İŞLETİM SİSTEMLERİ / ÇALIŞMA SORULARI

1) İşletim Sistemi nedir?
Mevcut **kaynakların ayırıcısı (resource allocator)** olarak tanımlayabiliriz. Belli programlar ve kullanıcı ihtiyacına göre kaynakları ayırma işlemi yapar. Aynı zamanda **bir kontrol programı**dır. I/O aygıtlarını ve kullanıcı programlarını kontrol ederek, hatalı kullanımın önüne geçmeye çalışır. Bilgisayar açıldığı andan itibaren **kapanana kadar çalışan tek program**dır ve buna Çekirdek (Kernel) denir.

2) İşletim Sistemi ne yapar?
Kaynakların verimli kullanımını sağlar. Kullanıcılar, rahatlık ve performans ister ve kaynak kullanımını önemsemezler. Böylece işletim sistemi, bilgisayar kullanıcılarının mutlu olmasını sağlar. Özellikle Workstation (İş İstasyonu) ve server (sunucu) gibi bilgisayarlarda kaynakların ayrımını yapıp verimli kullanımını sağlar. Kaynakları kısıtlı olan sistemlerde, kullanılabilirliği sağlar ve pil ömrünü optimize eder. Küçük arayüz veya arayüzü olmayan gömülü sistemlerin kullanılabilmesini sağlar. İşletim sistemi aynı zamanda bir kesme denetleyicisidir.

3) Kesme (Interrupt) nedir?
İşletim sistemi çalışmaya başladıktan sonra olayların oluşması için beklemeye başlar. Bir işin veya olayın oluşu sisteme "kesme" aracılığıyla bildirilir. **Donanım sistemi,** istediği zaman **sistem veriyolu ile** işlemciye kesme gönderebilir. **Yazılım ise sistem çağrısı ile** kesme gönderebilir**.** İşlemciye kesme sinyali geldiğinde, işlemci o anda yaptığı işi bırakır ve kesme tarafından belirlenen yere yönelir. Belirlenmiş yer genellikle kesmenin servis rutininin başlangıç adresini içerir. İşlemci **kesme servis rutinini gerçekleştirdikten sonra** kesmeden önce yapmakta olduğu yarım kalmış görevine döner ve onu yapmaya devam eder. Kesmelerin önemliliklerine göre aralarında öncelik durumları vardır. **Aynı anda birden çok kesme gelirse; kesmelerin öncelik durumlarına bakılır, yüksek öncelikli kesme önce işleme alınır, diğerleri maskelenir**. Kesmeler, kesilmiş işlemin de adresini kurtarmak zorundadır. Yeni sistemler kesilmiş işlemin adresini belleğe (stack) gönderirler. Kesme işlemi bitince, kurtarılan adres program sayacına yüklenir ve yarım kalmış işleme kaldığı yerden devam edilerek tamamlanması sağlanır.

4) Kesme Servis Rutini nedir?
Her kesmeden sorumlu ve kesmenin ayrıntılarını içeren bir servis rutini vardır.
Kesme servis rutini, **bir kesme oluştuğunda yapılması gereken işlemleri içeren komutların bütünü**dür.

5) İşaret Tablosu nedir?
Kesmelerin çabuk ele alınması amacıyla,
olabilecek kesmeler için önceden belirlenmiş rakamlar içeren bir tablodur. Kesmeyi işaret eder ve sistemi yöneltir.

6) Kesme Vektörü nedir?
İşaret tablosu düşük bellek alanında bulundurulur. Burası birçok aygıtın kesme servis rutininin adresini tutar.
Bu kısım kesme vektörü olarak adlandırılır.

7) Tuzak (Trap) ve İstisna (Exception) nedir?
Bir tuzak (trap) ya da istisna (exception), bir hata ya da kullanıcı isteğinin (request) neden olduğu ve yazılım tarafından oluşturulan bir kesmedir.

8) I/O işlemeye yönelik iki yöntem nedir?
* I/O başladıktan sonra, **kontrol yalnızca I/O tamamlandığında kullanıcı programına döner**.
* I/O başladıktan sonra, **kontrol I/O'nun tamamlanmasını beklemeden kullanıcı programına döner**.

9) Uygulama Programları I/O işlemini nasıl gerçekleştirirler?
İşletim sistemi üzerinden gerçekleştirirler. İstek, bir Sistem Çağrısı ile yapılır. İşletim sistemindeki sistem çağrısı rutini, işletim sistemindeki cihaz sürücüsü (device drive) rutinleri yardımıyla I/O işlemini gerçekleştirir.

10) DMA (Direct Memory Access) nedir?
Bellek ile I/O birimleri arasında (CPU bypass edilerek) **doğrudan veri değişimlerinin yapılması**dır. Doğrudan Bellek Erişimi (DMA) ile cihaz denetleyicisi, veri bloklarını CPU müdahalesi olmadan cihaz arabelleğinden ana belleğe aktarır.

11) RAM nedir?
Random Access Memory, bilgisayarda **ana bellek** olarak kullanılan yapıdır. RAM'de istenilen yerdeki veriye doğrudan erişim sağlanabilir. Manyetik teyplerdeki gibi, bir veriye ulaşmak için ondan önceki verileri geçmek gerekmez. Dynamic Random-Access Memory (DRAM)'in bir formudur. Ana bellek olan RAM, bilgisayar sisteminde bir önbellek olarak kullanılır.

12) Volatile nedir?
Uçucu veriler barındıran yapıdır. Yani güç kesildiğinde veriler kaybolur.

13) Depolama Hiyerarşisini anlatın.
Hız, Maliyet ve Uçuculuk açısından hiyerarşik olarak düzenlenir. Küçükten Büyüğe ve Hızlıdan Yavaşa Sıralanma aşağıdaki gibidir: İlk üç yapı volatile, diğerleri kalıcı (non-volatile) belleklerdir.
Registerlar **<** Donanım Önbellekleri L1, L2… **<** Ana Bellek **<** İkincil Bellek (Hard Disk) **<** Manyetik Teypler

14) Multiprocessors (Çok İşlemcili) sistem nedir? Avantajları ve Çeşitleri nelerdir?
Çok işlemcili bir bilgisayar sistemi, **iki veya daha fazla işlemcinin (CPU), ortak bir RAM'e tam erişim sağlayarak**, tüm işleri paylaştığı bir sistemdir. **Avantajları:** Daha çok işi daha kısa zamanda gerçekleştirebilmesi, çoklu PC kullanımından daha ucuz olması, daha güvenilir olması (hata toleransı). **Çeşitleri:** Asymmetric Multiprocessing'te her işlemciye özel bir görev atanır. Symmetric Multiprocessing'te her işlemci tüm görevleri yerine getirir.

15) Paralel Sistem nedir?
Bir sistemin paralel sistem olarak nitelendirilebilmesi için, birden çok işlemcisi veya birden çok çekirdeği olması ve kendisine tahsis edilen, işin belli kısımlarını **paralel olarak çalıştırabilmesi** gerekir.

16) Kümelenmiş Sistemler nedir?
Çok işlemcili sistemler gibidir, ancak birlikte çalışan **birden çok bilgisayar** sistemi vardır. Depolama birimlerini ortaklaşa kullanırlar. Genellikle Depolama Alanı Ağı (storage-area network **(SAN)) aracılığıyla depolama paylaşımı** vardır. Hatalardan kurtulan yüksek kullanılabilirlik (high availibility) hizmeti sağlar. Bir bilgisayarda hata oluşması durumunda, o bilgisayarın görevlerini sistemdeki diğer başka bir bilgisayar yerine getirir.

17) Multitasking (Timesharing) nedir?
Bir işletim sisteminde, bir kullanıcının birden fazla process'i aynı anda **işleme alabilmesi**dir. **Paralel çalıştırma değildir!** CPU, processleri sık sık değiştirir. Birden fazla process aynı anda çalışmaya hazırsa, CPU Scheduling yapılır.

18) Bilgisayar başladığında İşletim sistemi nasıl devreye girer?
Bootstrap programı, sistemi başlatmak için **yürütülen ilk kod**dur. Çekirdeğin (kernel) yüklenmesinden sorumludur. Çekirdek yüklenir. İşletim sisteminde geri planda çalışan programlar (system daemons) başlatılır. Kernel, kesmelerle yönetilir.

19) Çift Modlu Çalışma (Dual-Mode Operation) nedir?
İşletim sistemi kendisini ve diğer sistem bileşenlerini **korumak için** dual-mode özelliği sağlar. User ve Kernel Mod bitleri kullanılır. Sistemin kullanıcı kodunu veya çekirdek kodunu ne zaman çalıştırdığını **ayırt etme yeteneği sağlar**. Kullanıcı kodu yürütülürken mod biti "user", çekirdek kodu yürütülürken mod biti "kernel" olur. Peki kullanıcının mod bitini açıkça "kernel" olarak ayarlamayacağını nasıl garanti ederiz? Sistem çağrısı yapıldığında, işletim sistemi, modu "kernel" yapar. Çağrı sonrası mod biti yeniden "user" olur. **Ayrıcalıklı olarak belirlenmiş bazı talimatlar, yalnızca çekirdek modunda** çalıştırılabilir.

20) İşletim Sisteminin Temel Konseptleri/Parçaları/Fonksiyonları nelerdir?
* İşlem (Process) Yönetimi, * Bellek (Memory) Yönetimi, * Dosya Sistemi Yönetimi, * Depolama Yönetimi,
* I/O Kontrol ve Yönetimi, * Koruma (Protection).

21) İşlem (Process) Yönetimini anlatın.
Program pasif bir varlıktır; process ise aktif bir varlıktır. Program diskte, process RAM'de bulunur. Process, yürütülmekte olan programdır. Process görevini yerine getirmek için kaynaklara ihtiyaç duyar. CPU, Bellek, I/O gibi… Process'in sonlandırılması, yeniden kullanılabilir kaynakların geri alınmasını gerektirir. Bir process, **yürütülecek olan sonraki komutun yerini belirten bir program sayacına** sahiptir. Process, tamamlanana kadar talimatları (instructions) sırayla, birer birer yürütür. Çok thread'li processin, thread başına bir program sayacı vardır.

22) İşlem (Process) Yönetimi aktiviteleri/görevleri nelerdir?
Processlerin oluşturulması ve silinmesi, processleri askıya alma ve devam ettirme, process senkronizasyonu, process iletişimi ve kilitlenme ile başa çıkmak için mekanizmalar sağlamak.

23) Bellek (Memory) Yönetimini anlatın.
Çalıştırılacak olan program ve programın ihtiyaç duyduğu verilerin tamamı (veya bir kısmı) bellekte olmalıdır. Bellek yönetimi, **bellekte** neyin ne zaman olacağını belirler. CPU kullanımının optimize edilmesini sağlar.

24) Bellek (Memory) Yönetim aktiviteleri nelerdir?
Şu anda hafızanın hangi bölümlerinin kimler tarafından kullanıldığını **takip etme**, hangi işlemlerin ve verilerin belleğe girip çıkacağına **karar verme**, ihtiyaca göre **bellek ayırma** ve belleği **serbest bırakma** işlemlerini gerçekleştirmek.

25) İşletim sistemindeki "Dosya Sistem Yönetimi" ne yapar?
Dosya/dizin oluşturma ve silme işlemleri, **dosyaları/dizinleri değiştirmek** için temel öğeleri barındırır.
Dosyaları ikincil depolamaya **map eder**. **Dosyaların** kararlı (uçucu olmayan) depolama ortamına **yedeklenme**sini sağlar.

26) Depolama Yönetimindeki Caching (Önbellekleme) nedir?
Bilgisayarda birçok seviyede gerçekleştirilen (donanım, işletim sistemi, yazılım) önemli bir prensiptir. Kullanımdaki bilgiler geçici olarak daha yavaş depolamadan daha hızlı depolamaya kopyalanır. Bilgiye ulaşmak için **önce daha hızlı depolama (önbellek) kontrol edilir**. Öyleyse, bilgiler doğrudan önbellekten kullanılır (Hızlı). Değilse, veriler önbelleğe kopyalanır ve orada kullanılır. **Önbellek, önbelleğe alınan depolamadan daha küçüktür**. Önbellek yönetimi önemli bir tasarım sorunudur.

27) I/O bellek yönetiminin sorumlu olduğu durumlar nelerdir?
* **Buffering**: Ara belleğe alma (aktarılırken verilerin geçici olarak depolanması),
* **Caching:** Ön belleğe alma (performans için verilerin bölümlerinin daha hızlı bellek alanlarında tutulması),
* **Spooling**: Kuyruklama (bir işin çıktısının diğer işlerin girdileriyle örtüşmesi).

28) İşletim Sistemi Servisleri nelerdir?
* Kullanıcı arayüzü, * Program yürütülmesi, * I/O işlemleri,
* Dosya sistemi idaresi, * İletişimler, * Hata kontrolü,
* Kaynak paylaşımı, * Hesaplama, * Koruma.

29) Sistem Çağrısı nedir? nasıl kullanılır?
İşletim sistemi tarafından sağlanan **servisler için** (yani işletim sisteminin gerçekleştirebildiği tüm işlemler için) programlama arayüzüdür. Genellikle yüksek seviyeli bir dilde (C veya C++) yazılır. Doğrudan sistem çağrısı kullanımı yerine çoğunlukla API aracılığıyla, programlar tarafından erişilir. **En yaygın üç API; Win32 API**, **POSIX API** ve **Java API**'dir.
Her sistem çağrısı bir sayı ile ilişkilendirilir. Sistem çağrısı arayüzü, bu numaralara göre indekslenmiş bir tablo tutar. Sistem çağrısını çağıran uygulama, normalde sistem çağrısını nasıl gerçekleştirebileceğini bilmelidir. **API'de ise, sistem çağrısının nasıl uygulandığının bilinmesine gerek yoktur**. Sadece API'ye uyması ve işletim sisteminin ne yapacağını anlaması gerekir. İşletim sistemi arayüzünün çoğu detayı programcıdan API aracılığı ile gizlenmiştir.

30) Sistem Çağrısı Parametresi Geçişi nedir? Hangi yöntemler kullanılır?
Sistem çağrısı yaparken, çağrı numarasından fazla bilgi göndermek de mümkündür.
Gönderilecek bilginin tipi ve miktarı gibi. İşletim sistemine parametre göndermek için **3 yöntem** mevcuttur.
* **En basit yöntem Registers içerisine göndermek**tir. Register küçük olduğu için bu yöntemde bir sınır vardır.
* Blok yöntemi, **hafızada bir blokta veya tabloda tutma yöntemi**dir. Bu durumda hafızanın adresi register ile gönderilmelidir. Bu sistem Linux ve Solaris işletim sistemleri tarafından uygulanmaktadır.
* En son yöntem ise **Stack yöntemi**dir. Stack içerisine atılan parametreler işletim sistemi tarafından çekilir.
Blok ve Stack yöntemlerinde herhangi bir sınır bulunmaz.

31) Sistem Çağrısı Türleri nelerdir?
* İşlem Kontrolü, * Dosya Yönetimi, * Cihaz Yönetimi,
* Bilgi Sağlama, * İletişim, * Koruma.

32) Sistem Programları nedir?
Program geliştirme ve yürütme için uygun bir ortam sağlayan programlardır. Kullanıcıların işletim sisteminde gerçekleştirdikleri **birçok işlem, gerçek sistem çağrıları tarafından değil, sistem programları tarafından sağlanır**. Bazı sistem programları, basitçe sistem çağrılarına yönelik kullanıcı arayüzleridir, bazıları ise çok daha karmaşıktır.

33) İşletim Sistemi Türleri nelerdir?
* **Simple Structure (Basit Yapı):** Küçük bellek alanında pek çok işlev sağlanabilir. Modüllere bölünmez. (**MS-DOS**)
* **Layered Approach (Katmanlı Yaklaşım):** İşletim sistemi çeşitli katmanlara ayrılmıştır. En alt kısımda **(katman 0) donanım** yer alır, en üst kısımda ise **(katman N) kullanıcı arabirimleri** yer alır. Her katman, sadece alt düzey katmanlar tarafından sağlanan işlemleri yürütür.
* **Microkernel Approach (Mikro Kernel Yaklaşım):** Sadece en önemli işletim sistemi fonksiyonları vardır. Küçük boyutludur.
* **Modules Approach (Modül Yaklaşımı): Bir çok modern işletim sistemi çekirdek modüllerini uygular**. Loadable Kernel Modules (LKMs). **Nesne yönelimli yaklaşımı kullanırlar**. Her temel bileşen ayrıdır. Katman yapısına benzer, ancak daha esnektir. **Linux** ve **Solaris** örnek verilebilir.
* **Monolithic Structure: Original UNIX** – Donanım işlevselliği ile sınırlı bir yapıya sahiptir.
UNIX OS, iki ayrılabilir parçadan oluşur. Sistem programları ve Çekirdek.
* **Hibrit Sistemler: Çoğu modern işletim sistemi tek bir model üzerine kurulu değildir**. Hibrit yapıda, performans, güvenlik ve kullanılabilirlik ihtiyaçlarını karşılamak için birden çok yaklaşım birleştirilir. **Apple Mac OS X** hibrit ve katmanlı yapıdadır.
* **Temel İşletim Sistemi Katmanlı Yapı:** Uygulama, Kabuk, Çekirdek ve Donanım katmanlarından oluşur.

34) Temel İşletim Sistemi Katmanlı Yapıyı anlatın.

*** Uygulama Katmanı:** Kullanılan programlar bu katmanda yer alır. Word, Excel vb.
*** Kabuk (Shell) Katmanı:** Genellikle kullanıcı arayüzü de denilen, kullanıcı ile bilgisayar iletişimini sağlayan arabirimdir. MS-DOS komut istemi arayüzü, Linux'da root olarak girildiğinde #, kullanıcı olarak girildiğinde $ olarak görülen arabirim.
*** Çekirdek Katmanı:** Kabuk katmanından gelen komutlar doğrultusunda, donanım katmanı ile iletişime geçerek, gerekli işlemleri yürüten kısımdır.
*** Donanım Katmanı:** Donanım elemanlarının bulunduğu kısımdır.

35) Process nedir?

Program pasif bir varlıktır; process ise aktif bir varlıktır. Program diskte, process RAM'de bulunur. Process, yürütülmekte olan programdır. Çalıştırılabilir bir dosya belleğe yüklendiğinde **program processe dönüşür**. Bir program birden fazla process olarak yürütülebilir. Ana (Parent) process, çocuk (children) processleri oluşturur, bu çocuk processler de sırayla diğerlerini oluşturarak bir işlemler ağacı oluştururlar. Genel olarak processler, **"process id" (pid) aracılığıyla tanımlanır ve yönetilirler**.

36) Process neleri içerir?

Bir processte aşağıdakiler bulunur:
* **Program Kodu**.
* **Program Counter:** İşlemcide bir sonraki işletilecek komutun ana bellekteki adresini tutan 'register' dır.
* **Stack (Yığın):** Alt program değişkenleri, dönüş adresleri vb. değerleri içerir.
* **Veri:** Değişkenleri, Diziler, Listeleri içerir.
* **Heap:** Nesneleri içerir.

37) Process durumları nelerdir? Bir process hangi durumlarda bulunabilir?

* **Yeni (New):** Processin yeni oluşturulduğunu gösterir.
* **Çalışıyor (Running):** Processin komutları yürütülmektedir.
* **Bekliyor (Waiting):** Process bir olayın gerçekleşmesini beklemektedir. Örneğin bir **I/O işlemi**.
* **Hazır (Ready):** Process bir işlemciye atanmak için hazır haldedir.
* **Bitti (Terminated):** Process çalışmasını bitirmiştir.

38) Process Control Block içerisinde bulunan bilgiler nelerdir?

* **Process Durumu**.
* **Process Numarası:** Her process'in bir ID'si bulunmaktadır.
* **Program Sayacı:** Sayaç, process için yürütülmesi gereken bir sonraki komutun adresini gösterir.
* **CPU Registers (Mikroişlemci Kayıt Edicileri)**.
* **Mikroişlemci Programlama Bilgileri:** Bu bilgiler, işlem önceliğini, programın sıra göstergecini ve diğer programlama değişkenlerini içermektedir.
* **Bellek Yönetim Bilgileri:** Bu bilgiler base ve limit kayıtçılarının değerlerini, sayfa tablolarını veya işletim sistemi tarafından kullanılan bellek sistemine göre değişen segment tablosunu içerir.
* **Hesaplama Bilgileri**.

39) Context Switch nedir? nasıl gerçekleşir?

CPU **bir processten diğerine** geçtiğinde gerçekleşir. CPU başka bir processe geçtiğinde, sistem **eski işlemin durumunu kaydetmeli** ve yeni işlem için **kaydedilmiş durumu** bir bağlam anahtarı (context switch) **aracılığıyla yüklemeli**dir. Bağlam değiştirme süresi tamamen ek yüktür; **sistem kullanımı geçiş yaparken CPU boş (**CPU Idle**) durumdadır**. İşletim sistemi ve PCB ne kadar karmaşıksa, bağlam anahtarı da o kadar uzun olur.

40) Thread nedir?

Bir process'in **paralel olarak işlenen** her bir bölümüdür. Thread, **komut komut işlemlerin yürütülmesi**dir. Programlar ana bir thread ile çalışır. Bu thread'e single-thread, kontrol thread veya execution flow denir. Process içindeki bir thread, inputu beklerken, diğer bir thread başka bir işi yapabilir.

41) Thread ve Process arasındaki fark nedir?

Oluşturuluşu ve kaynakların paylaşılması açısından farklılıkları vardır. Çoğu durumda **threadler, processlerin içinde yer alır** ve onları oluştururlar. **Processler aynı adres uzayını paylaşmazlar** (bellek gibi). Fakat aynı process içindeki farklı thread'ler aynı adres uzayını paylaşabilirler. Bu açıdan process içinde **global bir değişken tanımlanırsa**, bu değişken process içindeki her threadde kullanılabilir olur.

42) CPU Scheduling nedir? CPU Scheduling'in amacı nedir?

Çalıştırılmaya hazır durumda bekleyen **processlerin çalıştırılma zamanlarının planlanması** CPU Scheduling'dir. Amacı, **CPU kullanımını maksimize etmek** için **her zaman çalışan processler olmasını sağlamak** ve ready kuyruğunda bekleyen **processlerin bekleme sürelerini azaltabilmek**tir.

43) CPU/Process Schedular nedir? görevi nedir?
CPU çekirdeğinde bir sonraki yürütme için mevcut **(ready) processler arasından** seçim yapar ve işlemciyi ona ayırır. Processlerin zamanlama sıralarını (scheduling queues) korur. **Seçme kararlarını CPU Scheduling algoritmalarına göre** verir.

44) Process Scheduling kuyrukları nelerdir?
* **İş (Job) kuyruğu:** Sistemdeki tüm işlemlerin kümesidir.
* **Hazır (Ready) kuyrukları:** Ana bellekte bulunan hazır ve yürütülmek için bekleyenler. (CPU'yu bekliyorlar).
* **Aygıt (Device) kuyrukları:** Bir I/O aygıt için bekleyen işlemlerin kümesidir.

45) Neden CPU Scheduling algoritmalarına ihtiyacımız var?
* CPU'nun **hiç boş bırakılmaması** ve CPU **kullanım zamanını maksimize** etmek için.
* Bir process sonlandıktan sonra, **ready kuyruğunda bekleyen process'lerden birinin seçilmesi** gerektiği için.
* CPU Scheduling algoritmaları **bu kararların doğru ve verilmli bir şekilde verilmesini** sağladığı için.

46) CPU Scheduling ne zaman devreye girer?
CPU boşta olduğunda, yani **"CPU Idle" denilen zamanda devreye girer**.
Bu da bir processin sonlanması, **bloklanması**, I/O işlemi veya **bir sistem çağrısı geldiğinde** olur.

47) Schedular; CPU Scheduling kararlarını, hangi process durumlarına göre, nasıl verir?
Process;
1. **Çalışma (running)** durumundan **bekleme (waiting)** durumuna geçerken,
2. **Çalışma (running)** durumundan **hazır (ready)** durumuna geçişte,
3. **Bekleme (waiting)** durumundan **hazır (ready)** durumuna geçişte,
4. **Sonlandığında (terminated)** verir.

İlk madde ve 4. madde için zamanlama açısından seçim yoktur. Ready kuyruğunda process varsa, (Non-Preemptive) sıradaki seçilir. Ancak 2. ve 3. maddeler için bir seçim vardır. Kısaca **Ready durumlarına geçişte (Preemptive)** bir seçim vardır.

48) Preemptive ve Non-Preemptive algoritma nedir?
* **Kesintili Algoritmalar (Preemptive):**
Yürütülen processin **bitirilmeden önce**, işlemciden kaldırılması ve istenilen başka bir processin işlemcide yürütülmesidir.
* **Kesmeyen Algoritmalar (Non-preemptive):**
Process işlemciye yerleştikten sonra; **tamamlanıncaya veya durana kadar** işlemciyi kullanır, müdahale edilmez.

49) Dispatcher nedir?
Görev Dağıtıcısı anlamına gelir. CPU Schedular tarafından **seçilen processe CPU'nun kontrolünün verilmesini sağlar** ve Context Switch yapar. **Processi seçen**, Schedular – **Kontrolü veren**, Dispatcher.

50) Dispatch Latency nedir?
Gönderim Gecikmesi anlamına gelir. Dispatcher (Görev Dağıtıcısı), CPU'nun kontrolünü bir processten diğerine verirken, önceki processin, process kontrol bloğunun kaydedilmesi ve sonraki processin process kontrol bloğunun yüklenmesi (yani context switch yapılması) gerekir ve o arada bir zaman kaybı olur, buna Dispatch Latency denir.

51) Planlama Kriterleri (Scheduling Criteria) nedir?
* **CPU Utilization (İşlemci Kullanımı):** İşlemciyi **olabildiğince meşgul** tutmak. (MAX)
* **Throughput (Üretilen iş): Birim zamanda bitirilen** process sayısı. (MAX)
* **Turnaround time:** Bir processin CPU'da **bitene kadar** geçirdiği toplam zaman. (Beklemeler ve Çalıştırılma dahil)
* **Waiting time (Bekleme zamanı):** Bir processin ready kuyruğunda **her bekleme için** geçirdiği toplam süre.
* **Response time:** Bir processin ready kuyruğuna ilk kez girmesinden sonra, **processe verilen ilk yanıt** arasında geçen süre.

İlk iki maddenin maksimum, diğerlerinin minimum olması beklenir / amaçlanır.

52) CPU Scheduling Algoritmaları nelerdir?
* **FCFS (First Come First Served):** Non-Preemptive'dir. **Kuyruğa ilk gelen processin işlenmesi** prensibine dayanır. Processler kuyruğa hangi sırayla dizilirse, o sırayla işlenir.
* **SJF (Shortest Job First):** Non-Preemptive'dir. Preemptive versiyonu, **SRTF (Shortest Remaining Time First)** olarak adlandırılır. **En kısa CPU burst time değerine sahip processi seçerek çalışır**. Her processle bir sonraki minimum burst time değerine sahip processi karşılaştırır ve hangisi daha küçükse onu çalıştırır. **SJF optimaldir, belirli bir işlem kümesi için minimum ortalama bekleme süresini** verir.
* **Priority Scheduling (Öncelikli Planlama):** Her bir processe öncelik sayısı (tamsayı) atanır. CPU **en öncelikli olan processe** tahsis edilir. **En yüksek öncelik; en küçük tam sayıya aittir**. Preemptive ve Non-Preemptive versiyonları vardır. Bu algoritmanın kullanımında, Starvation durumu ortaya çıkabilir.

* **Round Robin (RR):** Özellikle zaman paylaşımlı sistemler için tasarlanmıştır. İnteraktif işlemleri olan sistemler için kullanışlıdır. Round Robin için **bir zaman aralığı** (time quantum – **q**) tanımlanmıştır, bu zaman aralığı **10 ile 100 msn** arasında değişmektedir. **q** değerinin çok **büyük olması durumunda algoritma FCFS algoritmasına benzer**. q değerinin çok **küçük olması durumunda context switch işlemi çok fazla yapılacaktır**. Bu yüzden, CPU'nun context switch için harcadığı vakit gereğinden fazla olacaktır. Bu istenen bir durum değildir.
* **Multilevel Queue (Çok Düzeyli Kuyruk):** Ready kuyruğu **bölünüp, başka kuyruklar** oluşturulur. Her bir kuyrukta, farklı tipte processler bir araya getirilmiş olur. Örneğin, **Foreground-İnteraktif processler için** Round Robin algoritmasının kullanılacağı ayrı bir kuyruk, **Background processler için** de FCFS algoritmasının kullanılacağı ayrı bir kuyruk tasarlanabilir.
* **Çok Düzeyli Geri Bildirim (Multilevel Feedback Queue):** Bir process birçok **kuyruk arasında geçişler** yapabilir, kuyruklar arasında hareket edebilir. Process bir kuyrukta işini bitiremezse, diğer kuyruğa taşınır.

53) Starvation nedir? Nerede ortaya çıkar? Çözümü nedir?
Starvation, Priority Scheduling algoritmasının uygulanmasında veya Sabit öncelikli uygulanan Multilevel Queue algoritmasında ortaya çıkabilen bir problemdir. Sürekli yüksek öncelikli processlerin yürütülüp, düşük öncelikli processlerin hiç yürütülmemesi problemidir. Çözüm olarak kullanılan **Aging, zaman geçtikçe processin önceliğinin arttırılması**dır.

54) Sabit Öncelikli Planlama (Fixed priority scheduling) ve Time Slice nedir?
Multilevel Queue (Çok Düzeyli Kuyruk) algoritmasında kullanılan iki yöntemdir. **Sabit öncelikli planlama yöntemi**, örneğin önce foreground daha sonra background processlerin çalıştırılması şeklinde olabilir. Bu durumda **starvation problemi** ortaya çıkabilir. **Time slice** ise, belirli processlere %80 ve diğerlerine %20 zaman ayırma gibi uygulanabilir.

55) Thread'in Faydaları nelerdir?
* **Responsiveness (Cevap verebilirlik):** Özellikle **kullanıcı arayüzü** için önemlidir. Processin bir parçası engellenmişse, işlemin diğer bölümlerinin yürütülmesine izin verir.
* **Resource Sharing (Kaynak Paylaşımı):** Threadler **aynı adres uzayını paylaşırlar**. Bu durum, paylaşılan bellekten veya mesaj geçişinden daha kolay uygulanabilirdir.
* **Ekonomi:** Thread oluşturmak, process oluşturmaktan **daha ucuz**dur. Threadde context switch'in maliyeti daha azdır. Process değişiminde, önceki processin Process Control Block'unda son kaldığı yerin kaydedilmesi ve diğer processte kaldığı yerin okunması ve çalışmaya hazırlanması maliyetli bir iştir.
* **Scalability (Ölçeklenebilirlik):** Paralel çalışma için **yeni işlemci** almak yerine, **çok çekirdekli yapılar** kullanılabilir. Her bir thread, farklı bir çekirdek birimi ile eşleştirilir.

56) Bir sistemde Thread yapabilmek için neler gereklidir? * * * * *
* **Sistem Mimarisi**, thread oluşturmaya uygun yapıda olmalıdır. Multicore veya multiprocessor bir yapı olması gibi.
* **İşletim Sistemi**, thread kullanımını desteklemelidir. Örneğin, MS-DOS olursa thread oluşturulamaz.
* **Kullanılan program/yazılım**, thread işlemini destekleyen bir program/yazılım olmalıdır.

57) Paralellik ve Eşzamanlılık nedir?
* **Paralellik:** Bir sistemin aynı anda, birden fazla görevi yerine getirebilmesidir.
* **Eşzamanlılık:** İlerleme sağlayan birden fazla görevi destekler. Tek işlemcili ve tek çekirdekli yapılarda, eş zamanlılığı sağlayan şey "zamanlayıcı"dır.

58) Paralelleştirme Türleri nelerdir?
* **Veri Paralelleştirme:** Veriler bölünüp, birden çok çekirdeğe dağıtılır. Her bir çekirdekte **aynı işlem** yapılır.
* **Görev Paralelleştirme:** Tüm threadler tüm veriyi kullanır. Her bir thread **farklı bir görevi** gerçekleştirir.

59) Multicore / Multiprocessor sistemlerde programcıların dikkat etmesi gerekenler nelerdir?
* **Faaliyetleri Bölme:** Hangi faaliyetler thread yapmaya uygun?
* **Denge / Balance**.
* **Veri Bölme:** Threadler arasında verileri bölmek.
* **Veri Bağımlılığı:** Processin çalışması, başka bir verinin varlığına veya hesaplanmasına bağlı ise, onun beklenmesi gerekir.
* **Testing ve Debugging:** Thread sistemlerinin test edilmesi daha zordur.

60) Amdahl's Kanunu nedir?
Seri ve paralel bileşenlere sahip bir sisteme, **ek çekirdekler eklemekle elde edilen performans kazanımı**nı tanımlar. Bir uygulamanın seri kısmının, ek çekirdekler eklenerek elde edilen performans üzerinde "orantısız" etkisi vardır. Örneğin, uygulama %75 paralel ve %25 seri ise, 1 çekirdekten 2 çekirdeğe geçiş **1.6** kat hızlanma ile sonuçlanır. N sonsuza yaklaşırken hızlanma 1/S'ye yaklaşır. **SpeedUp <= 1 / [S + ((1 - S) / N)]**

61) Multithread Modeller nelerdir?
* **Many to One:** Birçok user thread'in, tek bir kernel thread'e map edilmesi ile oluşturulur. Kernel thread'in bloklanması, tüm user threadlerin de bloklanmasına yol açar. Çok çekirdekli bir yapı olsa bile, çoklu user threadlerin yalnızca biri çekirdekte çalıştığı için, **paralel olarak çalıştırılmayabilirler**. Örnek: Solaris Green Threads ve GNU Portable Threads.
* **One to One:** Her bir user thread, ayrı bir kernel thread'e map edilir. Her user level thread oluştuğunda, kernel thread oluşturulur. **Many to One modeline göre, daha iyi eşzamanlılık** sağlar. Örnek: Windows ve Linux.
* **Many to Many:** Birçok user thread, birçok kernel thread'e map edilir. **Pek yaygın değildir**. Örnek: ThreadFiber (Windows).

62) Pthread nedir?
Thread ve senkronizasyon oluşturmak için kullanılan **POSIX standartı bir API**'dir.
User ve Kernel düzeyinde sağlanabilir. UNIX işletim sistemlerinde (Linux ve Mac OS X) yaygındır.

63) UNIX ortamında ve C dilinde thread oluşturmak için kullanılan komutlar nelerdir?
* **pthread_t tid;** Oluşturulacak olan thread'i tanımlamak için kullanılır.
* **pthread_attr_init(&attr);** Thread, oluşturulma anında bir takım ilk değerler/ayarlar ile oluşturulacaksa kullanılır.
* **pthread_create(&tid, &attr, runner, argv[1]);** Thread oluşturma komutudur. İlk parametre tanımladığımız thread id'sinin adresidir. İkinci parametre, oluşturulma anında girilmek istenen ilk değerlerin adresidir. NULL olabilir. Üçüncü parametre, thread içerisinde çalıştırmak istediğimiz fonksiyonun adıdır. Dördüncü parametre, runner fonksiyonuna parametre girmek için kullanılır. Aynı zamanda, komut satırında girilen ilk parametredir.
* **pthread_join(tid, NULL);** Eğer bir thread'in yaptığı işin bitmesi beklenecekse, bu komut kullanılır. Birinci parametre, hangi thread'in bekleneceğini ve ikinci parametre de thread'in çalıştırdığı fonksiyondan dönecek değeri belirtir.
* **pthread_exit(0);** Thread'i sonlandırma komutudur. **0** değeri, döndürülen değerdir. Bu değer, pthread_join()'deki ikinci parametre ile istenirse elde edilebilir.

64) Interprocess Communication (IPC) nedir?
Aynı sistem içindeki farklı processlerin haberleşebilmeleri (bilgi alışverişi yapabilmeleri) için kullanılan işletim sisteminin sağladığı bir yapıdır.

65) Process'ler çalışma türü bakımından kaça ayrılır?
Bağımsız (Independent) ve İşbirliği halinde (Cooperating) olmak üzere iki şekilde çalışırlar. Bağımsız processler başka processlerden etkilenmez ve aralarında bilgi alışverişi yoktur. İşbirliği halinde olan processler, birbirleri ile haberleşirler.

66) İşbirliği halinde (Cooperating) çalışan process'lerin kullanılma nedenleri nelerdir?
* **Bilgi Paylaşımı:** Birçok kullanıcı aynı veriye ulaşmak isteyebilir. Veriye çoklu ulaşımı oluşturmamız gerekir.
* **Hesaplama Hızı:** Process'leri farklı görevlerde çalıştırarak, daha hızlı sonuçlar elde etmek.
* **Modülerlik:** Birçok açıdan faydası vardır. Ayrıca, belli bir modülde meydana gelen hatadan diğer modüller etkilenmez.
* **Rahatlık:** Kullanıcı birçok işlemin aynı anda yapılmasını isteyebilir. Bu açıdan kullanıcıya rahatlık sağlar.

67) IPC kaç farklı şekilde gerçekleşir?
IPC için iki farklı model vardır. Bunlar **Paylaşılan Bellek (Shared Memory)** ve **Mesaj Geçişi (Message Passing)**'dir.

68) Paylaşılan Bellek (Shared Memory) Yaklaşımı nedir?
Processlerin ortak bir bellek alanını kullandığı yaklaşımdır. Ortak alan, her processin adreslerinden farklı da olabilir. Herhangi bir processin bellek alanını ortak olarak da kullanabilirler. Bir sistem çağrısı ile birlikte kernel, ortak alanı (shared memory) oluşturduktan sonra devreden çıkar. Yani her seferinde bir kernel ihtiyacı yoktur. Bu tip bir iletişim, işletim sisteminin değil, processlerin kontrolü altındadır. Bu açıdan hızlı ve avantajlıdır. Konuyu daha iyi anlamak için Üretici-Tüketici (Producer-Consumer) problemini incelemek gerekir. Paylaşılan Bellek, "Buffer" olarak da adlandırılır.

69) Producer/Consumer (Üretici/Tüketici) problemi nedir?
Bu problemde bir adet üretici ve bir adet tüketici processi bulunur. Üretici burada veriyi üreten ve tüketici ise veriyi tüketendir. Burada webserver ve client örneği verilebilir. Server bu örnekte üretici, client ise tüketicidir.

70) Bounded (Sınırlı) Buffer ve Unbounded (Sınırsız) Buffer nedir?
Üretici veriyi buffer'a koyar ve tüketici veriyi bufferdan okur. Bounded, sabit bir buffer boyutu olduğunu varsayar. Unbounded buffer'da ise buffer boyutunda herhangi bir sınır olmadığı varsayılır. **Unbounded buffer'da üretici asla beklemez**. Bounded buffer'da ise üretici tüm buffer alanları doluysa bekler. Her ikisinde de tüketici, tüketilecek bir buffer yoksa bekler.

71) Race Condition (Yarış Durumu) nedir?
İki veya daha fazla prosesin, paylaşımlı kullandıkları bir veri üzerinde, yapmış oldukları **okuma ve yazma işlemleri**, **hangi processin ne zaman çalıştığına bağlı olarak**, **farklı bir son değere sahip oluyorsa** bu duruma yarış durumu denir.

72) Race Condition nasıl önlenir?
Örneğin, üretici-tüketici problemindeki yarış durumundan kurtulmak için **bir anda sadece bir proses**in counter değişkenini değiştireceği garanti edilmelidir. **Process'ler senkronize edilmeli**dir. Bu amaçla çeşitli kilit mekanizmaları kullanılabilir.

73) Mesaj Geçişi (Message Passing) nedir?
Processlerin, paylaşılan değişkenlere ihtiyaç duymadan birbirleri ile haberleşebildiği yapıdır. ***Send*** ve ***Receive*** sistem çağrısı ile çalışır. Her bir mesaj, sistem çağrısı ile gerçekleştirilir. Dolayısıyla kernel sürekli devrededir.

74) Mesaj Geçişi (Message Passing) yaklaşımının avantajı nedir?
Race Condition durumu ile karşılaşılmaması ve Senkronizasyon problemi yaşanmamasıdır.

75) Mesaj Geçişi sistemini dizayn ederken dikkat edilmesi gerekenler nelerdir?
* **Naming:** Doğrudan veya Dolaylı Haberleşme.
* **Senkronizasyon:** Bloklamalı veya Bloklamasız Haberleşme.
* **Kapasite:** Bounded veya Unbounded Buffer.

76) Doğrudan Haberleşme nedir?
Processler açıkça birbirini isimlendirmelidir. **İki process arasında** gerçekleşir. Bu komutlarla mesajın hangi processe gönderileceği ve hangi processin mesajı alacağı açıkça belirtilmiştir. **send**(P, message) ve **receive**(Q, message)

77) Dolaylı Haberleşme nedir?
Mesaj gönderilirken, alıcı direkt olarak belirtilmez. Bir mail/mesaj kutusu üzerinden iletişim kurulur. Mail kutusunun bir ismi vardır. Processler **yalnızca aynı mail kutusunu paylaşıyorlarsa** iletişim kurulabilir. Gönderici mesajı mail kutusuna gönderir. Alıcı da mesajı mail kutusundan okur. A, burada mail kutusunun ismidir. **send**(A, message) ve **receive**(A, message)

78) Doğrudan Haberleşme ile Dolaylı Haberleşme arasındaki fark nedir?
Doğrudan haberleşme **yalnızca iki process arasında** gerçekleşebilir. Dolaylı haberleşmede, ikiden fazla process arasında da iletişim sağlanabilir ve bir process birden fazla mail kutusu ile iletişim kurabilir.

79) Engellemeli Haberleşme (Blocking) nedir?
Gönderen, mesaj alınana kadar; Alıcı da bir mesaj mevcut olana kadar engellenir.

80) Engellemesiz Haberleşme (Non-Blocking) nedir?
Gönderen mesajı gönderir ve devam eder. Alıcı geçerli veya null bir mesaj alır.

81) POSIX standardında Shared Memory oluşturmak için gerekli komutlar nelerdir?
shm_open: Paylaşılacak bellek alanı segmentlerini açmak veya oluşturmak için kullanılır.
shm_unlink: Paylaşılan bellek alanının kaldırılması için kullanılır.
ftruncate(): Paylaşılan bellek alanının (nesnenin) boyutunun belirlenmesi için kullanılır.
mmap(): Bir dosya işaretçisini, paylaşılan bellek nesnesine eşlemek için kullanılır. Paylaşılan belleğe okuma ve yazma, mmap() tarafından döndürülen pointer (işaretçi) kullanılarak yapılır.

82) Kritik Bölge/Bölüm/Kısım (Critical Section) nedir?
Proseslerin, ortaklaşa kullandıkları veri alanlarına erişip, **veri okuma-yazma işlemlerini yaptıkları program parçaları**na denir.

83) Kritik Bölge Problemi nedir?
Paylaşımlı veriye aynı anda ulaşmak her zaman probleme açıktır ve tutarsızlığa neden olabilir. Tutarlılığı sağlamanın yolu, birlikte çalışan proseslerin veya threadlerin, **kesin bir sırayla** işlerini yapabilecekleri bir mekanizma oluşturmaktır. Kritik bölge problemi, bunu çözmek için protokol tasarlamaktır.

84) Kritik Bölge probleminin çözümü için gerekli şartlar nelerdir?
* **Mutual Exclusion (Karşılıklı dışlama):** Bir process kritik bölümü çalıştırıyorsa diğer process'lerin hiçbirisi kritik bölümü çalıştırmamalıdır.
* **Progress (İlerleme):** Hiçbir proses kritik bölümde çalışmıyorsa ve bu bölüme girmek isteyen birtakım prosesler varsa, kritik bölüme girme isteği uzun bir süre ertelenemez. Yani prosesler birbirlerinin sürekli iş yapmalarını engellememelidir.
* **Bounded Waiting (Sınırlı Bekleme):** Bir proses beklerken, diğer proseslerin kritik bölüme girme sayısının bir sınırı olmalıdır. Starvation'a yol açılmaması gerekir.

85) Kritik Bölge Problemi için bazı çözümler nelerdir?
Peterson Çözümü, Senkronizasyon Donanımı (test_and_set ve compare_and_swap), Mutex Kilitleri, Semaforlar, Monitörler.

86) Peterson Çözümü nedir? Özellikleri nelerdir?
Yazılım tabanlı bir çözümdür. **İki process için** tasarlanmıştır. Bu prosesler flag ve turn adlı nesneleri paylaşır ve bu şekilde senkronize olurlar. **i.** process **flag[i]** = **true** ve **turn** = **i** olunca kritik bölgeye girebilir. **turn** değişkeni, kritik bölgeye girme sırasının kimde olduğunu belirtir. **flag** ise, ilgili processin kritik bölgeye girmeye hazır olup olmadığını (isteğini) belirtir.

87) Senkronizasyon Donanımı Çözümü nedir?
Donanım tabanlı bir çözümdür. Temel çalışma mantığı, kritik bölgeyi kilitlemektir. İki çeşittir. Birinde prosesler kritik bölgeye girdiğinde, tüm kesmeler devre dışı bırakılır. Daha sonra yeniden devreye girerler. İkincisinde ise, kilit değişkenleri kullanılır. Bu amaçla test_and_set() ve compare_and_swap() şeklinde iki çözüm vardır. Tüm işlemler atomik olarak gerçekleşir.

88) Mutex Kilitleri (Mutex Locks veya Spinlock) nedir?
Bir process, kritik bölgeye girmeden önce bir (sistem çağrısıyla) lock talep eder. (acquire)
Kritik bölgeden ayrılınca lock değişkenini serbest bırakır. (release)
acquire() ve release() çağırımları atomik gerçekleşir ve bu fonksiyonun arka planında donanım tabanlı çözümler kullanılır.

89) Busy Waiting nedir?
Busy Waiting (Meşgul Bekleme) döngüsüdür. Bir process kritik bölgede iken, **kritik bölgeye girmek isteyen diğer processler**, sürekli olarak bir döngü içerisinde, **kritik bölgeye girmeye gayret ederler** ve başka bir iş yapamadan beklerler. Meşgul bekleme olduğundan, bekleyen process de işlemci zamanı harcar. İstenmeyen bir durumdur.

90) Mutex kilitlerindeki temel sorun nedir?
Temel sorun **Busy Waiting'e sebep olması**dır. Aynı zamanda, bir process kritik bölgeden çıkınca, bekleyen birden fazla process varsa, Starvation durumu da oluşabilmektedir.

91) Semaphore nedir?
Processleri senkronize etmenin bir başka yoludur. Busy Waiting, CPU zamanının harcanmasına ve performans düşüşüne neden olur. Bunu önlemek için Semaphore değişken (S) tanımlanır. S değişkeni bir tam sayıdır ve wait() ve signal() gibi iki temel fonksiyona sahiptir. Bu fonksiyonlar mutex kilitlerindeki acquire() ve release() fonksiyonları gibi atomik çalışırlar.

92) Semaphore nasıl çalışır?
wait() ile S'nin değeri azaltılır, **signal()** ile S'nin değeri artırılır. Kaynağı kullanmak isteyen her proses, semafor üzerinde **wait() işlemi gerçekleştirir** (sayaç azaltılır). Bir process kaynağı serbest bıraktığında ise signal() işlemini gerçekleştirir (sayaç artırılır). Semafor değeri **sıfır olduğunda**, tüm kaynaklar kullanılıyor durumdadır. Her semafor bekleyen proses listesine sahiptir. Bir proses, bir semafor üzerinde waiting durumunda ise semaforun proses listesine eklenir. Signal (kernel'de wakeup) fonksiyonu, prosesi semafor listesinden çıkarır.

93) Semaphore çeşitleri nelerdir?
*** Counting Semaphore:** Tamsayı değeri sınırsızdır. Belirli bir sayıda kaynağa erişimin denetlenmesi için kullanılır.
*** Binary Semaphore:** Tamsayı değeri **0** veya **1** olur. Bu tip semaforlar, mutex kilitleri gibi çalışırlar.

94) Semaphore, busy waiting problemini nasıl çözmeye çalışır?
Semafor, prosesleri **CPU üzerinde meşgul bekletmek yerine**, kernel'deki **block()** komutu ile bloke edip bekleme kuyruğuna alır. Bloke edilen proses, semaforla ilişkilendirilir, durumu **running -> waiting**'e alınır ve semafor ile ilişkili bir bekleme listesine yazılır. Bu sırada çalışan bir prosesin işi bittiğinde, listedeki bir proses **wakeup()** komutu ile uyandırılır ve durumu **waiting -> ready**'e alınır. Böylece **bekleme (busy waiting) olmadan, çalışır duruma geçer**. Buna rağmen, Semafor ile senkronizasyon çözümleri, zamana bağlı hatalar nedeni ile bazı durumlarda yetersiz olabilmektedir.

95) DEADLOCK nedir?
İki (veya daha fazla) prosessin (**veya thread**'in), karşılıklı olarak **aynı anda kaynak tutma ve kaynak talep etme** durumunda, her ikisinin de birbirini beklemesi ve çalışmaya devam edememesi (Kilitlenme) durumudur. Sözü edilen kaynaklar; Bellek, I/O, CPU zamanı ve benzerleri olarak sayılabilir.

96) DEADLOCK sebepleri nelerdir?
Deadlock oluşumu için dört sebep vardır ve deadlock olabilmesi için bu sebeplerin **hepsinin sağlanması** gerekir.
1) Mutual Exclusion: Bir kaynağı aynı anda yalnızca bir proses kullanıyorsa**;**
2) Hold and Wait: En az bir kaynağı tutan proses, diğer proseslerin tuttuğu, en az bir kaynağı da elde etmek istiyorsa**;**
3) No Preemption: Proses kesintiye uğratılamayan veya sadece prosesin kendisi tarafından serbest bırakılan durumdaysa**;**
4) Circular Wait: Döngüsel bir şekilde birbirini bekleyen proseslerden oluşan bir yapı varsa**;** deadlock oluşur.

97) Graph'te Cycle (Döngü) durumu olması, Deadlock oluşturur mu?
Graph; cycle içermiyorsa, kesinlikle **deadlock yoktur** diyebiliriz.
Graph; cycle içeriyorsa, diğer deadlock şartlarının sağlanıp sağlanmadığına bakmak gerekir.

98) Kilitlenmeleri ele alan kaç yöntem vardır? Bunlar nelerdir?
1) Sistemin asla kilitlenme durumuna girmeyeceğinden emin olmak. (Deadlock Prevention & Deadlock Avoidance)
2) Sistemin bir kilitlenme durumuna girmesine izin vermek ve ardından kurtarmak.
3) Sorunu görmezden gelmek ve sistemde asla kilitlenmeler oluşmadığını varsaymak.

99) Deadlock Prevention (Önleme) nedir?
Deadlock için gerekli dört koşuldan birini geçersiz kılmaktır.
Döngüsel bekleme koşulunun (**Circular Wait**) geçersiz kılınması en yaygın olanıdır.

100) Deadlock Avoidance (Kaçınma) nedir?
İstatistiklere göre, varsayımla hareket etmektir. Bunun için sistemin bazı ön bilgilere sahip olması gerekir. İki tür algoritması vardır. Bir kaynağın tek örneği varsa, **Kaynak Tahsis Grafları** kullanılır; çok örneği varsa, **Banker Algoritması** kullanılır.

101) Modern İşletim Sistemleri DEADLOCK durumunu nasıl çözer?
Modern İşletim Sistemleri, **kilitlenmeyi görmezden gelirler**. Çünkü çözmeye çalışmakla harcanan enerji, kazanımdan daha fazladır. Ayrıca; **Deadlock'ın çözülmesi, farklı problemlerin de ortaya çıkmasına sebep olur.**

102) Deadlock'ın çözülmesi durumunda, hangi problemler ortaya çıkabilir?
Örneğin, Deadlock Prevention'da, kilitlenmeyi önlemek için **Mutual Exclusion geçersiz kılınırsa**, bu sefer de **Race Condition** durumu ortaya çıkabilir. Aynı yöntemde **Hold and Wait geçersiz kılınırsa**, bu sefer de **Starvation** durumu ortaya çıkabilir.

103) Banker Algoritmasında kullanılan vektör ve matrisler nelerdir?
*** Available:** Her kaynak türünden, kaç adet örneğin kullanılabilir olduğunu gösteren vektördür. (1xm)
* **Max:** Her bir prosesin, her bir kaynak türünden en fazla ne kadar talep edebileceğini gösteren matristir. (nxm)
* **Allocation:** Her bir prosesin, her bir kaynak türünden anlık olarak ne kadar kullandığını gösteren matristir. (nxm)
* **Need:** Her prosesin, görevini tamamlamak için, her kaynak türünden ne kadar ihtiyacı olduğunu gösteren matristir. (nxm)

104) Banker Algoritmasında Need matrisi nasıl oluşturulur?
Need matrisi şu formülle oluşturulur: **Need[i,j] = Max[i,j] – Allocation[i,j]**

105) CPU hangi bellek alanlarına erişebilir?
CPU sadece Main Memory (RAM) ve Register'lara erişebilir. (RR)

106) Mantıksal (Logical) Adres nedir?
CPU tarafından oluşturulan adresler **Mantıksal (Logical) Adres** olarak adlandırılır. Mantıksal adres **sıfırdan başlar**.

107) Mantıksal Adresin kullanılma nedenleri nelerdir?
Mantıksal adresin kullanılmasının bir sebebi, **fiziksel adresin yeterli olmaması** durumudur. Bir diğer sebep ise, birçok programın kodunda kullanılan fiziksel adreslerin, birbiriyle çakışması ve çeşitli problemler ortaya çıkarmasıdır.

108) Fiziksel Adres nedir?
Bellek ünitelerinin gördüğü adreslere ise **fiziksel adres** denir. Program tarafından üretilen mantıksal adreslerin fiziksel adreslere çevrilmesi gerekir.

109) Bellek Yönetim Ünitesi (Memory Management Unit - MMU) nedir?
Mantıksal adresten fiziksel adrese **dönüşüm** (Address Binding – Adres Bağlama) yapan **donanım aygıtı**dır.

110) Adres Bağlama ne zaman gerçekleştirilir?
*** Compile Time:** Bellek konumu önceden (derleme anında) biliniyorsa (**absolute code**).
*** Load Time:** Derleme sırasında bellek konumu bilinmiyorsa (**relocatable code**).
*** Execution Time:** Proses, yürütme sırasında bir bellek bölümünden diğerine taşınabiliyorsa, bağlama çalışma zamanına kadar ertelenir.

111) Yer Değiştirme (Swapping), Swap Out ve Swap In nedir?
Yer Değiştirme (Swapping): Prosesin/Verinin, disk ve bellek arasında **sürekli taşınması**dır.
Swap Out: Bellekten Diske veri/proses taşıma işlemidir.
Swap In: Diskten Belleğe veri/proses taşıma işlemidir.

112) Prosesin disk ve bellek arasında, sürekli yer değiştirmesinin nedeni nedir?
Prosesin disk ve bellek arasında, sürekli yer değiştirmesinin sebebi, bellek alanının **yetersiz** olmasıdır.

113) Backing Store nedir?
Tüm kullanıcılar için, tüm bellek görüntülerinin kopyalarını barındıracak kadar büyük, **hızlı disk**tir.
Bu hafıza görüntülerine doğrudan erişim sağlanır.

114) Dinamik Depolama Tahsisinde kullanılan 3 yöntem nedir?
*** First Fit:** Yeterince büyük olarak bulunan, ilk alana veriler yerleştirilir.
*** Best Fit:** Bütün boş alanlara bakılır ve eldeki veri en az boşluk kalacak şekilde yerleştirilir.
*** Worst Fit:** Bulunabilen en büyük alana yerleştirme işlemi yapılır.

115) Fragmentation nedir? Kaç çeşittir?
Veri bloklarının, bellek adresleri içerisinde (aralarda boşluklu olacak şekilde) parçalı bir yapıda yer almasıdır. İki çeşittir.
1) External Fragmentation: Bir isteği karşılamak için yeterli bellek alanı vardır. Fakat bu alanlar bitişik değildir. Bellekte yerleşik durumda olan proseslerin arasında boşlukların olması gibi.
2) Internal Fragmentation: Ayrılan bellek, istenen bellekten biraz daha büyük olabilir; bu boyut farkı, bir bölümün dahili hafızasıdır, ancak kullanılmamaktadır.

116) Sıkıştırma (Compaction-Defragmentation) nedir?
Bütün prosesleri bir yönde kaydırıp, **boş bellek alanlarını birleştirmek**tir.
Sıkıştırma ile External ve Internal fragmentation problemleri **çözülmüş olur**.

117) Çerçeve (Frame), Sayfa (Page), Sayfa Tablosu (Page Table) ve Paging nedir?
Fiziksel bellek çerçeve (frame) adı verilen, Mantıksal bellek ise sayfa (page) adı verilen (512 byte ile 16 MB arası) eşit büyüklükte bloklara ayrılmıştır. Mantıksal adresleri fiziksel adreslere çevirmek için kullanılan tabloya **page table (sayfa tablosu)** denir. Bu işlemlerin hepsine birden Paging denir. Sayfa tabloları ana bellekte tutulur. N page'lik bir programı çalıştırmak için, N boş frame bulmanız ve programı yüklemeniz gerekir.

118) Paging faydaları nelerdir?
1) External fragmentation önlenir.
2) Değişken boyutlu bellek yığınları sorunu ortadan kalkar.

119) Paging yapılmasına rağmen çözülemeyen problem nedir?
Internal fragmentation, Paging yapılmasına rağmen devam eder, çözülmüş olmaz!

120) CPU tarafından üretilen adresler nelerdir?
*** Sayfa Numarası (p):** Fiziksel bellekteki her bir frame'in taban adresini tutan sayfa tablosundaki göstergedir.
* **Sayfa Offseti (d):** Taban adresi ile birleştirilerek, fiziksel bellekte frame'in içerisindeki yerin belirlenmesinde kullanılır.

121) PTBR ve PTLR nedir?
*** Page-table base register (PTBR)** page tablosunu işaret eder.
* **Page-table length register (PTLR)** page tablosunun boyutunu işaret eder.

122) TLB (Translation Look-Aside Buffer) nedir?
Hem page tablosu, hem de veri/komut için **iki defa RAM erişimi** gerekir. Bu istenmeyen bir durumdur. Bunu önleyen yapı TLB'dir. Page tablosunun **ön belleği**dir. Önbellekleme ile iki defa yerine, tek seferde aynı sonucu almak mümkün olur.

123) Effective Access Time (EAT) nedir?
Etkileşim süresidir. TLB yapısı ile daha az RAM erişimi olduğunda, etkileşim süresinde de azalma meydana gelir.
İsabet oranı **%80** olursa, yani erişimlerin %80'i 10 ns ve %20'si 20ns zaman alırsa; **EAT** = **0.8***10 + **0.2***20 = 12ns bulunur.

124) Bellek Koruması (Memory Protection) nedir?
Yalnızca okuma (read-only) veya okuma-yazma (read-write) erişimine izin verilip verilmediğini belirtmek için, koruma bitini her frame ile ilişkilendirerek uygulanan bir yaklaşımdır.

125) Valid – Invalid biti nedir?
Page tablosundaki her girişe eklenen geçerli-geçersiz bitidir.
*** Valid:** İlişkili sayfanın, prosesin mantıksal adres alanında olduğunu ve dolayısıyla **yasal bir sayfa** olduğunu belirtir.
*** Invalid:** Sayfanın, prosesin mantıksal adres alanında olmadığını gösterir.

126) Shared Pages (Paylaşılan Sayfalar) nedir?
Proseslerin, sayfa tablolarında tekrarlanan **(aynı olan) değerlerinin**, ortak bir veri alanında depolanmasıdır.
Böylece **bellek optimizasyonu** sağlanmış olur.

127) Virtual Memory (Sanal Bellek) nedir?
Kullanıcı **mantıksal belleğinin**, fiziksel bellekten ayrılmasıdır.

128) Sanal Bellek neden kullanılır? Amacı nedir?
Kodun yürütülebilmesi için bellekte olması gerekir, ancak programın tamamı nadiren kullanılır. O yüzden programın tüm kodu aynı anda gerekli değildir. **Program parçalı bir yapıda yüklenebilir**. Yani programın sadece yürütülecek olan kısmı belleğe getirilir. Diğer kısımlar ihtiyaç duyuldukça getirilir. Bu şekilde **mantıksal adres alanı, fiziksel alandan çok daha büyük olmuş olur** ve fiziksel belleğin sınırları ile kısıtlanmamış olur. Ayrıca programlar daha az bellek ayırır ve aynı anda daha fazla proses çalışabilir. Swap in ve Swap out işlemleri daha az olduğundan daha az I/O ihtiyacı olur ve programlar daha hızlı çalışır.

129) Virtual Memory kaç çeşittir?
İki çeşittir. Demand Paging (Sayfa Talebi) ve Demand Segmentation (Segmentasyon Talebi).

130) Demand Paging'in Faydaları nelerdir?
1) I/O ihtiyacı azalır, gereksiz I/O ihtiyacı olmaz.
2) Daha az **bellek** alanına ihtiyaç duyulur.
3) Daha hızlı **cevap** verilir.
4) Daha fazla kullanıcıya **hizmet** verilebilir.

131) Lazy Swapper nedir?
Bir sayfaya (page) ihtiyaç oluncaya kadar, o sayfanın belleğe getirilmemesidir.

132) Page Fault nedir?
MMU, adres çevirisi sırasında, eğer sayfa tablosu girişindeki valid-invalid biti **i** ise, sayfa hatası (page fault) meydana gelir. Yani **aranan bir sayfanın fiziksel bellekte olmama durumu**dur. Fiziksel bellek tamamen boş iken de meydana gelir

133) Copy on Write (COW) nedir?
Prosesler oluşturulurken, **parent ve child proseslerin bellekte aynı sayfayı paylaşmalarına izin verilmesi**dir. Proseslerden biri, paylaşılan sayfayı değiştirirse, ancak o zaman ilgili sayfanın kopyası oluşturulur. Bu şekilde, proseslerin oluşturulması, daha hızlı ve daha verimli bir hale gelmiş olur.

134) Page Replacement (Sayfa Değiştirme) nedir?
İhtiyaç halinde boş bir frame bulunamazsa, **victim frame** (kurban çerçeve) seçmek için,
bir sayfa değiştirme algoritması (FIFO, Optimal, LRU) kullanılmasıdır.

135) Page Replacement'ın amacı nedir?
En az page fault'u sağlamaktır. Frame sayısı arttıkça, Page Fault hep azalmaz. Belirli bir yere kadar azalır, sonra sabitleşir.

136) Belady Anomalisi nedir?
Page Replacement için FIFO algoritması kullanıldığında, yeni frame'ler eklemek **page fault'ın bazı anlarda artması**na da sebep olabilir. Buna **Belady Anomalisi** denir.

137) Page Replacement'ta FIFO algoritması nasıl çalışır?
Victim frame seçme işlemi ilk gelen frame'e göre yapılır. En önce veya en eski gelen frame silinir.

138) Page Replacement'ta Optimal algoritması nasıl çalışır?
Referans stringinde ileri bakılır ve hangisi daha uzun süre kullanılmayacaksa, o victim frame olarak seçilir. En az page fault durumu için bu algoritma idealdir. Fakat gerçekte uygulanan bir algoritma değildir. Sadece ölçü alınabilmesi için kullanılır.

139) Page Replacement'ta LRU algoritması nasıl çalışır?
Geçmişe göre bakılıp, en uzun süre kullanılmayan frame victim frame olarak seçilir. FIFO'dan daha iyi, Optimal'den daha kötü çalışır. Genel olarak iyi bir algoritma sayılır ve sıkça kullanılır. Stack ve Counter implementasyonları şeklinde ifade edilen iki türü vardır.

140) Thrashing nedir?
Bir prosesin sürekli swap in-swap out durumuna düşmesidir. Bu sebeple çok fazla page fault durumu meydana gelir. Kelime olarak da, boşa çalışmak anlamına gelir.