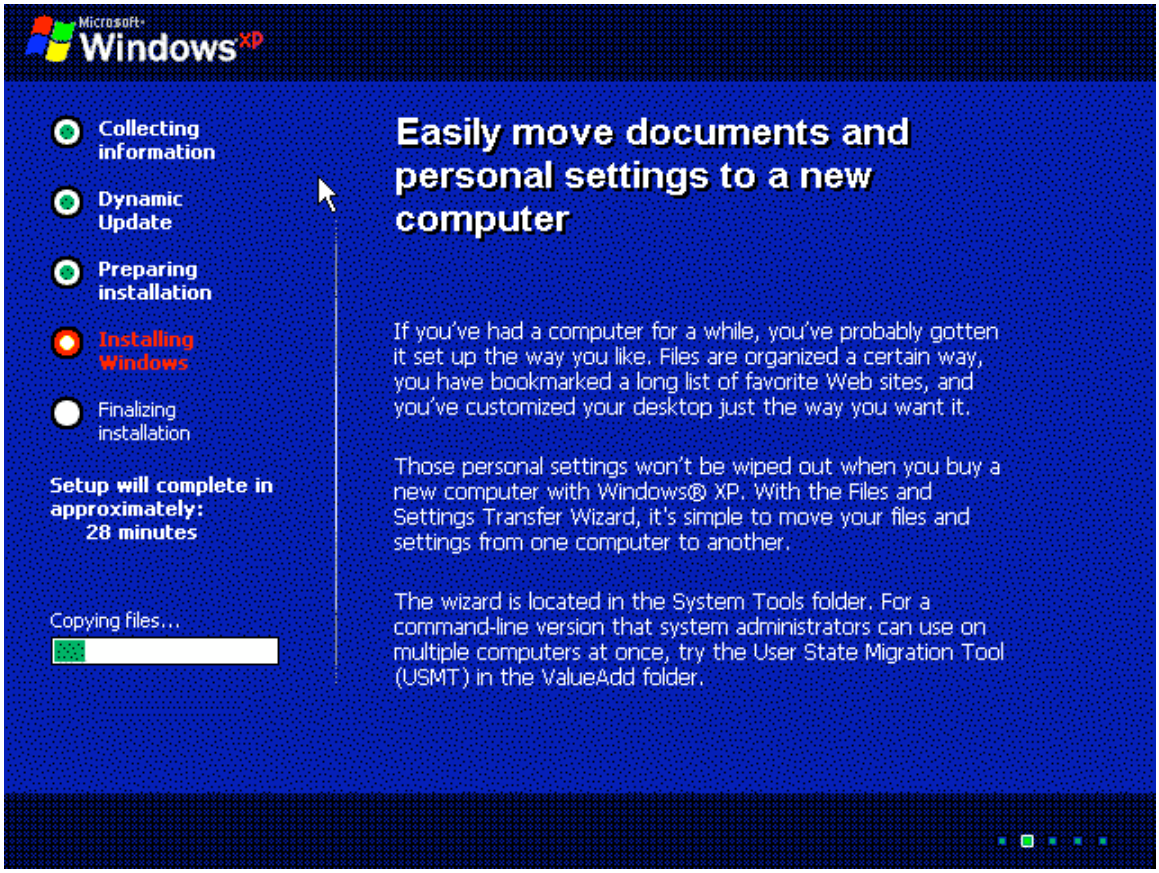
5. In the screen that appears, enter the current time, and then fill in the **Date** field and **Time Zone** field. Click **Next**.



6. After you make the previous configurations, the system installs your networking components, as shown in the following screen.

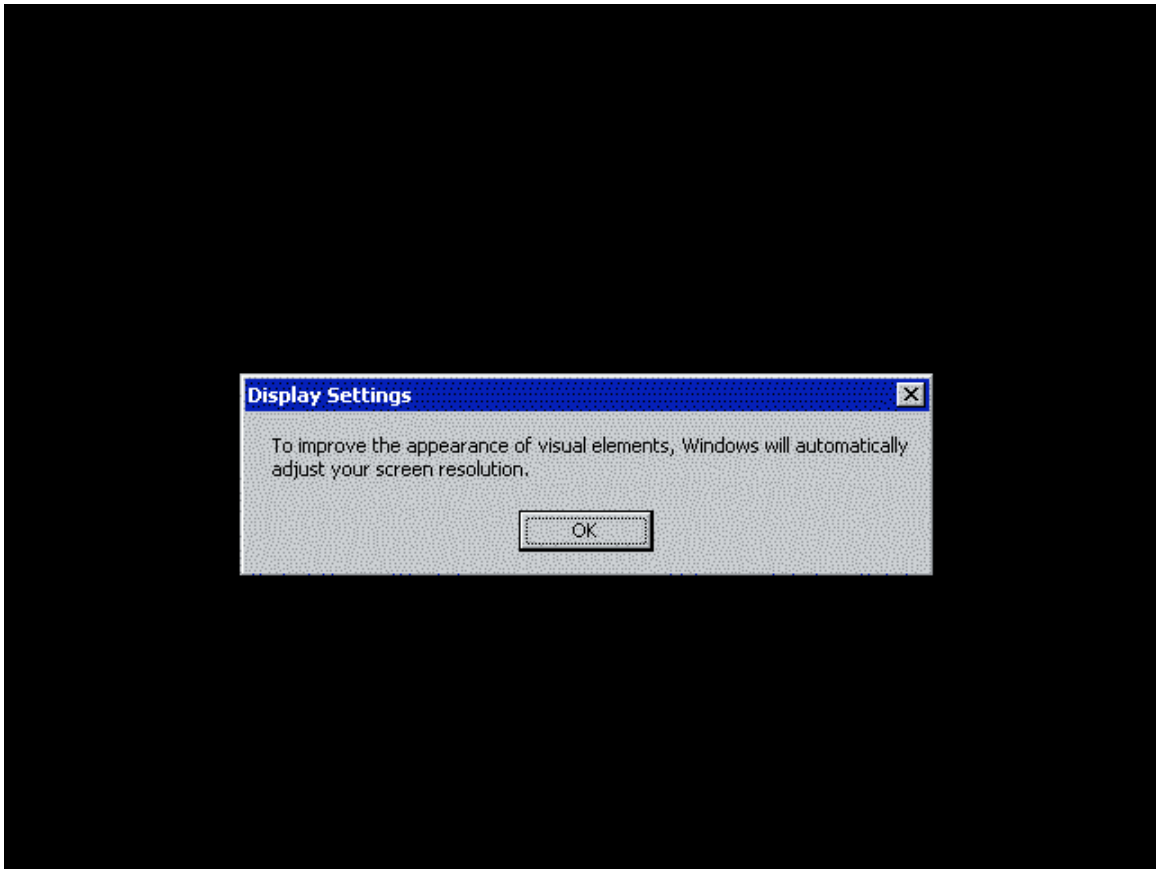


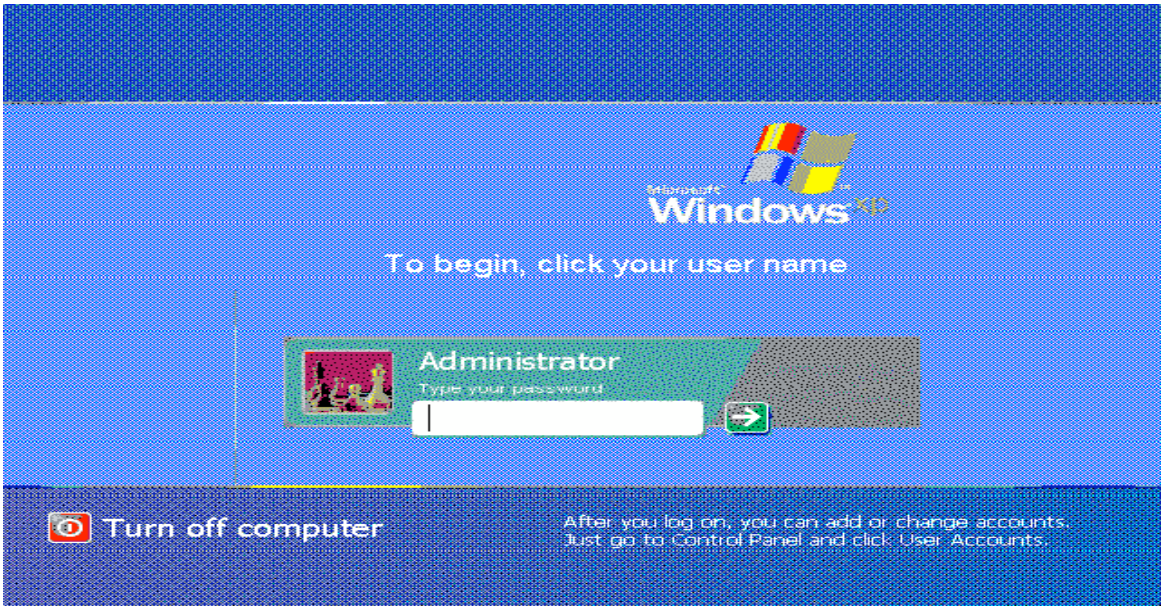
7. Windows completes the networking portion of the installation and moves on to its final tasks. This step takes a long time, so take the opportunity to grab another caffeine-laced beverage.



8. If you get the following screen, shout for joy. Congratulations, you have successfully installed Windows XP. Click **Finish**, and then remove the Windows CD-ROM before the system reboots so that you don't accidentally start the install process again. If you accidentally leave the CD-ROM in, and the install process starts again, simply remove the CD-ROM and hard-boot the machine (restart it).

9. After the next screen comes up, click the **OK** button.




Windows XP
To begin, click your user name
Administrator
Type your password
Turn off computer
After you log on, you can add or change accounts.
Just go to Control Panel and click User Accounts.

GJHHGDEXPERIMENT 2:PROGRAMS USING UNIX SYSTEM.

FEATURE OF UNIX

Multitasking is the capability of the operating system to perform various tasks.ie., A single user can perform various tasks.

Mutiuser capability

This allows several users to use the same computer to perform their tasks.

Security

Every user have a login name and a password. So, accessing another user's data is impossible without permission

Portability

UNIX is portable because it is written in a high level language ©. So UXIX can be run on different computers.

Communication:

UNIX supports the following communications.

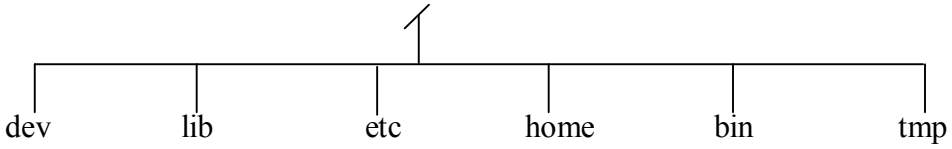
i) Between the different terminals connected to the UNIX server.

ii) Between the users of one computer to the users of another

Programming facility:

UNIX is highly programmable, the UNIX shell programming language has all the necessary ingredients like conditional and control structures (Loops) and variables.

Structure of a UNIX file system



/lib library files
/dev Contains file that link hardware devices
/tmp temporary storage of files
/home user home directories

Getting started with UNIX

Switching the system ON will provide the user with **login** prompt. Here we enter the login name. Then it prompts for the password. The password is not echoed on the screen to protect the privacy of the war. If both are correct, then we will get the **S** prompt.

UNIX Commands

Basic Commands

I File and Directory Related commands

1) pwd

This command prints the current working directory

2) ls

This command displays the list of files in the current working directory.

$ls –l Lists the files in the long format
$ls –t Lists in the order of last modification time
$ls –d Lists directory instead of contents
$ls -u Lists in order of last access time

3) cd

This command is used to change from the working directory to any other directory specified.

$cd directoryname

4) cd ..

This command is used to come out of the current working directory.

$cd ..

5) mkdir

This command helps us to make a directory.

$mkdir directoryname

6) rmdir

This command is used to remove a directory specified in the command line. It requires the specified directory to be empty before removing it.

$rmdir directoryname

7) cat

This command helps us to list the contents of a file we specify.

$cat [option][file]

cat > filename – This is used to create a new file.
cat >>filename – This is used to append the contents of the file

8) cp

This command helps us to create duplicate copies of ordinary files.

$cp source destination

9) mv

This command is used to move files.

$mv source destination

10) ln

This command is to establish an additional filename for the same ordinary file.

$ln firstname secondname

11) rm

This command is used to delete one or more files from the directory.

$rm [option] filename
$rm –i Asks the user if he wants to delete the file mentioned.

$rm –r Recursively delete the entire contents of the directory as well as the directory itself.

II) Process and status information commands

1) who

This command gives the details of who all have logged in to the UNIX system currently.

$ who

2) who am i

This command tells us as to when we had logged in and the system's name for the connection being used. $who am i

3) date

This command displays the current date in different formats.

+%D	mm/dd/yy	+%w	Day of the week
+%H	Hr-00 to 23	+%a	Abbr.Weekday
+%M	Min-00 to 59	+%h	Abbr.Month
+%S	Sec-00 to 59	+%r	Time in AM/PM
+%T	HH:MM:SS	+%y	Last two digits of the year

4) echo

This command will display the text typed from the keyboard.

$echo

Eg: $echo Have a nice day

O/p Have a nice day

II Text related commands

1. head

This command displays the initial part of the file. By default it displays first ten lines of the file.

$head [-count] [filename]

2. tail

This command displays the later part of the file. By default it displays last ten lines of the file.

$tail [-count] [filename]

3. wc

This command is used to count the number of lines, words or characters in a file.

$wc [-lwc] filename

4. find

The find command is used to locate files in a directory and in a subdirectory.

The –name option

This lists out the specific files in all directories beginning from the named directory. Wild cards can be used.

The –type option

This option is used to identify whether the name of files specified are ordinary files or directory files. If the name is a directory then use "-type d "and if it is a file then use "-type f".

The –mtime option

This option will allow us to find that file which has been modified before or after a specified time. The various options available are –mtime n(on a particular day),-mtime +n(before a particular day),-mtime –n(after a particular day)

The –exec option

This option is used to execute some commands on the files that are found by the find command.

IV File Permission commands

1) chmod

Changes the file/directory permission mode: $ chmod 777 file1

Gives full permission to owner, group and others

$ chmod o-w file1

Removes write permission for others.

V Useful Commands:

1) exit - Ends your work on the UNIX system.

2) Ctrl-l or clear

Clears the screen.

3) Ctrl-c

Stops the program currently running.

4) Ctrl-z

Pauses the currently running program.

5) man COMMAND

Looks up the UNIX command COMMAND in the online manual pages.

6) history

List all commands typed so far.

7) more FILE

Display the contents of FILE, pausing after each screenful.
There are several keys which control the output once a screenful has been printed.
<enter> Will advance the output one line at a time.
<space bar> Will advance the output by another full screenful.
"q" Will quit and return you to the UNIX prompt.

8) less FILE

"less" is a program similar to "more", but which allows backward movement in the file as well as forward movement.

9) lpr FILE

To print a UNIX text or PostScript file, type the following command at the system prompt:

Meta characters

Some special characters, called metacharacters may be used to specify multiple filenames. These characters substitute filenames or parts of filenames.

The "*"

This character is used to indicate any character(s)

$ cat ap*

This displays the contents of all files having a name starting with ap followed by any number of characters.

The "?" This character replaces any one character in the filename.

$ ls ?st

list all files starting with any character followed by st.

The [] These are used to specify range of characters.

$ ls [a-z]pple

Lists all files having names starting with any character from a to z.

Absolute path and relative path

Generally if a command is given it will affect only the current working directory. For example the following command will create a directory named curr in the current working directory.

$ mkdir curr

The directory can also be created else where in the file system using the absolute and relative

path.If the path is given with respect to the root directory then it is called full path or absolute path $ mkdir /home/it2006/it2k601/curr

The full path always start with the /, which represents the root directory.

If the path is given with respect to the current working directory or parent directory then it is called relative path.

$ mkdir ../curr

The above command will create a directory named curr in the parent directory.

$ mkdir ./first/curr

The above command will create a directory named curr inside first directory , where the directory first is located in the current working directory.

Note "." Represents current directory and ".." represents parent directory.

PIPES AND FILTERS

In UNIX commands were created to perform single tasks only. If we want to perform multiple tasks we can go for pipes and filters.

PIPES

A pipe is a mechanism, which takes the output of a command as its input for the next command.

$who | wc –l

$cat text.c | head –3

FILTERS

Filters are used to extract the lines, which contain a specific pattern, to arrange the contents of a file in a sorted order, to replace existing characters with some other characters, etc.

1.Sort filter

The sort filter arranges the input taken from the standard input in alphabetical order. The sort command when used with "-r" option will display the input taken from the keyboard in the reverse alphabetical order. When used with "-n" option arranges the numbers, alphabets and special characters according to their ASCII value. If we want to sort on any one field, then sort provides us with an option called "+pos1 –pos2" option.

2.Grep filter

This command is used to search for a particular pattern from a file or from standard input and display those lines on the standard output. Grep stands for "Global search for regular expression".

There are various options available with grep command.

-v displays only those lines which do not match the pattern specified.

-c displays only the count of those lines which match the pattern specified

-n displays matched lines with line numbers

-i displays matched pattern ignoring case distinction

3.Uniq filter

The uniq filter compares adjacent lines in the sorted input file and when used with different options displays single and multiple occurrences.

-d displays only the lines which are duplicated in the input file.

-u displays only the lines with single occurrences.

4.Pg and more filter

These commands display the output of the command on the screen page by page. The difference between pg and more filter is that the viewing screen of the latter can be done by pressing space bar while that of the former is done by pressing enter.

5.Cut command

One particular field from any file or from output of any command can be extracted and displayed using this cut command. One particular character can also be extracted using the –c option of this command.

6.Tr command

This command is used to translate characters taken from the standard input. This command when used with "-s" option is used to squeeze multiple spaces into a single space.

1.UNIX SYSTEM CALLS

Objective

To write a programs using the following system calls of UNIX operating system: fork, exec, getpid, exit, wait, close, stat, opendir, readdir

Description

When a computer is turned on, the program that gets executed first is called the ``*operating system*." It controls pretty much all activity in the computer. This includes who logs in, how disks are used, how memory is used, how the CPU is used, and how you talk with other computers. The operating system we use is called "Unix".

The way that programs talk to the operating system is via ``*system calls*." A system call looks like a procedure call (see below), but it's different -- **it is a request to the operating system to perform some activity**.

Getpid

Each process is identified by a unique *process id* (called a "pid"). The init process (which is the supreme parent to all processes) posesses id 1. All other processes have some other (possibly arbitrary) process id. The getpid system call returns the current process' id as an integer.

```
// …
int pid = getpid();
printf("This process' id is %d\n",pid);// …
```

fork

The fork system call creates a new child process. Actually, it's more accurate to say that it *forks* a currently running process. That is, it creates a *copy* of the current process as a new child process, and then both processes resume execution from the fork() call. Since it creates two processes, fork also returns two values; one to teach process. To the parent process, fork returns the *process id of the newly created child process*. To the child process, fork returns 0. The reason it returns 0 is precisely because this is an invalid process id. You would have no way of differentiating between the parent and child processes if fork returned an arbitrary positive integer to each.

Therefore, a typical call to fork looks something like this:

```
int pid;
if ( (pid = fork()) == 0 ) {
  /* child process executes inside here */
}
else {
 /* parent process executes inside here */
}
```

execvp & execlp

The exec functions (there are more than one) are a family of functions that *execute* some program *within* the current process space. So if I write a program that calls one of the exec functions, as soon as the function call succeeds the original process gets *replaced* with whatever

program I asked exec to execute. This is usually used in conjunction with a fork call. You would typically fork a child process, and then call exec from within the child process, to execute some other program in the new process entry created by fork.

Directory Operations

The complex nature of Unix File system directory entries means that it is not realistic to read such a directory using normal *read()* system calls and, in fact, attempting to use *read()* to read a directory will fail. Instead the system call getdents() is used to read directory entries, however this has a rather complex interface and for all normal purposes a set of library routines is provided, these, of course, work via *getdents()*.

There are eight such routines and two data types.

Directory handling data objects

DIR	Used to hold a pointer to an open directory. Analagous to *FILE* *

struct dirent	A structure holding information about the directory entry.
	`struct dirent`
	`{`
	`ino_t d_ino; /* i-number */`
	`off_t d_off; /* offset into directory file */`
	`ushort d_reclen; /* length of record */`
	`char d_name[1]; /* file name */`
	`}`

Both the above are *#define*'d in the standard header **dirent.h**. The eight routines are

Prototype	Function
DIR *opendir(const char **path*)	Opens a directory
struct dirent *readdir(DIR **dirp*)	Gets the next entry
struct dirent *readdir_r(DIR **dirp*,struct dirent **res*)	Similar to *readdir*, takes address of buffer as parameter. Intended for MT applications.
long telldir(DIR **dirp*)	returns current location
void seekdir(DIR **dirp*,long *loc*)	alters current position
void rewinddir(DIR **dirp*)	set current position to start
int closedir(DIR **dirp*)	closes directory

2. I/O System Calls

Objective

To Write a programs using the I/O system calls of UNIX operating system (open, read, write, etc).

Description

System Calls for I/O

There are 5 basic system calls that Unix provides for file I/O. The system object that is used to manipulate files is file descriptor. This is an integer number that is used by the various I/O system calls to access a memory area containing data about the open file.

Open

Open makes a request to the operating system to use a file. The call takes two parameters. The first argument '**path**' specifies what file you would like to use, and the '**flags**' and 'mode' arguments specify how you would like to use it. This call returns a file descriptor.

The mode may be any of the following:

O_RDONLY
: Open the file in read-only mode.

O_WRONLY
: Open the file in write-only mode.

O_RDWR
: Open the file for both reading and writing.

In addition, any of the following flags may be OR-ed with the mode flag:

O_CREAT
: If the file does not exist already - create it.

O_EXCL
: If used together with O_CREAT, the call will fail if the file already exists.

O_TRUNC
: If the file already exists, truncate it (i.e. erase its contents).

O_APPEND
: Open the file in append mode. Any data written to the file is appended at the end of the file.

O_NONBLOCK (or O_NDELAY)
: If any operation on the file is supposed to cause the calling process block, the system call instead will fail, and errno be set to EAGAIN. This requires caution on the part of the programmer, to handle these situations properly.

O_SYNC
: Open the file in synchronous mode. Any write operation to the file will block until the data is written to disk. This is useful in critical files (such as database files) that must always remain in a consistent state, even if the system crashes in the middle of a file operation.

Close

Close() tells the operating system that you are done with a file descriptor. The OS can then reuse that file descriptor. The usage is

Close(file descriptor)

Read

Read() tells the operating system to read "size" bytes from the file opened in file descriptor "fd", and to put those bytes into the location pointed to by "buf". It returns how many bytes were actually read. The prototype is

int read(fd, buf, size)

Write

Write() is just like read(), only it writes the bytes instead of reading them. It returns the number of bytes actually written, which is almost invariably "size".

rename

The rename() system call may be used to change the name (and possibly the directory) of an existing file. It gets two parameters: the path to the old location of the file (including the file name), and a path to the new location of the file (including the new file name). If the new name points to an already existing file, that file is deleted first. We are allowed to name either a file or a directory.

```
/* rename the file 'logme' to 'logme.1' */
if (rename("logme", "logme1") == -1) {
   perror("rename (1):");
   exit(1);
}
```

delete

Deleting a file is done using the unlink() system call. This one is very simple:

```
/* remove the file "/tmp/data" */
if (unlink("/tmp/data") == -1) {
   perror("unlink");
   exit(1);
}
```

EXPERIMENT 3. SIMULATION OF UNIX COMMANDS

Objective

To simulate the following unix commands
1)Is 2)grep

Description

ls

Use ls to see what files you have. Your files are kept in something called a directory.

ls---lists your files
ls -l --- lists your files in 'long format', which contains lots of useful information, e.g. the exact size of the file, who owns the file and who has the right to look at it, and when it was last modified.
ls -a --- lists all files, including the ones whose filenames begin in a dot, which you do not always want to see.

There are many more options, for example to list files by size, by date, recursively etc.

```
% ls
  foo       letter2
  foobar    letter3
  letter1   maple-assignment1
%
```

Note that you have six files. There are some useful variants of the **ls** command:

```
% ls l*
  letter1 letter2 letter3
%
```

Note what happened: all the files whose name begins with "l" are listed. The asterisk (*) is the " wildcard" character. It matches any string.

grep

grep ***string filename(s)*** --- looks for the string in the files. This can be useful a lot of purposes, e.g. finding the right file among many, figuring out which is the right version of something, and even doing serious corpus work. grep comes in several varieties (**grep**, **egrep**, and **fgrep**) and has a lot of very flexible options. Check out the man pages if this sounds good to you.

Use this command to search for information in a file or files. For example, suppose that we have a file *dict* whose contents are

```
red rojo
green verde
blue azul
white blanco
black negro
```

Then we can look up items in our file like this;

% **grep red dict**

red rojo

```
% grep blanco dict
  white blanco
% grep brown dict
%
```

Notice that no output was returned by grep brown. This is because "brown" is not in our dictionary file.

Grep can also be combined with other commands. For example, if one had a file of phone numbers named "ph", one entry per line, then the following command would give an alphabetical list of all persons whose name contains the string "sona".

```
% grep sona ph | sort
  sona College of technology salem-5
```

The symbol "|" is called "pipe." It pipes the output of the grep command into the input of the sort command.

EXPERIMENT 4. CPU SCHEDULING ALGORITHMS - I

Objective

To schedule the processes using FCFS(First Come First Served) and SJF(Shortest Job First) scheduling algorithms.

Description

When a computer is multi programmed , it has multiple processes competing for the CPU at the same time frequently. This situation occurs whenever two or more processes are simultaneously in the ready state. If only one CPU is available, a choice has to be made which process has to be in CPU. The part of the operating system that makes the choice is called the scheduler and the algorithm is called scheduling algorithm.

FCFS

In this scheduling policy the processes are assigned the CPU according to the order they arrive.

SJF

In this scheduling the process with shortest burst will be selected first. The processes are sorted in ascending order according to the CPU burst time.

Sample Input

Enter the number of processes:3
Process 1
Enter the CPU burst time: 5

Process 2
Enter the CPU burst time: 10

Process 3
Enter the CPU burst time:4

Sample Output

Process Name ArrivalTime BurstTime Wait time start End

The order in which the
processes are executed:

Waiting time for every
Proess Total waiting time is:

Average waiting time
for given FCFS :

Average turnaround time:

EXPERIMENT 5. CPU SCHEDULING ALGORITHMS - II

Objective

To schedule the processes using Priority and Round Robin scheduling algorithms.

Description

Priority

In this scheduling policy the processes are given certain priorities usually specified as a number. They are sorted according to the priorities and the process with highest priority is scheduled first.

Round Robin

In this algorithm, a time quantum is fixed for the process to get executed in the CPU. After that time quantum, the process is pre-empted and CPU is scheduled to another process. This will continue until all processes in the system complete their turn.

Sample Input:

Enter the number of processes:3
Process 1
Enter the CPU burst time: 5

Process 2
Enter the CPU burst time: 10

Process 3
Enter the CPU burst time:4

Sample Output:

Process Name ArrivalTime BurstTime Wait time start End

The order in which the
processes are executed:

Waiting time for every
Proess Total waiting time is:

Average waiting time
for given FCFS :

Average turnaround time:

EXPERIMENT 6. INTER PROCESS COMMUNICATION

Objective:

To implement Application using Inter Process communication (using shared memory, pipes or message queues).

Description:

Inter-process communication (**IPC**) is a set of techniques for the exchange of data among multiple threads in one or more processes. Processes may be running on one or more computers connected by a network. IPC techniques are divided into methods for message passing, synchronization, shared memory, and remote procedure calls (RPC). The method of IPC used may vary based on the bandwidth and latency of communication between the threads, and the type of data being communicated.

There are several reasons for providing an environment that allows process cooperation:

- Information sharing
- Computation speedup
- Modularity
- Convenience

IPC may also be referred to as *inter-thread communication* and *inter-application communication.*

1. **Pipes :** This allows the flow of data in one direction only. Data from the output is usually buffered until the input process receives it which must have a common origin.
2. **Named Pipes :** This is a pipe with a specific name. It can be used in processes that do not have a shared common process origin. Example is FIFO where the data is written to a pipe is first named.
3. **Message queuing:** This allows messages to be passed between messages using either a single queue or several message queues. This is managed by the system kernel. These messages are co-ordinated using an application program interface (API)
4. **Semaphores:** This is used in solving problems associated with synchronization and avoiding race conditions. They are integers values which are greater than or equal to zero
5. **Shared Memory:** This allows the interchange of data through a defined area of memory. Semaphore value has to be obtained before data can get access to shared memory.
6. **Sockets:** This method is mostly used to communicate over a network, between a client and a server. It allows for a standard connection which I computer and operating system independent.

EXPERIMENT 7. Unix Shell Scripts

Introduction

In previous discussions we have talked about many of the facilities of the C shell, such as command aliasing, job control, etc. In addition, any collection of csh commands may be stored in a file, and csh can be invoked to execute the commands in that file. Such a file is known as a shell script file. The language used in that file is called shell script language. Like other programming languages it has variables and flow control statements (e.g. if-then-else, while, for, goto).

In Unix there are several shells that can be used, the C shell (csh and its extension, the T C shell tcsh), the Bourne Shell (sh and its extensions the Bourne Again Shell bash and the highly programmable Korn shell ksh) being the more commonly used.

Note that you can run any shell simply by typing its name. For example, if I am now running csh and wish to switch to ksh, I simply type ksh, and a Korn shell will start up for me. All my commands from that point on will be read and processed by the Korn shell (though when I eventually want to log off, exiting the Korn shell will still leave me in the C shell, so I will have to exit from it too).

2. INVOKING SHELL SCRIPTS
There are two ways to invoke a shell script file.

2.1 Direct Interpretation

In direct interpretation, the command

```
csh filename [arg ...]
```

invokes the program csh to interpret the script contained in the file 'filename'.

2.2 Indirect Interpretation

In indirect interpretation, we must insert as the first line of the file

```
#!
/bin/csh
```

3 Shell Variables

Like other programming languages the csh language has variables. Some variables are used to control the operation of the shell, such as $path and $history, which we discussed earlier. Other variables can be created and used to control the operation of a shell script file.

3.1 Setting Variables

Values of shell variable are all character-based: A value is formally defined to be a list of zero or more elements, and an element is formally defined to be a character string. In other words, a shell variable consists of an array of strings.

For example,

```
set X
```

will set the variable $X to have an empty list as its value. The command

```
set V = abc
```

will set V to have the string 'abc' as its value. The command

```
set V = (123 def ghi)
```

will set V to a list of three elements, which are the strings '123', 'def' and 'ghi'.

The several elements of a list can be treated like array elements. Thus for V in the last example above, $V[2] is the string 'def'. We could change it, say to 'abc', by the command `set V[2] = abc`

3.2 Referencing and Testing Shell Variables

The value of a shell variable can be referenced by placing a $ before the name of the variable. The

command `echo $path`

will output the value of the variable $path. Or you can access the variable by enclosing the variable name in curly brace characters, and then prefixing it with a $. The command

`echo ${pat`

would have the same result as the last example. The second method is used when something is to be appended to the contents of the variable. For example, consider the commands

```
set fname=
prog1 rm
${fname}.c
```

These would delete the file 'prog1.c'.

To see how many elements are in a variable's list, we prefix with a # then a $. The command

```
echo $#V
```

above would print 3 to the screen, while

```
echo $#path
```

would reveal the number of directories in your search path.

The @ command can be used for computations. For example, if you have shell variables $X and $Y, you can set a third variable $Z to their sum by

```
@Z = $X +
$Y
```

4 Command Arguments

Most commands have arguments (parameters), and these are accessible via the shell variable $argv. The first parameter will be $argv[1], the second $argv[2], and so on. You can also refer to them as $1, $2, etc. The number of such arguments (analogous to argc in the C language) is $#argv.

For example, consider the following script file, say named Swap:

```
#! /bin/csh -
f

set tmp =
$argv[1]
cp $argv[2]
$argv[1]
cp $tmp
$argv[2]
```

This would do what its name implies, i.e. swap two files. If, say, I have files x and y, and I type

```
Swap x
y
```

then the new contents of x would be what used to be y, and the new contents of y would be what used to be

Examples

A Shell Script For Deleting Files

This code, which we will call Del, will delete files like rm does, prompting for your confirmation for each file to be deleted, including directory files (which the -i option of rm won't do).

```
#! /bin/csh -f

foreach name
($argv)
if ( -f $name ) then
echo -n "delete the file '${name}'
(y/n/q)?" else
echo -n "delete the entire directory '${name}'
(y/n/q)? " endif
set ans = $<
switch ($ans)
case n:
continu
e case q:
```

```
            exi
        t case
        y:

            rm -r
            $name
            continue
    endsw
end
```