Bölüm 17: Soru Cevap

İşletim Sistemleri

Soru

- İşletim sistemi çekirdeğinin amacı nedir?

- A) Kullanıcılara arayüz sağlamak
- B) Donanım kaynaklarını yönetmek
- C) Uygulama yazılımlarını yürütmek
- D) Kullanıcı verilerini depolamak

Cevap

- Cevap: B

- İşletim sistemi çekirdeği, donanım kaynaklarını yönetmekten sorumludur. Bellek, işlemci, disk ve diğer donanım bileşenlerinin etkin ve güvenilir bir şekilde kullanılmasını sağlar. Çekirdek, donanım ve yazılım arasında iletişimi sağlar, giriş/çıkış işlemlerini yönetir, bellek yönetimini gerçekleştirir ve işlemciler arasında çizelgeleme yapar. Bu nedenle, işletim sistemi çekirdeği, bilgisayar sistemlerinin temel işlevselliğini sağlayan kritik bir bileşendir.

Soru

- Aşağıdakilerden hangisi işletim sisteminin bir fonksiyonu değildir?

- A) Süreç yönetimi
- B) Bellek yönetimi
- C) Ağ yönlendirme
- D) Dosya sistemi yönetimi

Cevap

- Cevap: C

- İşletim sistemi, süreç yönetimi, bellek yönetimi ve dosya sistem yönetimi gibi temel görevleri yerine getirir. Ancak, ağ yönlendirme işlevi işletim sistemi tarafından değil, ağ yönlendiricileri veya benzer ağ aygıtları tarafından gerçekleştirilir.

Soru

- İşletim sisteminde çizelgeleyicinin (*scheduler*) temel sorumluluğu nedir?

- A) Bellek kaynaklarını tahsis etmek
- B) Giriş/çıkış işlemlerini yönetmek
- C) CPU'ya atanacak bir sonraki süreci belirlemek
- D) Kullanıcı kimlik doğrulamasını sağlamak

Cevap

- Cevap: C

- Çizelgeleyici (*scheduler*), işletim sistemindeki temel görevlerden biri olan CPU zamanının etkin bir şekilde kullanılmasını sağlar. Bir sonraki çalıştırılacak süreci belirleyerek, işlemciye hangi işin verileceğini belirler. Sistemdeki farklı süreçler arasında adil bir paylaşım sağlar ve işlemcideki boş zamanı minimumda tutar.

Soru

- Aygıt sürücüsünün amacı nedir?

- A) Çevre birimlerini yönetmek
- B) Bellek kaynaklarını tahsis etmek
- C) CPU görevlerini çizelgelemek
- D) Yüksek seviye programlama dillerini yorumlamak

Cevap

- Cevap: A

- Aygıt sürücüsü, işletim sistemi ile donanım arasında iletişim sağlayan yazılım bileşenidir. Temel amacı, çeşitli donanım aygıtlarını (örneğin, yazıcı, fare, klavye, disk sürücüleri vb.) işletim sistemi ile uyumlu hale getirmek ve bu aygıtların doğru çalışmasını sağlamaktır.

Soru

- Modern bilgisayar sistemlerinde CPU önbelleğinin temel işlevi nedir?

- A) Uygulama verilerini geçici olarak depolamak
- B) CPU ve ana bellek arasındaki veri erişimini hızlandırmak
- C) Çevre birimlerini yönetmek
- D) Makine komutlarını yürütmek

Cevap

- Cevap: B

- CPU önbelleği, işlemcinin ana bellek (RAM) ile iletişimini hızlandırmak için kullanılan küçük ve yüksek hızlı bellek birimidir. Ana belleğe göre çok daha hızlıdır ve sıkça kullanılan verileri veya komutları depolayarak CPU'nun erişim sürelerini kısaltır. İşlemcinin performansını artırır ve sistem genelinde veri erişimini iyileştirir.

Soru

- Kesmeleri yönetmek ve donanım işlemlerini koordine etmekten sorumlu bileşen hangisidir?

- A) Merkezi İşlem Birimi (*CPU*)
- B) Rastgele Erişimli Bellek (*RAM*)
- C) Giriş/Çıkış Denetleyicisi (*I/O Controller*)
- D) Güç Kaynağı Ünitesi (*PSU*)

Cevap

- Cevap: C

- Giriş/Çıkış Denetleyicisi (*I/O Controller*), harici aygıtlarla iletişimi sağlayan ve kesmeleri yöneten bileşendir. Denetleyiciler, işlemcinin (CPU) yükünü azaltarak ve donanım işlemlerini koordine ederek sistem performansını artırır.

Soru

- Bilgisayarın ayağa kalkma sırasında BIOS (Temel Giriş/Çıkış Sistemi), hangi rolü oynar?

- A) Bellek kaynaklarını tahsis etmek
- B) İşletim sistemi çekirdeğini belleğe yüklemek
- C) Donanım bileşenlerini başlatmak
- D) Dosya sistemlerini yönetmek

Cevap

- Cevap: C

- BIOS (Temel Giriş/Çıkış Sistemi), bilgisayarın başlatma sürecinin ilk adımında donanım bileşenlerini başlatmak ve sistem için temel ayarları yapmakla sorumludur. İşletim sistemi yüklenmeden önce ana donanım bileşenlerinin doğru şekilde çalışmasını sağlar.

Soru

- Aşağıdakilerden hangisi modern bilgisayar sistemlerinde kullanılan bir tür kararlı bellek (*non-volatile memory*) tipi değildir?

- A) Sabit Disk Sürücüsü (*HDD*)
- B) Katı Hal Sürücüsü (*SSD*)
- C) Manyetik Bant
- D) Dinamik Rastgele Erişimli Bellek (*DRAM*)

Cevap

- Cevap: D

- Dinamik Rastgele Erişimli Bellek (DRAM), kararlı olmayan (*volatile*) bellek türündendir ve güç kesildiğinde verileri korumaz. Sabit Disk Sürücüsü (HDD), Katı Hal Sürücüsü (SSD) ve Manyetik Bant kararlı bellek türleridir ve veriler güç kaynağı kesilse bile korunur.

Soru

- CPU'yu ana belleğe bağlamak için hangi veri yol standardı kullanılır?

- A) SATA
- B) PCIe
- C) USB
- D) DDR

Cevap

- Cevap: D

- DDR (*Double Data Rate*), modern bilgisayar mimarilerinde CPU ile ana bellek arasındaki veri iletişimini sağlamak için kullanılan bir veri yol standardıdır. SATA, depolama aygıtlarını bağlamak için kullanılan arabirim standardıdır. PCIe, genişleme kartları ve diğer aygıtları bağlamak için kullanılan yüksek hızlı seri veri yoludur. USB çeşitli harici aygıtları bağlamak için kullanılan arayüz standardıdır.

Soru

- Kabuğun (*shell*) temel işlevi nedir?

- A) Donanım kaynaklarını yönetmek
- B) Çekirdeğe erişim için kullanıcı arayüzü sağlamak
- C) Çevre birimlerini kontrol etmek
- D) Makine kodu komutlarını yürütmek

Cevap

- Cevap: B

- Kabuk, kullanıcıların işletim sistemiyle etkileşime girmelerini sağlar. Kullanıcıların komutları girip çalıştırmasına, dosyaları yönetmesine ve sistem kaynaklarını kullanmasına izin verir.

Soru

- Bir dizinin içeriğini listelemek için hangi komut kullanılır?

- A) cd
- B) mv
- C) ls
- D) rm

Cevap

- Cevap: C

- "ls" komutu, bir dizinin içeriğini listelemek için kullanılır.

Soru

- GUI'nin (Grafiksel Kullanıcı Arayüzü) temel amacı nedir?

- A) Çekirdek ile etkileşim kurmak
- B) Kullanıcılar için metin tabanlı bir arayüz sağlamak
- C) Kullanıcıların grafiksel öğeler aracılığıyla etkileşimini kolaylaştırmak
- D) Donanım kaynaklarını yönetmek

Cevap

- Cevap: C

- GUI, kullanıcıların bilgisayar sistemleriyle etkileşimde bulunmasını sağlayan grafiksel bir arayüzdür. Grafiksel öğeler, pencereler, menüler, düğmeler vb. aracılığıyla kullanıcılarla etkileşim kurulmasını sağlar. Kullanıcılar bu arayüz sayesinde dosyaları yönetebilir, uygulamaları çalıştırabilir, ayarları yapılandırabilir ve daha birçok işlemi gerçekleştirebilir.

Soru

- Aygıt Yöneticisinin (*Device Manager*) temel işlevi nedir?

- A) Kullanıcı hesaplarını yönetmek
- B) Ağ trafiğini izlemek
- C) Donanım aygıtlarını yapılandırmak ve sorun gidermek
- D) CPU performansını optimize etmek

Cevap

- Cevap: C

- Aygıt Yöneticisi, işletim sisteminde donanım aygıtlarını yönetmek, yapılandırmak ve sorun gidermekten sorumludur. Bu araç, bilgisayar donanımını tanımlar, yükler ve yönetir. Donanım sürücülerini güncelleme, eksik veya hatalı donanımı tanıma ve sorunları giderme gibi işlevleri de içerir.

Soru

- Aygıt Yöneticisi bağlamında aygıt sürücülerinin amacı nedir?

- A) Aygıtları fiziksel olarak bilgisayara bağlamak
- B) Donanım bileşenleri için güç ayarlarını yönetmek
- C) Donanım aygıtlarıyla etkileşim için yazılım arayüzleri sağlamak
- D) Cihaz performansını optimize etmek

Cevap

- Cevap: C

- Aygıt sürücüleri, işletim sistemindeki Aygıt Yöneticisi tarafından kullanılan yazılım bileşenleridir. Sürücüler, donanım aygıtlarıyla iletişim kurmak, tanımak, yönetmek ve kontrol etmek için gereken yazılım arayüzlerini sağlar.

Soru

- Çekirdek (*kernel*) ile kullanıcı (*user*) modu arasındaki temel fark nedir?

- A) Çekirdek modu doğrudan donanım kaynaklarına erişime izin verirken, kullanıcı modu böyle bir erişimi kısıtlar.
- B) Kullanıcı modu, çekirdek moduna kıyasla sistem yöneticilerine daha yüksek ayrıcalıklar sağlar.
- C) Çekirdek modu, kullanıcı uygulamalarını sistem işlemlerinin önüne geçirirken, kullanıcı modu sistem işlemlerini önceliklendirir.
- D) Kullanıcı modu sistem belleğini yönetirken, çekirdek modu giriş/çıkış işlemlerini yönetir.

Cevap

- Cevap: A

- Çekirdek (*kernel*) modu, işletim sisteminin en temel ve en yüksek ayrıcalıklı modudur. Bu mod, doğrudan donanım kaynaklarına erişim sağlar ve işletim sistemi çekirdeği tarafından çalıştırılan kritik işlemleri gerçekleştirir. Kullanıcı (*user*) modu ise, kullanıcıların uygulamalarının çalıştığı moddur ve sınırlı erişim yetkilerine sahiptir.

Soru

- Doğrudan donanım kaynaklarına erişimi kısıtlayan ve güvenlik amaçları için süreç izolasyonunu sağlayan mod hangisidir?

- A) Çekirdek (*kernel*) modu
- B) Kullanıcı (*user*) modu
- C) Denetleyici (*controller*) modu
- D) Korunan (*protected*) modu

Cevap

- Cevap: B

- Kullanıcı modu (*User mode*), işletim sisteminde kullanıcıların uygulamalarının çalıştığı ve sınırlı erişim yetkilerine sahip olduğu moddur. Bu modda, doğrudan donanım kaynaklarına erişim kısıtlanır ve güvenlik amacıyla süreç izolasyonu sağlanır. Kullanıcı modunda çalışan uygulamalar, işletim sistemi tarafından belirlenen sınırlar içinde çalışır ve birbirlerinden ve işletim sistemi çekirdeğinden izole edilirler. Bu sayede, sistem güvenliği artırılır ve istenmeyen erişim ve zararlı yazılım saldırıları engellenir.

Soru

- İşletim sisteminde çekirdek ve kullanıcı modlarının kullanılmasının temel avantajı nedir?

- A) Aygıt sürücülerini yazma işlemini basitleştirmesi
- B) Kritik görevleri kullanıcı uygulamalarından izole ederek sistem istikrarını ve güvenliğini arttırması
- C) Kullanıcı programlarının daha hızlı yürütülmesine olanak tanıması
- D) Bellek yönetiminin karmaşıklığını azaltması

Cevap

- Cevap: B

- Ayrı çekirdek ve kullanıcı modları, işletim sisteminde kritik görevleri (çekirdek modu) kullanıcı uygulamalarından (kullanıcı modu) ayırarak sistem istikrarını ve güvenliğini artırır. Çekirdek modu, işletim sisteminin en temel ve en yüksek ayrıcalıklı modudur ve doğrudan donanım kaynaklarına erişim sağlar. Bu mod, kritik işlemleri gerçekleştirir ve işletim sistemi çekirdeği tarafından çalıştırılır. Kullanıcı modu ise, kullanıcıların uygulamalarının çalıştığı ve sınırlı erişim yetkilerine sahip olduğu moddur. Kullanıcı modunda çalışan uygulamalar, işletim sistemi tarafından belirlenen sınırlar içerisinde çalışır ve birbirlerinden ve işletim sistemi çekirdeğinden izole edilirler.

Soru

- Bir mikroişlemcinin iç yapısı ve tasarımını ifade eden terim hangisidir?

- A) Fiziksel Aygıt
- B) Mikro Mimari
- C) Makine Dili
- D) İşletim Sistemi

Cevap

- Cevap: B

- Mikro mimari (*Microarchitecture*), mikroişlemcinin iç yapısal özelliklerini ve tasarımını ifade eder. Bu, mikroişlemcinin içindeki işlem birimleri, veri yolları, bellek yönetimi, komut kümesi ve diğer önemli bileşenleri kapsar. Mikro mimari, mikroişlemcinin performansı, güç tüketimi, sıcaklık yönetimi gibi özelliklerini belirler.

Soru

- CPU tarafından doğrudan yürütülebilen ikili (*binary*) komutları içeren dil hangisidir?

- A) Yüksek seviyeli dil
- B) Çevirici dil
- C) Makine dili
- D) Komut dosyası dili

Cevap

- Cevap: C

- Makine dili (*Machine language*), bilgisayarın anlayabileceği en temel dil olarak bilinir. Bu dil, ikili formatta yani sıfırlar ve birler olarak yazılmış komutlar (*buyruk*) içerir. Her bir ikili kod, belirli bir işlemi veya komutu temsil eder. Makine dili, doğrudan işlemcinin yürütebileceği en temel seviyede talimatları içerir.

Soru

- Aşağıdakilerden hangisi mikro mimari'nin özelliklerinden değildir?

- A) Komut kümesi tasarımı
- B) İşlemci organizasyonu
- C) Önbellek yönetimi
- D) Dosya sistemi yapısı

Cevap

- Cevap: D

- Mikro mimarinin temel özellikleri, işlemci içindeki komut seti tasarımı, işlemci organizasyonu ve önbellek yönetimi gibi donanım düzeyindeki özelliklerdir. Dosya sistemi yapısı ise bir işletim sistemi konseptidir ve işlemci mikro mimarisi'nin bir parçası değildir.

Soru

- CPU'ya en yakın ve en küçük kapasiteye sahip olan önbellek hangisidir?

- A) L1 önbelleği
- B) L2 önbelleği
- C) L3 önbelleği
- D) Sanal Bellek

Cevap

- Cevap: A

- L1 önbellek (*Level 1 cache*), CPU'ye en yakın ve en düşük kapasiteye sahip ve en hızlı önbellek seviyesidir. L1 önbelleği, CPU çekirdeklerine doğrudan bağlıdır ve işlemcinin hızlı erişim sağlamak için kullandığı küçük ve yüksek hızlı bellek birimidir.

Soru

- Programcıların makine komutlarını temsil etmek için hatırlatıcı (*mnemonik*) kodlarını kullanmalarına izin veren dil hangisidir?

- A) Yüksek seviyeli dil
- B) Çevirici dil
- C) Makine dili
- D) Komut dosyası dili

- *mnemonik*, karmaşık kodları veya talimatları hatırlamayı kolaylaştırmak için kullanılan kısa, akılda kalıcı sembolleri veya kısaltmaları ifade eder.

Cevap

- Cevap: B

- Çevirici dil (*Assembly language*), programcıların makine buyruklarını temsil etmek için kullanabilecekleri mnemonik kodları içeren düşük seviyeli programlama dilidir. Bu mnemonik kodlar, makine dilindeki talimatları daha anlaşılır hale getirir. Çevirici dil, bir derleyici veya çevirici aracılığıyla makine diline çevrilir ve bilgisayar tarafından doğrudan yürütülebilir.

Soru

- İşletim sistemleri bağlamında soyutlama nedir?

- A) Bilgisayar sistemlerinin fiziksel bileşenlerine işaret eder.
- B) Karmaşık detayları gizlemeyi sağlar.
- C) CPU performansını optimize etmeye odaklanır.
- D) Donanım ve yazılım arasındaki etkileşimle ilgilidir.

Cevap

- Cevap: B

- Soyutlama, karmaşık detayları gizlemeyi ve kullanıcılara ve uygulamalara kaynakların basitleştirilmiş bir görünümünü sağlamayı içerir. Bu, kullanıcıların ve uygulamaların işletim sistemi kaynaklarına erişimini yönetmeyi ve karmaşık yapıları gizlemeyi içerir. Örneğin, dosya sistemi soyutlaması, kullanıcılara dosyaları ve dizinleri yönetmek için bir arayüz sağlar ve fiziksel depolama cihazlarının karmaşık detaylarını gizler.

Soru

- İşletim sistemlerinde soyutlama, kullanıcılara ve geliştiricilere ne sağlar?

- A) Donanım mimarisine detaylı bakış açısı
- B) Basitleştirilmiş ve tutarlı arayüzler
- C) Düşük seviyeli sistem kaynaklarına erişim
- D) Gerçek zamanlı performans izleme

Cevap

- Cevap: B

- Soyutlama, işletim sistemlerinde kullanıcılara ve geliştiricilere basitleştirilmiş ve tutarlı arayüzler sağlar. Bu, kullanıcıların ve geliştiricilerin karmaşık detaylarla uğraşmadan kaynaklara erişmelerini ve işletim sistemi hizmetlerinden yararlanmalarını sağlar. Örneğin, bir dosya sistemi arayüzü, kullanıcılara dosyaları yönetmek için basit ve tutarlı bir yol sunar, ancak bu, dosyaların fiziksel olarak nasıl depolandığına veya yönetildiğine dair ayrıntıları gizler.

Soru

- Uygulama programlarından fiziksel bellek konumlarını gizleyen soyutlama hangisidir?

- A) CPU çizelgeleme
- B) Süreç yönetimi
- C) Sanal bellek
- D) Dosya sistemi

Cevap

- Cevap: C

- Sanal bellek, uygulama programlarına fiziksel bellek konumlarını gizleyen bir soyutlamadır. Bu, uygulama programlarının bellek üzerinde çalışırken fiziksel bellek adreslerini doğrudan kullanmasını engeller. Her bir uygulama programı, kendine ayrılmış sanal bellek adreslerini kullanır ve işletim sistemi bu sanal adresleri fiziksel bellek adreslerine eşler. Bu sayede, uygulama programlarının birbirlerini veya işletim sistemini etkilemeden bellek üzerinde çalışması sağlanır.

Soru

- Tek bir CPU üzerinde birden fazla sürecin eş zamanlı olarak çalışmasına olanak tanıyan soyutlama hangisidir?

- A) Dosya sistemi
- B) Süreç yönetimi
- C) Aygıt sürücüleri
- D) Ağ protokolleri

Cevap

- Cevap: B

- Süreç yönetimi, tek bir CPU üzerinde birden fazla sürecin eş zamanlı olarak çalışmasına olanak tanır. İşletim sistemi, süreç yönetimi aracılığıyla süreçleri oluşturur, yönetir, zamanında yürütür ve sonlandırır. Bu, birden çok sürecin aynı anda çalışmasını ve CPU kaynaklarının etkin bir şekilde kullanılmasını sağlar.

Soru

- İşletim sistemlerinde dosya sistemi soyutlamasının amacı nedir?

- A) Ağ bağlantılarını yönetmek
- B) Verileri depolamak ve verilere erişmek için arayüz sağlamak
- C) CPU kaynaklarını kontrol etmek
- D) Bellek tahsisini optimize etmek

Cevap

- Cevap: B

- Dosya sistemi soyutlaması, işletim sistemi tarafından verileri depolamak ve verilere erişmek için birleşik bir arayüz sağlar. Bu soyutlama, kullanıcıların ve uygulamaların dosyaları oluşturmasını, okumasını, yazmasını ve silmesini sağlar. Ayrıca, dosya sistemi, dosyaları organize eder, depolama aygıtlarına erişimi yönetir ve dosyaları korur. Bu sayede, kullanıcılar ve uygulamalar, dosyaları işlemek için dosya sistemi aracılığıyla tek bir arayüz kullanırlar.

Soru

- Aygıtlar ve CPU arasındaki giriş/çıkış işlemlerini yönetmekten sorumlu soyutlama hangisidir?

- A) Süreç yönetimi
- B) CPU çizelgeleme
- C) Aygıt sürücüleri
- D) Sanal bellek

Cevap

- Cevap: C

- Aygıt sürücüleri, işletim sisteminde aygıtlarla (örneğin, yazıcı, klavye, disk sürücüleri) ve CPU arasındaki giriş/çıkış işlemlerini yönetmekten sorumludur. Aygıt sürücüleri, işletim sistemi ve donanım arasında arabirim görevi görür ve kullanıcı programlarına veya işletim sistemine gelen giriş/çıkış taleplerini işler.

Soru

- Kullanıcıların depolama aygıtlarına depolanan verileri düzenlemesine ve işlemesine olanak tanıyan soyutlama hangisidir?

- A) CPU çizelgeleme
- B) Sanal bellek
- C) Dosya sistemi
- D) Süreç yönetimi

Cevap

- Cevap: C

- Dosya sistemi, kullanıcıların depolama aygıtlarında depolanan verileri düzenlemesine ve işlemesine olanak tanır. Dosya sistemi, dosyaları organize eder, depolama aygıtlarına erişimi yönetir ve dosyaları korur. Kullanıcılar ve uygulamalar, dosyaları oluşturabilir, okuyabilir, yazabilir, silip yeniden adlandırabilir veya taşıyabilirler.

Soru

- Çevre birimi (*peripheral*) aygıtların fiziksel özelliklerini uygulama yazılımından gizleyen soyutlama hangisidir?

- A) Sanal bellek
- B) CPU çizelgeleme
- C) Aygıt sürücüleri
- D) Dosya sistemi

Cevap

- Cevap: C

- Aygıt sürücüleri, çevre birimi cihazların fiziksel özelliklerini uygulama yazılımından gizler. Bu, uygulama yazılımının belirli bir aygıtla iletişim kurarken, cihazın fiziksel yapısı veya bağlantı protokolleri gibi detaylarla uğraşmasını engeller. Aygıt sürücüleri, işletim sistemi ve donanım arasında arabirim görevi görür. Bu sayede, uygulama yazılımı, aygıt türüne özgü detaylarla uğraşmak zorunda kalmaz ve farklı aygıtlar arasında kolayca geçiş yapabilir.

Soru

- Linux'ta *proc* dizininin amacı nedir?

- A) Sistem ikili dosyalarını içerir.
- B) Kullanıcı uygulamalarının yapılandırma dosyalarını saklar.
- C) Çalışan süreçler ve sistem kaynakları hakkında bilgi sağlar.
- D) Kullanıcı ana dizinlerine ev sahipliği yapar.

Cevap

- Cevap: C

- *proc* dizini, Linux işletim sisteminde çalışan süreçler ve sistem kaynakları hakkında bilgi sağlar. Bu dizin altında, süreç numaraları, bellek kullanımı, CPU kullanımı, ağ istatistikleri gibi birçok sistem ve süreç bilgisi yer alır. *proc*, gerçekte bir dizin değil, çalışan süreçler ve sistem bilgilerini dinamik olarak oluşturan bir sanal dizindir.

Soru

- *proc* dizinindeki */proc/sys* alt dizini neyi içerir?

- A) Çeşitli sistem hizmetleri için yapılandırma dosyalarını
- B) Çalışan süreçler hakkında bilgi
- C) Çekirdek parametreleri ve ayarlar
- D) Kullanıcı ana dizinlerini

Cevap

- Cevap: C

- */proc/sys* alt dizini, Linux işletim sistemi çekirdeğiyle ilgili parametreleri ve ayarları içerir. Bu alt dizin altında, çeşitli sistem ayarlarına ilişkin dosyalar bulunur ve bu dosyalar, sistem davranışını değiştirmek için kullanılır. Örneğin, bellek yönetimi, ağ yapılandırması, güvenlik ayarları gibi çeşitli sistem parametreleri bu dizin altında bulunur.

Soru

- *proc* dizinindeki */proc/[pid]* dizinlerinin amacı nedir?

- A) Bağlanmış dosya sistemleri hakkında bilgi içerirler.
- B) Süreç kimlikleri (PIDs) tanımlanan süreçler hakkında bilgi sağlarlar.
- C) Sistem günlüklerini saklarlar.
- D) Sistem genelinde çekirdek parametrelerini içerirler.

Cevap

- Cevap: B

- */proc/[pid]* dizinleri, süreç kimlikleri (PIDs) tanımlanan süreçler hakkında ayrıntılı bilgi sağlar. Her bir */proc/[pid]* dizini, o süreçle ilgili çeşitli bilgiler içerir. Örneğin, bellek kullanımı, CPU kullanımı, dosya tanımlayıcıları, süreç durumu gibi bilgiler bu dizin altında tutulur. Sistem üzerinde çalışan her süreç için ayrı bir dizin oluşturur ve bu dizinler aracılığıyla sistem tarafından süreç hakkında bilgi alınabilir veya etkileşimde bulunulabilir.

Soru

- *proc* dizinindeki hangi dosya, çekirdek sürümü ve derleme parametreleri hakkında bilgi verir?

- A) /proc/version
- B) /proc/net
- C) /proc/mounts
- D) /proc/filesystems

Cevap

- Cevap: A

- */proc/version* dosyası, Linux çekirdeğinin sürüm numarası ve derleme parametreleri gibi bilgileri içerir.

Soru

- *Von Neumann* mimarisinde, komutlar ve veriler nerede saklanır?

- A) Ayrı bellek birimlerinde
- B) CPU önbelleğinde
- C) Yazmaçlarda
- D) Aynı bellek biriminde

Cevap

- Cevap: D

- *Von Neumann* mimarisinde, komutlar ve veriler aynı bellek biriminde saklanır. Bu bellek birimi, hem program kodunu (talimatlar) hem de programın çalıştığı verileri içerir.

Soru

- *Von Neumann* mimarisinin hangi yönü, performans açısından potansiyel bir darboğaz oluşturur?

- A) Ayrı depolama ve işlem birimleri
- B) Entegre grafik işleme
- C) Paralel işleme yeteneği
- D) Bellek ile CPU arasındaki sınırlı bant genişliği

Cevap

- Cevap: D

- *Von Neumann* mimarisinde, bellek ve işlemci arasındaki iletişim bellek ile işlemci arasındaki bant genişliği ile sınırlıdır. Bu, işlemcinin bellekten veri okuma ve yazma işlemlerini gerçekleştirirken verimlilik açısından bir darboğaz oluşturabilir. Çünkü işlemci, belleğe erişmek için belirli bir bant genişliğine sahiptir ve bu bant genişliği diğer sistem bileşenleriyle (örneğin, diğer işlemciler, giriş/çıkış aygıtları) de paylaşılabilir.

Soru

- *Von Neumann* mimarisindeki saklanmış program (*stored program*) kavramının önemi nedir?

- A) Donanım bileşenleriyle doğrudan etkileşim sağlar.
- B) Komutların veri olarak saklanmasına ve işlenmesine izin verir.
- C) Giriş/çıkış aygıtlarına ihtiyacı ortadan kaldırır.
- D) CPU çalışmasını hızlandırır.

Cevap

- Cevap: B

- Saklanmış program kavramı, *Von Neumann* mimarisindeki önemli bir kavramdır çünkü bu mimaride talimatlar ve veriler aynı bellek biriminde saklanır. Bu, talimatların bellekte veri olarak saklanmasına ve işlenmesine izin verir. Yani, programın kendisi de bellekte veri olarak saklanabilir ve işlenebilir. Bu, programın çalışma zamanında değiştirilebileceği ve programların diğer programlar tarafından manipüle edilebileceği anlamına gelir. Bu da esneklik ve programların dinamik olarak oluşturulması ve değiştirilmesi açısından büyük bir önem taşır.

Soru

- Kesmenin (*interrupt*) temel amacı nedir?

- a) Mevcut sürecin yürütmesini durdurmak
- b) Bir sürecin CPU kontrolünü isteğe bağlı olarak bırakmasına izin vermek
- c) Asenkron olayları işlemek ve önceliklendirmek
- d) Süreçler arasında iletişimi kolaylaştırmak

Cevap

- Cevap: C

- İşletim sistemi, birçok görevi aynı anda yönetmek zorundadır. Bu görevler arasında kullanıcının klavyeden bir tuşa basması, ağa veri alıp gönderme, diskten veri okuma/yazma gibi işlemler bulunur. Bu tür olaylar "asenkron" olarak adlandırılır, çünkü işlemciden bağımsız ve önceden belirlenemeyen zamanlarda gerçekleşir. Bir kesme, işletim sisteminin normal akışını keserek asenkron olaylara müdahale etmesini ve önceliklerini belirlemesini sağlar. Örneğin, bir klavyeden bir tuşa basıldığında, klavye kesmesi (*keyboard interrupt*) oluşur ve işletim sistemi bu kesmeyi işler, klavyeden gelen veriyi okur ve uygun işlemi gerçekleştirir.

Soru

- Kesmelerin (*interrupts*) öncelikli olduğu bir önleyici (*preemptive*) çoklu görevli sistemde işletim sistemi, daha yüksek öncelikli bir kesme aldığında hangi eylemi gerçekleştirir?

- a) Mevcut süreci tamamlar ve sonra kesmeyi işler
- b) Hemen kesme işleyicisinin yürütülmesine geçer
- c) Kesmeyi bir kuyruğa yerleştirir ve mevcut süreçten sonra işler
- d) Mevcut süreç tamamlanana kadar kesmeyi görmezden gelir

Cevap

- Cevap: B

- Öncelikli bir kesme geldiğinde, işletim sistemi hemen mevcut süreci duraklatır ve kesme işleyicisinin yürütülmesine geçer. Bu, yüksek öncelikli bir olayın hızlı bir şekilde işlenmesini sağlar ve sistemin daha duyarlı olmasını sağlar. Örneğin, bir acil durum alarmı gibi kritik bir olay gerçekleştiğinde, işletim sistemi diğer işlemleri askıya alarak bu olayı derhal işler. Bu, sistem performansını artırır ve kritik işlevlerin zamanında gerçekleştirilmesini sağlar.

Soru

- Aşağıdaki senaryolardan hangisi maskelenemez kesme (*non-maskable interrupt - NMI*) örneğidir?

- a) Bir tuşa basılarak oluşturulan klavye kesmesi
- b) Zaman diliminin sona ermesiyle tetiklenen zamanlayıcı kesmesi
- c) Sistem tarafından algılanan kritik donanım hatası
- d) Bir sistem çağrısı ile çağrılan yazılım kesmesi

Cevap

- Cevap: C

- Maskelenemez kesme (*Non-maskable interrupt*), kritik durumları işlemek için kullanılır ve donanım hatası gibi ciddi durumlar için ayrılmıştır. Bu tür bir kesme, diğer normal kesmelerden farklı olarak, işletim sistemi veya kullanıcı tarafından engellenemez veya durdurulamaz. Bu tür hatalar genellikle sistem bütünlüğünü tehlikeye atan veya ciddi hizmet kesintilerine neden olabilecek aygıt arızalarını ifade eder.

Soru

- İşletim sistemleri kesme işlemlerini yönetmek için hangi mekanizmayı kullanır?

- a) Sorgulama (*polling*)
- b) Vektörlendirme (*vectorized*)
- c) Round-robin çizelgeleme
- d) Öncelik ters çevirme (*priority inversion*)

Cevap

- Cevap: B

- İşletim sistemleri, kesme oluştuğunda hangi işlem kodunun çalıştırılacağını belirlemek için vektörlendirme mekanizmasını kullanır. Bu, kesmelerin işlenmesinde verimlilik ve doğruluk sağlar. Vektör tablosu kesmelerin tanımlandığı ve ilgili işlem kodlarının bulunduğu bir tablodur. Bu sayede, bir kesme oluştuğunda işletim sistemi, vektör tablosundaki girdiye karşılık gelen işlem kodunu bulur ve çalıştırır.

Soru

- Vektörlü kesme işleme şemasında, kesme işleyicisi nasıl belirlenir?

- a) Her bir kesmeye atanan sabit bir öncelikle
- b) Kesme numarası tarafından indekslenen bir işaretçi tablosu ile
- c) Kesmelerin alındığı sıraya göre
- d) Kesme denetleyici donanımı tarafından

Cevap

- Cevap: B

- Vektörlü kesme işleme şemasında, bir kesme işleyicisinin belirlenmesi için, kesme numarasına göre indekslenen bir işaretçi tablosuna başvurulur. Bu tablo, her kesme için bir işaretçiye sahip olup, her bir işaretçi kesmenin işleyicisinin konumunu gösterir. Bir kesme oluştuğunda, işletim sistemi bu tabloya bakarak ilgili işaretçiyi bulur ve bu işaretçinin gösterdiği adrese giderek ilgili kesme işleyicisini başlatır. Bu sayede, her bir kesme için farklı işleyiciler belirlenebilir ve etkin bir şekilde yönetilebilir.

Soru

- Kesme (*interrupt*) ile tuzak (*trap*) arasındaki fark nedir?

- a) Tuzaklar asenkron olaylar, kesmeler ise senkrondur.
- b) Tuzaklar işletim sistemi, kesmeler aygıt sürücüleri tarafından işlenir.
- c) Kesmeler kontrolü harici bir aygıta, tuzaklar işletim sistemine aktarır.
- d) Kesmeler donanım tarafından, tuzaklar yazılım tarafından başlatılır.

Cevap

- Cevap: D

- Kesmeler, harici bir donanımın işlemcinin dikkatini çektiği ve işlemcinin normal akışını geçici olarak keserek kesmeyi işlemesi gerektiği durumlarda donanım tarafından başlatılır. Tuzaklar ise yazılım tarafından bilerek ve isteyerek oluşturulan özel işaretlerdir. Bir programın yürütmesi sırasında hata durumunda veya belirli bir koşul gerçekleştiğinde, yazılım tuzakları kullanarak işletim sistemine veya belirli bir sürece kontrolü aktarabilir. Bu nedenle, kesmeler donanımın etkileşimiyle ilgili iken, tuzaklar yazılım tarafından başlatılır ve işletim sistemi tarafından kullanılır.

Soru

- Kesme Servis Rutini'nin (ISR) amacı nedir?

- a) Eş zamanlı olarak birden fazla sürecin yürütülmesini yönetmek
- b) Kesmelerin tespit ve onaylanmasını işlemek
- c) Sanal bellek adreslerini fiziksel bellek adreslerine çevirmek
- d) Donanım aygıtları arasında iletişim için arayüz sağlamak

Cevap

- Cevap: B

- Kesme Servis Rutini (ISR), bir kesme gerçekleştiğinde işletim sistemi tarafından çağrılan özel kod parçasıdır. Bu rutin, kesmenin algılanmasını ve onaylanmasını yönetir. Kesme algılandığında, işletim sistemi, ilgili ISR'yi çağırarak kesmeyi işler ve gereken aksiyonu alır. Örneğin, bir kesme, bir aygıtın hazır olduğunu veya bir hata durumunun meydana geldiğini bildirebilir ve ISR, bu durumu ele almak için gereken işlemleri gerçekleştirebilir.

Soru

- Doğrudan Bellek Erişimini (DMA) en iyi tanımlayan ifade hangisidir?

- a) İşletim sistemini dahil etmeden CPU'nun doğrudan belleğe erişmesine izin verir.
- b) Çevre birimlerinin CPU müdahalesi olmadan bellek arasında veri aktarımına izin verir.
- c) Modern işletim sistemlerinde sanal bellek yönetimi için bir mekanizmadır.
- d) İşletim sistemi içinde süreçler arası iletişim için kullanılır.

Cevap

- Cevap: B

- Doğrudan Bellek Erişimi (DMA), çevre birimlerinin (örneğin, disk sürücüsü, ağ kartı) bellek iletişimini yönetmek için kullanılır. DMA, CPU'nun iş yükünü azaltarak, çevre birimlerinin bellekten veri okuması veya belleğe veri yazması gibi işlemleri doğrudan gerçekleştirmesine izin verir. Bu sayede, CPU'nun bu tür veri transferlerini denetlemesi gerekmez, CPU işlem gücünü diğer görevlere odaklayabilir.

Soru

- Aşağıdakilerden hangisi DMA kullanmanın faydalarından değildir?

- a) Azaltılmış CPU iş yükü
- b) Artırılmış veri transfer hızı
- c) Bellek erişimi için artırılmış güvenlik
- d) İyileştirilmiş sistem tepkiselliği

Cevap

- Cevap: C

- DMA kullanmanın avantajları arasında, azaltılmış CPU iş yükü, artırılmış veri transfer hızı ve iyileştirilmiş sistem tepkiselliği bulunmaktadır. Ancak, DMA'nın bellek erişimi için artırılmış güvenlik sağladığı söylenemez. DMA, doğrudan çevre birimlerinin belleğe erişmesine izin verirken, bu durum bazı güvenlik risklerine neden olabilir. Özellikle kötü amaçlı yazılımların, DMA aracılığıyla belleğe doğrudan erişim sağlaması ve hassas verilere zarar vermesi gibi durumlar söz konusu olabilir.

Soru

- Sistemde DMA transferlerini başlatan bileşen hangisidir?

- a) Merkezi İşlem Birimi (CPU)
- b) Çevre birimleri
- c) DMA denetleyici
- d) Bellek Yönetim Birimi (MMU)

Cevap

- Cevap: B

- DMA transferlerini çevre birimleri başlatır. DMA, çevre birimlerinin doğrudan belleğe erişmesine olanak tanır, bu nedenle çevre birimleri DMA transferlerini başlatır ve kontrol eder. DMA denetleyicisi, bu transferleri yönetir ve koordine eder.

Soru

- Ortalama dönüş süresini minimize etmek için tasarlanmış olan CPU çizelgeleme algoritması hangisidir?

- a) Round Robin
- b) En Kısa İşlem Önce (*SJN*)
- c) İlk Gelen İlk Hizmet (*FCFS*)
- d) En Kısa Kalan Süre İlk (*SRTF*)

Cevap

- Cevap: D

- En Kısa Kalan Süre İlk (*SRTF*) CPU çizelgeleme algoritması, mevcut süreçler arasından sonlanması için en kısa süresi kalan süreci seçer. Bu şekilde, ortalama dönüş süresi minimize edilir, çünkü her zaman en kısa süreli süreç öncelikli olarak seçilir ve daha kısa sürede tamamlanır.

Soru

- Önleyici (*preemptive*) CPU çizelgelemenin temel amacı nedir?

- a) CPU kullanımını maksimize etmek
- b) Etkileşimli (*interactive*) görevler için yanıt süresini minimize etmek
- c) İşlemler arasında CPU tahsisinde adil olmayı sağlamak
- d) Bağlam anahtarlama işleminden kaynaklanan iş yükünü minimize etmek

Cevap

- Cevap: B

- Önleyici (*preemptive*) CPU çizelgeleme, öncelikle kullanıcı etkileşimli görevler için yanıt süresini minimize etmeyi amaçlar. Bu, kullanıcıların işlemlere hızlı şekilde yanıt almasını sağlar. Süreçler önceliklerine göre kesmelerle (*interrupts*) kontrol edilir ve işlemcideki yürütülen süreç sırası sık sık değişebilir.

Soru

- Hangisi CPU çizelgeleme algoritmalarının performansını etkilemez?

- a) Sistemdeki CPU sayısı
- b) CPU-yoğun ve G/Ç-yoğun işlemlerin karışımı
- c) Sistem belleğinin boyutu
- d) Bağlam anahtarlama işleminden kaynaklanan iş yükü

Cevap

- Cevap: C

- CPU çizelgeleme algoritmalarının performansını etkileyen faktörler arasında, sistemdeki CPU sayısı, CPU-yoğun ve G/Ç-yoğun işlemlerin karışımı ve bağlam anahtarlama işleminden kaynaklanan ek iş yükü gibi faktörler bulunur. Ancak, sistem belleğinin boyutu, çizelgeleme algoritmalarının performansını doğrudan etkilemez.

Soru

- İlk Gelen İlk Hizmet (*FCFS*) algoritmasının başlıca dezavantajı nedir?

- a) Ortalama bekleme süresinin yüksek olması
- b) Zayıf CPU kullanımı
- c) Düşük öncelikli süreçlerin belirsiz bir süre bloke olması
- d) Önemli süreçlere öncelik verememe

Cevap

- Cevap: A

- İlk Gelen İlk Hizmet (FCFS), gelen süreçleri sırayla işlemek için kullanılan basit bir algoritmadır. Ancak, bu yaklaşımın başlıca dezavantajı, süreçlerin işlemciye geliş sırasına göre işlenmesi nedeniyle yüksek ortalama bekleme süresidir. Öncelikli olan süreçler, öncelikli olmayan işlemlerin tamamlanmasını beklemek zorunda kalır.

Soru

- CPU görev dağıtıcısının (*dispatcher*) temel amacı nedir?

- a) CPU kaynaklarını süreçlere tahsis etmek
- b) Çevre birimleri tarafından oluşturulan kesmeleri işlemek
- c) Sistem çağrılarının yürütülmesini yönetmek
- d) Süreçler arasında paylaşılan kaynaklara erişimi senkronize etmek

Cevap

- Cevap: A

- CPU görev dağıtıcısı, işletim sistemi içinde bir bileşen ve temel amacı CPU kaynaklarını süreçlere tahsis etmektir. Bir süreç hazır duruma geldiğinde veya bir süreç tamamlandığında, CPU görev dağıtıcısı bu süreçlere CPU kaynaklarını tahsis eder. İşletim sisteminin çoklu işlem (*multiprocessing*) yeteneklerini yönetmesine ve süreçleri adil bir şekilde CPU üzerinde çalıştırmasına olanak tanır.

Soru

- Önbelleğin (*cache*) temel amacı nedir?

- a) Daha hızlı erişim için sık erişilen veri ve komutları saklamak
- b) Bellek yoğun uygulamalar için ek bellek sağlamak
- c) Disk G/Ç işlemleri için bir arabellek (tampon) görevi görmek
- d) Süreçler arası iletişimi kolaylaştırmak

Cevap

- Cevap: A

- Önbellek, işletim sistemlerinde sık erişilen veri ve komutları saklar. Önbellek, ana belleğe kıyasla daha hızlı erişilebilir ve işlemciye daha yakın bir konumdadır. Bu nedenle, sık sık kullanılan veri ve komutlar önbelleğe kopyalanır, işlemci bu verilere daha hızlı erişebilir.

Soru

- Tipik olarak en küçük kapasiteye ve en hızlı erişim süresine sahip önbellek türü hangisidir?

- a) L1 önbelleği
- b) L2 önbelleği
- c) L3 önbelleği
- d) Ana bellek

Cevap

- Cevap: A

- Önbellek hiyerarşisinde, L1 önbelleği en küçük kapasiteye ve en hızlı erişim süresine sahip olan seviyedir. L1 önbelleği, işlemcinin hemen yanında yer alır ve çok hızlı bir şekilde erişilir. Ancak, küçük kapasiteye sahiptir, çünkü önbellek içine sığdırılabilecek veri miktarı sınırlıdır. Diğer önbellek seviyeleri (L2 ve L3) daha büyük kapasiteye ve daha yüksek erişim sürelerine sahiptir.

Soru

- Çok işlemcili sistemlerde önbellek tutarlılığı ile ilişkili temel zorluk nedir?

- a) Önbellek seviyeleri arasındaki veri tutarlılığını sağlamak
- b) Önbellek erişim gecikmesini en aza indirmek
- c) Önbellek yerine koyma politikalarını optimize etmek
- d) Önbellek boyutu ile hız arasındaki dengeyi sağlamak

Cevap

- Cevap: A

- Çok işlemcili sistemlerde, her işlemci kendi önbelleğine sahiptir ve bu önbellekler arasındaki veri tutarlılığını sağlamak zordur. Birden fazla işlemci aynı bellek hücresine erişirse ve bu hücredeki veriyi değiştirirse, diğer işlemcilerin kendi önbelleklerinde bu değişikliği fark etmeleri ve güncelleme yapmaları gerekir.

Soru

- İşletim sistemi önyükleme (*boot*) sürecinde önyükleyicinin (*bootloader*) rolü nedir?

- a) Donanım bileşenlerini başlatmak
- b) İşletim sistemi çekirdeğini belleğe yüklemek
- c) Kullanıcıya özgü ayarları yapılandırmak
- d) Disk bölümlerini yönetmek

Cevap

- Cevap: B

- Önyükleyicinin temel görevi, işletim sistemi çekirdeğini diskten belleğe yüklemektir. Bilgisayar başlatıldığında, önyükleyici (*bootloader*), işletim sistemi çekirdeğini sabit diskten okur ve belleğe yükler. Önyükleme işlemi, donanımı başlatmak, önyükleme aygıtlarını tanımlamak ve işletim sistemi çekirdeğini yüklemek gibi adımları içerir. Ancak, önyükleyici işletim sistemi çekirdeğini belleğe yüklemekle sınırlıdır ve diğer sistem yapılandırma veya kullanıcı ayarları işlemleri için kullanılmaz. Bu görevler, işletim sistemi tarafından veya başlatma sonrası betikler (*script*) tarafından gerçekleştirilir.

Soru

- Önyükleme sürecinde Master Boot Record (*MBR*)'in amacı nedir?

- a) İşletim sistemi çekirdeğini saklamak
- b) Önyükleme yükleyici programını yüklemek
- c) Diski mantıksal birimlere bölmek
- d) Donanım kesmelerini yönetmek

Cevap

- Cevap: B

- *Master Boot Record (MBR)*, önyükleme yükleyici programını yüklemek için kullanılan bir alandır. MBR, sabit diskin ilk sektöründe bulunur ve disk üzerindeki bölümleri ve bu bölümlerin dosya sistemlerini tanımlar. MBR, bilgisayarın önyükleme işlemi sırasında ilk olarak yüklenen kod parçasıdır. MBR, önyükleme yükleyici programını yükler ve bu program, işletim sistemi çekirdeğini diskten belleğe yükler.

Soru

- Önyükleme sürecinin *POST* (*Power-On Self-Test*) aşamasında ne olur?

- a) İşletim sistemi çekirdeği belleğe yüklenir
- b) Donanım bileşenleri başlatılır ve test edilir
- c) Kullanıcıya özgü yapılandırma ayarları uygulanır
- d) Disk bölüm bilgileri *Master Boot Record* (*MBR*)'den okunur

Cevap

- Cevap: B

- Önyükleme sürecinin *POST* (*Power-On Self-Test*) aşaması, bilgisayarın donanım bileşenlerinin başlatılması ve test edilmesi için gereklidir. Bu aşamada, sistem belleği, işlemci, sabit disk, grafik kartı ve diğer donanım bileşenleri kontrol ve test edilir. *POST*, bilgisayarın başlangıç noktasıdır ve donanımın düzgün çalışıp çalışmadığını belirlemek için önemlidir.

Soru

- Linux tabanlı işletim sistemlerinin önyükleme sürecinde *GRUB* (*Grand Unified Bootloader*)'un rolü nedir?

- a) Donanım bileşenlerini başlatır
- b) İşletim sistemi çekirdeğini belleğe yükler
- c) Disk bölümlendirme ve biçimlendirme işlemlerini yönetir
- d) Farklı işletim sistemi yapılandırmalarını seçmek için menü sağlar

Cevap

- Cevap: B

- *GRUB* (*Grand Unified Bootloader*), Linux tabanlı işletim sistemlerinde önyükleme sürecinde işletim sistemi çekirdeğini belleğe yüklemekten sorumlu olan önyükleme yükleyici programıdır. *GRUB*, bilgisayar başlatıldığında yüklenir ve kullanıcıya seçenekler sunarak hangi işletim sisteminin yükleneceğini seçmesine izin verir. Seçilen işletim sistemi çekirdeği daha sonra belleğe yüklenir ve işletim sistemi başlatılır.

Soru

- Önyükleme sürecinde çekirdek yükleyicisinin amacı nedir?

- a) Donanım bileşenlerini başlatmak
- b) Kullanıcıya özgü ayarları yapılandırmak
- c) Aygıt sürücülerini ve sistem modüllerini belleğe yüklemek
- d) Disk bölümlendirme ve biçimlendirme işlemlerini yönetmek

Cevap

- Cevap: C

- Çekirdek yükleyicisi, önyükleme sürecinde, işletim sistemi çekirdeği tarafından kullanılan aygıt sürücülerini ve sistem modüllerini belleğe yükler. İşletim sisteminin gerekli donanım sürücülerini yüklemesini sağlar ve işletim sisteminin kullanıma hazır hale gelmesini sağlar.

Soru

- İşletim sistemlerinde monolitik çekirdek yapısının özelliği nedir?

- a) Çekirdek bileşenlerini tekil kullanıcı alanı (*user space*) süreçlerine ayırır.
- b) Bileşenler arasında iyi tanımlanmış arayüzler sağlar.
- c) Tüm işletim sistemi işlevselliğini büyük bir ikili dosyada birleştirir.
- d) Süreçler arası iletişim için mikro çekirdek prensiplerine dayanır.

Cevap

- Cevap: C

- Monolitik çekirdek yapısı, işletim sistemi işlevselliğini tek bir büyük ikili dosyada birleştirir. Bu, çekirdeğin temel işlevlerinin, sürücülerin, sistem çağrılarının ve diğer işletim sistemi bileşenlerinin tümünün aynı ikili dosyada bulunduğu anlamına gelir. Bu yaklaşım, çekirdeğin işlevselliğini konsolide eder ve çekirdeğin boyutunu artırır. Monolitik çekirdek, modüler bir yapıya sahip değildir, birçok işlevi doğrudan çekirdeğin içine entegre eder.

Soru

- Monolitik çekirdek mimarisinde, aygıt sürücüleri nasıl uygulanır?

- a) Ayrı kullanıcı alanı süreçleri olarak
- b) Dinamik olarak yüklenebilir çekirdek modülleri olarak
- c) Bağımsız uygulamalar olarak
- d) Çekirdek ikili dosyanın bir parçası olarak

Cevap

- Cevap: D

- Monolitik çekirdek mimarisinde, aygıt sürücüleri çekirdeğin parçası olarak uygulanır. Aygıt sürücüleri doğrudan çekirdek koduna dahil edilir. Aygıt sürücüleri, işletim sisteminin diğer temel bileşenleriyle birlikte tek bir büyük ikili dosyada bulunur ve tüm çekirdek işlevselliği bir arada yürütülür. Bu yaklaşım, aygıt sürücülerinin çekirdek içinde doğrudan erişilebilir ve hızlı çalışmasını sağlar. Çekirdek boyutunu artırır ve çekirdeğin değiştirilmesini veya güncellenmesini zorlaştırır.

Soru

- Monolitik çekirdek, kullanıcı alanı (*user space*) uygulamalarından gelen sistem çağrılarını nasıl işler?

- a) Sistem çağrı işleme görevini ayrı kullanıcı alanı süreçlerine devrederek
- b) Çekirdek adres alanı içinde sistem çağrısı işleyicilerini çağırarak
- c) Çekirdek modülleri arasında mesaj iletişimi mekanizması kullanarak
- d) Sistem çağrılarını ayrı bir mikro çekirdek bileşenine yönlendirerek

Cevap

- Cevap: B

- Monolitik çekirdek, kullanıcı alanı uygulamalarının yaptığı sistem çağrılarını doğrudan çekirdek adres alanı içindeki sistem çağrısı işleyicilerini çağırarak işler. Sistem çağrısı işleyicileri, kullanıcı uygulamalarının taleplerini alır, uygun çekirdek işlevlerini çağırır ve sonuçları uygun şekilde işler. Bu yaklaşım, monolitik çekirdeğin hızlı ve verimli çalışmasını sağlar, çünkü sistem çağrıları doğrudan çekirdek içinde gerçekleştirilir ve gereksiz ara işlemlere gerek kalmaz.

Soru

- İşletim sistemlerinde mikro çekirdek yapısının belirleyici özelliği nedir?

- a) Tüm işletim sistemi işlevselliğini büyük bir ikili dosyada birleştirir.
- b) Aygıt sürücülerinin çekirdek alanı içinde diğer bileşenler ile birlikte çalışmasına izin verir.
- c) Çekirdek boyutunu en aza indirerek yalnızca temel işlevleri çekirdek alanında (*Kernel space*) uygular.
- d) Süreçler arası verimli iletişim için monolitik bir mimariye dayanır.

Cevap

- Cevap: C

- Mikro çekirdek yapısının belirleyici özelliği, çekirdek boyutunu en aza indirerek yalnızca temel işlevleri çekirdek alanında uygulamasıdır. Mikro çekirdek, işletim sistemi işlevselliğini mümkün olduğunca küçük bir çekirdek içine yerleştirir ve daha fazla işlevselliği kullanıcı alanına veya çekirdek dışı modüllere taşır. Bu, çekirdeğin boyutunu azaltır ve işletim sistemi daha modüler hale gelir. Bileşenler daha bağımsız olarak geliştirilebilir ve değiştirilebilir hale gelir. Mikro çekirdek mimarisi, esneklik, güvenilirlik ve bakım kolaylığı sağlar.

Soru

- Mikro çekirdek mimarisinde, aygıt sürücüleri nerede bulunur?

- a) Çekirdek alanında
- b) Kullanıcı alanında ayrı süreçler olarak
- c) Ayrılmış donanım katmanında
- d) Yazılım katmanında

Cevap

- Cevap: B

- Mikro çekirdek mimarisinde, aygıt sürücüleri kullanıcı alanında ayrı süreçler olarak bulunur. Aygıt sürücüleri çekirdek dışında, kullanıcı alanında çalışır. Mikro çekirdek yapısı, çekirdeğin boyutunu en aza indirmeyi ve çekirdek içinde sadece temel işlevleri tutmayı amaçlar. Bu nedenle, aygıt sürücüleri gibi özelleştirilmiş işlevler, kullanıcı alanında ayrı süreçler olarak yürütülür ve çekirdek dışında çalışır.

Soru

- Mikro çekirdek mimarisi, monolitik çekirdeğe kıyasla performans açısından hangi zorlukla karşılaşır?

- a) Bağlam anahtarlama maliyetinin yüksek olması
- b) Güvenlik açıklarına karşı hassas olması
- c) Özelleştirme için esnekliğin az olması
- d) Çekirdek bileşenleri arasında iletişim gecikmesinin yüksek olması

Cevap

- Cevap: A

- Mikro çekirdek mimarisi, monolitik çekirdeklerle karşılaştırıldığında artan bağlam anahtarlama maliyetleri ile karşılaşır. Mikro çekirdek mimarisinde, işlevler çekirdek dışında, ayrı süreçlerde yürütülür ve bu nedenle süreçler arası iletişim ve geçişler daha sık gerçekleşir. Bu durum, mikro çekirdek mimarisinin performansını etkileyebilir. Monolitik çekirdeklerin, işlevlerin doğrudan çekirdek içinde çalıştığı için, bu tür maliyetleri daha düşüktür.

Soru

- Mikro çekirdek mimaride Süreçler Arası İletişim (*IPC*) mekanizması hangi role sahiptir?

- a) Çekirdek ve kullanıcı alanı süreçleri arasındaki iletişimi yönetir.
- b) Farklı çekirdek modülleri arasındaki iletişimi kolaylaştırır.
- c) Çekirdek ve kullanıcı uygulaması arasında bellek yönetimini koordine eder.
- d) Dosya ve aygıtlar gibi sistem kaynaklarına erişimi kontrol eder.

Cevap

- Cevap: B

- Mikro çekirdek mimaride Süreçler Arası İletişim (IPC) mekanizması, farklı çekirdek modülleri arasındaki iletişimi kolaylaştırır. Mikro çekirdek mimarisinde, işlevler ayrı çekirdek modülleri olarak uygulanır ve bu modüller arasındaki iletişim IPC mekanizması aracılığıyla gerçekleşir. Örneğin, bir aygıt sürücüsü çekirdek içinde bir modül olabilir ve bir sistem çağrısı ile iletişim kurarak veya mesaj ileterek diğer çekirdek modülleriyle etkileşimde bulunabilir. IPC mekanizması, çekirdek modüllerinin işbirliğini sağlar ve farklı süreçler arasında veri ve kontrol akışını yönetir.

Soru

- Mikro çekirdek tabanlı işletim sisteminin işlevselliğini genişletmek için hangi yaklaşım kullanılır?

- a) Çekirdek modüllerinin dinamik olarak bağlanması
- b) Aygıt sürücülerinin çekirdek alanına entegrasyonu
- c) Önyükleme sırasında ek çekirdek bileşenlerinin yüklenmesi
- d) Kritik hizmetlerin çekirdek ikili dosyasının bir parçası olarak çalıştırılması

Cevap

- Cevap: A

- Mikro çekirdek tabanlı işletim sisteminin işlevselliğini genişletmek için çekirdek modülleri dinamik olarak bağlanır. Bu yaklaşım, işletim sisteminin çekirdek alanında yer alan temel işlevlerini genişletmek veya özelleştirmek için harici modüllerin dinamik olarak yüklenmesini sağlar. Çekirdek modülleri, çalışma zamanında yüklenebilir ve çekirdek dışındaki bir alanda saklanabilir. Bu, işletim sistemi işlevselliğinin esnek bir şekilde genişletilmesini sağlar ve çekirdek boyutunu azaltır.

Soru

- Sistem performansı açısından mikro çekirdek mimarisinin dezavantajı nedir?

- a) Sistem yönetiminin artan karmaşıklığı
- b) Sistem çağrıları ve IPC için yüksek ek maliyet
- c) Üçüncü taraf aygıt sürücülerini entegre etmede zorluk
- d) Büyük ölçekli sistemler için sınırlı ölçeklenebilirlik

Cevap

- Cevap: B

- Mikro çekirdek mimarisinin sistem performansı açısından potansiyel bir dezavantajı, sistem çağrıları ve süreçler arası iletişim (*IPC*) için ek maliyettir. Mikro çekirdek mimarisinde, işlevler ayrı çekirdek modülleri olarak uygulanır ve bu modüller arasındaki iletişim sistem çağrıları veya IPC mekanizması aracılığıyla gerçekleşir. Her sistem çağrısı veya IPC işlemi, çekirdek ve kullanıcı alanı arasında veri kopyalamayı gerektirebilir ve bu, işlemci zamanı ve bellek kullanımı açısından maliyetlidir.

Soru

- Katmanlı (*layered*) işletim sistemi yapısında, donanım soyutlama katmanının (*HAL*) asıl amacı nedir?

- a) Kullanıcı uygulamalarının işletim sistemiyle etkileşimi için bir arayüz.
- b) Sistem çağrıları ve çekirdek alanı görevlerin yürütülmesini yönetmek.
- c) İşletim sisteminin farklı katmanları arasında iletişimi kolaylaştırmak.
- d) Donanıma özgü ayrıntıları soyutlamak ve üst katmanlara bir arayüz.

Cevap

- Cevap: D

- Donanım Soyutlama Katmanı (*Hardware Abstraction Layer*), donanıma özgü ayrıntıları soyutlar ve üst katmanlara (çekirdek ve kullanıcı arayüzü gibi) birleşik bir arayüz sağlar. Bu katman, işletim sistemi yazılımının donanımdan bağımsız olmasını sağlar. Böylece, işletim sistemi farklı donanım yapılandırmalarıyla çalışabilir ve üst katmanlar, donanımın nasıl çalıştığına dair ayrıntılar hakkında endişelenmeden işletim sistemine erişebilir. HAL, donanım değişikliklerini gizler ve üst katmanlara istikrarlı bir arayüz sunar, bu da işletim sisteminin genişletilmesini ve bakımını kolaylaştırır.

Soru

- İşletim sistemi hiyerarşisinde hangi katman, kullanıcı uygulamaları ile çekirdek arasındaki etkileşimi yönetmekten sorumludur?

- a) Çekirdek katmanı
- b) Donanım soyutlama katmanı
- c) Sistem çağrısı katmanı
- d) Kabuk katmanı

Cevap

- Cevap: C

- Kullanıcı uygulamaları ile çekirdek arasındaki etkileşimi sistem çağrısı katmanı yönetir. Bu katman, kullanıcı uygulamalarının çekirdeğe erişimini sağlar ve sistem çağrıları aracılığıyla çeşitli işletim sistemi hizmetlerine erişim sağlar. Kullanıcı uygulamaları, sistem çağrıları yaparak çekirdekten hizmet talep ederler ve çekirdek, bu çağrıları işleyerek gerekli işlemleri gerçekleştirir.

Soru

- Katmanlı işletim sistemi yapısının temel avantajı nedir?

- a) Sistem işlevselliğini özelleştirmek ve genişletmek için esneklik
- b) Sistem kaynaklarının ve süreçlerin kolay yönetimi
- c) Katmanlar arasında iletişimde azalan ek maliyet, artan performans
- d) Sistem bileşenlerinin izolasyonu ve sarmalama (*encapsulation*) yoluyla artan güvenlik

Cevap

- Cevap: D

- Katmanlı işletim sistemi yapısının temel avantajı, sistem bileşenlerinin izolasyonu ve sarmalama yoluyla artan güvenliktir. Katmanlı yapı, her katmanın belirli bir işlevsellik seviyesine sahip olduğu ve alt katmanların üst katmanlardan gizlendiği bir yapı sunar. Bu, her katmanın sadece altındaki katmanlarla iletişim kurmasını ve dışarıdaki katmanlardan izole olmasını sağlar. Böylece, bir katman, diğer katmanlardan etkilenmez ve sistem bütünlüğü korunur. Bu, güvenlik açısından önemlidir çünkü saldırganların sistem içinde hareket etmelerini ve zarar vermelerini önler. Katmanlı yapı karmaşıklığı azaltır, sistem yönetimini ve bakımını kolaylaştırır.

Soru

- Sanallaştırmada, hipervizörün temel amacı nedir?

- a) Kullanıcı uygulamalarının sanal makinelerle etkileşim kurması için bir arayüz sağlamak.
- b) Sanal makinelerin CPU, bellek ve depolama gibi sistem kaynaklarını yönetmek.
- c) Sanal makinelerin işletim sistemlerini çalıştırabilmesi için donanım bileşenlerini taklit etmek.
- d) Tek bir fiziksel ana bilgisayarda birden fazla sanal makine oluşturmak ve yönetmek.

Cevap

- Cevap: D

- Bir hipervizörün temel amacı, tek bir fiziksel ana bilgisayarda birden fazla sanal makine oluşturmak ve yönetmektir. Hipervizör, donanım kaynaklarını sanal makineler arasında paylaştırır ve her bir sanal makineyi izole bir ortamda çalıştırır. Bu, bir fiziksel sunucu üzerinde birden fazla işletim sistemi çalıştırmak için ideal bir çözümdür ve kaynakları etkin şekilde kullanmayı sağlar. Hipervizör, sanal makinelerin oluşturulması, başlatılması, durdurulması ve yönetilmesi gibi görevleri yerine getirir ve sanal makineler arasında kaynak paylaşımını düzenler.

Soru

- Tip 1 hipervizörleri, Tip 2 hipervizörlerinden ayıran nedir?

- a) Tip 1 fiziksel donanım üzerinde doğrudan, Tip 2 bir ana işletim sisteminin üstünde çalışır.
- b) Tip 1 tam sanallaştırmayı, Tip 2 yarı-sanallaştırmayı destekler.
- c) Tip 1 masaüstü için, Tip 2 sunucu sanallaştırması için tasarlanmıştır.
- d) Tip 1, doğrudan donanım erişimine sahip olduğu için, Tip 2'ye göre daha iyi performans sağlar.

Cevap

- Cevap: A

- Tip 1 hipervizör fiziksel donanımın üzerinde doğrudan çalışırken, Tip 2 hipervizör bir ana işletim sisteminin üstünde çalışır. Tip 1 hipervizör, işletim sistemi ile donanım arasında aracı olmadan çalışır ve bu nedenle daha az katmana sahiptir. Tip 2 hipervizör bir ana işletim sistemi üzerinde çalışır ve bu nedenle ana işletim sistemi ve donanım arasında bir aracı olarak işlev görür. Bu fark, her iki tip hipervizörün çalışma ortamını ve performansını etkiler. Tip 1 hipervizör, doğrudan donanıma eriştiği için daha iyi performans sunarken, Tip 2 hipervizörler esnek dağıtım seçenekleri sunar.

Soru

- Gömülü sistemlerde bekçi zamanlayıcının (*watchdog timer*) amacı nedir?

- a) Donanım bileşenleri arasında zamanlama sinyallerini senkronize etmek
- b) Sistem sağlığını izlemek ve yanıt vermeyen sistemi sıfırlamak
- c) Kritik sistem görevlerinin yürütme süresini ölçmek
- d) Senkronizasyon amaçları için kesin zaman referansları oluşturmak

Cevap

- Cevap: B

- Bir bekçi zamanlayıcının (watchdog timer) gömülü sistemlerdeki amacı, sistemin sağlığını izlemek ve yanıt vermediği durumlarda sistemde sıfırlama işlemi gerçekleştirmektir. Gömülü sistemler genellikle kritik görevleri yerine getirir ve sistemdeki herhangi bir hata veya kilitlenme, ciddi sonuçlara yol açabilir. Bekçi zamanlayıcı, sistemdeki belirli bir süre boyunca düzenli olarak bir sinyal göndermezse, sistem bu durumu algılar ve otomatik olarak sıfırlanır. Bu, sistemdeki hatalı durumları ele almanın ve sistem istikrarını korumanın önemli bir yoludur.

Soru

- İşletim sistemlerinde zamanlayıcılar tarafından kullanılan sistem saati hangi bileşen tarafından üretilir?

- a) Gerçek zamanlı saat (*RTC*)
- b) Merkezi İşlem Birimi (*CPU*)
- c) Programlanabilir aralık zamanlayıcı (*PIT*)
- d) Bellek Yönetim Birimi (*MMU*)

Cevap

- Cevap: A

- İşletim sistemlerinde zamanlayıcılar tarafından kullanılan sistem saatini üreten bileşen gerçek zamanlı saattir (RTC). Gerçek zamanlı saat, anakartın bir parçası olarak bulunan donanım bileşenidir. Bu saat, gerçek dünya zamanını temsil eder ve işletim sistemi tarafından zamanlama işlemleri için kullanılır. Zamanlayıcılar, bu gerçek zamanlı saati kullanarak belirli sürelerde işlemler yaparlar, zamanlanmış görevleri yürütürler ve zamanlayıcı olaylarını işlerler.

Soru

- Modern işletim sistemlerinde sistem zamanlaması için hangi çözünürlüğü kullanılır?

- a) Milisaniye
- b) Mikro saniye
- c) Nano saniye
- d) Piko saniye

Cevap

- Cevap: B

- Modern işletim sistemlerinde, sistem zamanlaması için tipik olarak mikro saniye (μs) çözünürlüğü kullanılır. Mikro saniye, işletim sisteminin çeşitli zamanlama işlemleri için yeterli ayrıntı düzeyi sağlar ve kullanıcıların algısal sınırlarının altındaki zaman aralıklarını ölçmek için yeterlidir.

Soru

- Sistem çağrısının (*system call*) temel amacı nedir?

- a) CPU ve bellek tahsisi gibi sistem kaynaklarını yönetmek
- b) Aynı sistemde çalışan farklı süreçler arasında iletişimi kolaylaştırmak
- c) Kullanıcı düzeyi uygulamaların çekirdekten hizmet talep etmesi için arayüz sağlamak
- d) Sisteme bağlı çevre birimi aygıtlarının oluşturduğu kesmeleri işlemek

Cevap

- Cevap: C

- Sistem çağrıları (*system call*), kullanıcı düzeyi uygulamaların çekirdekten hizmet talep etmesi için arayüz sağlar. Sistem çağrıları, kullanıcı uygulamalarının işletim sistemi çekirdeğinin sağladığı hizmetlere erişmesini sağlar. Örneğin, dosya sistemi erişimi, ağ iletişimi, bellek yönetimi gibi işlemler için sistem çağrıları kullanılır.

Soru

- Hangi işletim sistemi bileşeni sistem çağrılarını işlemekten sorumludur?

- a) Çekirdek (*Kernel*)
- b) Kabuk (*Shell*)
- c) Aygıt sürücüleri (*Device drivers*)
- d) Kesme işleyicisi (*Interrupt handler*)

Cevap

- Cevap: A

- Sistem çağrılarını işlemekten sorumlu bileşen çekirdektir (*kernel*). Çekirdek, işletim sisteminin temel parçasıdır ve sistem kaynaklarını yönetir, süreçleri çizelgeler, belleği yönetir ve sistem çağrılarını ele alır. Sistem çağrıları, kullanıcı uygulamalarının çekirdekten belirli hizmetleri talep etmesini sağlar. Örneğin, dosya okuma/yazma, ağ iletişimi, bellek tahsisi gibi işlemler sistem çağrıları aracılığıyla gerçekleştirilir.

Soru

- Yeni süreç oluşturmak için hangi sistem çağrısı kullanılır?

- a) fork()
- b) exec()
- c) wait()
- d) exit()

Cevap

- Cevap: A

- Yeni süreç oluşturmak için fork() sistem çağrısı kullanılır. fork() çağrısı, mevcut süreci çoğaltarak yeni bir süreç yaratır. Yeni süreç, çağıran süreçle aynı program kodunu ve durumu paylaşır, ancak farklı bir süreç kimliği (PID) ile yürütülür. Bir sürecin, çocuk süreç oluşturmasını ve ardından çocuk sürecin başka bir programı çalıştırması için kullanılır. Örneğin, ata süreç yeni bir süreç yaratmak için fork() kullanır ve sonra exec() çağrısını kullanarak yeni sürecin bir programı çalıştırmasını sağlayabilir.

Soru

- Ata sürecin, çocuk sürecin sonlanmasını beklemek için kullandığı sistem çağrısı hangisidir?

- a) fork()
- b) exec()
- c) wait()
- d) exit()

Cevap

- Cevap: C

- Ata süreç, çocuk sürecin sonlanmasını beklemek için wait() sistem çağrısını kullanır. wait() çağrısı, ata sürecin bir veya daha fazla çocuk sürecin sonlanmasını beklemesini sağlar. Eğer çocuk süreç henüz sonlanmamışsa, ata süreç wait() çağrısı üzerinde bekler ve çocuk süreç sonlandığında devam eder. Eğer çocuk süreç zaten sonlanmışsa, wait() çağrısı hemen geri döner ve ata süreç, çocuğun sonlanma durumunu (çocuğun geri dönüş değerini) alabilir.

Soru

- Bir sürecin yürütmesini sonlandırmak için hangi sistem çağrısı kullanılır?

- a) fork()
- b) exec()
- c) wait()
- d) exit()

Cevap

- Cevap: D

- Unix benzeri bir işletim sisteminde, bir sürecin yürütmesini sonlandırmak için exit() sistem çağrısı kullanılır. exit() çağrısı, mevcut süreci sonlandırır ve kontrolü çağıran sürece bırakır. Süreç tamamlandığında veya bir hata durumunda çağrılır. Süreç, exit() çağrısını yaptıktan sonra, sürece ait kaynaklar serbest bırakılır ve süreç sonlandırılır.

Soru

- Dosya yönetiminde open() sistem çağrısının amacı nedir?

- a) Dosya sisteminde yeni bir dosya oluşturmak
- b) Var olan dosyayı açmak ve dosya tanımlayıcısı elde etmek
- c) Açık olan bir dosyayı kapatmak ve ilişkili kaynakları serbest bırakmak
- d) Bir dosyadan verileri bellek tamponuna okumak

Cevap

- Cevap: B

- open() sistem çağrısı, var olan bir dosyayı açmak ve bu dosya üzerinde yapılacak işlemler için bir dosya tanımlayıcısı (*file descriptor*) elde etmek için kullanılır. Dosyayı açarken gerekli izinleri ve dosyanın yolunu belirtir. Dosya tanımlayıcısı, işlem sırasında dosyaya erişim sağlamak için kullanılır. open() çağrısı, dosya oluşturma, dosya okuma, yazma veya değiştirme gibi çeşitli dosya işlemlerinin başlatılmasını sağlar.

Soru

- Süreç denetim bloğunun (*PCB*) temel amacı nedir?

- a) Bir sürecin yürütülebilir kodunu saklamak
- b) Sistem kaynaklarının süreçlere tahsisini yönetmek
- c) Bir sürecin durumu ve özellikleri hakkındaki bilgileri saklamak
- d) Birden çok süreç arasında iletişimi sağlamak

Cevap

- Cevap: C

- Süreç denetim bloğu (*PCB*), sürecin durumu ve özellikleri hakkındaki bilgileri saklar. *PCB*, işletim sistemi tarafından her süreç için oluşturulur. Sürecin durumu (çalışıyor, hazır, beklemede vb.), program sayacı (PC), kaynaklar (bellek, açık dosyalar, CPU zamanı vb.) gibi bilgileri içerir. İşletim sistemi tarafından süreç yönetimi ve çizelgeleme için kullanılır. Süreç durumu veya süreç kaynak talepleri değiştiğinde PCB güncellenir. Bu sayede işletim sistemi, süreçleri etkin bir şekilde yönetebilir ve kaynakları adil bir şekilde dağıtabilir.

Soru

- Çoklu iş parçacığı (*multithreaded*) sürecin özelliği nedir?

- a) Farklı CPU çekirdeklerinde aynı anda birden fazla işlem yürütür
- b) Adres alanı ve kaynakları birden fazla iş parçacığı arasında paylaşır
- c) İşletim sistemi çizelgeleyicisi tarafından kesintiye uğratılamaz
- d) Her iş parçacığı ayrı bir süreç denetim bloğuna (PCB) sahiptir

Cevap

- Cevap: B

- Çoklu iş parçacığı (*multithreaded*) süreç, aynı adres alanını ve kaynakları birden fazla iş parçacığı arasında paylaşır. Aynı süreç içinde birden fazla iş parçacığı (*thread*) çalışabilir. İş parçacıkları, süreç içindeki farklı görevleri eşzamanlı olarak gerçekleştirmek için kullanılır. Ancak, tüm iş parçacıkları süreç içindeki aynı adres alanını ve kaynakları paylaşır. Bu, iş parçacıklarının veri paylaşımı ve iletişimi için uygun bir ortam sağlar. Örneğin, bir iş parçacığı bir dosyayı okurken, diğer bir iş parçacığı aynı dosyaya yazabilir veya başka bir iş parçacığı tarafından tahsis edilen bellek bölgesine erişebilir.

Soru

- Bir olayın tamamlanmasını bekleyen (örneğin G/Ç tamamlanması) bir süreci temsil eden durum hangisidir?

- a) Çalışıyor (*Running*)
- b) Hazır (*Ready*)
- c) Engellenmiş (*Blocked*)
- d) Sonlanmış (*Terminated*)

Cevap

- Cevap: C

- Bir süreç, bir olayın tamamlanmasını beklerken (örneğin, bir G/Ç işleminin tamamlanması), engellenmiş (*blocked*) durumda olur. Bu durumda, süreç kaynak talebi nedeniyle beklemek zorunda kalır ve işletim sistemi diğer süreçleri yürütmeye devam eder. Süreç, beklediği olay gerçekleştiğinde veya belirli bir zaman aşımında yeniden hazır duruma gelir ve işletim sistemi tarafından tekrar yürütülür.

Soru

- Hazır kuyruğundaki süreçlerin ortalama bekleme süresini en aza indirmeyi amaçlayan çizelgeleme algoritması hangisidir?

- a) Sıralı (*Round Robin*)
- b) En kısa iş önce (*Shortest Job Next*)
- c) İlk gelen ilk hizmet alır (*First Come, First Served*)
- d) Sonlanmasına en kısa süre kalan (*Shortest Remaining Time First*)

Cevap

- Cevap: D

- *Shortest Remaining Time First* (*SRTF*), en az işlem süresi kalan süreci önceliklendirir. *SRTF*, hazır kuyruğundaki tüm süreçlerin kalan işlem sürelerini izler ve en az süresi kalanı seçer. Bu şekilde, kısa süreçler öncelikli olarak işlenir ve ortalama bekleme süresi en aza indirilir. Diğer çizelgeleme algoritmalarına göre, *SRTF* daha düşük bekleme süreleri sağlar, ancak işlem süreleri dinamik olarak değiştiği için işlem sırası sık sık değişebilir.

Soru

- Semaforun amacı nedir?

- a) Kaynaklara karşılıklı dışlama sağlayarak yarış koşullarını önlemek
- b) Süreçler arasında mesaj iletimi için iletişim kanalı sağlamak
- c) Eşzamanlı bir programda kritik bölgelerin yürütülmesini planlamak
- d) Süreç yürütme sırasında bellek tahsis etmek ve serbest bırakmak

Cevap

- Cevap: A

- Semafor işaret flaması (*semaphore*), senkronizasyon aracıdır. Paylaşılan kaynaklara karşılıklı dışlama sağlar. Yarış koşullarını önlemek için, birden çok süreç aynı anda paylaşılan bir kaynağa erişmeye çalışırken, semaforlar kullanılarak kritik bölgelere girişin düzenlenmesi sağlanır. Semaforlar, kritik bölgelerde aynı anda yalnızca bir sürecin çalışmasına izin vererek, paylaşılan kaynakların güvenli bir şekilde kullanılmasını sağlar.

Soru

- Süreç ve iş parçacığını doğru bir şekilde ayırt eden ifade hangisidir?

- a) Süreç birden çok iş parçacığı içerir, iş parçacığı hafif bir işlemdir.
- b) Süreç işletim sistemi tarafından oluşturulur, iş parçacığı uygulama tarafından oluşturulur.
- c) Süreç kendi adres alanına sahiptir, iş parçacıkları aynı süreç içinde aynı adres alanını paylaşır.
- d) Süreç birden çok komutu aynı anda yürütebilir, iş parçacığı komutları ardışık olarak yürütür.

Cevap

- Cevap: C

- Süreç ve iş parçacığı arasındaki temel fark, bellek yönetimidir. Süreç, işletim sistemi tarafından ayrılan ve izole edilen kendi bellek alanına sahiptir. Süreç kendisine ait veri, kod ve kaynaklara sahiptir, diğer süreçlerden izole edilmiştir. Bu izolasyon, süreçlerin bağımsız çalışmasını sağlar ve bir hata oluştuğunda diğer süreçleri etkilemez. İş parçacıkları aynı süreç içinde aynı görevi yerine getirmek üzere kullanılır ve birbirleriyle iletişim kurmak için paylaşılan bellek alanını kullanırlar. Süreçler daha fazla izolasyon ve güvenlik sağlarken, iş parçacıkları daha hafif ve daha hızlıdır. Bir süreç birden çok iş parçacığını barındırabilir.

Soru

- Paylaşılan kaynaklara eş zamanlı erişimle ilgili ana zorluk nedir?

- a) Ölümcül kitlenme (*deadlock*)
- b) Açlık (*starvation*)
- c) Yarış koşulları (*race condition*)
- d) Öncelik ters çevirme (*priority inversion*)

Cevap

- Cevap: C

- Yarış koşulları (*race conditions*), çok süreçli sistemlerde paylaşılan kaynaklara aynı anda erişmeye çalışan süreçler arasında beklenmeyen ve istenmeyen sonuçların ortaya çıkmasına neden olur. Bu durumlar, programın sırası veya zamanlaması ile ilgili olduğunda ortaya çıkar ve programın doğruluğunu veya güvenilirliğini etkiler.

Soru

- Her bir sürece eşit CPU zamanı sağlamayı hedefleyen çizelgeleme algoritması hangisidir?

- a) Sıralı (*Round Robin*)
- b) İlk Gelen İlk Hizmet (*FCFS*)
- c) En Kısa İş İlk (*SJN*)
- d) Öncelikli çizelgeleme (*Priority scheduling*)

Cevap

- Cevap: A

- *Round Robin*, çok süreçli sistemlerde CPU zamanını süreçlere eşit bir şekilde dağıtmayı hedefler. Her süreç belirli bir zaman dilimi (*quantum*) için CPU'ya atanır. Zaman dilimi sona erdiğinde, işlemci başka bir sürece geçer. Bu işlem, süreçler arasında adil bir dağılım sağlar ve her sürece eşit fırsat verir.

Soru

- Eşzamanlı programlamada kritik bölgenin amacı nedir?

- a) Birçok süreç tarafından aynı anda yürütülebilen kod bölümü tanımlamak
- b) Süreçlerin iletişimini ve faaliyetlerini senkronize etmesini sağlamak
- c) Paylaşılan kaynakları birçok sürecin eşzamanlı erişiminden korumak
- d) Süreçlere önceliklerine bağlı olarak CPU zamanı tahsis etmek

Cevap

- Cevap: C

- Kritik bölge (*critical region*), eşzamanlı programlamada paylaşılan kaynaklara eş zamanlı erişimi kontrol etmek için kullanılan bir mekanizmadır. Birden çok süreç aynı anda paylaşılan bir kaynağa erişmeye çalıştığında, bu kaynağın veri tutarlılığını sağlamak önemlidir. Kritik bölge, yalnızca bir sürecin aynı anda paylaşılan kaynağa erişebileceği şekilde tasarlanır.

Soru

- Senkronizasyon için kilit kullanmanın potansiyel dezavantajı nedir?

- a) Kilitlenme (*deadlock*)
- b) Öncelik tersine çevirme (*priority inversion*)
- c) Meşgul bekleme (*busy waiting*)
- d) Açlık (*starvation*)

Cevap

- Cevap: C

- Kilitler, çok süreçli sistemlerde senkronizasyonu sağlamak için kullanılır. Bu mekanizmanın dezavantajı, meşgul bekleme (*busy waiting*) durumudur. Meşgul bekleme, iş parçacığının kilit çözülene kadar sürekli kontrol ederek işlemciyi bekletme durumudur. İşlemci işlem gerçekleştiremez ve kilit çözülünceye kadar bekler. İşlemciyi verimli bir şekilde kullanmak yerine, kilit sürekli kontrol edilir ve kaynaklar boşa harcanır.

Soru

- Eşzamanlı programlamada monitör yapısının rolü nedir?

- a) Süreçlerin iletişimi ve faaliyetlerini koordine etmesi için mekanizma
- b) Paylaşılan kaynakları eş zamanlı erişimden korumak
- c) Yalnızca bir sürecin kritik kod bölümünü yürütmesine izin vermek
- d) Paylaşılan verilere ve bunlara erişen süreçleri sarmalamak ve karşılıklı dışlama sağlamak

Cevap

- Cevap: D

- Monitör, eşzamanlı programlamada paylaşılan verileri ve bu verilere erişen süreçleri sarmalamak için kullanılan yapılardır. Monitörler, veri ve süreçleri bir araya getirerek, bu verilere aynı anda sadece bir sürecin erişmesine ve değiştirmesine izin verirler. Bu, karşılıklı dışlama (*mutual exclusion*) sağlar ve paylaşılan verilere eş zamanlı erişimden kaynaklanan sorunları önler. Monitörler aynı zamanda koşullu değişkenler gibi senkronizasyon araçlarını da içerebilir, böylece süreçler arasında iletişim ve koordinasyonu sağlarlar.

Soru

- *Zombie* (Ölü) durumunda olan süreç nedir?

- a) Giriş/Çıkış gibi bir olayın gerçekleşmesini bekleyen süreç.
- b) Sonlandırılmış ancak hala süreç tablosunda tutulan süreç.
- c) Askıya alınmış ve daha sonra devam ettirilebilecek süreç.
- d) CPU'da etkin olarak komutları yürüten aktif süreç.

Cevap

- Cevap: B

- Bir süreç, ata süreç tarafından sonlandırıldıktan sonra hala süreç tablosunda tutuluyor olabilir. Bu durum, süreç tablosundaki kaynağın boşa harcanmasına ve gereksiz kaynak tüketimine neden olur. Ölü (*zombie*) bir süreç, çocuk süreç sona erdikten sonra ata süreci sona erdirmediğinde veya ata süreç sona erdiğinde ancak çocuk süreç sonlandırma sinyalini almadığında oluşur. Ölü bir süreç, sonlandırıldıktan sonra sistem kaynaklarını serbest bırakmak için özel bir temizlik işlemi ile kaldırılmalıdır.

Soru

- Bir sürecin *Sonlandırılmış* (*terminated*) durumu neyi ifade eder?

- a) G/Ç işleminin tamamlanması gibi bir olayın gerçekleşmesi bekleniyor.
- b) Yürütme tamamladı ve ayrılan kaynaklar serbest bırakıldı, bekliyor.
- c) Süreç, giriş veya çıkış işlemlerinin tamamlanmasını bekliyor.
- d) Süreç, CPU'da komutları aktif bir şekilde yürütüyor.

Cevap

- Cevap: B

- Bir sürecin *Sonlandırılmış* durumu, yürütmesinin tamamlandığını ve tüm ayrılan kaynakları serbest bıraktığını gösterir. Bu durum, süreç artık sistem kaynaklarına ihtiyaç duymadığı ve sistem tarafından temizlenmeye veya sonlandırılmaya hazır olduğu anlamına gelir. İşletim sistemi, sonlandırılmış bir işlemi süreç tablosundan kaldırarak ve kullanılan diğer kaynakları serbest bırakarak bu durumu işler.

Soru

- Sürecin Giriş/Çıkış işlemlerinin tamamlanması gibi olayın gerçekleşmesini beklediği durum hangisidir?

- a) Çalışıyor (*Running*)
- b) Hazır (*Ready*)
- c) Engellenmiş (*Blocked*)
- d) Yeni (*New*)

Cevap

- Cevap: C

- *Engellenmiş* durum, bir sürecin Giriş/Çıkış (I/O) tamamlanması gibi bir olayın gerçekleşmesini beklediği durumu ifade eder. Bir süreç, bir Giriş/Çıkış işlemi gerçekleştirdiğinde, süreç genellikle işlemcinin dışındaki bir aygıtla etkileşime girer ve sonucu bekler. Örneğin, bir dosya okuma işlemi gerçekleştirirken, işlemci dosyanın okunmasını bekler. Bu süreçte, işlem *Engellenmiş* durumuna geçer ve yürütülmek için bekler. İşletim sistemi, Giriş/Çıkış işlemi tamamlandığında süreci uyandırır ve yeniden çalıştırır.

Soru

- Süreç yönetiminde *Hazır* durumunun rolü nedir?

- a) Sürecin yürütmesini tamamladığını kaynakları serbest bıraktığını belirtir.
- b) Sürecin G/Ç işlemi gibi bir olayın gerçekleşmesini beklediğini gösterir.
- c) Sürecin giriş veya çıkış işlemlerinin tamamlanmasını beklediğini belirtir.
- d) Bir sürecin hazır ve CPU'ya atanmayı beklediğini belirtir.

Cevap

- Cevap: D

- *Hazır* durumu, bir sürecin CPU'ya atanma ve yürütme için hazır olduğunu belirtir. İşletim sistemi, CPU'ya atamak için, hazır durumda olan süreçlerden birini seçer. Hazır durumundaki süreç, CPU zamanı almak için bekler, ancak o anda yürütülmez. İşletim sistemi, bu süreçler arasında zaman paylaşımı yaparak ve işlemciyi verimli kullanarak süreçleri sırayla yürütür.

Soru

- Sürecin komutları etkin bir şekilde yürüttüğünü belirten durum hangisidir?

- a) Hazır (*ready*)
- b) Engellenmiş (*blocked*)
- c) Çalışıyor (*running*)
- d) Sonlandırıldı (*terminated*)

Cevap

- Cevap: C

- *Çalışıyor* durumu, bir sürecin işlemcide aktif olarak komutları yürüttüğünü ifade eder. Süreç, CPU üzerindeki işlemci zamanını kullanarak talimatları gerçekleştirir. Bu durumda, süreç sistem kaynaklarına erişir. Çalışan bir süreç, işletim sistemi tarafından CPU'dan alınana veya bir kesme gerçekleşene kadar çalışmaya devam eder.

Soru

- Bağlam anahtarlama (*context switch*) işleminin temel amacı nedir?

- a) Bir sürecin durumunu kaydetmek ve yürütme için başka bir sürecin durumunu geri yüklemek.
- b) Süreçlere öncelik ve çizelgeleme algoritmasına dayalı olarak CPU zamanı tahsis etmek.
- c) Eşzamanlı yürütülen süreçler arasında paylaşılan kaynaklara erişimi senkronize etmek.
- d) Sisteme bağlı çevre birimleri tarafından üretilen kesmeleri işlemek.

Cevap

- Cevap: A

- Bağlam anahtarlama, işletim sistemi tarafından bir sürecin durumunu kaydetmek ve CPU'da başka bir süreci yürütmek üzere kaydedilen durumunu geri yüklemek için gerçekleştirilir. Bu işlem, çok işlemli (*multiprocessing*) sistemlerde CPU'nun farklı süreçler arasında geçiş yapmasını sağlar. Bir süreç, CPU'da çalışırken, bir kesme meydana geldiğinde veya işletim sistemi farklı bir sürecin çalışması gerektiğine karar verdiğinde, süreç durumu kaydedilir ve CPU'ya başka bir süreç atanır.

Soru

- Bağlam anahtarlamayı gerçekleştirmekten hangi bileşen sorumludur?

- a) Çekirdek (*Kernel*)
- b) Kabuk (*Shell*)
- c) Aygıt sürücüleri (*Device drivers*)
- d) Çizelgeleyici (*Scheduler*)

Cevap

- Cevap: A

- Bağlam anahtarlama, işletim sistemi içerisinde gerçekleşen önemli bir işlemdir. İşletim sistemi çekirdeği (*kernel*) tarafından yönetilir. Çekirdek, işletim sisteminin merkezi parçasıdır ve temel işlevleri, süreçleri yönetmek, kaynakları (örneğin, CPU, bellek, dosya sistemi) denetlemek ve sistem kaynaklarını yönetmektir.

Soru

- Sık sık bağlam anahtarlama yapılmasının sonucu nedir?

- a) Geliştirilmiş sistem tepkisi ve işlem kapasitesi
- b) Süreç durumu kaydedilmesi ve geri yüklenmesi nedeniyle artan ek yük
- c) Sistem kaynaklarının yönetiminde azalan karmaşıklık
- d) Artan hata toleransı ve güvenilirlik

Cevap

- Cevap: B

- Sık sık bağlam anahtarlama işlemleri, işletim sistemi tarafından yönetilen çok görevli (*multitasking*) sistemde yaygın bir durumdur. Ancak, sık bağlam değişimlerinin bir sonucu olarak, süreç durumunun sürekli olarak kaydedilmesi ve geri yüklenmesi işlemi artar. Bu, ekstra işlemci zamanı ve sistem kaynakları gerektirir, işletim sistemi üzerinde ek bir yük oluşturur.

Soru

- Bağlam anahtarlamanın süresini etkileyen faktör hangisidir?

- a) CPU'nun hızı
- b) Kullanılabilir bellek miktarı
- c) Sistemdeki süreç sayısı
- d) Çizelgeleme algoritmasının verimliliği

Cevap

- Cevap: A

- Bağlam anahtarlama, işlemcinin yürütülen süreci durdurup hazır bekleyen başka bir sürece geçiş yapmasıdır. Bağlam anahtarlama süresi, birçok faktöre bağlıdır ancak en önemlisi CPU'nun hızıdır. Çünkü işlemci tarafından gerçekleştirilir ve işlemci hızı, bu işlemin ne kadar hızlı gerçekleşeceğini belirler.

Soru

- Bağlam anahtarlama sırasında bir sürecin durumunun kaydedilmesinin amacı nedir?

- a) Sürecin kesintiye uğradığı noktadan devam etmesini sağlamak.
- b) Süreçlere öncelik ve çizelgeleme algoritmasına bağlı olarak CPU zamanı tahsis etmek.
- c) Eşzamanlı yürütülen süreçler arasında paylaşılan kaynaklara erişimi senkronize etmek.
- d) Sisteme bağlı çevre birimleri tarafından üretilen kesmeleri işlemek.

Cevap

- Cevap: A

- Bağlam anahtarlama sırasında, bir süreç durumunun kaydedilmesinin amacı, süreci kesintiye uğradığı noktadan devam ettirebilmektir. Bu, işlemci bir başka sürece geçtiğinde veya bir kesme (*interrupt*) meydana geldiğinde, süreci durdurmak ve daha sonra işlemciye geri döndüğünde sürecin kaldığı noktadan devam etmesini sağlamak için gereklidir.

Soru

- İş kesme üstünlüğü ile çok görevli (*preemptive multitasking*) sistemlerde, bağlam anahtarlamayı tetikleyen nedir?

- a) Bir sürecin CPU'nun kontrolünü gönüllü olarak bırakması.
- b) Bir kesme meydana gelmesi (zamanlayıcı veya Giriş/Çıkış işleminin tamamlanması).
- c) Bir sürecin paylaşılan bir kaynağa erişim isteğinde bulunması.
- d) Bir sürecin yürütmesini tamamlayıp çıkış yapması.

Cevap

- Cevap: B

- İş kesme üstünlüğü ile çok görevli sistemlerde, bağlam anahtarlama bir kesme (*interrupt*) meydana geldiğinde tetiklenir. Örneğin, bir zamanlayıcı kesmesi (*timer interrupt*) belirli bir süre dolduğunda veya bir Giriş/Çıkış işlemi tamamlandığında gerçekleşebilir. Bu tür bir kesme, işletim sistemi çekirdeği tarafından algılanır ve işlemci mevcut süreci duraklatır ve kesme işlemi tarafından işaret edilen başka bir sürece geçiş yapar.

Soru

- Tekrarlanamaz (*non-reentrant*) yürütme bağlamında, bir sürecin bir olayın gerçekleşmesini beklerken sürekli geciktiği durumu hangisi tanımlar?

- a) Kitlenme (*Deadlock*)
- b) Açlık (*Starvation*)
- c) Boşa çabalama (*Thrashing*)
- d) Boşa dönme (*Spinning*)

Cevap

- Cevap: B

- *Açlık* terimi, sürecin bir olayın gerçekleşmesini beklerken sürekli olarak geciktiği durumu tanımlar. Bu durumda, süreç bir olayın gerçekleşmesini beklerken, diğer süreçler sürekli olarak öncelikli hale gelirse veya beklenen olay hiç gerçekleşmezse bekleme durumda kalır. Bu durum, süreç için istenen kaynakların sürekli olarak diğer süreçler tarafından alınmasından veya kaynakların hiç serbest bırakılmamasından kaynaklanabilir.

Soru

- Tekrarlanamaz (*non-reentrant*) yürütme bağlamında, bir sürecin başka bir süreç tarafından tutulan kaynağı sonsuza kadar beklediği durumu hangi terim tanımlar?

- a) Açlık (*Starvation*)
- b) Kilitlenme (*Deadlock*)
- c) Boşa çabalama (*Thrashing*)
- d) Boşa dönme (*Spinning*)

Cevap

- Cevap: B

- *Kilitlenme* terimi, bir sürecin belirli bir kaynağı elinde tutan diğer süreci beklemesi nedeniyle ilerleyememesi durumudur. Bu durumda, her süreç diğerinin elindeki kaynağa erişmek için beklerken, hiçbiri ilerleyemez ve sistemde bir döngü oluşur. Örneğin, Süreç A, bir kaynağı elinde tutarken, Süreç B aynı kaynağı elde etmek için beklerken, Süreç A da başka bir kaynağı elde etmek için Süreç B'nin kaynağını bekleyebilir. Sonuç olarak, her iki süreç de kaynakları serbest bırakmayı bekler ve sistemde ilerleme olmaz.

Soru

- Birden fazla süreç arasında paylaşılan bellek bölgesine doğrudan erişim sağlayan IPC mekanizması hangisidir?

- a) Mesaj iletişimi (*Message passing*)
- b) Paylaşımlı bellek (*Shared memory*)
- c) İşaret flaması (*Semaphore*)
- d) Boru hattı (*Pipes*)

Cevap

- Cevap: B

- Paylaşımlı bellek mekanizması, birden fazla süreç arasında veri paylaşımını sağlar. Süreçlerin aynı bellek bölgesine doğrudan erişim sağlamasına izin verir, bu da veri kopyalama ve iletişim maliyetini azaltır. Süreçler, aynı bellek bölgesine yazabilir ve oradan okuyabilirler. Bu, veri aktarımının hızlı ve etkin olmasını sağlar. Paylaşımlı bellek kullanırken senkronizasyon önemlidir çünkü aynı bellek bölgesine birden fazla süreç aynı anda erişebilir ve bu da istenmeyen veri tutarsızlığı sonuçlarına yol açabilir.

Soru

- Süreçler arası iletişimde (*IPC*) senkron mesaj iletiminin özelliği nedir?

- a) Gönderici ve alıcı, iletişim sırasında senkronize olmak zorunda değildir.
- b) Gönderici, alıcının iletiyi aldığını doğrulayana kadar bekler.
- c) İletiler, alıcıya asenkron gönderim için bir tampon bellekte saklanır.
- d) Süreçler arasında koordinasyon gerektirmeden iletişim gerçekleşir.

Cevap

- Cevap: B

- Senkron mesaj iletiminde, gönderici bir ileti gönderdikten sonra iletiyi alıcının almasını bekler. Bu, gönderici ve alıcı arasında senkronizasyonu sağlar ve iletişimin güvenilirliğini artırır. Gönderici, iletiyi gönderdikten sonra alıcının onu aldığını doğrulamadan devam etmez. Bu, iletişimin güvenilir ve güvenli bir şekilde gerçekleşmesini sağlar, ancak iletişimde gecikmelere neden olabilir.

Soru

- Aynı ağ üzerinde farklı bilgisayarlarda çalışan süreçler arasındaki iletişim için hangi *IPC* mekanizması kullanılır?

- a) Paylaşımlı bellek (*Shared memory*)
- b) Mesaj iletişimi (*Message passing*)
- c) İşaret flaması (*Semaphore*)
- d) Boru hattı (*Pipes*)

Cevap

- Cevap: B

- Aynı ağ üzerinde farklı bilgisayarlarda çalışan süreçler arasında iletişim için genellikle mesaj iletişimi (*message passing*) kullanılır. Mesaj iletişimi, bir süreç veya bilgisayar tarafından diğerine mesaj gönderme ve almayı içerir. Mesaj iletişimi, iletişimdeki taraflar arasında bağlantı gerektirmeyen güvenilir bir iletişim mekanizmasıdır. Örneğin, ağ üzerindeki bir bilgisayar bir istemci süreci başlatır ve sunucuya bir istek mesajı gönderir. Sunucu, isteği alır, işler ve yanıt gönderir.

Soru

- *IPC* bağlamında, işaret flaması (semaphore) amacı nedir?

- a) Süreçler arasında paylaşılan belleğe doğrudan erişim
- b) Süreçler arasında veriyi senkronize bir şekilde transfer etme
- c) Eşzamanlı süreçler arasında paylaşılan kaynaklara senkronize erişim
- d) Süreçler arasında tek yönlü iletişim kanalı oluşturma

Cevap

- Cevap: C

- Semaforlar, eşzamanlı çalışan süreçler arasında paylaşılan kaynaklara erişimi senkronize eder. Birden fazla süreç aynı anda bir kaynağa erişmeye çalışıyorsa, semaforlar bu erişimi düzenleyerek çakışmaları önler ve kaynakların güvenli kullanılmasını sağlar. Örneğin, bir süreç bir dosyaya yazarken, diğer süreçlerin aynı dosyaya aynı anda yazmamasını sağlamak için kullanılabilir.

Soru

- Hangi *IPC* mekanizması, iki süreç arasında tek yönlü iletişim kanalı sağlar?

- a) Paylaşımlı bellek (*shared memory*)
- b) Mesaj iletişimi (*message passing*)
- c) İşaret flaması (*semaphore*)
- d) Boru hattı (*pipes*)

Cevap

- Cevap: D

- Boru hattı (*pipes*), bir çift süreç arasında tek yönlü iletişim kanalı sağlar. Bu kanal, veri akışının bir yönden diğerine doğru boru hattı gibi düşünülebilir. Bir süreç, boru hattına veri yazabilir ve diğer süreç bu hattan veriyi okuyabilir. Boru hattı bir süreç tarafından oluşturulur ve diğer süreçle iletişim kurmak için kullanılır. Örneğin, bir süreçten çıktı almak ve başka bir süreçten girdi almak için sıklıkla kullanılır. Boru hatları tek yönlü bir iletişim kanalı sağlar ve çift yönlü iletişim için kullanılamaz.

Soru

- Soketler, ağ ortamında süreçler arası iletişimde hangi rolü oynar?

- a) Uzak süreçler arasındaki paylaşılan bellek bölgelerine erişim sağlar.
- b) Aynı bilgisayarda çalışan süreçler arasında mesaj iletişimini kolaylaştırır.
- c) Farklı bilgisayarlarda çalışan süreçler için iletişim kanalları oluştururlar.
- d) Eş zamanlı süreçler arasında paylaşılan kaynaklara erişimi senkronize eder.

Cevap

- Cevap: C

- Soketler, farklı bilgisayarlarda çalışan süreçler arasında iletişim kurmak için kullanılır. İki farklı bilgisayar arasında bağlantı sağlayarak, veri alışverişi yapılmasını mümkün kılar. Soketler TCP/IP veya UDP/IP protokollerini kullanarak iletişim kurarlar ve çeşitli ağ uygulamalarında (web tarayıcıları, e-posta istemcileri, dosya paylaşımı programları vb.) kullanılırlar.

Soru

- Bir iş parçacığının paylaşılan kaynakta değişiklik yaptığında diğer iş parçacığının onu okuma veya değiştirme işlemi sürerken yaşanan durumu hangi terim tanımlar?

- a) Kilitlenme (*Deadlock*)
- b) Açlık (*Starvation*)
- c) Yarış koşulu (*Race condition*)
- d) Kritik bölge (*Critical region*)

Cevap

- Cevap: C

- Yarış koşulu (*Race condition*), çok iş parçacıklı programlamada (*multi-threaded programming*) karşılaşılan bir durumdur. Bir iş parçacığı paylaşılan kaynağı okurken veya değiştirirken, başka bir iş parçacığı da aynı kaynağa erişmeye çalışır. Eğer iş parçacıkları arasında senkronizasyon sağlanmamışsa ve kaynağın durumu kontrol edilmeden işlemler gerçekleştirilirse, beklenmedik sonuçlar ortaya çıkabilir. Örneğin, bir iş parçacığı bir değişkeni artırırken, diğeri aynı değişkeni okuyabilir veya değiştirebilir. Bu durumda, değişkenin son değeri tutarsız olabilir.

Soru

- Yarış koşullarını azaltmada *mutex* kilidin temel rolü nedir?

- a) İş parçacıkları arasında paylaşılan kaynaklara karşılıklı dışlama
- b) İş parçacıklarının aynı anda kaynaklara erişmesine izin verme
- c) İş parçacıkları arasında iletişimi kolaylaştırma
- d) İş parçacıklarına önceliklerine göre CPU zamanı tahsis etme

Cevap

- Cevap: A

- *Mutex* kilidi, yarış koşullarını önlemede önemli rol oynar. Paylaşılan bir kaynağa aynı anda sadece bir iş parçacığının erişmesini sağlayarak, diğer iş parçacıklarının beklemesini ve kaynağın güvenli bir şekilde kullanılmasını sağlar. Bu, eşzamanlı çalışan iş parçacıklarının birbirlerinin çalışmasını engeller ve veri bütünlüğünü korur. *Mutex* kilitleri, kritik bölge adı verilen kod parçasının aynı anda yalnızca bir iş parçacığı tarafından çalıştırılmasını sağlar.

Soru

- Kritik bölgenin amacı nedir?

- a) Kodu birden çok iş parçacığında eşzamanlı olarak yürütmek
- b) Süreçler arasında paylaşılan kaynaklara erişimi senkronize etmek
- c) Süreçlerin mesaj iletişimi yoluyla iletişim kurmasına izin vermek
- d) Süreçlere önceliklerine göre CPU zamanı tahsis etmek

Cevap

- Cevap: B

- Kritik bölge, karşılıklı dışlamanın sağlandığı kod parçasıdır. Bu bölgede, yalnızca bir süreç veya iş parçacığı aynı anda çalışabilir. Bu, paylaşılan bir kaynağa erişimi senkronize eder ve birden fazla süreç veya iş parçacığının aynı anda bu kaynağa erişimini önler. Bu nedenle, kritik bölge, paylaşılan kaynaklara güvenli bir şekilde erişim sağlamak için kullanılır ve yarış koşullarını önler.

Soru

- Karşılıklı dışlama bağlamında, *spinlock* kullanmanın dezavantajı nedir?

- a) Çok sık bağlam anahtarlama nedeniyle yüksek işlem maliyeti
- b) Kilidin serbest bırakılmasını bekleyen iş parçacıkları arasında kilitlenme
- c) CPU kaynaklarının verimsiz kullanımı
- d) Çok sayıda iş parçacığı için sınırlı ölçeklenebilirlik

Cevap

- Cevap: C

- *Spinlock*, bir kaynağa erişimi beklerken işlemciyi meşgul eder. Bu, işlemcinin sürekli olarak belirli bir kilit durumunu kontrol etmesine ve kilit serbest bırakılana kadar tekrar etmesine neden olur. Bu süreç, CPU kaynaklarının verimsiz kullanımına yol açar, çünkü işlemci kilit serbest bırakılana kadar başka işleri yapmak yerine sürekli olarak aynı işlemi tekrarlar.

Soru

- Karşılıklı dışlamanın performansa etkisini azaltmak için alınabilecek önlem nedir?

- a) Paylaşılan kaynaklara erişen iş parçacığı sayısını artırmak
- b) Kilitlerin süresini azaltmak için kritik bölgeyi optimize etmek
- c) Senkronizasyon için *mutex* kilitler yerine *spinlock* kullanmak
- d) Birden çok iş parçacığının kritik bölgeye eşzamanlı erişimine izin vermek

Cevap

- Cevap: B

- Karşılıklı dışlamanın performansa etkisini azaltmanın bir yolu, kritik bölgeyi optimize etmektir. Kritik bölge, bir iş parçacığının paylaşılan kaynağa erişirken diğer iş parçacıklarını engellemek için kullanılan kod parçasıdır. Kritik bölge, kilitlenme süresini en aza indirmek için mümkün olduğunca küçük olmalıdır. Kilitlerin süresini azaltmak için kritik bölgedeki işlemleri optimize etmek, iş parçacıklarının kilitlerin serbest bırakılmasını beklerken harcadığı süreyi azaltır.

Soru

- İşletim sistemlerinde kritik bölge nasıl tanımlanır?

- a) Sistem işlemleri için ayrılmış bellek bölümüdür.
- b) Paylaşılan kaynaklara erişilen ve eşzamanlı erişimden korunması gereken kod bölümüdür.
- c) Sık erişilen verilerin saklandığı CPU önbelleğinin bir parçasıdır.
- d) Sistem günlükleri ve hata mesajlarını saklamak için kullanılan disk depolama alanının bir parçasıdır.

Cevap

- Cevap: B

- Kritik bölge, bir işletim sistemi içinde, birden fazla iş parçacığı veya süreç tarafından kullanılan ve paylaşılan kaynaklara erişimi içeren kod parçasıdır. Bu kaynaklar genellikle bellek, dosya sistemi gibi sistem kaynakları veya donanım aygıtları olabilir. Kritik bölgeye girildiğinde, sadece bir iş parçacığı veya süreç tarafından erişilmesine izin verilir ve diğerlerinin erişimi engellenir. Bu, veri bütünlüğünün korunmasını sağlar ve yarış koşullarını önler.

Soru

- Kritik bölgeleri yönetmede semaforun amacı hangisidir?

- a) Yalnızca bir sürecin aynı anda kritik bölgeye girmesini sağlar.
- b) Birden çok sürecin aynı anda paylaşılan kaynağa erişmesine izin verir.
- c) Süreçler arasında mesaj iletişimini kolaylaştırır.
- d) Paylaşılan kaynaklara erişimi düzenleyerek yarış koşullarını önler.

Cevap

- Cevap: D

- Semaforlar, kritik bölgelere erişimi koordine ve senkronize ederek, birden çok iş parçacığının aynı anda paylaşılan kaynaklara erişmesini önler. Bu, yarış koşullarını ve veri bütünlüğünü korur. Semaforlar genellikle iş parçacıklarının veya süreçlerin kritik bölgeye girmesine izin veren veya engelleyen bir tür senkronizasyon mekanizması olarak kullanılır.

Soru

- *Peterson* çözümü, işletim sistemlerindeki hangi sorunu çözmeyi amaçlar?

- a) Kilitlenme (*Deadlock*)
- b) Yarış koşulu (*Race condition*)
- c) Açlık (*Starvation*)
- d) Boşa çabalama (*Thrashing*)

Cevap

- Cevap: B

- *Peterson* çözümü, karşılıklı dışlama ve senkronizasyonun sağlanması gereken çok iş parçacıklı veya çok işlemcili sistemlerde, yani eş zamanlı programlama ortamlarında yarış koşullarını çözmeyi amaçlar. Yarış koşulları, birden çok iş parçacığının veya sürecin aynı anda paylaşılan bir kaynağa erişmeye çalışması durumunda ortaya çıkan sorunlardır. Peterson çözümü, bu tür yarış koşullarını önlemek için bir algoritma sağlar, böylece paylaşılan kaynaklara güvenli bir şekilde erişim sağlanır ve veri bütünlüğü korunur.

Soru

- *Peterson* çözümünü doğru bir şekilde tanımlayan ifade hangisidir?

- a) Aynı anda yalnızca bir sürecin kritik bölgeye girebilmesini sağlar.
- b) Aynı anda birden çok sürecin paylaşılan kaynağa erişimine izin verir.
- c) Süreçler arasında mesaj iletişimini kolaylaştırır.
- d) Yarış koşullarını, paylaşılan kaynaklara erişimi düzenleyerek önler.

Cevap

- Cevap: A

- *Peterson* çözümü, eş zamanlı programlama ortamında paylaşılan kaynaklara güvenli bir şekilde erişimi sağlamak için kullanılır. Bu çözüm, aynı anda yalnızca bir sürecin kritik bölgeye girebileceğini garanti eder. Bu, kritik bölgedeki paylaşılan kaynakların güvenli bir şekilde kullanılmasını sağlar ve yarış koşullarını önler.

Soru

- *Peterson* çözümü karşılıklı dışlamayı nasıl sağlar?

- a) Kritik bölgede meşgul bekleyerek
- b) *Mutex* kilitleri yerine *spinlock* kullanarak
- c) Süreçler arasında koordinasyon ve senkronizasyon kullanarak
- d) Birden çok sürecin aynı anda kilitleri tutmasına izin vererek

Cevap

- Cevap: C

- *Peterson* çözümü, süreçler arasında karşılıklı dışlamayı sağlamak için uygun koordinasyon ve senkronizasyon mekanizmalarını kullanır. Süreçlerin kritik bölgeye girişlerini ve çıkışlarını düzenler ve aynı anda yalnızca bir sürecin kritik bölgeye girmesini sağlar.

Soru

- *Peterson* çözümünde paylaşılan kaynaklara erişimi koordine etmek için kullanılan veri yapısı nedir?

- a) Semafor (*semaphore*)
- b) Mutex kilidi (*mutex lock*)
- c) Bariyer (*barriers*)
- d) Bayrak dizisi (*flag array*)

Cevap

- Cevap: D

- *Peterson* çözümünde, süreçler arasında paylaşılan kaynaklara erişimi koordine etmek için bayrak dizisi kullanılır. Bu dizi, her süreç için ayrı bir bayrağın tutulduğu veri yapısıdır. Her süreç, kritik bölgeye girmek istediğinde önce kendi bayrağını kaldırır ve daha sonra diğer süreçlerin bayraklarını kontrol ederek kritik bölgeye girer.

Soru

- Uyandırma (*wake-up*) işleminin amacı nedir?

- a) Bir süreci uyutarak geçici olarak çalışmasını askıya almak
- b) Bir süreci sonlandırarak ayrılmış kaynaklarını serbest bırakmak
- c) Daha önce uyuyan bir sürecin çalışmasını yeniden başlatmak
- d) Bir sürece önceliğine bağlı olarak CPU zamanı tahsis etmek

Cevap

- Cevap: C

- *Uyandırma* işlemi, uyuyan bir sürecin çalışmasını yeniden başlatır. Bir süreç, bir olayın gerçekleşmesini beklerken uyur duruma geçer. Beklenen olay gerçekleştiğinde, işletim sistemi bu süreci uyandırır ve süreç tekrar çalışmaya başlar. Bu durum, örneğin bir G/Ç işlemi tamamlandığında veya bir sinyal alındığında gerçekleşebilir. Bu sayede, süreç sırayla uyku ve uyanıklık durumları arasında geçiş yapabilir ve işletim sistemi kaynakları etkin bir şekilde yönetebilir.

Soru

- İşletim sisteminde *wake-up sleep* mekanizmasının temel avantajı nedir?

- a) Bağlam anahtarlamanın getirdiği ek yükü azaltır
- b) Süreçlerin mesaj iletişimi yoluyla iletişim kurmasını sağlar
- c) CPU kaynaklarını verimli yöneterek sistem yanıt verilebilirliğini artırır
- d) Eş zamanlı süreçler arasında adil davranarak açlık durumunu önler

Cevap

- Cevap: C

- *wake-up sleep* mekanizmasının temel avantajı, CPU kaynaklarını verimli bir şekilde kullanarak sistem yanıt verme tepki süresini azaltmasıdır. Bu mekanizma, işletim sisteminin uyku durumundaki süreçleri uyandırmasına ve süreçlere CPU kaynakları sağlamasına olanak tanır.

Soru

- Boru hattının (*pipe*) asıl amacı nedir?

- a) Bellek kaynaklarını süreçlere tahsis etmek
- b) Eş zamanlı süreçler arasında paylaşılan kaynaklara senkronize erişim
- c) Süreçler arasında tek yönlü iletişim kanalı oluşturmak
- d) Süreçlere önceliklerine göre CPU zamanı tahsis etmek

Cevap

- Cevap: C

- Boru hattı, süreçler arasında tek yönlü iletişim kanalı oluşturur. Bir süreç tarafından yazılan veriler başka bir süreç tarafından okunabilir. Genellikle bir sürecin çıktısını diğer bir sürece girdi olarak aktarmak için kullanılır. Örneğin, bir süreç bir komutun çıktısını üretirken, bu çıktı bir boru hattı aracılığıyla başka bir sürece aktarılabilir ve bu süreç bu çıktıyı kullanarak başka bir işlemi gerçekleştirebilir.

Soru

- Boru hattı (*pipe*) hakkında hangi ifade doğrudur?

- a) Süreçler arasında çift yönlü iletişime olanak tanır.
- b) Süreçler arasında paylaşılan bellek bölgelerine doğrudan erişim sağlar.
- c) Süreçler arası iletişimde kullanılan *first-in-first-out* (*FIFO*) veri yapısıdır.
- d) Süreçler arasındaki iletişimi mesaj geçişi yoluyla kolaylaştırır.

Cevap

- Cevap: C

- Boru hattı (*pipe*), süreçler arasında *first-in-first-out* (*FIFO*) mantığına göre çalışır. Bir süreç tarafından gönderilen veriler, alıcı tarafından aynı sırayla alınır. Bir süreç bir veri gönderdiğinde, alıcı bu veriyi aldıktan sonra bir sonraki gönderilen veriyi alabilir.

Soru

- Boru hattının (*pipe*) özelliği nedir?

- a) Süreçler arasında çift yönlü iletişime olanak tanır.
- b) İki süreç arasında noktadan noktaya bağlantı kurar.
- c) Süreçler arasında paylaşılan bellek bölgelerine doğrudan erişim sağlar.
- d) Birden fazla süreç arasında çoklu yayın iletişimini destekler.

Cevap

- Cevap: B

- Boru hattı (*pipe*), iki süreç arasında noktadan noktaya bağlantı kurar. Bir boru hattı bir süreç tarafından oluşturulur ve diğer süreç tarafından kullanılır. Bu bağlantı, bir süreç tarafından yazılan verilerin diğer süreç tarafından okunmasını sağlar.

Soru

- Bir süreç okuma yapacağında boru hattı boş ise ne olur?

- a) Süreç, boru hattında veri mevcut olana kadar engellenir.
- b) Boru hattının boş olduğunu belirten bir hata döndürülür.
- c) Süreç, hiçbir verinin mevcut olmadığını belirten *null* cevabı alır.
- d) Süreç, boru hattında veri mevcut olana kadar uyku durumuna geçer.

Cevap

- Cevap: A

- Bir süreç, boru hattı boş iken, okuma yapmaya çalıştığında beklemeye alınır. Yani, süreç boru hattında veri mevcut olana kadar bekler ve o zamana kadar askıya alınır. Bu işlem, boru hattına veri yazılana kadar devam eder.

Soru

- Boru hattını (*pipe*) uygulamak için hangi mekanizma kullanılır?

- a) Paylaşımlı bellek
- b) Semafor tabanlı senkronizasyon
- c) Dosya tanımlayıcıları
- d) Sistem çağrıları

Cevap

- Cevap: C

- Boru hattını uygulamak için dosya tanımlayıcıları kullanılır. Boru hatları, dosya benzeri bir arayüz sağlar ve bu nedenle işletim sistemi bunları dosya tanımlayıcıları aracılığıyla yönetir. Bir süreç, bir boru hattına yazmak veya bir boru hattından okumak için ilgili dosya tanımlayıcısını (*file identifier*) kullanır.

Soru

- Yemek yiyen filozoflar probleminde (*Dining Philosophers*) her filozof neyi temsil eder?

- a) Ortak kaynağa erişim için rekabet eden bir süreci temsil eder.
- b) Bir *mutex* kilidine erişmeye çalışan iş parçacığını temsil eder.
- c) Kritik kod bölgesini yürüten CPU çekirdeğini temsil eder.
- d) Bir kesmeye hizmet etmek için bekleyen aygıtı temsil eder.

Cevap

- Cevap: A

- Yemek yiyen filozoflar problemi, bir grup filozofun bir masada yemek yemesi ve ortak bir kaynak olan çatal setini kullanması durumunu modelleyen senkronizasyon sorunudur. Her filozof, bir süreci temsil eder ve ortak kaynağa (çatal setine) erişmek için rekabet eder. Bu problemde, filozofların çatalları nasıl kullanacakları ve kullandıktan sonra diğer filozoflara ne zaman serbest bırakacakları gibi senkronizasyon sorunları bulunmaktadır.

Soru

- Yemek yiyen filozoflar problemi (*Dining Philosophers*), hangi ana zorlukları ortaya çıkarır?

- a) Kilitlenme (*Deadlock*)
- b) Açlık (*Starvation*)
- c) Yarış koşulu (*Race condition*)
- d) Bağlam anahtarlama ek yükü (*Context switching overhead*)

Cevap

- Cevap: A

- Yemek yiyen filozoflar problemi, kilitlenme durumlarına yol açan senkronizasyon problemidir. *Deadlock*, sistemdeki süreçlerin birbirlerinin tamamlanmasını beklerken kilitlenmesi durumudur. Yemek yiyen filozoflar problemi özelinde, her filozofun sağındaki ve solundaki çatalı kullanması gerekmektedir. Ancak, bu çatal seti paylaşılan bir kaynaktır ve doğru sırayla çatalı kullanmaları gerekmektedir. Eğer her filozof çatalı almak için beklerse kilitlenme durumu ortaya çıkar.

Soru

- Yemek yiyen filozoflar probleminde (*Dining Philosophers*) her filozof kaç tane kaynağa (çatala) ihtiyaç duyar?

- a) Bir
- b) İki
- c) Üç
- d) Dört

Cevap

- Cevap: B

- Yemek yiyen filozoflar problemi, her filozofun iki adet kaynağa (çatala) ihtiyaç duyduğu klasik bir senkronizasyon problemidir. Her filozof, yemek yemek için sağında ve solundaki çatalı kullanmalıdır. Dolayısıyla, her filozof iki adet çatala ihtiyaç duyar.

Soru

- Yemek yiyen filozoflar probleminde (*Dining Philosophers*) semafor kullanmanın amacı nedir?

- a) Filozoflar arasında kilitlenmeyi önlemek için
- b) Paylaşılan kaynaklara (çatal) karşılıklı dışlama sağlamak için
- c) Birden fazla filozofun aynı anda çatala erişmesine izin vermek için
- d) Filozoflar arasında yemek masasına erişimi senkronize etmek için

Cevap

- Cevap: B

- Yemek yiyen filozoflar problemi, filozofların ortak kaynak olan çatal setine güvenli bir şekilde erişmesi için senkronizasyon mekanizmalarının kullanılmasını gerektirir. Semaforlar, paylaşılan kaynaklara (çatala) karşılıklı dışlama sağlayarak, yani aynı anda sadece bir filozofun bir çatalı almasını sağlayarak, kaynakların doğru şekilde kullanılmasını sağlar.

Soru

- Yemek yiyen filozoflar probleminde kilitlenmeyi önlemek için hangi koşul sağlanmalıdır?

- a) En az bir filozof tüm gerekli kaynakları edinebilmelidir.
- b) Her filozof, sol çatalı almadan önce sağ çatalı beklemelidir.
- c) Filozoflar, dairesel bekleme (*circular wait*) durumunu önlemek için kaynakları sıkı bir sıra ile serbest bırakmalıdır.
- d) Filozoflar senkronizasyon olmadan kaynaklara aynı anda erişmelidir.

Cevap

- Cevap: C

- Yemek yiyen filozoflar problemi, kilitlenme oluşumunu engellemek için filozofların çatalı sıkı bir sıra ile alıp bırakmasını gerektirir. Bu, dairesel bekleme durumunu önler. Örneğin, sağ çatalı alamayan filozof sol çatalı serbest bırakmalıdır. Bu sayede, herhangi bir filozof, bir çatalı almak için beklerken diğer çatalı gereksiz yere tutmayacak, ve dairesel bir bekleme durumu oluşturmayacaktır.

Soru

- Çok iş parçacıklı programlama kullanmanın temel avantajı nedir?

- a) Azaltılmış bellek tüketimi
- b) Geliştirilmiş hata toleransı
- c) Daha iyi yanıt verme ve eş zamanlılık
- d) Basitleştirilmiş programlama modeli

Cevap

- Cevap: C

- Çok iş parçacıklı programlama (*multithreading*), birçok avantaj sunar, en önemlisi sistem yanıt tepki süresini azaltması ve eş zamanlılık sağlamasıdır. Çoklu iş parçacığı sayesinde, bir süreç içindeki birden fazla iş parçacığı aynı anda çalışabilir, böylece sistem daha hızlı ve daha etkili bir şekilde yanıt verebilir. Aynı zamanda, birden fazla iş parçacığı arasındaki işbirliği ve veri paylaşımı daha kolay hale gelir, bu da genel sistem performansını artırır.

Soru

- Çok iş parçacıklı (*multithreading*) kavramında, bir iş parçacığı (*thread*) neyi temsil eder?

- a) Kendi bellek alanına sahip ayrı bir süreci
- b) Süreç içindeki aynı bellek alanını paylaşan yürütme birimini
- c) Mesaj iletimi yoluyla iletişim kuran bağımsız bir süreci
- d) Süreçler arasında iletişimi kolaylaştıran donanım bileşenini

Cevap

- Cevap: B

- İş parçacığı (*thread*), süreç içinde yürütme birimlerden biridir ve aynı süreç içindeki diğer iş parçacıkları ile aynı bellek alanını paylaşır. Bu, iş parçacıklarının birbirleriyle doğrudan iletişim kurmalarını ve aynı bellek alanını kullanarak veri paylaşmalarını sağlar. Bir iş parçacığı, bir süreç içindeki diğer iş parçacıkları ile işbirliği yaparak belirli bir görevi gerçekleştirmek için birlikte çalışır.

Soru

- Çoklu iş parçacığı (*multithreading*) çoklu süreçten (*multiprocessing*) nasıl farklıdır?

- a) Çoklu iş parçacığı, farklı işlemcilerde yürütülen birden fazla süreci içerir.
- b) Çoklu süreç, bir süreç içinde birden fazla iş parçacığının yürütülmesine izin verir.
- c) Çoklu iş parçacığı, bir süreç içindeki iş parçacıkları arasında aynı bellek alanını paylaşır.
- d) Çoklu süreç, süreçleri birbirinden izole ederek hata toleransını artırır.

Cevap

- Cevap: C

- Çoklu iş parçacığı (*multithreading*) ve çoklu süreç (*multiprocessing*), paralel yürütme sağlayan iki farklı yaklaşımdır. Çoklu iş parçacığı, bir süreç içinde birden fazla iş parçacığının yürütülmesini sağlar ve bu iş parçacıkları aynı süreç içindeki diğer iş parçacıkları ile aynı bellek alanını paylaşır. Bu, veri paylaşımını ve işbirliğini kolaylaştırır. Öte yandan, çoklu süreç (multiprocessing), farklı işlemciler üzerinde farklı süreçlerin yürütülmesine izin verir. Her süreç kendi ayrı bellek alanına sahiptir ve iş parçacıkları arasında veri paylaşımı doğrudan olmaz.

Soru

- Kullanıcı düzeyi iş parçacıklarını (*user-level threads*) çekirdek düzeyi iş parçacıklarından (*kernel-level threads*) ayıran özellik nedir?

- a) Kullanıcı düzeyi iş parçacıkları, doğrudan donanıma erişime sahiptir.
- b) Çekirdek düzeyi iş parçacıkları, işletim sistemi tarafından yönetilir.
- c) Kullanıcı düzeyi iş parçacıkları, CPU'ya yakınlığı nedeniyle daha hızlıdır.
- d) Çekirdek düzeyi iş parçacıkları, kullanıcı uygulamaları tarafından oluşturulabilir ve yönetilebilir.

Cevap

- Cevap: B

- Kullanıcı düzeyi ve çekirdek düzeyi iş parçacıkları arasındaki ana fark, nasıl yönetildikleridir. Çekirdek düzeyi iş parçacıkları, işletim sisteminin çekirdeği tarafından yönetilir. İşletim sistemi, iş parçacıklarını oluşturur, çizelgeler, duraklatır ve sonlandırır. Kullanıcı düzeyi iş parçacıkları ise kullanıcı tarafından oluşturulur ve yönetilir. İşletim sistemi, kullanıcı düzeyi iş parçacıklarını doğrudan yönetmez. Kullanıcı, iş parçacıklarını oluşturur, çizelgeler ve çeşitli iş parçacığı kütüphaneleri aracılığıyla iş parçacıklarının yönetimini sağlar.

Soru

- Kullanıcı düzeyi iş parçacığı kütüphanesi, engelleyici (*blocking*) sistem çağrılarını nasıl işler?

- a) Çağrı tamamlanana kadar farklı bir kullanıcı iş parçacığına geçer.
- b) Çağrıyı çekirdek düzeyi iş parçacığı yöneticisine devreder.
- c) Çağrı tamamlanana kadar tüm kullanıcı iş parçacıklarını askıya alır.
- d) Engellenen çağrıyı iptal eder ve yürütmeye devam eder.

Cevap

- Cevap: A

- Kullanıcı düzeyi iş parçacığı kütüphanesi, engelleyici sistem çağrılarını işlerken blokeden kaçınmak için bazı yöntemler kullanır. Bu yöntemlerden biri, engellenen bir çağrının etkilerini azaltmak için o anda bloke olan iş parçacığını başka bir kullanıcı düzeyi iş parçacığıyla değiştirmektir. Böylece, engellenen çağrının tamamlanmasını beklerken başka bir iş parçacığı yürütülür ve işlemci zamanı daha verimli kullanılır.

Soru

- Kullanıcı düzeyi iş parçacıklarının dezavantajına yol açabilecek bir senaryo nedir?

- a) Ayrıntılı (*fine-grained*) iş parçacığı kontrolüne ihtiyaç duyulduğunda
- b) Farklı işletim sistemleri arasında taşınabilirlik gerektiğinde
- c) Uygulama yüksek performans ve ölçeklenebilirlik gerektirdiğinde
- d) Uygulama, G/Ç işlemleri için sistem çağrılarına yoğun bir şekilde bağımlı olduğunda

Cevap

- Cevap: D

- Kullanıcı düzeyi iş parçacıkları, iş parçacıklarının yönetimi konusunda kullanıcıya daha fazla kontrol sağlar, ancak bu durum bazı senaryolarda dezavantajlara yol açabilir. Özellikle, uygulamanın G/Ç (giriş/çıkış) işlemleri için sistem çağrılarına bağımlı olduğunda, kullanıcı düzeyi iş parçacıkları dezavantajlı olur. Sistem çağrıları, işletim sistemi çekirdeği içinde gerçekleşir ve iş parçacıkları bu çağrıların tamamlanmasını beklerken bloke olur. Eğer bir iş parçacığı G/Ç işlemi başlatırsa ve bu işlem bloke olursa, bu iş parçacığı tarafından kullanılan diğer iş parçacıkları beklemek zorunda kalabilir. Bu durum, uygulamanın yanıt verme süresini ve performansı etkileyebilir.

Soru

- Öncelik tabanlı çizelgeleme algoritmanın özellikleri nelerdir?

- a) Tüm süreçlere eşit CPU zamanı ayırarak adil bir dağılım sağlar.
- b) Süreçleri varış zamanlarına göre önceliklendirir.
- c) Her sürece sayısal bir öncelik değeri atar.
- d) Çizelgeleme kararları için süreçlerin çalışma süresine bakar.

Cevap

- Cevap: C

- Öncelik tabanlı çizelgeleme algoritmaları, her sürece öncelik değeri atayarak çalışır. Bu öncelik değerleri, süreçlerin önem derecesini veya önceliğini temsil eder. Öncelik değeri yüksek olan işlemler, düşük olanlara göre önce çalıştırılır.

Soru

- Süreçlerin ortalama dönüş (*turnaround*) süresini en aza indirmeyi hedefleyen çizelgeleme algoritması hangisidir?

- a) En kısa süresi kalan önce (*Shortest remaining time first - SRTF*)
- b) Sıralı (*Round robin - RR*)
- c) En kısa iş önce (*Shortest job next - SJN*)
- d) İlk gelen, ilk hizmet (*First-come first-served - FCFS*)

Cevap

- Cevap: A

- En kısa süresi kalan önce (*SRTF*) algoritması, süreçleri geriye kalan tamamlanma sürelerine göre önceliklendirir. Bu algoritma, her zaman CPU'ya sonlanmasına en kısa süresi kalan süreci atar, ortalama dönüş süresini minimize eder.

Soru

- En kısa iş önce (*SJN*) çizelgelemenin dezavantajlarından biri nedir?

- a) Uzun süre çalışan süreçlerin açlık yaşamasına neden olabilir.
- b) Her sürecin yürütme süresinin bilinmesini gerektirir.
- c) Kesme tabanlı çizelgeleme (*preemptive scheduling*) desteği sağlamaz.
- d) Bağlam anahtarlama ek maliyeti yüksek olur.

Cevap

- Cevap: B

- En kısa iş önce (*SJN*), her sürecin yürütme süresinin önceden bilinmesine dayanır ve işlemciye en kısa sürecek süreci seçer. Ancak, gerçek dünyada her sürecin yürütme süresini önceden bilmek imkansız. *SJN* çizelgeleme pratikte uygulanması zor.

Soru

- Basit ve adil olması ile tanınan çizelgeleme algoritması hangisidir?

- a) İlk gelen, ilk hizmet (*FCFS*)
- b) Round robin (*RR*)
- c) Çok seviyeli geri besleme kuyruğu (*Multilevel feedback queue*)
- d) En kısa iş önce (*SJN*)

Cevap

- Cevap: B

- *Round robin*, basitliği ve adil davranışıyla bilinir. Süreçlere eşit parçalar halinde zaman paylaşımı yapar ve her bir süreç için sırayla CPU zamanı tahsis eder. Tüm süreçler eşit fırsatlara sahiptir ve adil bir şekilde işlemci kaynaklarından yararlanır.

Soru

- Kesme tabanlı (preemptive) ile kesme tabanlı olmayan (non-preemptive) çizelgeleme arasındaki fark nedir?

- a) Kesme tabanlı, süreçlerin CPU'yu gönüllü bırakmasına izin verir.
- b) Kesme tabanlı, süreçleri zorla duraklatarak, CPU zamanını daha yüksek öncelikli süreçlere tahsis eder.
- c) Kesme tabanlı olmayan, süreçlerin CPU'yu gönüllü bırakana kadar çalışmasına izin verir.
- d) Kesme tabanlı olmayan, CPU zamanını süreç önceliğine göre ayırır.

Cevap

- Cevap: B

- Kesme tabanlı çizelgeleme, süreçleri zorla duraklatarak CPU zamanını daha yüksek öncelikli süreçlere tahsis eder. Bu durum, yüksek öncelikli süreçlerin öncelikli olarak çalışmasını sağlar ve düşük öncelikli süreçlerin CPU'yu kullanmasını engeller. Kesme tabanlı çizelgeleme, CPU zamanını sürekli olarak yüksek öncelikli süreçlere atayarak sistem yanıt süresini artırabilir.

Soru

- En kısa iş önce (*SJN*) algoritmasının sınırlaması nedir?

- a) Açlığa (*starvation*) neden olabilir.
- b) CPU kullanma (*burst*) sürelerinin bilgisini gerektirir.
- c) Kesme tabanlı çizelgeleme için uygun değildir.
- d) Sık sık bağlam anahtarlamadan dolayı yüksek ek iş yüküne neden olur.

Cevap

- Cevap: B

- En kısa iş önce (*SJN*), her süreç için tahmini CPU kullanım (*burst*) sürelerine dayanır ve her zaman en kısa sürecek süreci seçer. Ancak, gerçek CPU kullanım sürelerini tam olarak bilmek imkansızdır. Bu nedenle, *SJN* gerçek dünya senaryolarında uygulanması zor.

Soru

- Hangi çizelgeleme algoritması işlem önceliklerini dinamik olarak ayarlar?

- a) En kısa süresi kalan önce (*Shortest remaining time first*)
- b) Çok seviyeli geri besleme kuyruğu (*Multilevel feedback queue*)
- c) Öncelik tabanlı (*Priority based*)
- d) Sıralı (*Round robin*)

Cevap

- Cevap: B

- Çok seviyeli geri besleme kuyruğu çizelgeleme algoritması, süreç önceliklerini dinamik olarak ayarlayarak süreçlerin davranışlarına uyum sağlar. Süreçlerin önceliklerini, davranışlarına (örneğin, CPU kullanımı, vb.) göre değiştirebilir. Öncelikli süreçler daha fazla CPU zamanı alırken, az öncelikli süreçler daha az CPU zamanı alır. Sistemdeki iş yükünü daha dengeli bir şekilde dağıtır.

Soru

- Hangi çizelgeleme algoritması, etkileşimli süreçler için daha iyi yanıt süresi sağlamayı hedefler?

- a) İlk gelen, ilk hizmet (*FCFS*)
- b) Öncelik tabanlı (*Priority*)
- c) En kısa iş önce (*SJN*)
- d) Round robin (*RR*)

Cevap

- Cevap: B

- Öncelik tabanlı çizelgeleme, süreçlere öncelik değerleri atar ve bu değerlere göre süreçleri önceliklendirir. Yüksek öncelikli süreçlerin daha hızlı CPU'ya erişmesini sağlar. Etkileşimli süreçler daha yüksek önceliklere sahiptir çünkü kullanıcılarının hızlı yanıt almasını gerektirirler.

Soru

- Hangi çizelgeleme algoritması konvoy etkisine neden olabilir?

- a) Öncelik tabanlı (*Priority based*)
- b) En kısa süresi kalan önce (*SRTF*)
- c) İlk gelen, ilk hizmet (*FCFS*)
- d) Round robin (*RR*)

Cevap

- Cevap: C

- İlk gelen, ilk hizmet (FCFS) algoritması, süreçleri sırayla işleme alır ve işlem sırası gelene işlemci kaynaklarını tahsis eder. Uzun süre çalışan bir süreç diğer süreçlerinin tamamlanmasını beklemek zorunda kalabilir, bu durum kısa sürecek süreçlerin bekleme sürelerini artırabilir. Kısa süreçlerin *konvoy* halinde uzun süreli süreçlerin arkasında beklemesine neden olabilir.

Soru

- Hangi çizelgeleme algoritması yaşlanmaya (*aging*) eğilimlidir?

- a) En kısa iş önce (*SJN*)
- b) Öncelik tabanlı (*Priority*)
- c) Çok seviyeli geri besleme kuyruğu (*MLFQ*)
- d) Round robin (*RR*)

Cevap

- Cevap: C

- Çok seviyeli geri besleme kuyruğu algoritması, süreçlerin önceliklerini ve işlem sıralamasını dinamik olarak ayarlar. Bir süreç belirli bir süre boyunca düşük öncelik seviyesinde kaldıysa, önceliği artırılarak öne çıkarılır. Bu şekilde, uzun süre bekleyen yaşlı süreçlerin önceliği artar ve öncelikli bir şekilde işlenir, bu da *yaşlanma fenomenini* önler.

Soru

- *spinlock* kullanmanın potansiyel dezavantajı nedir?

- a) Sık sık bağlam anahtarlamadan kaynaklanan artan ek iş yükü
- b) Öncelik tersine çevirme (*priority inversion*) riski
- c) Daha yüksek kilitlenme olasılığı
- d) Daha yüksek CPU kullanımı

Cevap

- Cevap: D

- *Spinlock*, bir kaynağa erişim sağlamak için sürekli olarak kaynağı kontrol ederek iş parçacıklarını bekletme stratejisi kullanır. İşlemciyi sürekli meşgul eder ve çalışır durumda tutar.

Soru

- Hangi senkronizasyon ilkesi, iş parçacıklarının kısa bir süreliğine kilidi bırakmasına ve bir koşulun karşılanmasını beklemesine izin verir?

- a) Bariyer (*Barrier*)
- b) Mutex kilidi (*Mutex lock*)
- c) Semafor (*Semaphore*)
- d) Koşul değişkeni (*Condition variable*)

Cevap

- Cevap: D

- Koşul değişkeni, bir iş parçacığının kritik bölgeden (kilitlemeyle korunan bölge) çıkmasına ve bir koşulun karşılanmasını beklemesine izin verir. İş parçacığı, koşul değişkeni üzerinde sinyal alana kadar bekler ve bu sinyal, diğer bir iş parçacığı tarafından belirli bir koşulun karşılandığını gösterir. Bu sayede iş parçacığı, kilidi bırakabilir ve beklemeye geçebilir, diğer iş parçacıkları kritik bölgeye erişebilir. Koşul değişkeni, iş parçacıklarının birbirleriyle iletişim kurmasını ve işbirliği yapmasını sağlar.

Soru

- Hangi senkronizasyon mekanizması, iş parçacıklarının yürütmesini belirlenen bir noktada senkronize etmelerine izin verir?

- a) Mutex kilidi (*Mutex lock*)
- b) Bariyer (*Barrier*)
- c) Koşul değişkeni (*Condition variable*)
- d) Semafor (*Semaphore*)

Cevap

- Cevap: B

- Bariyer, bir dizi iş parçacığının belirli bir noktada eşleşmesini sağlar. İş parçacıkları, bariyer noktasına ulaştıklarında diğer iş parçacıklarının da aynı noktaya ulaşmasını beklerler. Tüm iş parçacıkları belirli bir noktaya ulaştığında, bariyer kaldırılır ve iş parçacıkları eşzamanlı olarak çalımaya devam eder.

Soru

- *mutex* kilidini *spinlock*'tan ayıran özellik nedir?

- a) Mutex kilitleri bloke etmeyen (*non-blocking*) yapılara sahiptir, *spinlock* iş parçacıklarının belirsiz süre beklemesine neden olabilir.
- b) Mutex kilitleri öncelik devrini (*inversion*) sağlar, spinlock sağlamaz.
- c) Mutex kilitleri kullanıcı alanında (*user space*), spinlock çekirdek alanında (*kernel space*) uygulanır.
- d) Mutex kilitleri, spinlock'lara kıyasla kısa kritik bölgelerde verimsizdir.

Cevap

- Cevap: A

- *Mutex* kilitleri, kilitleme işlemi başarısız olduğunda iş parçacıklarının beklemesine ve uyumasına neden olurken, *spinlock* iş parçacıklarının belirli bir koşulu beklemesi için sürekli olarak döngüye girerler. *spinlock* kullanılırken, bir iş parçacığı diğer iş parçacığının kilidini bırakmasını beklerken sürekli işlemci kaynağı kullanır ve bu, işlemciyi gereksiz bir şekilde meşgul eder. Bunun aksine, mutex kilit kullanılırken, iş parçacıkları diğer iş parçacıklarının kilidi bırakmasını beklerken uyuyabilir ve işlemci kaynağı daha etkin kullanılır.

Soru

- Senkronizasyonda *okuyucu-yazar* kilidinin asıl amacı nedir?

- a) Birden fazla iş parçacığının paylaşılan kaynağı aynı anda okumasına izin vermek, ancak yalnızca bir iş parçacığının yazmasına izin vermek
- b) Aynı kaynağa erişen birden fazla okuyucu arasında kilitlenmeyi önlemek
- c) Yazarların paylaşılan kaynağa erişiminde okuyuculara karşı önceliğini sağlamak
- d) Paylaşılan kaynağa erişen hem okuyuculara hem yazarlara karşılıklı dışlama sağlamak

Cevap

- Cevap: A

- Okuyucu-yazar kilidi, paylaşılan kaynağa aynı anda birden fazla okuma sürecine izin verirken, yalnızca bir yazma sürecine izin veren senkronizasyon mekanizmasıdır. Birden çok iş parçacığının kaynağı okumasına izin vererek performansı artırırken, aynı anda yalnızca bir iş parçacığının kaynağı değiştirmesine izin vererek veri tutarlılığı sağlar.

Soru

- Süreçlerin birbirlerini beklemesi sonucu oluşan kilitlenme (*deadlock*) için gerekli koşul hangisidir?

- A) Karşılıklı dışlama (*Mutual exclusion*)
- B) Tut ve bekle (*Hold and wait*)
- C) Kesme yok (*No preemption*)
- D) Döngüsel bekleme (*Circular wait*)

Cevap

- Cevap: D

- Kilitlenme, sistemde birden fazla sürecin birbirlerinin kaynaklarını kullanabilmeleri için birbirlerini beklemeleri ve böylece hiçbir sürecin ilerlememesi durumudur. Kilitlenme için gerekli olan koşullardan biri döngüsel beklemedir. Bu, süreçler arasında bir zincirleme etkisi yaratarak hiçbir sürecin ilerlemesine izin vermez.

Soru

- Sistemde kilitlenmeleri (*deadlock*) önlemek için hangi algoritma bekle-öl şeması (*wait-die scheme)* kullanır?

- A) Banker's algoritması (*Banker's algorithm*)
- B) Bekleme çizgesi (*Wait-for graph*)
- C) Kaynak tahsis çizgesi (*Resource allocation graph*)
- D) Öncelik devralma protokolü (*Priority inheritance protocol*)

Cevap

- Cevap: C

- Kaynak Tahsis Çizgesi Algoritması, zaman damgalarıyla birlikte kullanıldığında bekle-öl stratejisi uygulamak için kullanılabilir. Algoritma, kaynak tahsisini ve süreç taleplerini izler. Daha eski zaman damgasına sahip süreç (T1) başka bir süreç (T2) tarafından tutulan bir kaynak talep ederse, T1, T2'nin kaynağı serbest bırakmasını bekler. Yeni zaman damgasına sahip bir süreç (T1) başka bir süreç (T2) tarafından tutulan bir kaynak talep ederse, T1 iptal edilir ve sonra yeniden başlatılır.

Soru

- Banker's algoritmasının kilitlenme önleme için kullanılmasının başlıca dezavantajı nedir?

- A) Kaynak yetersizliğine neden olabilir
- B) Merkezi bir otorite gerektirir
- C) Kilitlenmeleri tespit edemez
- D) Hesaplama maliyeti yüksektir

Cevap

- Cevap: D

- Banker's algoritması, sistemdeki mevcut kaynak miktarını ve her bir sürecin maksimum kaynak talebini göz önünde bulundurarak kaynak tahsisini gerçekleştirir. Sistemdeki her bir sürecin herhangi bir anda talep edebileceği maksimum kaynak miktarını tahmin etmeyi gerektirir. Sistemdeki kaynakların büyük bir kısmının kullanımda olabileceği durumları hesaba katmak gerekir. Algoritmanın karmaşıklığı ve hesaplama maliyeti yüksektir. Sürekli hesaplama ve güncelleme işlemi, sistemdeki performansı düşürebilir ve kaynakların etkin kullanımını zorlaştırabilir.

Soru

- Aşağıdakilerden hangisi kilitlenmeleri (*deadlock*) ele almak için kullanılan yöntemlerden değildir?

- A) Önleme (*Deadlock prevention*)
- B) Tespit ve kurtarma (*Deadlock detection and recovery*)
- C) Kaçınma (*Deadlock avoidance*)
- D) Kabul (*Deadlock acceptance*)

Cevap

- Cevap: D

- İşletim sistemleri, kilitlenmelerle başa çıkmak için *önleme*, *tespit ve kurtarma*, ve *kaçınma* yöntemlerini kullanır. Kabul, bir başa çıkma yöntemi değildir.

Soru

- Hangi yaklaşım, sistemin kilitlenme durumuna girmesine izin verir ve ardından kurtulur?

- A) Kilitlenmeyi önleme
- B) Kilitlenme tespiti
- C) Kilitlenmeden kaçınma
- D) Kilitlenmeden kurtarma

Cevap

- Cevap: D

- Kilitlenme tespit edildiğinde, sistem belirli bir algoritma veya strateji kullanarak kilitlenmenin gerçekleştiğini belirler. Kilitlenmeyi çözmek için sistemdeki kaynakları serbest bırakacak uygun adımlar atılır. Örneğin, kilitlenen süreçlerin durdurulması, kaynakların serbest bırakılması veya kilitlenmeye neden olan koşulların değiştirilmesi gibi.

Soru

- Kilitlenme önleme durumunda, sistem *tut ve bekle (hold and wait)* koşulunu önlemek için ne yapar?

- A) Kaynak tahsisi, dairesel bekleme olasılığını ortadan kaldıracak şekilde yapılır
- B) Gerektiğinde kaynaklar önceden alınır
- C) Kaynakları sağlanamayan süreçler sonlandırılır
- D) Süreçlere istedikleri tüm kaynaklar birden sağlanır

Cevap

- Cevap: D

- *Tut ve bekle* koşulunda, bir süreç ihtiyaç duyduğu kaynakları beklerken başka bir süreç tarafından tutulan kaynaklara ihtiyaç duyar ve bu da bir döngüsel bekleyişe yol açar. Kilitlenmeyi (deadlock) önlemek için "tut ve bekle" (hold and wait) koşulunu ortadan kaldırmak gerekir. Bunun için, süreçlerin tüm ihtiyaç duydukları kaynakları başlamadan önce alması sağlanır. Böylece süreç, bir kaynağı alıp başka bir kaynak için bekleyerek kilitlenme durumuna düşmez.

Soru

- Bir sürecin tüm kaynakları tek seferde değil, aşamalı olarak talep ederek kilitlenme olasılığını azaltan hangi tekniktir?

- A) Kaynak tahsis grafiği algoritması
- B) Banker algoritması
- C) Öncelik mirası protokolü
- D) Aşamalı kaynak tahsisi

Cevap

- Cevap: D

- Aşamalı kaynak tahsisi, bir sürecin tüm gereksinimlerini tek seferde talep etmek yerine aşamalı olarak talep etmesine izin verir. Bu yaklaşım, sürecin sadece ihtiyaç duyduğu kaynakları belirli bir sırayla talep etmesine ve kullanmasına olanak tanır. Bu durum, bazı kaynakları diğer süreçlerin serbest bırakmasını beklerken, serbest kaynakları kullanabilmesini sağlar.

Soru

- Banker algoritmasında, güvenlik (*safety*) algoritmasının amacı nedir?

- A) Kilitlenmeleri tespit etmek
- B) Kaynakları en uygun şekilde tahsis etmek
- C) Kaynak tahsisinin güvenli bir sırası olduğundan emin olmak
- D) Kilitlenmelerden kurtulmak

Cevap

- Cevap: C

- *Güvenlik* (*safety*) algoritması, sistemdeki mevcut kaynak durumunu değerlendirerek, herhangi bir kaynak talebini karşılamak için güvenli bir sıra olup olmadığını kontrol eder. Güvenli sıra, tüm süreçlerin kaynak ihtiyaçlarının karşılanabileceği sıradır ve kilitlenme durumunun oluşmasını önler.

Soru

- Kaynak tahsis çizgesinde döngü (*cycle*) neyi temsil eder?

- A) Potansiyel bir kilitlenme durumunu
- B) Kaynak tahsis geçmişini
- C) Süreçlerin önceliğini
- D) Bir kaynağın örnek nesne (*instance*) sayısını

Cevap

- Cevap: A

- Kaynak tahsis çizgesinde döngü, döngü içindeki süreçlerin birbirlerinin kaynaklarını beklemelerine neden olur. Bu durum, her bir sürecin kaynağı beklerken diğer sürecin elindeki kaynağı serbest bırakmasını beklemesiyle sonuçlanabilir. Dolayısıyla, kaynak tahsis çizgesindeki döngüler potansiyel kilitlenme durumlarını gösterir.

Soru

- Kaynak tahsis çizgesinde döngüleri tespit için hangi teknik kullanılır?

- A) Derinlik öncelikli arama (*Depth-first search*)
- B) Genişlik öncelikli arama (*Breadth-first search*)
- C) Topolojik sıralama (*Topological sorting*)
- D) Dijkstra algoritması (*Dijkstra's algorithm*)

Cevap

- Cevap: A

- Kaynak tahsis çizgesindeki döngüleri tespit etmek için derinlik öncelikli arama (*DFS*) algoritması kullanılır. Bu algoritma, bir çizge üzerinde dolaşırken döngü varsa tespit eder. DFS, bir başlangıç noktasından başlayarak derinlemesine bir yol izler ve bu yol boyunca tüm olası yolları keşfeder. Bir döngü DFS sırasında zaten ziyaret edilen bir düğüme geri dönüş olduğunu gösterir.

Soru

- Kilitlenmeleri önlemek için kullanılmayan kaynak tahsis stratejisi aşağıdakilerden hangisidir?

- A) Kaynak Üstünlüğü (*Resource preemption*)
- B) İyimser Kaynak Tahsisi (*Optimistic resource allocation*)
- C) Bekle ve Öldür Şeması (*Wait-die scheme*)
- D) Vazgeç ve Bekle Şeması (*Wound-wait scheme*)

Cevap

- Cevap: B

- İyimser kaynak tahsisi yaklaşımı, kaynakların muhtemelen kullanılabilir olacağını varsayar. Süreçler serbestçe kaynak talep eder ve bir talep o anda karşılanamazsa, süreç bekler ve daha sonra yeniden dener. Bu iyimser yaklaşım, kilitlenmeleri aktif olarak önlemez. Süreçlerin kaynak talep etmeye ve beklemeye devam etmesi, nihayetinde döngüsel bir bekleme durumu ve kilitlenme oluşturması mümkündür.

Soru

- Kaynak tahsis çizgesinde, süreç düğümünden kaynak düğümüne yönlü kenar neyi temsil eder?

- A) Süreç, kaynağı tutuyor
- B) Süreç, kaynağı talep ediyor
- C) Kaynak, sürece tahsis edildi
- D) Kaynak tahsisi için kullanılabilir

Cevap

- Cevap: B

- Kenarlar, süreç düğümünden kaynak düğümüne yönlendirildiğinde, sürecin o kaynağı talep ettiğini gösterir.

Soru

- Kaynak tahsis çizgesi hakkında aşağıdakilerden hangisi doğrudur?

- A) Çizgede döngü olması kilitlenmeyi (*deadlock*) gösterir
- B) Döngüsüz çizgeler kaynak tahsisi senaryolarını gösteremez
- C) Çizgede sadece süreçler düğüm olarak temsil edilir
- D) Kaynak düğümleri her zaman süreç düğümlerine yönlü kenarlara sahiptir

Cevap

- Cevap: A

- Kaynak tahsis çizgesinde bir döngü, kilitlenme durumunu gösterir. Kilitlenme, birden fazla sürecin birbirlerinin tuttukları kaynakları beklemesi ve bu kaynakların serbest bırakılmaları gerektiği durumu ifade eder. Bir döngü, bir sürecin bir kaynağı tuttuğu ve diğer süreçlerin aynı döngüdeki kaynakları elde etmek için beklediği durumu simgeler.

Soru

- Kaynak tahsis çizgesinde, kaynak düğümünden süreç düğümüne yönlü kenar neyi temsil eder?

- A) Kaynak, tahsise hazır
- B) Kaynak, süreç tarafından serbest bırakılıyor
- C) Süreç, kaynağı elinde tutuyor
- D) Kaynak, süreç tarafından talep ediliyor

Cevap

- Cevap: C

- Kaynak tahsis çizgesinde (*resource allocation graph*), kaynak düğümünden (*resource node*) süreç düğümüne (*process node*) yönlendirilmiş bir kenar, o kaynağın ilgili süreç tarafından tutulduğunu (kullanıldığını) gösterir. Çizge, kaynağın hangi süreç tarafından kullanıldığını ve kaynakların nasıl dağıtıldığını görselleştirir.

Soru

- Kaynak tahsis çizgesinde, bekleme (*wait-for*) kenarları nasıl temsil edilir?

- A) Süreçlerden diğer süreçlere doğru yönlü kenarlar
- B) Kaynaklardan süreçlere doğru yönlü kenarlar
- C) Süreçlerden kaynaklara doğru yönlü kenarlar
- D) Süreçler arasında yönsüz kenarlar

Cevap

- Cevap: A

- Bekleme kenarları, bir süreç düğümünden başka bir süreç düğümüne yönlü kenarlardır. Süreç bir kaynağı beklerken diğer bir sürecin o kaynağı elinde tuttuğunu ifade eder.

Soru

- Kaynak tahsis çizgesinin temel amacı nedir?

- A) Fiziksel belleğin tahsisini görselleştirmek
- B) CPU kaynaklarının tahsisini temsil etmek
- C) Potansiyel kilitlenmeleri tespit ve analiz etmek
- D) G/Ç işlemlerinin zamanlamasını optimize etmek

Cevap

- Cevap: C

- Kaynak tahsis çizgesi, işletim sistemlerinde potansiyel kilitlenme durumlarını tespit ve analiz etmek için kullanılır. Süreçlerin kaynakları kullanma ve talep etme etkileşimlerini görselleştirmeye olanak tanır. Kaynakları kimin kullandığı, kimin beklediği ve hangi kaynakların serbest olduğu gibi bilgileri sağlayarak kilitlenme durumlarını belirlemede yardımcı olur.

Soru

- Bekleme (*wait-for*) çizge algoritması, kaynak talep ve serbest bırakılmalarını nasıl ele alır?

- A) Sistem durumunun anlık görüntüsünü alır
- B) Çizgeyi dinamik olarak günceller
- C) Süreçlerden açık bildirim gerektirir
- D) Kaynak taleplerini ve serbest bırakmaları görmezden gelir

Cevap

- Cevap: B

- Bekleme çizgesi algoritması, kaynak talepleri ve serbest bırakmalarını dinamik olarak günceller. Süreçlerin kaynakları talep ettiğinde veya serbest bıraktığında, çizgede ilgili değişiklikler yapılır. Bu güncellemeler, kilitlenme durumlarını tespit etmek ve çözmek için çizgedeki ilişkileri doğru şekilde yansıtmaya yardımcı olur.

Soru

- Birden fazla türde kaynak için kilitlenme tespit algoritmasında, kaynakların tahsis ve talep durumunu temsil etmek için hangi veri yapısı kullanılır?

- A) Bekleme çizgesi
- B) Kaynak tahsis çizgesi
- C) Öncelik kuyruğu
- D) İkili ağaç

Cevap

- Cevap: B

- Kaynak tahsis çizgesi, birden fazla türde kaynak için kilitlenme tespit algoritmalarında kullanılır. Sistemdeki süreçlerin hangi kaynakları talep ettiğini ve hangi kaynakları elinde tuttuğunu gösterir. Bekleme çizgesi (*Wait-for graph*), bir sürecin diğer bir süreci beklediği durumları gösterirken, kaynak tahsis çizgesi (*Resource allocation graph*), kaynakların tahsis ve talep durumlarını temsil eder.

Soru

- Birden fazla türde kaynak için Banker algoritmasında, *güvenli sıra* terimi neyi ifade eder?

- A) Güvenle sonlandırılabilecek süreç dizisi
- B) Güvenle tahsis edilebilecek kaynak talebi dizisi
- C) Kilitlenmeye neden olmadan güvenle yürütülebilecek süreç dizisi
- D) Kilitlenmeleri önleyebilecek kaynak serbest bırakma dizisi

Cevap

- Cevap: C

- *Güvenli sıra*, Banker algoritması içinde kilitlenme oluşturmadan güvenle işletilebilecek bir süreç dizisini ifade eder. Bu sıra, sistemdeki mevcut kaynak durumuna ve her bir sürecin talep ettiği ve elinde bulunduğu kaynaklara bağlı olarak belirlenir. Banker algoritması, sistemde bir güvenli sıra varsa, bu sıranın işletilmesini güvence altına alır ve kilitlenme olasılığını önler.

Soru

- Banker algoritmasında bir kaynak talebinin kabul edilmesi için hangi koşulların sağlanması gerekmektedir?

- A) Talep karşılandıktan sonra sistem güvenli bir durumda olmalıdır
- B) Talep eden süreç en yüksek önceliğe sahip olmalıdır
- C) Talep eden süreç en uzun süre bekleyen olmalıdır
- D) Talep eden süreç zaten en fazla kaynağa sahip olmalıdır

Cevap

- Cevap: A

- Banker algoritmasında bir kaynak talebinin kabul edilmesi için, talebin karşılandıktan sonra sistemin güvenli durumunun korunması gerekir. Talep edilen kaynağın tahsis edilmesi durumunda kilitlenme olasılığı olmamalıdır. Kilitlenme olasılığı, sistemdeki mevcut kaynak durumuna ve her bir sürecin talep ettiği ve elinde bulunduğu kaynaklara bağlı olarak belirlenir. Eğer talep edilen kaynak karşılandıktan sonra sistem güvenli bir durumdaysa, talep kabul edilir.

Soru

- Banker algoritmasında, *maksimum* matrisinin önemi nedir?

- A) Her sürecin talep edebileceği maksimum kaynakları temsil eder
- B) Sistemdeki toplam mevcut kaynakları temsil eder
- C) Her sürece o anda tahsis edilen kaynakları temsil eder
- D) O anda tahsis edilebilecek kaynakları temsil eder

Cevap

- Cevap: A

- *Maksimum* matrisi, Banker algoritmasında her bir sürecin talep edebileceği maksimum kaynak miktarlarını temsil eder. Sistemin güvenli bir şekilde işlemesi ve kilitlenmelerin önlenmesi için kullanılır. Her bir satır, bir süreci temsil eder ve her bir sütun, bir türdeki kaynağı temsil eder. Bir hücredeki değer, ilgili sürecin o kaynağı talep edebileceği maksimum miktarı belirtir.

Soru

- Banker algoritmasının *gereksinim* (*need*) matrisinin amacı nedir?

- A) Her sürece o anda tahsis edilen kaynakları temsil eder
- B) O anda tahsis edilebilecek kaynakları temsil eder
- C) Her sürecin maksimum kaynakları ile o anda tahsis edilen kaynaklar arasındaki farkı temsil eder
- D) Sistemdeki toplam mevcut kaynakları temsil eder

Cevap

- Cevap: C

- *Gereksinim* matrisi, Banker algoritmasında her bir sürecin maksimum kaynak talepleri ile o anda tahsis edilen kaynaklar arasındaki farkı temsil eder. Her bir sürecin ihtiyaç duyduğu kaynak miktarını gösterir. Her bir satır, bir süreci temsil eder ve her bir sütun, bir türdeki bir kaynağı temsil eder. Bir hücredeki değer, ilgili sürecin o kaynağı talep etme miktarıdır.

Soru

- Tek tip kaynaklar için kilitlenme tespit algoritmalarında kaynakların tahsis ve talep durumunu temsil etmek için hangi veri yapısı kullanılır?

- A) Bekleme (*wait-for*) çizgesi
- B) Kaynak tahsis çizgesi
- C) Öncelik kuyruğu
- D) İkili ağaç

Cevap

- Cevap: A

- Tek tip kaynaklar için kilitlenme tespit algoritmalarında, kaynakların tahsis ve talep durumunu temsil etmek için bekleme çizgesi kullanılır. Bir sürecin diğer bir süreci beklediği durumları gösterir ve bu şekilde kilitlenmeleri tespit etmek için kullanılır. Bir süreç bir kaynağı talep ettiğinde ve bu kaynak o anda başka bir süreç tarafından kullanılıyorsa, bekleme çizgesi bu ilişkiyi temsil eder.

Soru

- Eşzamanlılık kontrolü bağlamında, *iki aşamalı kilitlenme* protokolü neyi amaçlar?

- A) İletişim kilitlenmelerinin önlenmesi
- B) Canlı kilitlenme durumlarının ortadan kaldırılması
- C) Açlık durumunun önlenmesi
- D) Süreçlerin serileştirilmesinin sağlanması

Cevap

- Cevap: D

- İki aşamalı kilitlenme protokolü, eşzamanlılık kontrolü bağlamında süreçlerin serileştirilmesini sağlar. Süreçlerin kilit alma ve serbest bırakma aşamalarını iki parçaya böler. İlk aşamada, süreç tüm gerekli kilitleri alır; ikinci aşamada ise, süreç tüm kilitleri bırakır. Bu şekilde, herhangi bir süreç, diğer süreçlerle uygun bir şekilde sıralanarak, eşzamanlılık sırasında doğru sonuçların elde edilmesini sağlar.

Soru

- İki aşamalı kilitlenme protokolünü diğer kontrol mekanizmalarından ayıran özellik nedir?

- A) Süreçlerin tamamlanmadan önce kilitleri serbest bırakmasına izin verir
- B) Tüm kilitlerin aynı anda alınmasını gerektirir
- C) Kilitlerin iki ayrı aşamada alınmasına ve serbest bırakılmasına izin verir
- D) Süreçleri zaman damgalarına göre önceliklendirir

Cevap

- Cevap: C

- İki aşamalı kilitlenme protokolü, kilitlerin iki ayrı aşamada alınmasına ve serbest bırakılmasına izin verir. İlk aşamada tüm gerekli kilitler alınır ve süreç başlar, ikinci aşamada ise tüm kilitler serbest bırakılır ve süreç tamamlanır. Süreçlerin kilitlenme ve serbest bırakma işlemlerini net bir şekilde ayırılarak, süreçlerin uygun bir şekilde sıralanması ve eşzamanlılığı sağlanır.

Soru

- Canlı kilidi (*livelock*) ölümcül kilitlenmeden (*deadlock*) ayıran özellik nedir?

- A) Süreçlerin sürekli durumlarını değiştirmeleri ancak ilerleyememeleri
- B) Süreçlerin kaynaklar için belirsiz süre beklemeleri
- C) Süreçlerin çakışan kaynak talepleri nedeniyle ilerleyememeleri
- D) Süreçler arasında döngüsel bir bağımlılık zincirinden kaynaklanması

Cevap

- Cevap: A

- *Livelock* (canlı kilitlenme) ile kilitlenme (*deadlock*) arasındaki temel fark, davranış açısından ortaya çıkar. Livelock durumunda, süreçler sürekli olarak durumlarını değiştirirler ancak ilerleme kaydetmezler. Süreçler birbirlerine yanıt verirler ve kaynakları serbest bırakmaya çalışırlar, ancak hiçbir ilerleme sağlamazlar.

Soru

- Canlı kilitlenme durumlarını azaltmak için bir strateji nedir?

- A) Sistemin zaman aşımı eşiklerini (*timeout thresholds*) artırmak
- B) Çizelgelemeye (*scheduling*) rastgelelik (*randomness*) katmak
- C) Kaynak tahsisi için geri çekilme (*backoff*) mekanizmaları kullanmak
- D) Etkilenen süreçleri yeniden başlatmak

Cevap

- Cevap: C

- Canlı kilitlenme durumlarını azaltmak için, kaynak tahsisinde geri çekilme mekanizmaları kullanılır. Bir sürecin kaynağı alamadığı durumda rastgele bir süre beklemesini ve ardından tekrar denemesini sağlar. Bu, süreçlerin kaynakları elde etme girişimlerini sıraya koymasını ve tekrarlanan kaynak taleplerinin aynı anda gerçekleşmesini önler.

Soru

- Kaynak tahsisi bağlamında, *açlık* (*starvation*) terimi neyi ifade eder?

- A) Süreçlerin çakışan kaynak talepleri nedeniyle ilerleyememeleri
- B) Süreçlerin sürekli durum değiştirmeleri ancak ilerleyememeleri
- C) Süreçlerin kaynaklar için belirsiz süre beklemesi
- D) Süreçlerin ihtiyaç duydukları kaynaklara erişiminin engellenmesi

Cevap

- Cevap: C

- *Açlık* terimi, kaynak tahsisi bağlamında süreçlerin kaynaklar için belirsiz bir süre beklediği durumdur. Bir süreç, gerekli kaynaklara erişememekten dolayı sürekli olarak bekler. Süreç kaynakları alamadığı için çalışmaya devam edemez. Açlık durumu adil olmayan kaynak tahsis politikaları veya önceliklendirme eksikliği gibi nedenlerden kaynaklanır.

Soru

- Sistemde *açlık* durumunun sonucu nedir?

- A) Sık sık kilitlenmelerin meydana gelmesi
- B) Süreçlerin aşırı CPU döngüleri tüketmesi
- C) Azalan sistem verimliliği
- D) Canlı kilitlenme durumlarının artması

Cevap

- Cevap: C

- Açlık durumu, süreçlerin bekleyerek kaynaklara erişemediği durumdur. Bazı süreçler belirli kaynakları elde edemeyecekleri için kaynakları kullanamaz ve bu da sistem verimliliğini azaltır. Özellikle kritik kaynaklara erişim gerektiren süreçler açlık durumu yaşarsa, sistemde verimlilik düşüşü yaşanır.

Soru

- Çizelgeleme algoritmalarında *açlığı* (*starvation*) ele almak için hangi mekanizma kullanılır?

- A) Yaşlandırma (*Aging*)
- B) İş kesme üstünlüğü (*Preemption*)
- C) Round robin
- D) Öncelik tersine çevirme (*Priority inversion*)

Cevap

- Cevap: A

- *Açlık* durumunu ele almak için çizelgeleme algoritmalarında *yaşlandırma* (*aging*) uygulanır. Süreçlerin uzun süre beklemesi durumunda önceliklerinin artırılmasını sağlar. Uzun süre bekleyen süreçler daha yüksek önceliğe sahip olurlar ve sistem tarafından öncelikli olarak işlenirler. Bu durum, süreçlerin açlık durumundan kurtulmasını ve sistemde adil bir kaynak dağıtımı sağlanmasını amaçlar.

Soru

- Dağıtık bir sistemde *canlı kilitlenmeye* yol açacak senaryo hangisidir?

- A) İki sürecin sürekli mesajlarına yanıt vermesi ancak ilerleyememeleri
- B) Bir sürecin uzaktaki bir sunucudan süresiz yanıt beklemesi
- C) İki sürecin birbirlerinin elindeki kaynakları beklemesi
- D) Bir sürecin, başka süreç tarafından tutulan kilit için süresiz beklemesi

Cevap

- Cevap: A

- Dağıtık bir sistemde canlı kilitlenme durumuna yol açacak senaryo, iki sürecin birbirlerinin mesajlarına sürekli yanıt vermesi ancak ilerleme kaydetmemeleridir. Sonuç olarak, süreçler birbirlerinin mesajlarına yanıt vermeye devam eder ancak herhangi bir ilerleme kaydedilmez.

SON