Question 2: (15 Marks) The Sleeping-Barber Problem:

A barbershop consists of a waiting room with n chairs, and the barber room containing the barber chair. If there are no customers to be served, the barber goes to sleep. If a customer enters the barbershop and all chairs are occupied, then the customer leaves the shop. If the barber is busy, but chairs are available, then the customer sits in one of the free chairs. If the barber is asleep, the customer wakes up the barber.
Write an interactive program in C / C++ to synchronize/coordinate the barber and the customers.

Answer 2 :

```
# define CHAIRS 5                          // chairs for waiting customers
typedef int semaphore;                     // use this for imagination
semaphore customers = 0;                   // number of customers waiting for service
semaphore barbers – 0;                     // number of barbers waiting for customers
semaphore mutex = 1;                       // for mutual exclusion
int waiting = 0;                           //customers who are waiting for a haircut
void barber(void)
while (TRUE) {
down(&customers);                          //go to sleep if no of customers are zero
down(&mutex);                              //acquire access to waiting
waiting = waiting -1 ;                     //decrement count of waiting customers
up(&barbers);                              //one barber is now ready for cut hair
up(&mutex);                                //release waiting
cut_hair();                                //this is out of critical region for hair cut
}
}
```

```
void customer(void)

{

down(&mutex); //enter critical region

if (waiting < CHAIRS) //if there are no free chairs, leave

{

waiting = waiting +1; //increment count of waiting customers

up(&customers); //wake up barber if necessary up(&mutex);

//release access to waiting

down(&barbers); //go to sleep if no of free barbers is

zero get_haircut(); //be seated and be serviced

} else

{

up (&mutex); // shop is full: do no wait

}

}
```

Question 3: **(10 Marks)**

Study and implement the Lamport's Bakery Algorithm for Interprocess synchronization using C/C++ programming language

Answer 3 :

Lamport's Bakery Algorithm provides a decentralised implementation of the "take a number" idea.

For each process, i, there are two values, C[i] and N[i], giving the status of process I and the number it has picked. In more detail:

N[i] = 0 --> Process i is not in the bakery.

N[i] > 0 --> Process i has picked a number and is in the bakery.

C[i] = 0 --> Process i is not trying to pick a number.

C[i] = 1 --> Process i is trying to pick a

number. when

N[i] = min(for all j, N[j] where N[j] > 0)

Process i is allowed into the critical section.

Here is the basic algorithm used to pick a number:

C[i] := 1;

N[i] := max(for all j, N[j]) + 1;

C[i] := 0;

In effect, the customer walks into the bakery, checks the numbers of all the waiting customers, and then picks a number one larger than the number of any waiting customer.

If two customers each walk in at the same time, they are each likely to pick the same number.

A process does the following to wait for the baker:

Step 1:

while (for some j, C(j) = 1) do nothing;

First, wait until any process which might have tied with you has finished selecting their numbers. Since we require customers to raise their hands while they pick numbers, each customer waits until all hands are down after picking a number in order to guarantee that all ties will be cleanly recognised in the next step.

Step 2:

repeat

W := (the set of j such that N[j] > 0)

(where W is the set of indeces of waiting processes)

M := (the set of j in W

such that N[j] <=

N[k] for all k in W)

(where M is the set of process indices with minimum

numbers) j := min(M)

(where is in M and the tie is

broken) until i = j;

Question 4: **(20 Marks)**

Ubuntu is an Operating System based on the Debian Linux Distribution and distributed as free and open source software, using its own desktop environment. Discuss in detail the features, process scheduling, file handling and protection & protection mechanism in it.

Answer 4 :

UNIX is one of the most popular OS today because of its abilities like multi-user, multi-tasking environment, stability and portability. The fundamental features which made UNIX such a phenomenally successful operating systems are:

• **Multi-user:** More than one user can use the machine at a time supported via terminals (serial or network connection).

• **Multi-tasking:** More than one program can be run at a time.

• **Hierarchical file system:** To support file organization and maintenance in a easier manner.

• **Portability:** Only the kernel (<10%) is written in assembler. This means that the operating system could be easily converted to run on different hardware.

• **Tools for program development:** Supports a wide range of support tools (debuggers, compilers).

File and Directory Permissions

Each file or a directory on a UNIX system has three types of permissions. These permissions are read (r), write (w) and execute (x), and the three categories of users are user or owner (u), group (g) and others (o).

These file and directory permissions can only be modified by:

• their owners

• by the superuser (root)

by using the chmod(change [file or directory])mode system utility. $ chmod options files chmod accepts options in two forms. Firstly, permissions may be specified as a sequence of 3 octal digits. Each octal digit represents the access permissions for the user/owner, group and others respectively. The mapping of permissions onto their corresponding octal digits is as follows:

---	0
--x	1
-w-	2
-wx	3
r--	4
r-x	5
rw-	6
rwx	7

The context of a process is essentially a snapshot of its current runtime environment, including its address space, stack space, etc. At any given time, a process can be in user-mode, kernel-mode, sleeping, waiting on I/O, and so on. The process scheduling subsystem within the kernel uses a time slice of typically 20ms to rotate among currently running processes. Each process is given its share of the CPU for 20ms, then left to sleep until its turn again at the CPU. This process of moving processes in and out of the CPU is called context switching. The kernel makes the operating system appear to be multi-tasking (i.e. running processes concurrently) via the use of efficient context-switching.

At each context switch, the context of the process to be swapped out of the CPU is saved to RAM. It is restored when the process is scheduled its share of the CPU again. All this happens very fast, in microseconds.

Question 5: **(20 Marks)**

Discuss in detail the Process management, Memory management, I/O management, File management and Security and Protection in WINDOWS 8 Operating System.

Answer 5 :

Each process in Windows 2000 operating system contains its own independent virtual address space with both code and data, protected from other processes. Each process, in turn, contains one or more independently executing *threads*. A thread running within a process can create new threads, create new independent processes, and manage communication and synchronization between the objects.

By creating and managing processes, applications can have multiple, concurrent tasks processing files, performing computations, or communicating with other networked systems. It is even possible to exploit multiple processors to speed processing.

The following sections explain the basics of process management and also introduce the basic synchronization operations.

- **Job:** collection of processes that share quotas and limits
- **Process:** container for holding resources
- **Thread:** Entity scheduled by the kernel
- **Fiber:** Lightweight thread managed entirely in user space.

Interprocess Communication

Threads can communicate in many ways including: pipes, named pipes, mailslots, sockets, remote procedure calls, and shared files

Semaphore

A semaphore is created using function 'CreateSemaphore' API function. As Semaphores are kernel objects and thus have security descriptors and handles. The handle for a semaphore can be duplicated and as a result multiple process can be synchronised on the same semaphore. Calls for up (ReleaseSemaphore) and down (WaitForSingleObject) are present. A calling thread can be released eventually using WaitForSingleObject, even if the semaphore remains at 0.

Mutexes

Mutexes are kernel objects used for synchronization and are simpler than semaphores as they do not have counters. They are locks, with API functions for locking (WaitForSingleObject) and unlocking (releaseMutex). Like Semaphores, mutex handles can be duplicated.

Critical Sections or Critical Regions

This is the third synchronization mechanism which is similar to mutexes. It is pertinent to note that Critical Section are not kernel objects, they do not have handles or security descriptors and cannot be passed between the processes. Locking and unlocking is done using EnterCriticalSection and LeaveCriticalSection, respectively. As these API functions are performed initially in user space and only make kernel calls when blocking is needed, they are faster than mutexes.

Events

This synchronization mechanism uses kernel objects called events, which are of two types: manual-reset events and auto-reset events. The number of Win32 API calls dealing with processes, threads, and fibres is nearly 100. Windows 2000 knows nothing about fibres and fibres are implemented in user space. As a result, the CreateFibre call does its work entirely in user space without making 12-13 system calls.

Scheduling There is no central scheduling thread in Windows 2000 operating system. But when a thread cannot run any more, the thread moves to kernel mode and runs the scheduler itself to see which thread to switch to and the concurrency control is reached through the following conditions

• Thread blocks on a semaphore, mutex, event, I/O, etc.: In this situation the thread is already running in kernel mode to carry out the operation on the dispatcher or I/O object. It cannot possibly continue, so it must save its own context, run the scheduler to pick its successor, and load that threat's context to start it.

• It signals an object – In this situation, thread is running in kernel mode and after signaling some object it can continue as signaling an object never blocks. Thread runs the scheduler to verify if the result of its action has created a higher priority thread. If so, a thread switch occurs because Windows 2000 is fully pre-emptive.

• The running thread's quantum expires: In this case, a trap to kernel mode occurs, thread executes the scheduler to see who runs next. The same thread may be selected again and gets a new quantum and continue running, else a thread switch takes place.

MEMORY MANAGEMENT

Windows 2000 consists of an advanced virtual memory management system. It provides a number of Win32 functions for using it and part of the executives and six dedicated kernel threads for managing it. In Windows 2000, each user process has its own virtual address space, which is 32 bit long (or 4 GB of virtual address space). The lower 2 GB minus approx 256 MB are reserved for process's code and data; the upper 2 GB map onto to kernel memory in a protected way. The virtual address space is demand paged with fixed pages size.

INPUT/OUTPUT IN WINDOWS 2000

The goal of the Windows 2000 I/O system is to provide a framework for efficiently handle various I/O devices. These includes keyboards, mice, touch pads, joysticks, scanners, still cameras, television cameras, bar code readers, microphones, monitors, printers, plotters, beamers, CD-records, sound cards, clocks, networks, telephones, and camcorders.

FILE SYSTEM MANAGEMENT

Windows 2000 supports several file systems like FAT-16, FAT-32, and NTFS. Windows 2000 also supports read-only file systems for CD-ROMs and DVDs. It is possible to have access to multiple file system types on the same running system.

New Technology File System (NTFS)

NTFS stands for **N**ew **T**echnology **F**ile **S**ystem. Microsoft created NTFS to compensate for the features it felt FAT was lacking. These features include increased fault tolerance and enhanced security. Individual File names in NTFS are limited to 255 characters; full paths are limited to 32,767 characters. Files are in Unicode. But Win32 API does not fully support case-sensitivity for file names. NTFS consists of multiple attributes, each of which is represented by a stream of bytes

Security

NTFS has many security options. You can grant various permissions to directories and to individual files. These permissions protect files and directories locally and remotely. NT operating system was designed to meet the U.S. Department of Defense's C2 (the orange book security requirements). Windows 2000 was not designed for C2 compliance, but it inherits many security properties from NT, including the following: (1) secure login with anti-spoofing mechanism, (2) Discretionary access controls, (3) Privileged

access controls, (4) Address space protection per process, (5) New pages must be zeroed before being mapped in, (6) security auditing

File Compression

Another advantage to NTFS is native support for file compression. The NTFS compression offers you the chance to compress individual files and folders of your choice.