20K-1901 Abdullah Imtiaz BSE-4B

## Q8.12(a)

1) Mutual Exclusion: One vehicle at a time can use the intersection point if another car request that resource the requesting vehicle must be delayed

2) Hold & wait: vehicle holding one intersection is waiting to acquire additional intersection point held by other vehicles

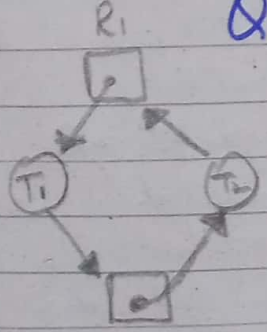
3) Non Preemption: An intersection point can only be released after a vehicle moves forward

4) Circular Wait: There exist a circular chain each intersection required to specific vehicle is held by another vehicles & so on.

## Q8.12(b)

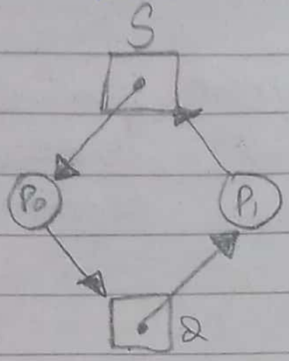
Install traffic lights to control the traffic & make sure that at a time only one ~~visible~~ vehicle cross the intersection point.

## Q8.13




- $T_1$ is holding resource $R_1$
- $T_2$ is holding resource $R_2$
- $T_1$ is waiting for $T_2$ to release $R_2$
- $T_2$ is waiting for $T_1$ to release $R_1$

## Q8.14




## Q8.15

To determine if deadlock is possible when using multiple reader-writer locks we can apply the four necessary conditions for deadlock:

Mutual Exclusion: This condition holds for reader writer locks as they allow multiple readers or a single writer to hold the lock

Hold & Wait: This condition can potentially hold for the reader-writer locks if a thread holds a reader lock & later requests a writer lock. The thread will have to wait for all other readers to release the lock before it can acquire the writer lock

Circular Wait: This condition can potentially hold for multiple reader locks if a cycle of threads exists where each thread holds one reader lock & is waiting for a different reader lock held by the next thread in the cycle. However this scenario is highly unlikely as reader-writer locks are typically used to allow concurrent access to resources.

No Preemption: This condition holds for reader writer locks as a lock can be released voluntarily by the thread holding it.

In conclusion, it is possible for deadlock to occur with multiple reader writer locks if the hold & wait condition is violated & a thread requests a writer lock while holding a reader lock. However circular wait scenarios are unlikely to occur with reader writer locks.

## Q8.16

Deadlock can be avoided if CPU Scheduler will either schedule thread 1 first or thread 2 to acquire locks & after releasing lock allowing the other threads to occupy. However deadlock can occur in this if both threads at a time tries to acquire lock.

## Q8.17

```
void Transaction(Account from, Account to, double amount)
{ mutex lock1, lock2;
  if (from.getaccountnumber() < to.getaccountnumber())
    { lock1 = from.getlock;
      lock2 = to.getlock; }
  else { lock1 = to.getlock;
         lock2 = from.getlock; }
  acquire(lock1); acquire(lock2);
  withdraw(from.amount);
  deposit(to.amount);
  release(lock2); release(lock1);
}
```

# Q8.18

a) Deadlock will not occur:
- $T_1$ & $T_3$ held $R_1$, either $T_2$ or $T_3$ will release $R_1$ which is then allocated to $T_1$
- $T_2$ & $T_3$ held $R_2$ either $T_2$ or $T_3$ will release $R_2$ which is then allocated to $T_1$

e) Deadlock will not occur:
$T_2$ & $T_4$ holding $R_2$ either $T_2$ or $T_4$ will release $R_2$ then $T_1$ will use $R_2$ which will break the cycle resulting in no deadlock

f) Deadlock will not occur:
$T_2$ & $T_4$ are holding $R_2$ either $T_2$ or $T_4$ release $R_2$ which is then used by $T_1$ which will break the cycle resulting in no deadlock

c) Deadlock ~~occur~~ will not occur:
- $T_2$ & $T_3$ held $R_1$ either $T_2$ or $T_3$ release $R_1$ which is then allocated to $T_1$
- $T_2$ & $T_3$ held $R_2$ either $T_2$ & $T_3$ release $R_2$ which is then allocated to $T_1$

b) Deadlock occur: $T_1 - R_3 - T_3 - R_1 - T_1$ (cycle)

d) Deadlock occur:
- $T_1 - R_2 - T_3 - R_1 - T_1$
- $T_1 - R_2 - T_4 - R_1 - T_1$
- $T_2 - R_2 - T_4 - R_1 - T_2$
- $T_2 - R_2 - T_3 - R_1 - T_2$

## Q8.19

* Runtime Overhead:

• Circular wait scheme: it incurs higher runtime overhead since it involves detecting and breaking circular wait conditions by aborting or rolling back some of the process

• Deadlock avoidance scheme: These schemes also have some runtime overhead, but they aim to prevent deadlocks by ensuring that resources are allocated safely.

* System throughput:

• Circular wait Scheme: It can lead to reduced system throughput since aborting or rolling back processes can waste resources & increase processing time

• Deadlock avoidance scheme: They maximize system throughput by avoiding deadlocks while ensuring that resources are allocated safely.

## Q9.12

To achieve this linker combines multiple object modules into a single program binary & assign memory address to instructions & data contained in each module.

Relocation: It involves modifying the object modules code & data sections

Symbol resolution: Includes resolving references b/w object module by updating address of the called function.

To facilitate the memory binding tasks of the linker the compiler must pass information about external references & global symbols to linker.

## Q9.16

On a paging system, a process cannot access memory that it doesn't own because it is protected by the OS. The OS can allow access to additional memory by allocating new memory pages to the process but this should only be done if its necessary & justified. Allowing access to additional memory can pose security risks & appropriate access controls & security measures should be in place to protect the system and its users.

## Q9.17

Mobile Operating systems like iOS & Android do not support swapping because it can negatively impact the performance & battery life. Swapping is slow & mobile devices have limited storage and slower storage technologies which are not well-suited for swapping. Instead, mobile OS use other memory management techniques like memory compression & aggresive memory management to maximize available memory & minimize the need for swapping.

## Q10.15

* TLB miss with no page fault: The page is in physical memory but removed from TLB.
* TLB miss with page fault: The page is not in physical memory also it is not in TLB.
* TLB hit with no page fault: The page is in physical memory as well as in TLB
* TLB hit with page fault: This scenario is not possible because if TLB hit it means that the physical address of memory location is already known & there is no need for page fault which means page is not in physical memory & therefore cant be in TLB.

## Q10.16

a) The state will change & become blocked because it will & wait for pages.

b) It will not change state. It executes instructions as normal using the updated mapping provided by TLB.

c) No, thread ~~state~~ will not change state thread state is not affected by page table lookup process

# Q10.17

a) When process first starts execution, the page fault rate is typically high. This is because the process needs to access large number of pages that are not currently in physical memory.

b) When loaded into memory, the page fault will decrease because it's in memory.

c) Designer may have few applications
1) Increasing the size of physical memory
2) Using page replacement algorithm
3) Implement a priority scheme
4) Use a multi-level page table.

# Q10.18

0x2A1 → 0xAA1
0x4E6 → 0x9E6
0x94A → 0x14A
0x316 → 0xF16

## Q10.19

Copy on write is a memory management technique where multiple processes can share the same memory pages until a process attempts to write a page. It defers copying until its necessary, allowing for memory to be shared b/w processes without unnecessary copying overhead. It is useful when multiple processes needs to access the same data but only a few process needs to modify it. Hardware support is not required but some hardware architecture provide support through memory management unit capabilities such as page protection bits & page fault handling.

## Q10.20

When a user process generates a virtual address the system uses software & hardware operations to establish the corresponding physical location. The virtual address is divided into a page number & an offset within the page. The page number is used to index the page table, which is stored in memory to retrieve the corresponding page frame number. The page number is combined with the offset to form the physical address. Hardware operations include memory management unit to perform

address translation & access physical memory. Software operations include managing the page table & performing page replacement algorithms when necessary.

## Q10.21

EAT = (1-P) x mAt + P(hit memory time)

200 = (1-P) x 100 + P x (8x10^6 x 0.3 + 20x10^6 + 0.7 + 100)

P = 0.186 = 18.6 %

Maximum acceptable page fault rate for an Eat is $6.1 \times 10^{-6}$