CMR INSTITUTE OF TECHNOLOGY

SCHEME OF EVALUATION

Internal Assessment Test I – September. 2016

Sub:	OPERATING SYSTEMS			Code:	10CS53
Date: 7/9/2016	Duration: 90 mins	Max Marks: 50	Sem: 5 A,B,C	Branch:	CSE

Note: Answer FIVE full Questions,selecting at least TWO from each part.

Question No.		Description	Distribution of marks		Total Marks
1.	a)	• Explain all the services of OS	1Mx10	10M	10M
2.	a)	• Diagram of VMware • Explanation	3 2	5M	10M
	b)	• Diagram of JVM • Explanation	3 2	5 M	
3.	a)	Explanation of Message passing and IPC	5	5 M	10M
	b)	• Diagrams of models • Explanation of models	3 2	5 M	
4.	a)	• Digram of Process States • Explanation of each state in one line	3 2	5M	10M
	b)	• Synchronization • Swap Instructions • Semaphores	1 2 2	5M	
5.	a)	• Definition of System call. • Types of System calls	1 4	5M	10M
	b)	• Gantt Chart • Avg TAT • AVG WT	3 1 1	5M	
6.	a)	• Diagram of PCB • Explanation of Componets of PCB	2 3	5M	10M
	b)	• Gantt Chart • Avg TAT • AVG WT	3 1 1	5M	
7.	a)	• Memory-management functions • Process management functions	2.5 2.5	5M	10M
	b)	• Gantt Chart • Avg TAT • AVG WT	3 1 1	5M	
8.	a)	• 5 differences between process and threads	1x5	5M	10M
	b)	• W: wait(S); X: Signal(T); Y: wait(T); Z: Signal(S);	5	5M	

SOLUTIONS

PART - A

1a) Explain the services provided by Operating Systems in detail.

An operating system provides an environment for the execution of programs. It provides certain services to programs and to the users of those programs. The specific services provided, of course, differ from one operating system to another, but we can identify common classes. These operating-system services are provided for the convenience of the programmer, to make the programming task easier.
One set of operating-system services provides functions that are helpful to the user.

User Interface: Almost all operating systems have a User Interface (UI). This interface can take several forms. One is a DTrace Command Line Interface (CLI) which uses text commands and a method for entering them (say, a program to allow entering and editing of commands). Another is a batch Interface, in which commands and directives to control those commands are entered into files, and those files are executed. Most commonly, a graphical user interface (GUI) is used. Here, the interface is a window system with a pointing device to direct I/O, choose from menus, and make selections and a keyboard to enter text. Some systems provide two or all three of these variations.
Program execution: The system must be able to load a program into memory and to run that program. The program must be able to end its execution, either normally or abnormally (indicating error).
I/O operations: A running program may require I/O, which may involve a file or an I/O device. For specific devices, special functions may be desired (such as recording to a CD or DVD drive or blanking a display screen). For efficiency and protection, users usually cannot control I/O devices directly. Therefore, the operating system must provide a means to do I/O.
File-system manipulation: The file system is of particular interest. Obviously, programs need to read and write files and directories. They also need to create and delete them by name, search for a given file, and list file information. Finally, some programs include permissions management to allow or deny access to files or directories based on file ownership. Many operating systems provide a variety of file systems, sometimes to allow personal choice, and sometimes to provide specific features or performance characteristics.
Communications: There are many circumstances in which one process needs to exchange information with another process. Such communication may occur between processes that are executing on the same computer or between processes that are executing on different computer systems tied together by a computer network. Communications may be implemented via *shared memory* or through *message passing,* in which packets of information are moved between processes by the operating system.
Error detection: The operating system needs to be constantly aware of possible errors. Errors may occur in the CPU and memory hardware (such as a memory error or a power failure), in I/O devices (such as a parity error on tape, a connection failure on a network, or lack of paper in the printer), and in the user program (such as an arithmetic overflow, an attempt to access an illegal memory location, or a too-great use of CPU time). For each
type of error, the operating system should take the appropriate action to ensure correct and consistent computing. Of course, there is variation in how operating systems react to and correct errors. Debugging facilities can greatly enhance the user's and programmer's abilities to use the system efficiently.

Another set of operating-system functions exists not for helping the user but rather for ensuring the efficient operation of the system itself. Systems with multiple users can gain efficiency by sharing the computer resources among the users.

Resource allocation: When there are multiple users or multiple jobs running at the same time, resources must be allocated to each of them. Many d1Herent -types of resources are managed by the operating system. Some (such as CPU cycles, main memory, and file storage) may have special allocation code, whereas others (such as I/O devices) may have much more general request and release code. For instance, in determining how best to use the CPU, operating systems have CPU-scheduling routines that take into account the speed of the CPU, the jobs that must be executed, the number of registers available, and other factors. There may also be routines to allocate printers, modems, USB storage drives, and other peripheral devices.

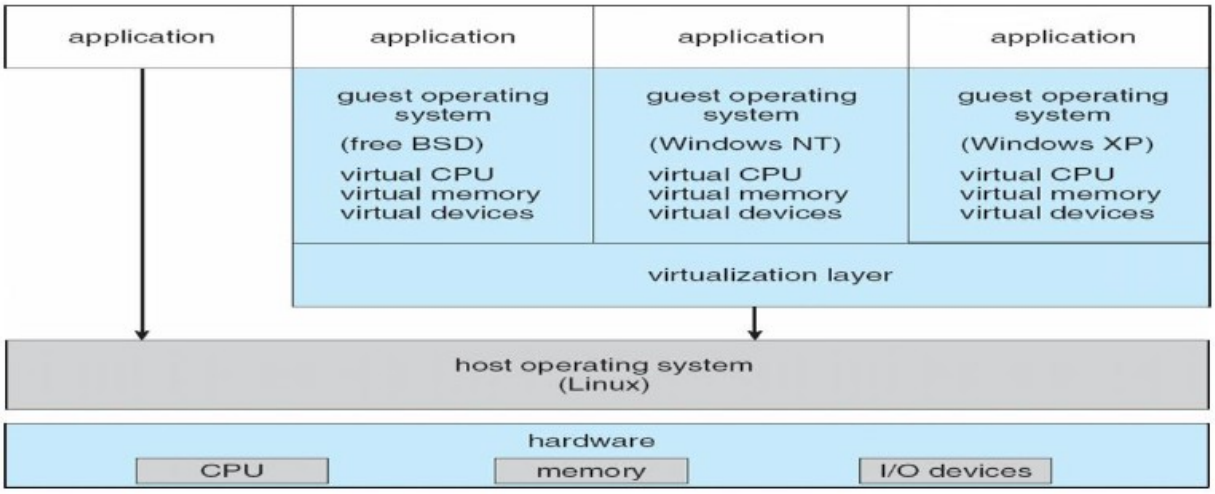
Accounting: We want to keep track of which users use how much and what kinds of computer resources. This record keeping may be used for accounting (so that users can be billed) or simply for accumulating usage statistics. Usage statistics may be a valuable tool for researchers who wish to reconfigure the system to improve computing services.

Protection and security: The owners of information stored in a multiuser or networked computer system may want to control use of that information. When several separate processes execute concurrently, it should not be possible for one process to interfere with the others or with the operating system itself. Protection involves ensuring that all access to system- resources is controlled. Security of the system from outsiders is also important. Such security starts with requiring each user to authenticate himself or herself to the system, usually by means of a password, to gain access to system resources. It extends to defending external I/O devices, including modems and network adapters, from invalid access attempts and to recording all such connections for detection of break-ins. If a system is to be protected and secure, precautions must be instituted throughout it. A chain is only as strong as its weakest link.

2a) Explain VMware with neat diagram.

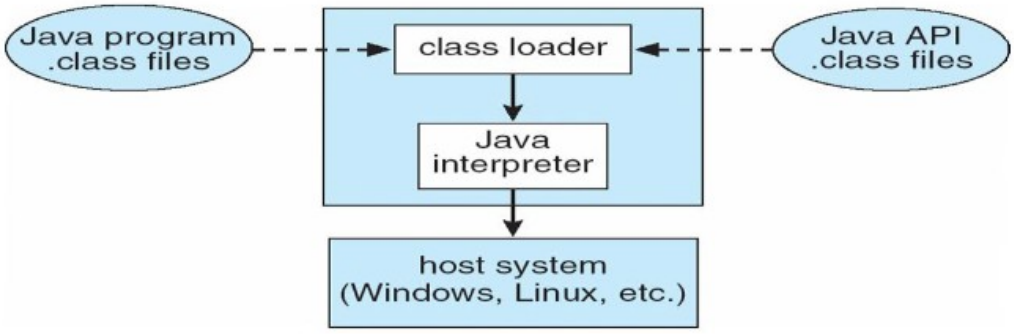
A virtual machine takes the layered approach to its logical conclusion. It treats hardware and the operating system kernel as though they were all hardware.

VMware Architecture



2b) Explain JVM and Just in time compilation with neat diagram.

The Java Virtual Machine



3a What is IPC? Explain different methods for logical implementation of message passing systems. Explain any one.

Interprocess Communication:

Message Passing

- Mechanism for processes to communicate and to synchronize their actions
- Message system – processes communicate with each other without resorting to shared variables
- IPC facility provides two operations:
- **send**(*message*) – message size fixed or variable
- **receive**(*message*)
- If *P* and *Q* wish to communicate, they need to:
- establish a *communication link* between them
- exchange messages via send/receive
- Implementation of communication link
- physical (e.g., shared memory, hardware bus)
- logical (e.g., logical properties)

Direct Communication

- Processes must name each other explicitly:
- **send** (*P, message*) – send a message to process P
- **receive**(*Q, message*) – receive a message from process Q
- Properties of communication link
- Links are established automatically
- A link is associated with exactly one pair of communicating processes
- Between each pair there exists exactly one link
- The link may be unidirectional, but is usually bi-directional

Indirect Communication

- Messages are directed and received from mailboxes (also referred to as ports)
- Each mailbox has a unique id
- Processes can communicate only if they share a mailbox
- Properties of communication link
- Link established only if processes share a common mailbox
- A link may be associated with many processes
- Each pair of processes may share several communication links
- Link may be unidirectional or bi-directional
- Operations
- create a new mailbox
- send and receive messages through mailbox
- destroy a mailbox
- Primitives are defined as:
- **send**(*A, message*) – send a message to mailbox A
- **receive**(*A, message*) – receive a message from mailbox A
- Mailbox sharing
- P_1, P_2, and P_3 share mailbox A

- P_1, sends; P_2 and P_3 receive
- Who gets the message?
- Solutions
- Allow a link to be associated with at most two processes
- Allow only one process at a time to execute a receive operation
- Allow the system to select arbitrarily the receiver. Sender is notified who the receiver was.

Synchronization

i) Message passing may be either blocking or non-blocking
ii) **Blocking** is considered **synchronous**
iii) **Blocking send** has the sender block until the message is received
iv) **Blocking receive** has the receiver block until a message is available
v) **Non-blocking** is considered **asynchronous**
vi) **Non-blocking** send has the sender send the message and continue
vii) **Non-blocking** receive has the receiver receive a valid message or null

Buffering

Queue of messages attached to the link; implemented in one of three ways

1. Zero capacity – 0 messages

Sender must wait for receiver (rendezvous)

2. Bounded capacity – finite length of *n* messages

Sender must wait if link full

3. Unbounded capacity – infinite length

Sender never waits

3b Explain with diagrams different thread models.

User Threads

- Thread management done by user-level threads librarynThree primary thread libraries:
- POSIX Pthreadsl Win32 threads
- Java threads

Kernel Threads

Supported by the Kernel

Examples

- Windows XP/2000
- Solaris
- Linux
- Tru64 UNIX
- Mac OS X

Multithreading Models

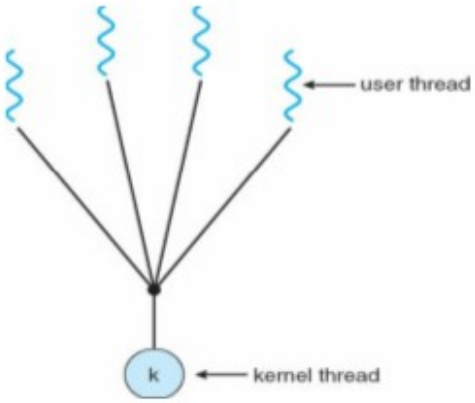
- Many-to-One
- One-to-One
- Many-to-Many

Many-to-One

Many user-level threads mapped to single kernel thread

Examples:

- Solaris Green Threads
- GNU Portable Threads



One-to-One

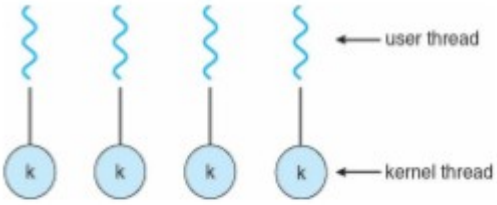
Each user-level thread maps to kernel thread

Examples

Windows NT/XP/2000

Linux

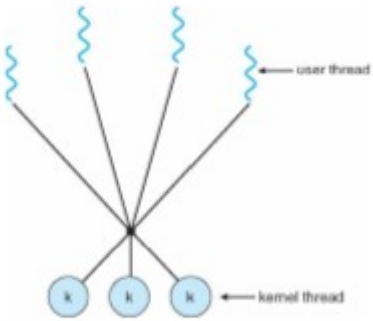
Solaris 9 and later



Many-to-Many Model

- Allows many user level threads to be mapped to many kernel threads
- Allows the operating system to create a sufficient number of kernel threads
- Solaris prior to version 9

Windows NT/2000 with the *ThreadFiber* package

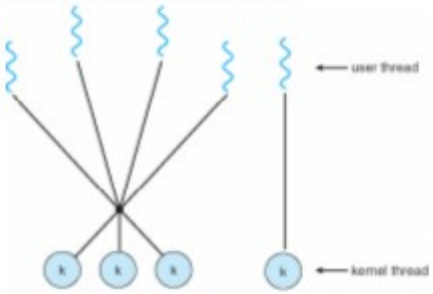


Two-level Model

Similar to M:M, except that it allows a user thread to be **bound** to kernel thread

Examples

- IRIX
- HP-UX
- Tru64 UNIX
- Solaris 8 and earlier



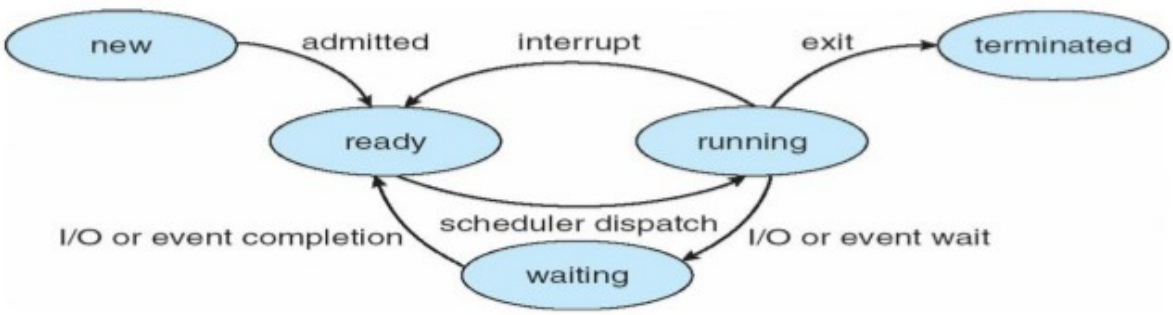
4a)Explain process states with neat diagram.

Process State

As a process executes, it changes *state*

- **new**: The process is being created
- **running**: Instructions are being executed
- **waiting**: The process is waiting for some event to occur
- **ready**: The process is waiting to be assigned to a processor
- **terminated**: The process has finished execution

Diagram of Process State



4b.) What is Synchronization? Explain Semaphores and Swap in hardware synchronization.

Swap Instruction

Definition:

```
void Swap (boolean *a, boolean *b)
{
    boolean temp = *a;
    *a = *b;
    *b = temp:
}
```

Solution using Swap

Shared Boolean variable lock initialized to FALSE; Each process has a local Boolean variable key

Solution:

```
do {
    key = TRUE;
    while ( key == TRUE)
        Swap (&lock, &key );

        //  critical section
    lock = FALSE:

        //  remainder section
} while (TRUE);
```

Semaphore as General Synchronization Tool

- Counting semaphore – integer value can range over an unrestricted domain
- Binary semaphore – integer value can range only between 0 and 1; can be simpler to implement
- Also known as mutex locksnCan implement a counting semaphore S as a binary semaphore
- Provides mutual exclusionSemaphore mutex; // initialized to do {
  wait (mutex);
  // Critical Section
  signal (mutex);

PART – B

5a What are system calls? Explain different types of System calls.

System Calls

- Programming interface to the services provided by the OS
- Typically written in a high-level language (C or C++)
- Mostly accessed by programs via a high-level Application Program Interface (API) rather than direct system call usenThree most common APIs are Win32 API for Windows, POSIX API for POSIX-based systems (including virtually all versions of UNIX, Linux, and Mac OS X), and Java API for the Java virtual machine (JVM)

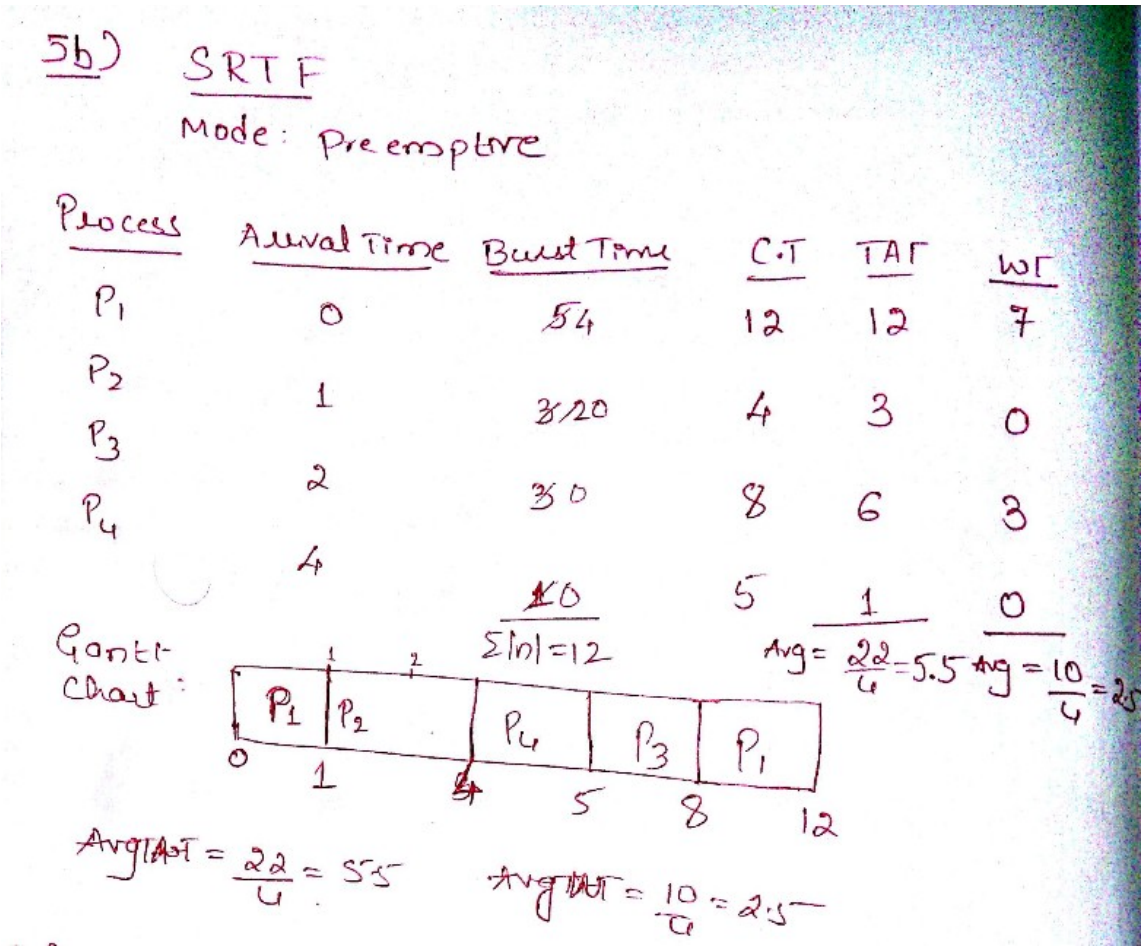
Types of System Calls

- Process control
- File management
- Device management
- Information maintenance
- Communications
- Protection

Examples of Windows and Unix System Calls

	Windows	Unix
Process Control	CreateProcess() ExitProcess() WaitForSingleObject()	fork() exit() wait()
File Manipulation	CreateFile() ReadFile() WriteFile() CloseHandle()	open() read() write() close()
Device Manipulation	SetConsoleMode() ReadConsole() WriteConsole()	ioctl() read() write()
Information Maintenance	GetCurrentProcessID() SetTimer() Sleep()	getpid() alarm() sleep()
Communication	CreatePipe() CreateFileMapping() MapViewOfFile()	pipe() shmget() mmap()
Protection	SetFileSecurity() InitializeSecurityDescriptor() SetSecurityDescriptorGroup()	chmod() umask() chown()

5b)

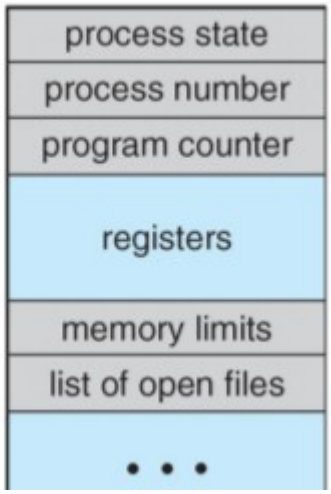


6a) Explain the components of PCB with neat diagram.

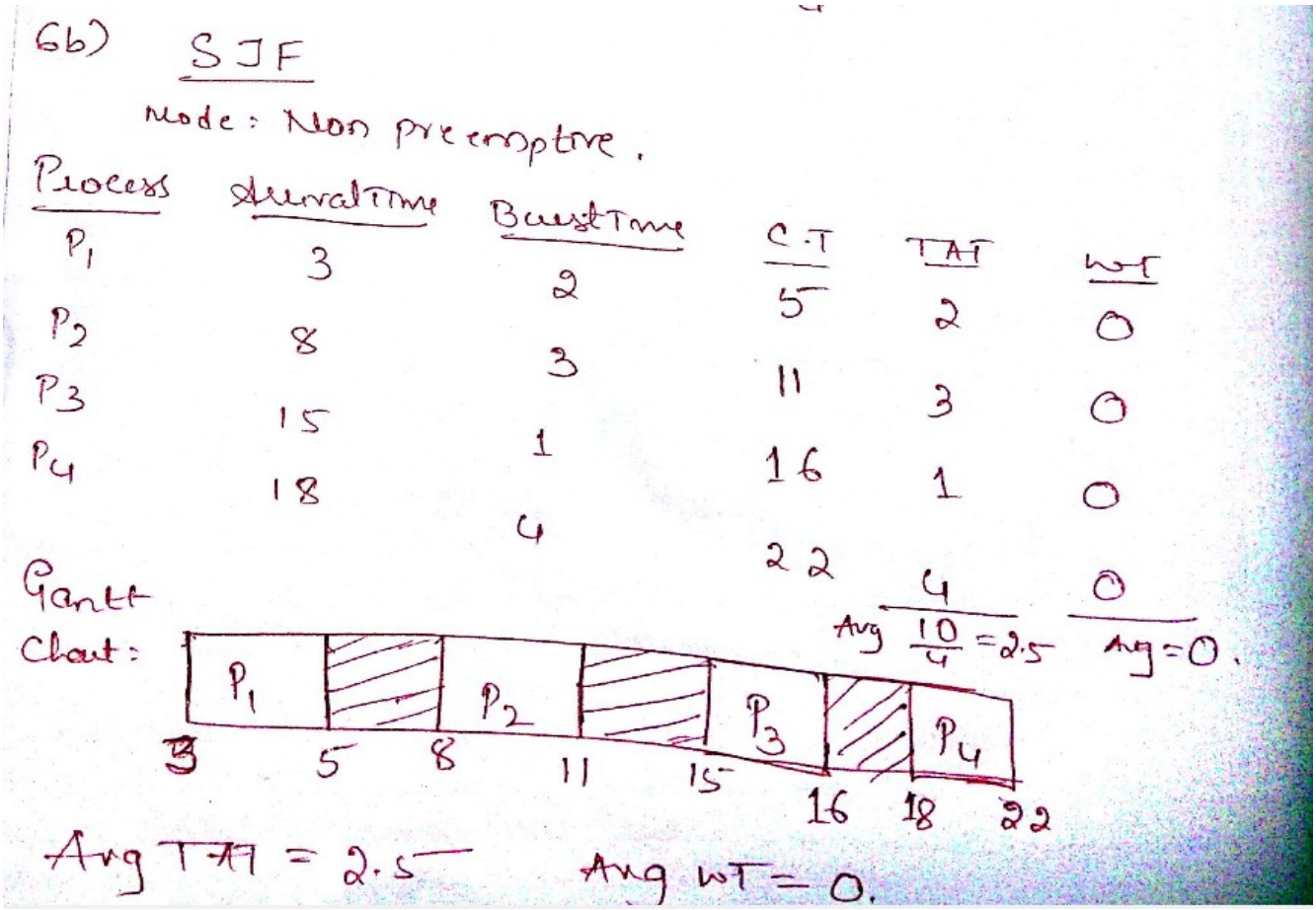
Process Control Block (PCB)

Information associated with each process

- Process state
- Program counter
- CPU registers
- CPU scheduling information
- Memory-management information
- Accounting information
- I/O status information



6b)



7a) Explain the various functions of operating system with respect process management and memory management.

Process Management Activities

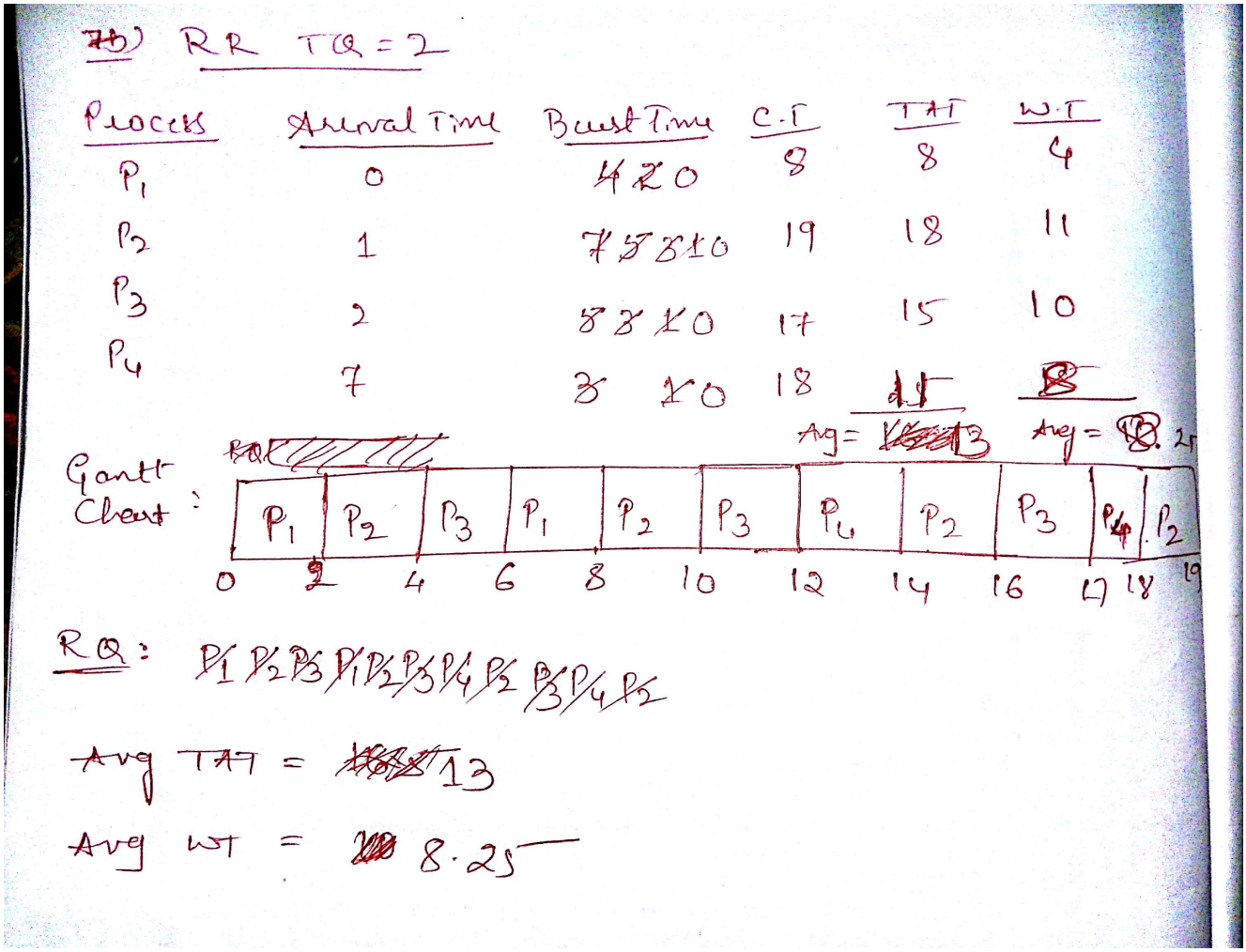
- The operating system is responsible for the following activities in connection with process management:
- Creating and deleting both user and system processes
- Suspending and resuming processes
- Providing mechanisms for process synchronization
- Providing mechanisms for process communication
- Providing mechanisms for deadlock handling

Memory Management

- All data in memory before and after processing
- All instructions in memory in order to execute
- Memory management determines what is in memory when
- Optimizing CPU utilization and computer response to users

- **Memory management activities**
- Keeping track of which parts of memory are currently being used and by whom
- Deciding which processes (or parts thereof) and data to move into and out of memory
- Allocating and deallocating memory space as needed

7b)



8a) Differentiate between process and threads.

Process

Each process provides the resources needed to execute a program. A process has a virtual address space, executable code, open handles to system objects, a security context, a unique process identifier, environment variables, a priority class, minimum and maximum working set sizes, and at least one thread of execution. Each process is started with a single thread, often called the primary thread, but can create additional threads from any of its threads.

Thread

A thread is the entity within a process that can be scheduled for execution. All threads of a process share its virtual address space and system resources. In addition, each thread maintains exception handlers, a scheduling priority, thread local storage, a unique thread identifier, and a set of structures the system will use to save the thread context until it is scheduled. The thread context includes the thread's set of machine registers, the kernel stack, a thread environment block, and a user stack in the address space of the thread's process. Threads can also have their own security context, which can be used for impersonating clients.

8b)

Suppose we want to synchronize two concurrent processes P and Q using binary semaphores S and T. The code for the processes P and Q is shown below.

Process P:	Process Q:

while (1) {
W:_____
print '0';
print '0';
X:_____
}
	while (1) {
	Y:_____
	print '1';
	print '1';
	Z:_____
	}

Synchronization statements can be inserted only at points W, X, Y and Z. Initially S=1 and T=0. Fill W,X,Y,Z such that it **will always lead to an output starting with '001100110011'**

W: wait(S);
X: Signal(T);
Y: wait(T);
Z: Signal(S);