ANISH ADNANI
D17B/01

DC
(1)

ASSIGNMENT 2

Q1) Compare process and Thread

→

Process	Thread
1. A program in execution is a process	1. Thread means a segment of a process
2. A process consists of multiple threads	2. A Thread is a smallest part of the process that can execute concurrently (threads) of process
3. A process has its own address space	3. A thread uses the process's address space and share it with the other threads of the process
4. A process can communicate with other process by using inter process communication.	4. A Thread can communicate with other thread (of the same process) directly using the methods like wait() notify() notifyall()
5. The process is mostly isolated and doesn't store data	5. Thread share memory.

	process	Thread
6.	Creation of each process requires seperate system call for each process	6. Single system call can create more than one thread.
7.	Per process items 1) Address space 2) Global variables 3) open files 4) child processes 5) Pending alarm 6) signal and signal handler 7) Accounting info	7. Per Thread items 1) Program counter 2) Registers 3) stack 4) state

Q2) Explain desirable features of a scheduling algorithm.

→ 1. No a priori knowledge about processes:
- user does not want to specify information about characteristic & requirement

2. Dynamic in nature:
The decision regarding the assignment of process should be dynamic, which means that it should be based on the current load of the systems and not on the same state policy.

3. Quick decision making capability:
Algorithm must make quick decision about the assignment of task nodes of systems

4. Balanced system performance and scheduling overhead:
Great amount of information gives more intelligent decision but increases overhead.

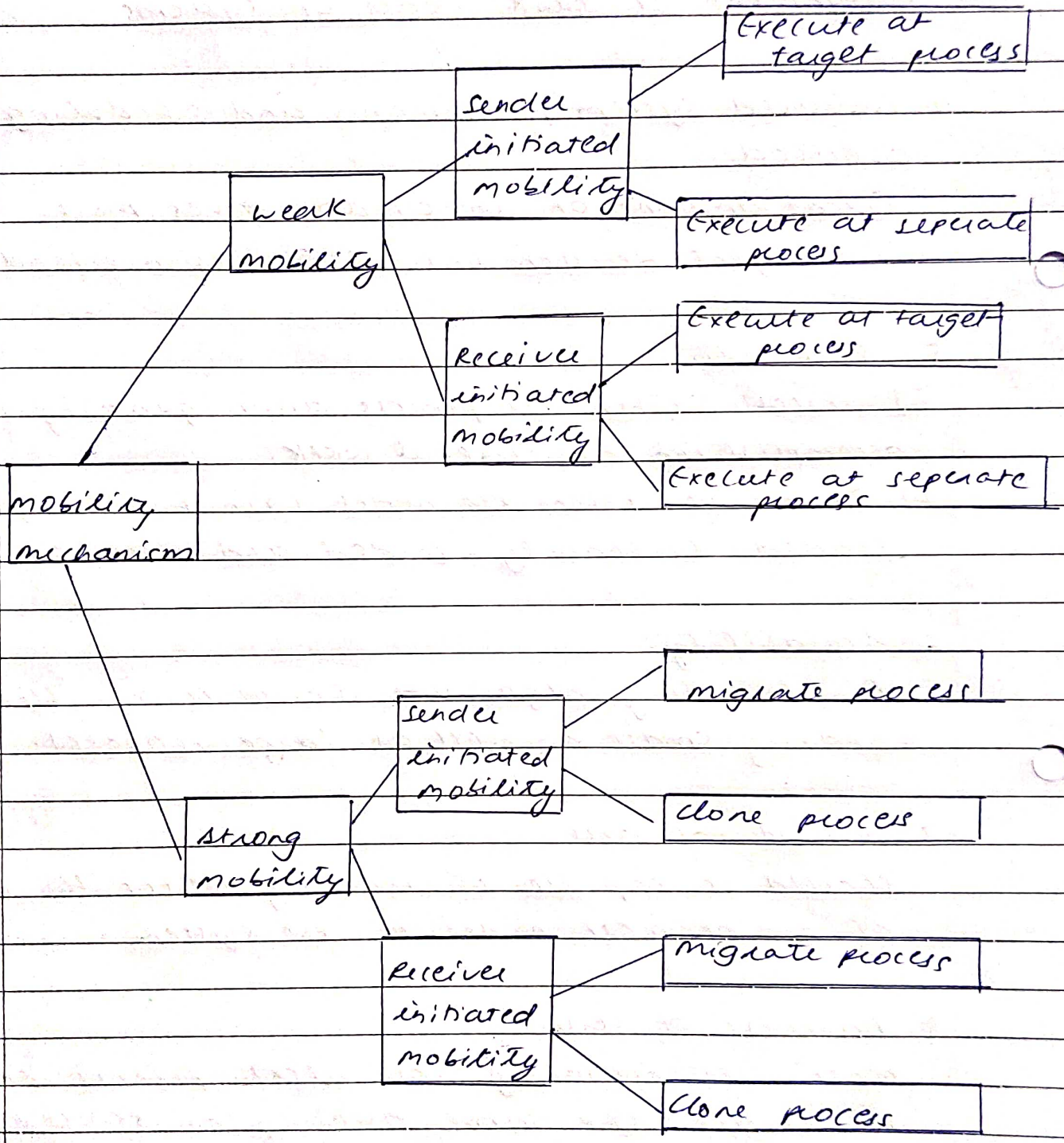
5. Stability:
a) unstable when all process are migrating without accomplishing any useful work.
b) It occurs when the nodes turn from lightly-loaded to heavily-loaded and viceversa.

6. Scalability:
A scheduling algorithm should be capable of handling small as well as large networks.

7. Fault Tolerance:
Should be capable of working after the crash of one or more nodes of the system.

8. Fairness of service:
More users initiating equivalent process expect to receive the same quality of service.

(Q3) Draw and explain models of code migration.



Models of Migration

Weak mobility: transfer only code segment along with perhaps some initialization data.

Strong mobility: a running process can be stopped subsequently moved to another machine and then resume execution where it left off.

Sender initiated migration: It is initiated at the machine where the code currently resides or is being executed

Receiver initiated migration: Initiation is taken by the target machine.

Cloned process: It is executed in parallel to original process

Q4) Explain process to resource and resource to machine binding

→

Resource to machine binding

process to resource binding		Unattached MV (or GR)	Fastened GR (or MV)	Fixed GR
	By identifier			
	By value	CP (or MV, GR)	GR (or CP)	GR
	By Type	RB (or MV (P))	RB (or GR (P))	RB (or GR)

GR- establish a global system wide reference
MV- move to resource
CP- copy the value of the resource
RB- rebind process to locally available resource.

1. Binding by identifier: is when a process uses a URL to refer to a specific web side or where it refers to an FTP server by means of that server's address

2. Binding by type: is exemplified by reference to local devices such as monitors, printers & so on.

3. Binding by value: is when a program relies on standard libraries

4. unattached resources: can be easily moved between different machine and are typically files associated only with the program that is migrated

5. Fastened resources: may be possible to transfer but only at relatively high costs.

6. Fixed resources: intimately bound to a specific machine or environment and cannot be moved.

Q5) Explain process migration in heterogenous environment.

→ 1. Distributed systems are connected on a heterogenous collection of platforms each having their own operating and machine architecture.

2. migration in such systems requires that each platform is supported that is, that the code segment can be executed on each platform, perhaps after recompiling the original source.

Process:

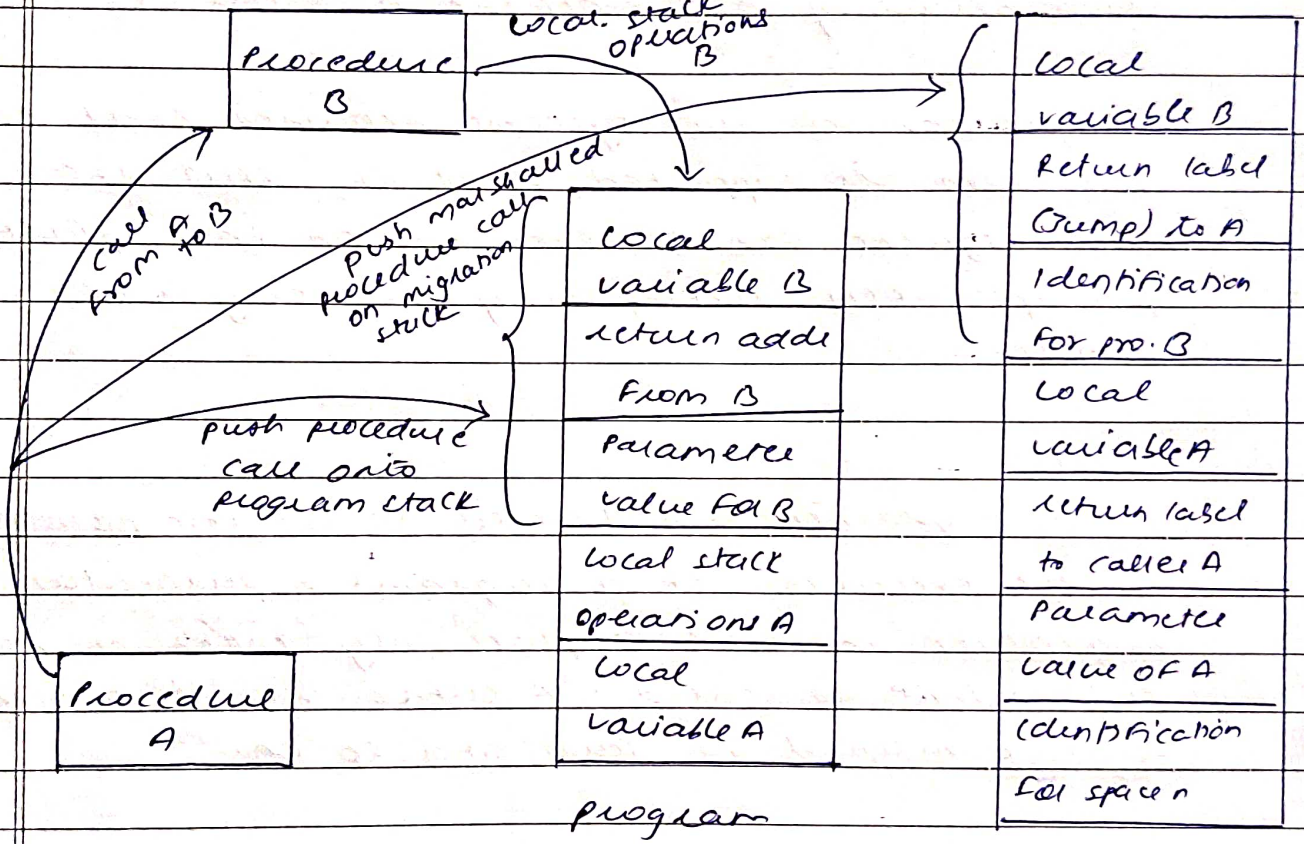
a) Code migration is restricted to specific points in the execution of a program. In particular, migration can be take place only when a next subroutine is called. A subroutine is a function in C, a method in Java and so on.

b) The runtime system maintains its own copy of the program stack, but in a machine independent way. The migration stack is updated when a subroutine is called or when execution returns from a subroutine.

c) When a subroutine is called the runtime system marshalls the data that have been pushed into

stack since the call.

The marshalled data are then pushed on to the migration stack along with an identifier for the called subroutine



e) If code migration takes place at the point where a subroutine is called the subroutine system marshalls all global program specific data forming part of the execution segment.

f) machine specific data are ignored as well as the current stack. marshalled data are transferred to destination along with migration stack.

g) marshalled data belonging to execution segment are unmarshalled and new runtime stack is constructed by unmarshalling the migration stack. Execution can then be resumed by simply entering the subroutine that was called at the original site.

Q6) List Types of virtualization

→ virtualization can take place in 2 ways

1. Process virtual machine:

a) we can build a runtime system that essentially provides an abstract instruction set that is to be used for executing application

b) Instructions can be
- interpreted
- emulated as is done for running windows application on UNIX platforms

c) This type of virtualization leads to call a process virtual machine stressing that virtualization is done essentially only for a single process

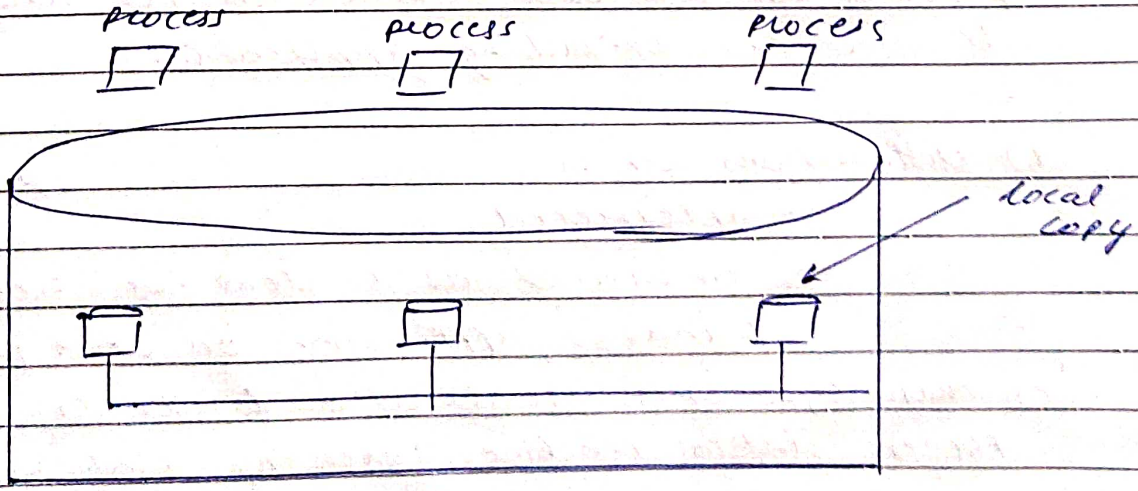
2) A Native virtual machine report:

a) To provide a system that is essentially implemented as a layer completely shielding the original hardware but now offering the complete instruction set of the same (or other hardware) as interface.

b) It is now possible to have multiple and different OS run independently and concurrently on the same platform.

Q7) Explain data centric consistency model:

→ A data stores may be physically distributed across multiple machines. Each process that can access data from the store is assumed to have a local nearby copy available of entire stores.



a) Strict consistency

Defn: Any read on a data item x returns value corresponding to the result of the most recent on x

- Existence of absolute global time
- Available in uni-processor system

P1: W(x) a

P2: R(x) a

b) Sequential consistency: The result of any execution is same as if operation by all process on the data store operates of each individual process appear in this sequence

P1:	W(x)a			P1:	W(x)a		
P2:		W(x)b		P2:		W(x)b	
P3:		R(x)b	R(x)a	P3:		R(x)b	R(x)a
P4:		R(x)b	R(x)a	P4:		R(x)a	R(x)b
	Sequential				Non-sequential		

c) Linearizability: Its also sequentially consistent but is more expensive to implement

In addition if TS(x) < TS(y) then OP1(x) should proceed OP2(y) in this sequence

d) Casual consistency : It provides a total order of casually related write operations only

- A read is casually related to write that provided the data the read got
- A write is casually related to a read that happened before this write in the same process
- if write1 → read and read → write2
  then write1 → write2

P1: W(x)a		P1: W(x)a
P2:	R(x)a W(x)b	P2: W(x)b
P3:	R(x)b R(x)a	P3: R(x)b R(x)a
P4:	R(x)a R(x)b	P4: R(x)a R(x)b

e) FIFO consistency :
- Write done by
a) single process are seen by all other process in order in which they were issued different order by different process.

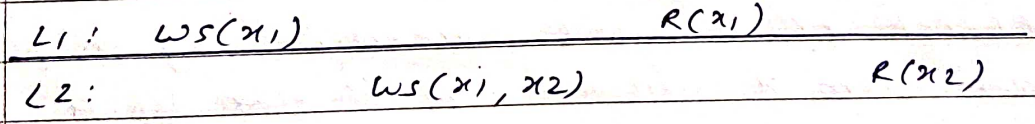
f) weak consistency :
- Enforcing consistency on a group of memory reference operations rather than individual operations
- When a process access a synchronization variable, the entire memory is synchronized by making visible changes made to the memory to all processes.

Q8) Explain client centric consistency model?

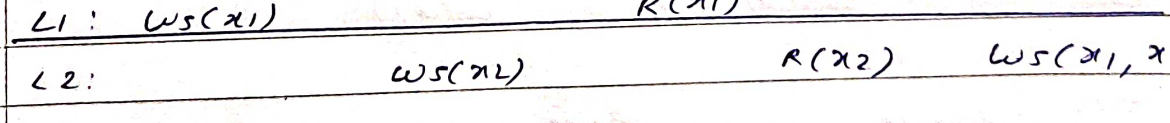
→ Goal: show how we can perhaps avoid system wide consistency by concentrating on what specific client want instead of what should be maintained by servers

a) Eventual consistency: In system that tolerate high degree of consistency, if no update take place for long time all replicas will gradually and eventually become consistent

b) Monotonic Reads:
If process reads the value of a data item x, any successive read operation on x by that process will always return that same or more recent values.



monotonic-read consistent datastore



Data store that doesn't provide monotonic reads

c) Monotonic writes : If a write operation by a process on a data item x is completed before any successive write operation on x by same process

L1:	W(x1)		L1:	W(x1)	
L2:		W(x2) W(x2)	L2:		W(x2)

Monotonic write consistent datastore — data store that does not provide monotonic write consistency

d) Read your writes: The effect of a write operation by a process on data item will always be seen by a successive read operation on x by same process.

L1:	W(x1)		L1:	W(x1)	
L2:		W(x1, x2) R(x2)	L2:		W(x2) R(x2)

e) Writes Follow read:
The effect of a write operation by a process on data item x will always be seen by a successive read operation on x by the same process

L1:	WS(x1)			R(x1)	
L2:		WS(x1, x2)			W(x2)

Write follows read consistency.