Internals - 02

OS (18CS43)

Shahul Hameed . S
1KN 18CS097
CSE 'A' Sec

3a. when processor request a source . and if the resources are not available at the time the process enters into waiting State. waiting process may not change its state because the resources they are requested are held by other process. This is called deadlock :

Conditions :-

Mutual exclusion :- only one process must hold the resource at a time. If any other process request for the resources, the requesting process must be delayed until the resource has been released.

Hold and wait :- A process must be holding at least one resource and waiting to acquire additional ~~req~~ resources that are currently being held by the other process.

No preemption :- Resources can't be preempted i.e., only the process holding the resources must release it after the process has completed its task.

Circular wait :- A set [p0, p1... pn] of waiting process must exist such that p0 is waiting for a resources i.e held by p1 p2 is waiting for a resources i.e held by p2, pn-1 is waiting for resources held by process pn. pn is waiting for resources ~~held~~ held by p1.

Shahul Hameed S

3b.

	Allocation	Max	Available
P_0	0 1 0	7 5 3	3 3 2
P_1	2 0 0	3 2 2	
P_2	3 0 2	9 0 2	
P_3	2 1 1	2 2 2	
P_4	0 0 2	4 3 3	

need = Max - allocation.

Need matrix,

	Need		
	A	B	C
P_0	7	4	3
P_1	1	2	2
P_2	5	0	0
P_3	0	1	1
P_4	4	3	1

Applying the safety algorithm on the given system,

Step 1: Initialization.

work = 332

	P_0	P_1	P_2	P_3	P_4
finish :	false	false	false	false	false

Step 2: for i = 0

finish[P_i] = false and need [P_i] <= work (743) <= (332) ~~true~~ false

So P_0 must wait.

Step 2: for i = 1

finish P[1] = false and need [P_1] <= work (122) < (332) True

Step 3: work = work + allocation [P_1] = (332) + (200) = (532) false

	P_0	P_1	P_2	P_3	P_4
finish =	false	true	false	false	false

Step 2: for i = 2

finish [P_2] = false and need [P_2] <= work (600) <= (532) false

So P_2 must wait.

Step 2:- for i = 3

finish [P3] = false & need [P3] <= work (011) <= (533)

true

So P3 must be kept in safe sequence.

Step 3:- work = work + Allocation [P3] = (532) + (211) = (743)

	P0	P1	P2	P3	P4
finish =	False	true	false	true	false

Step 2:- for i = 4

finish [P4] = false & need [P4] <= work (431) <= (743)

true

So P4 must be kept in safe place.

Step 3:- work = work + Allocation [P4] = (743) + (002) = (745)

	P0	P1	P2	P3	P4
finish =	false	true	false	true	true

Step 2:- for i = 0

finish P[0] = false and need [P0] <= work (743) <= (745)

true

So P0 must be kept in safe place.

Step 3:- work = work + allocation [P0] = (745) + (010) = (755)

finish =	true	true	false	true	true

Step 2:- for i = 2

finish [P2] = false & need [P2] <= work (600) <= (745)

true

So P2 must be kept in safe place.

Step 3:- work = work + allocation [P2] = (755) + (302)

= (1057)

	P0	P1	P2	P3	P4
finish =	true	true	true	true	true

Shahul Hameed .S

Step 4: finish [Pi] : true for 0 <= i < n

Hence, the system is currently in a safe place.

The safe sequence is < P_1, P_3, P_4, P_0, P_2 >

Conclusion: yes the system is currently in safe place.

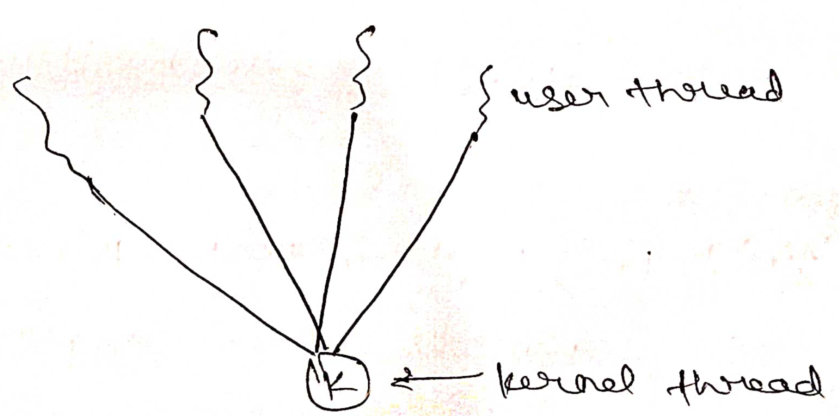
1a. A thread is a basic unit of CPU utilization. It consists of thread ID, program counter, a stack & a set of registers.

Multiple threading Models:

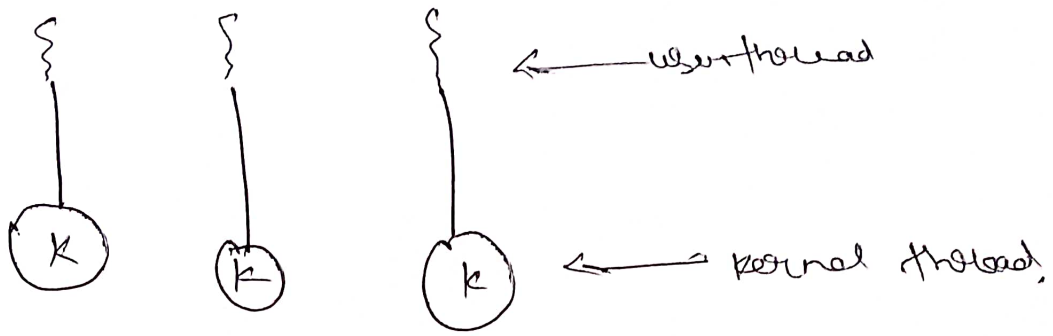
Many to One Model

Here, Many user level threads are mapped to a single kernel thread. thread management handled by thread library in user space, which is very efficient.

Only one thread user can access the kernel at a time, as there is only one kernel thread. thus the threads are unable to run in the parallel on multiprocess.



One to one model :- It creates a separate kernel threads handle each user thread. It overcomes problems like blocking system calls and the splitting of process across multiple CPU's , the overhead of managing the one-to-one model is significant, involving more overhead & slowing down the system.

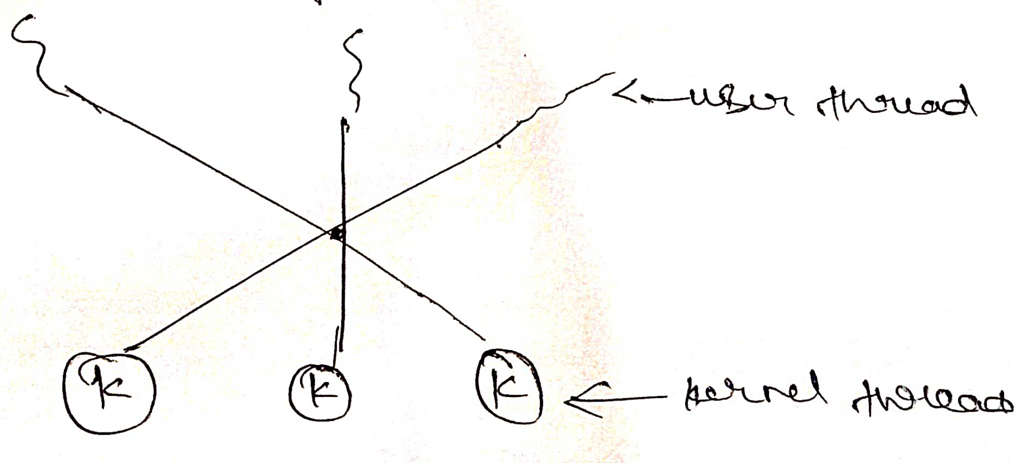


one to one

Many to many thread :-

It multiplexes any number of user threads onto an equal ~~and~~ or smaller number of kernel threads.

- user have no restrictions on number of threads created
- Blocking kernel system calls do not block entire process
- processes can be split across multiple processors.
- This is also called as two tier model.
- Supported by OS such as IRIX, UNIX, HP-UX,



many to many