

NAME : Muhammad Yonis Afendi

I.D : 12701

Subject : operating system.

(1)

(Q) In deadlock prevention strategy
do you think it is necessary to
check that either safe state exists
or not? Give reason to support
your answer?

Ans: "Deadlock Prevention".

→ We can design a system to
avoid deadlock by making any of the
4 conditions impossible.

Breaking mutual exclusion.

In some cases, deadlock can be
mitigated by making resource more
shareable. For example, using a reader/
writer lock instead of a mutex.

Can make deadlock less likely (since
many readers can share the read locks)
Using a lock-free data structure is
another way to allow multiple threads
to access a data structure simultaneously
(without blocking).

However, many resources are inherently
non-shareable (e.g. printer: can't print
two documents simultaneously). Mutual
exclusion is a good condition to
break if you can, but often you can't.

(3) Breaks "Breaks No-preemption"
In some situations we can make
resources preemptable. If a process
tries to acquire a resource that
is held by another process, we
can make it possible for the
new process to steal the
resource.

(2)

In order to do this, we need some mechanism for rollback. We need to be able to restore whatever program invariants that the resources was held in order to satisfy.

For example: if the resource is a lock protecting a shared variable we could roll back the thread that holds the lock by restoring the state of the shared variable to the state it held before the lock was acquired, and restarting the processes that was performing the update.

once we allow computations to be rolled back, we introduce the possibility that two threads can continue to preempt each other forever. Although the system is not deadlocked (both threads seem to be making forward progress) the system may never actually finish its tasks if competing threads are continuously being rolled back before they can finish.

It is not possible to make all resource preemptible. It is a well-known impediment to rollback, once some output has been performed it may be impossible to make all

(3)

all resource preemptible t/o
is a known impediment to
roll back. once some output has
been performed, it may be
impossible to return to a
consistent state. once you tell the
user you've started processing
their order you can't take it back.

(1) Break hold-and-wait.
can break hold-and-wait by
having threads release all locks
and re-acquire them all one
at once.

Releasing locks may require rollbacks
which leads to the same issue
described above.

monitors partially use this strategy
to avoid-and-wait calling wait &
condition variable automatically. release
the locks, so that acquiring the
monitors locks cannot cause dead
locks. However, it is still possible
to create a form of deadlock
with a monitor, where one
thread needs bot some before
operating. state is a way satisfied
another predicate while a second
thread waits bot the second
predicate before making the
first predicate true.

(4)

Q2 Differentiate between Dynamic Loading and dynamic linking with the help of examples?

Ans: Dynamic Loading means loading the library (or any other binary for that matter) into the memory during load or run-time. Dynamic loading can be imagined to be similar to plugins, that is an exe can actually execute before the dynamic loading happens (The dynamic loading for load library call in C or C++).

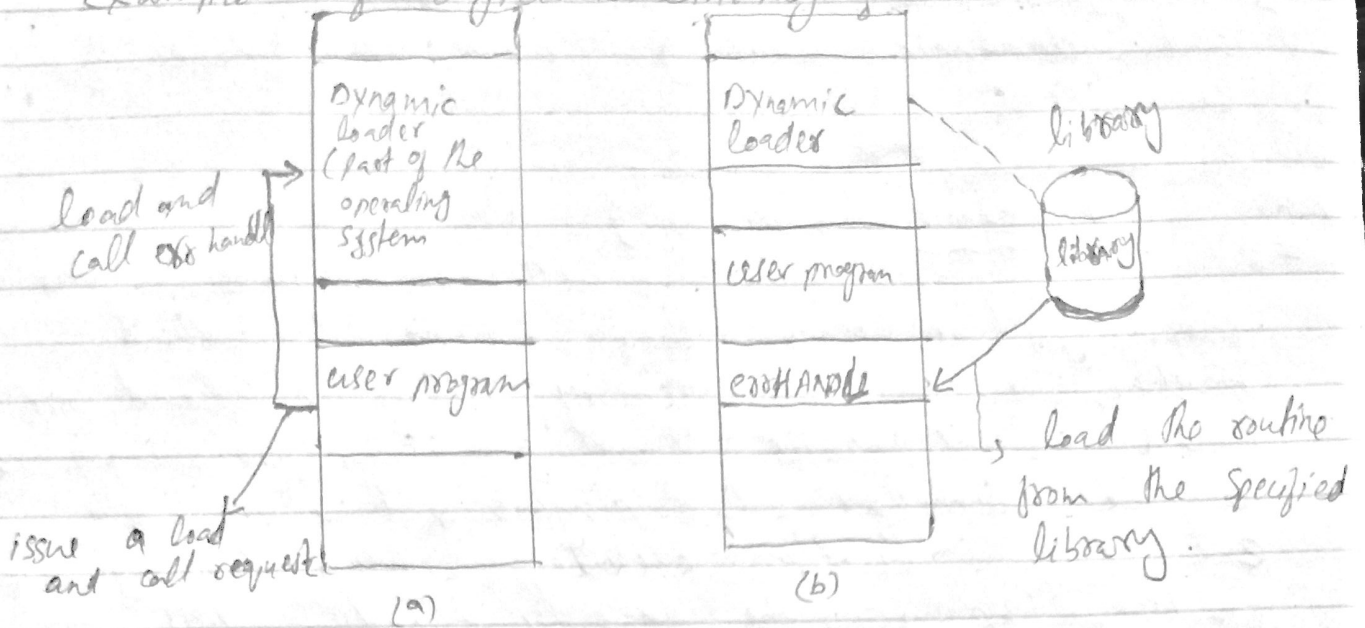
Example: Dynamic loading has found its main use as a basis for plugin systems for applications like notepad++ for examples, which loads plugins from all files in a certain directory in its folder structure.

Dynamic linking refers to the linking that is done during load or run-time and not when the exe is created. In case of dynamic linking the linker while creating the exe does minimal work. For the dynamic linker to work it actually has to load the libraries.

P.T.O

(5)

Examples of dynamic linking:



(6)

Q3 Which component of an operating system is best suited to ensure fair, secure, orderly and efficient use of memory? also identify some more tasks managed by this component.

Ans The kernel is a computer program at the core of a computer operating system with complete control over everything in the system. It is an integral part of any operating system. It is the portion of the operating system code that is always resident in memory.

When the OS allows you to perform more than one task at a time, it is multitasking programs called device drivers facilitate communication between devices attached to the computer and the OS.



(7)

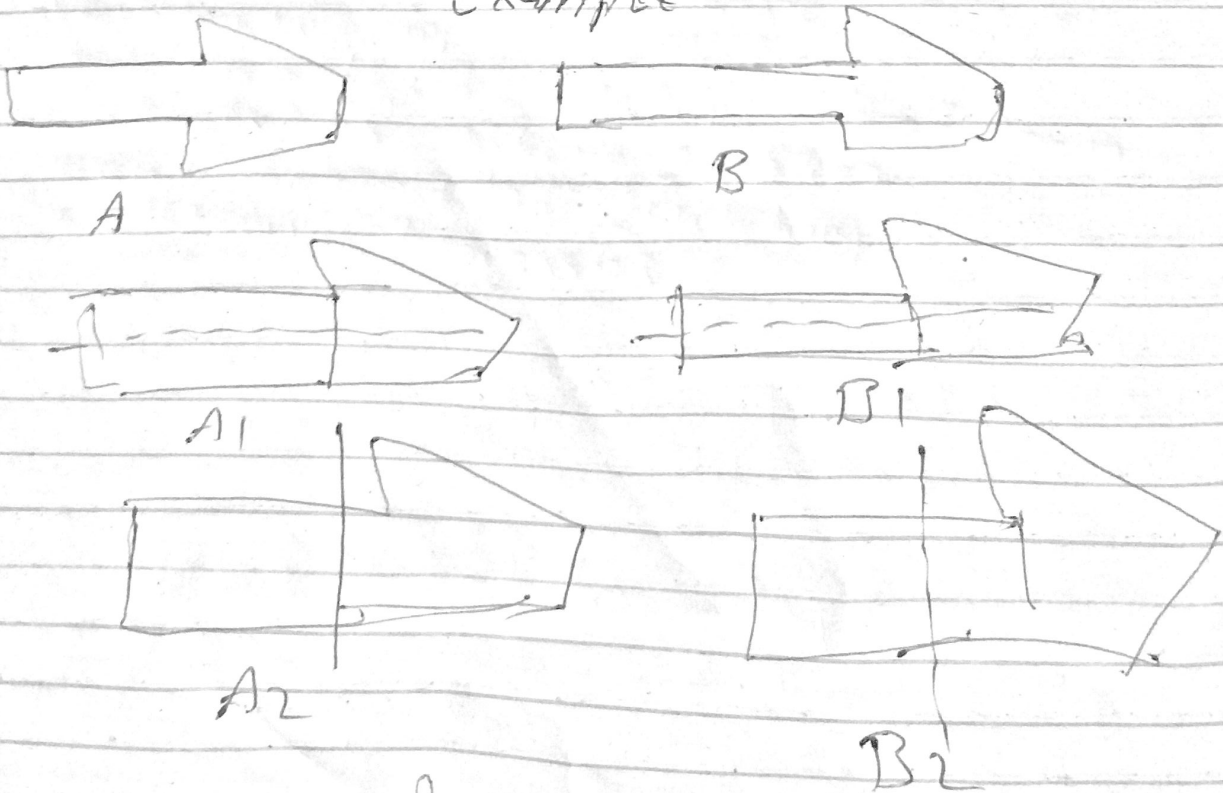
Q1)

Q1) Differentiate between symmetric and Asymmetric encryption with the help of example?

Ans, "Symmetric"

Encryption is a process to change the form of any message in order to protect it from revealing by anyone. In symmetric-key encryption the message is encrypted by using a key and the same encryption the message is encrypted by using a key and the same key is used to decrypt the message which makes it easy to use but less secure. It also requires a safe method to transfer the key from one party to another.

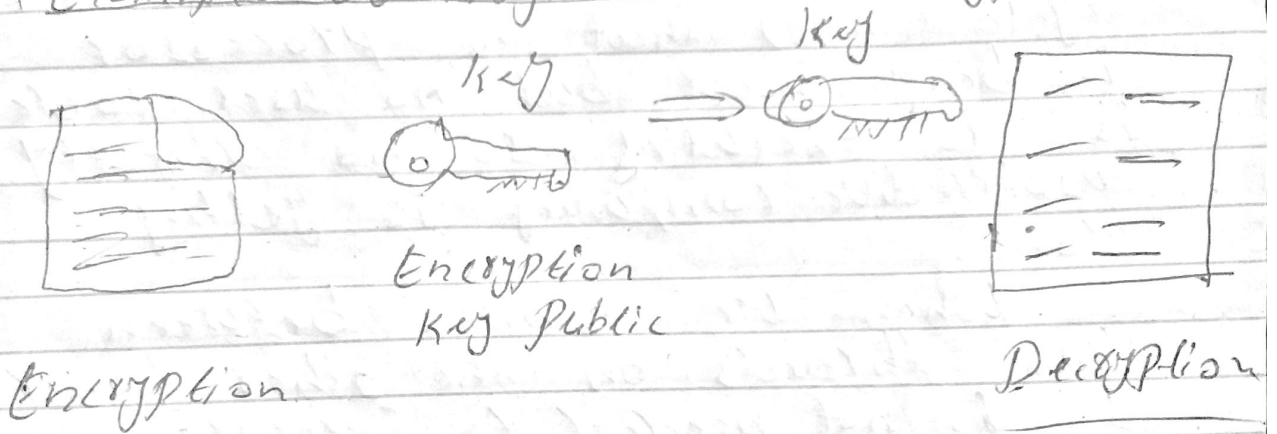
"Example"



Asymmetric Key Encryption.

Asymmetric key encryption is based on public and private key encryption technique. It uses two different keys to encrypt and ~~also~~ decrypt the message. It is more secure than symmetric key encryption technique but is much slower.

Example of Asymmetric Encryption

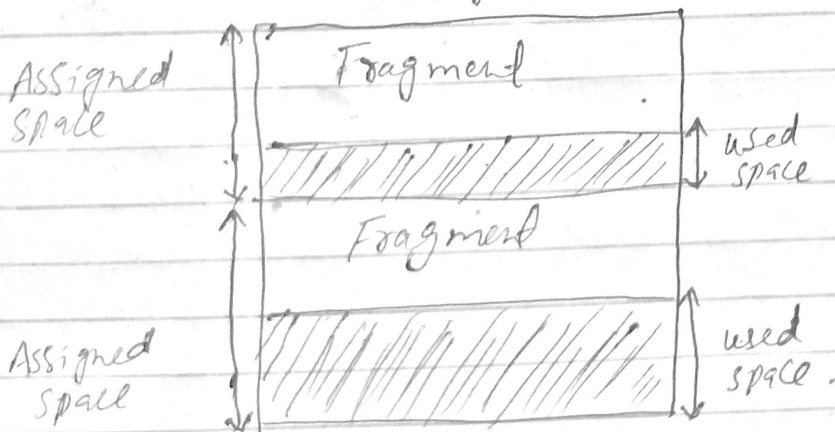


(9)

Q5 Describe the differentiate between external and internal fragmentation, why should they be avoid.

Ans Internal Fragmentation :-

Internal fragmentation happens when the memory is split into mounted sized blocks. whenever a method request for the memory, the mounted sized block is allotted to the method. Just in case the memory ~~also~~ allotted to the method. Just in case the memory allotted to the method is somewhat larger than the memory requested, then the distinction between allotted and requested memory is that the internal fragmentation.

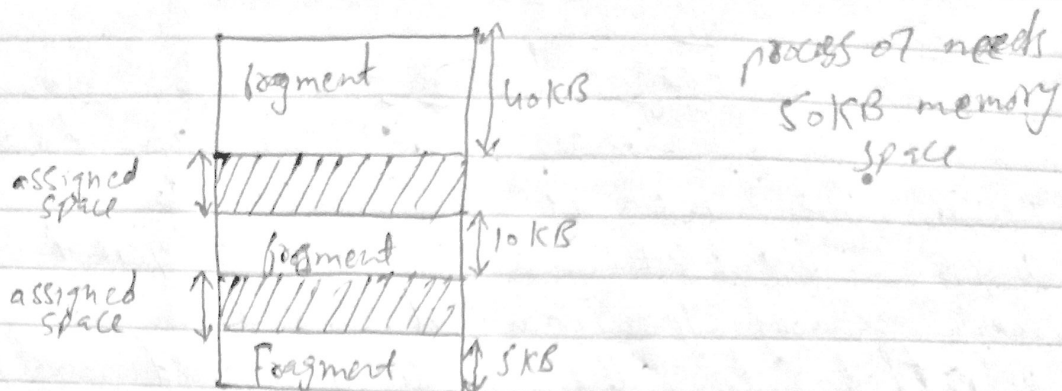


External fragmentation :-

External fragmentation happens when there's a sufficient quantity of area within the memory of satisfy the memory request of a method. however the process's memory request cannot be fulfilled because the memory offered is during non-contiguous manner. either you apply first-fit or best-fit memory

(10)

allocation strategy it'll cause external fragmentation.



However, both the phenomena can be avoided, Internal fragmentation can be reduced by allocating memory to process dynamically, whereas external fragmentation can be best avoid be compaction, paging and segmentation.



Q6 List and describe the four memory allocation algorithms covered in lectures, which two of the four are more commonly used in practice?

Ans The four memory allocation algorithms (in the scheme of dynamic partitioning placement) are.

First-fit is the linked list of available in the memory addresses, we place the data in the first entry that will fit its data. It's aim to minimise the amount of searching, but leads to external fragmentation later on.

Next-Fit Similar to first-fit, but instead of searching from the ~~beginning~~ beginning of each block, it searches from the last successful allocation. greatly reduces the amount of searching but leaves external fragmentation at the beginning of memory.

Worst-fit : traverses the memory and gives the partitions as large space as possible to leave usable fragments left over, needs to search the complete list and such is a poor performer.

Best-fit Carefully scans the memory for spaces that perfectly fit the RAM we want. However, the search is likely to take a very long time. we must commonly use first-fit and next-fit in practice.

(12)

Q7 Why is the context switch overhead of a user-level threading as compared to the overhead for processes? Explain.

Ans7 Context-switch time is pure overhead because the system does no useful work while switching --- context switching is overhead because it is cycle (time) that the processor is being used but no user code is executing, so no directly productive computing is getting done.

When we switch between two threads, on the other hand it is not needed to invalidate the TLB because all threads share the same address space and thus have the same contents in the cache. Thus the cost of switching between threads is much smaller than the cost of switching between processes.