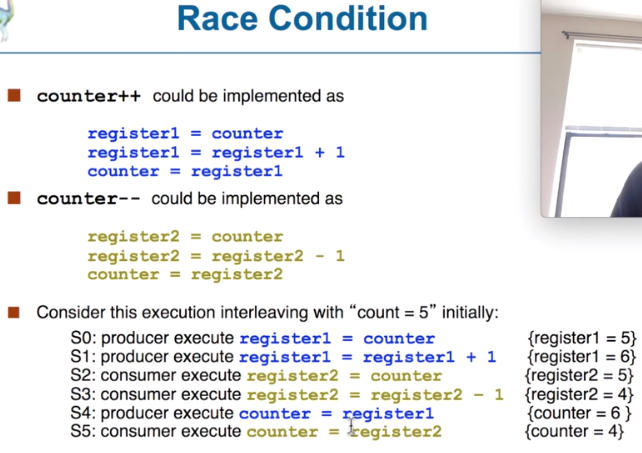
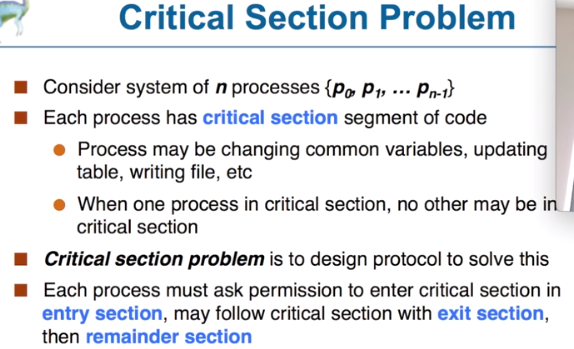


Mesela dosya erisimini ele alalim. Ayni anda iki process ayni dosyaya erismeye calisirsa tabiki burada bir sorun cikacaktir. Mesela burda bir yaris vardir. Hangi processin once erisecegini bilemeyiz. (**Race condition**)



Kodu yazan kisi birden fazla processin, threadin falan calistigi durumlarda risk olusturacak yerleri bilmesi gerek. Kodu yazan kisi bilmezse kimse bilmez. Dolayisiyla kodu yazan kisi bu risklere onlem almak zorundadir. (**Critical Section**)



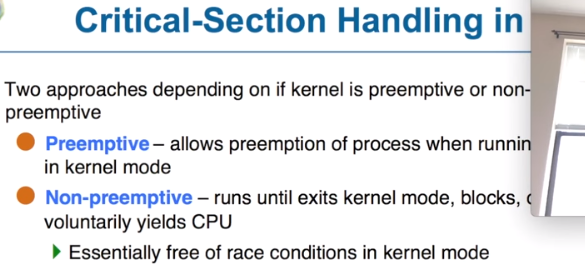
**SOLUTION to CRITICAL-SECTION PROBLEM**

**1 - Mutual Exclusion (Karsilikli Dislama)** – Ben bir degiskeni artirirken oteki gidip azaltirsa ben sonuc olarak bekledigim degeri yaptigim seyi alamam ve sonucu hatali alirim yani mesela. Bu yuzden bunu engellemek icin mutual exclusion vardir yani bir process calisiyorken oteki calismayacaktir.

**2 – Progress** – Bu durumda ise sirayla calisacaklar yani atiyorum bir uretici calisacak bir tuketici sonra tekrardan bir uretici calisacak ve tekrardan tuketici boyle gidecek …

**3 – Bounded Waiting** – Bu durumda ise bazi esneklikler tanimlarsiniz yani mesela iste producer calistiktan sonra consumer 10 saniye calismasa da olur hani bir sikinti olmaz gibi esnekliklerle cozulur durum.

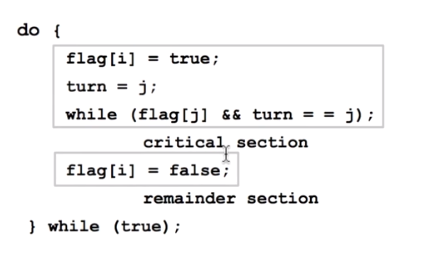
Mesela iki process calisiyor tamam ikisi de ayri calisiyor yani mutual exclusion saglaniyor ve ardindan bunlari sirasi da dogru yani atiyorum once producer calisiyor daha sonra consumer calisiyor son olarak ise iste atiyorum birden fazla kez producer calismasina izin veriyoruz falan bu da bounded waiting oluyor.



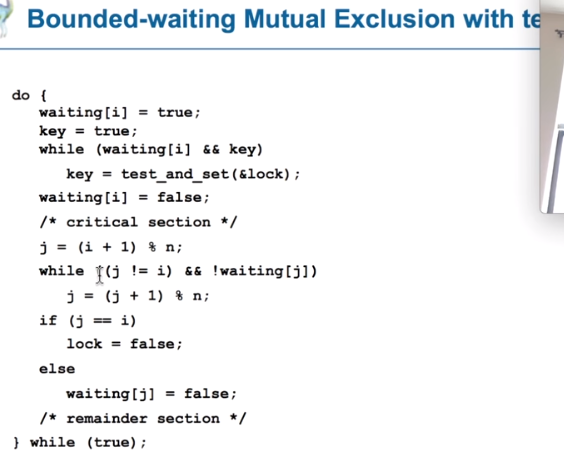
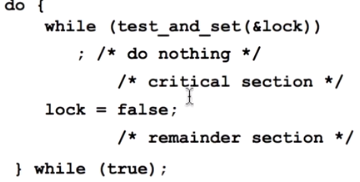
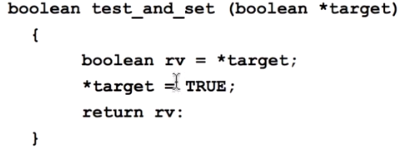
Preemptive = CPU birden fazla is yapiyor.

Non-preemptive = CPU yalnizca bir is icin ayriliyor ve o is bitmeden baska bisey yapmiyor.

**PETERSON SOLUTION (Gercek cozum degil)**



Bazi donanimlar bize direct senkranizasyon ozellik saglanabilir. Yani **Atomic** donanim sagliyor bize. Yani CPU o isi yaparken digerlerini kitliyor ve CPU’ya baskasi giremiyor. O is bittikten sonra kilit tekrar aciliyor.

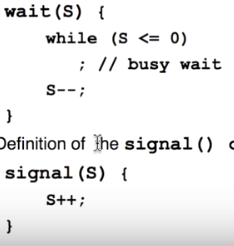


Eger bir lock CPU’yu surekli isgal ediyorsa buna **busy waiting** ismi veriliyor. Ve bu lock’a da **spinlock** denir. Dolayisiyla boyle lock’lari istemiyoruz.

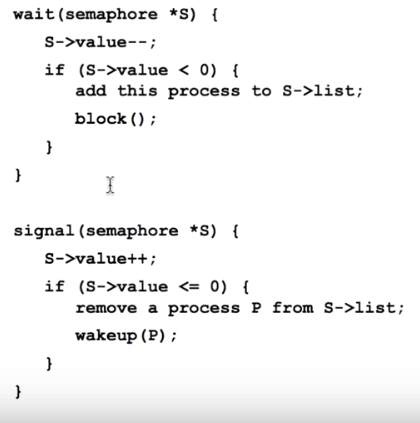
SEMAPHORE BURDA ANLATILIYOR.

**Semaphore (Isaret vermek gibi bir anlami var)**

Counting semaphore (bir arttiran bir azaltan asagidaki uygulamamiz) and binary semaphore (mutex lock’larla aynidir baska kimsenin calismamasini sagliyor)



S’in ilk degeri ilk asamada ayni anda kac processin calisacagini soyler. 3 verirseniz 3 process birden girip calismaya baslar ve artik kim once bitirirse isini S—ile S’in degerini bir azaltir ve yeni bir process iceri girer. Icerde hep 3 process olur diyebiliriz.



**Deadlock and Starvation**

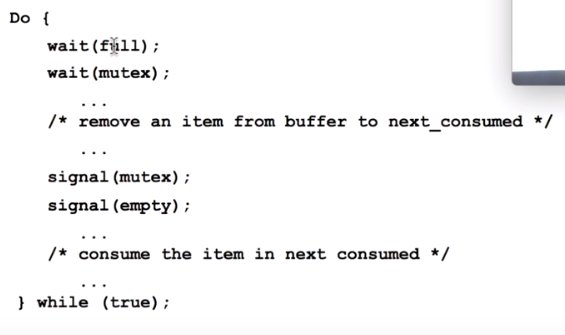
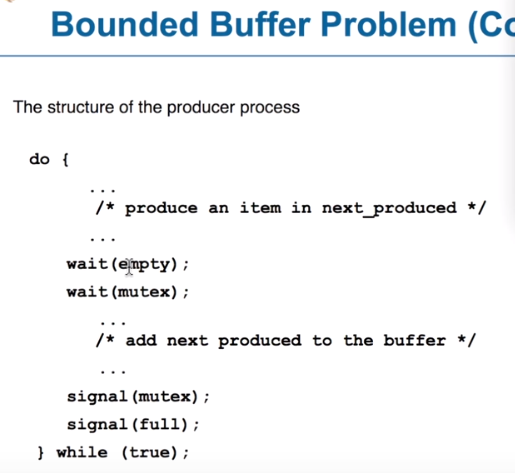
Iki process de birbirinin islem yapacagini zannedip birbirlerini bekliyorlar ve hicbir process calismiyor. Sonsuza kadar bu boyle kalir iste bu **deadlock**’tir.



P0 calisti ve wait(S) yapti tam o sirada P1 calisti ve wait(Q) calisti. Iste bu durumda hem S kilitli hem Q kilitli bu durumda ne P0 devam edebilecek ne de P1 hareket edebilecek. (**Deadlock**)

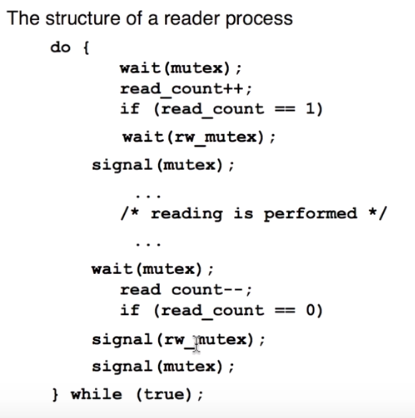
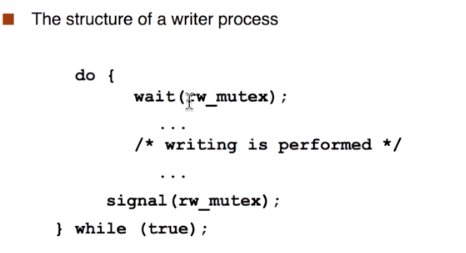
SImdi islemlerin oncelik sirasina gore gerceklestirildigi durumlarda mesela atiyorum orta oncelikli biri sirada sirasinin gelmesini beklerken surekli CPU’ya yuksek oncelikli isler gelirse hep onlar one gececegi icin bu orta islem hic calisamayacaktir calismayabilir veya uzun sure bekler falan. Iste bu duruma da **starvation (aclik)** denir.

**Priority Inversion** – Mesela iste dusuk oncelikli bir process bir sekilde calismaya basladi yani sirasi geldi ve calismaya basladi. Calisirken de iste bir dosyaya erismesi gerekti ve ben bu dosyayla calisacam diyip kimse erismesin diye dosyayi kilitledi. Tam o sirada da yuksek oncelikli bir process geldi ve islem onceligine gore bizim processimizin onune gecti. Simdi yuksek onceliklinin isi bitmeden bizimkinin isine gecilemez. Yuksek oncelikli bizim dosyamiza erismeye calissa erisemeyecek bizimkinin kilidi kaldirmasi lazim e yuksek bitmeden de bizimki calisamaz yani kilit kalkamaz boylelikle deadlock olacak ve islemler kitlenecek. Bunlari cozmek programcinin isidir ☺

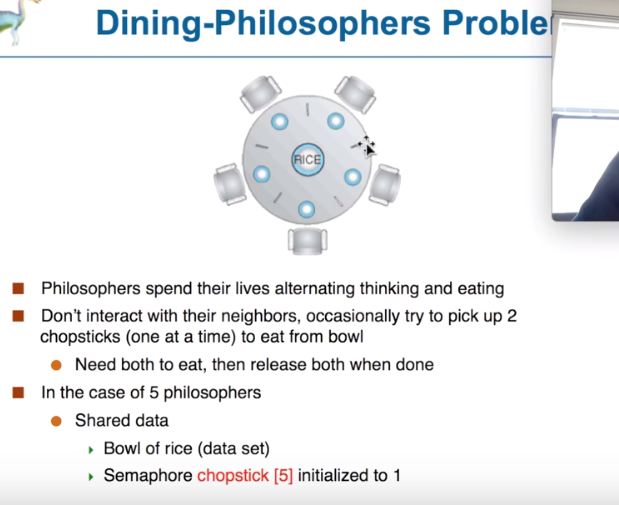


**Readers-Writers Problem**

Mesela bir cok kisi ayni dosyaya eristi ve okumaya ve yazmaya calisti. Simdi bu durumda dosya okumak isteyenlerde sorun yok isterse 1000 lerce kisi acsin okusun dosyayi fakat eger tam o sirada bir yazan varsa iste bu yazanin yaninda ne okuyana ne de baska bir yazana izin vermememiz gerekir aksi takdirde sorunla karsilasiriz. Bir tane writer eristiyse dosyaya baska kimse erisemez.



**Dining-Philosophers Problem**



* Mesela en fazla 4 process’e izin vermek bir cozum. Yani 4 filozofa izin vermek sadece.
* Sagdakine bak, soldakine bak ve iki chopstick’I ayni anda al ve bunu yaparken baska bir isleme (processe) izin verme (Atomic)
* Diger bir cozum iste 0-5 arasi numara ver sonra bi tekleri calistir ciftler dursun sonra ciftleri calistir tekler dursun falan diye ilerler.