BIL 301 İşletim Sistemleri
2025 Güz Dönemi
Vize Sınavı

- **Fotoğraflı üniversite kimliği** olmayan öğrenciler sınava giremez.
- Her türlü kağıt, kitap, not, hesap makinesi, telefon, saat vb. elektronik aygıt kullanımı **kopya** sayılır.
- **TOPLAM SÜRE 90 DAKİKADIR.**
- Toplam 100 puan üzerinden değerlendirme yapılacaktır.
- Cevaplarınızı okunaklı bir şekilde ayrılan kısımlara yazınız.

İsim ve Soyisim : _____

Öğrenci No : _____

İmza : _____

Question	Points	Score
1	4	
2	4	
3	4	
4	4	
5	4	
6	6	
7	10	
8	10	
9	5	
10	8	
11	6	
12	15	
13	5	
14	8	
15	10	
16	5	
17	6	
Total:	114	

1. 4 puan **(İngilizce)** *Which mechanism allows user programs to request privileged operations from the kernel?*
(Türkçe) Kullanıcı programlarının ayrıcalıklı işlemleri kernel'dan talep etmek için kullandığı mekanizma hangisidir?

A. Hardware interrupt B. API calls **C. System calls** D. Shared memory

Solution: System calls (Sistem çağrıları) kullanıcı programlarının kernel tarafından sağlanan ayrıcalıklı işlemlere erişmesini sağlayan mekanizmadır.

2. 4 puan **(İngilizce)** *When a running process is preempted due to a timer interrupt, what state does it enter?*
(Türkçe) Çalışan bir process timer interrupt nedeniyle kesintiye uğradığında hangi duruma geçer?

A. New B. Running C. Waiting **D. Ready** E. Terminated

Solution: Timer interrupt ile kesintiye uğrayan process "Ready" (Hazır) durumuna geçer ve tekrar CPU'ya atanmayı bekler.

3. 4 puan **(İngilizce)** *In a system with 4 CPU cores (single-threaded each), what is the maximum number of processes that can be in the "running" state simultaneously?*
(Türkçe) 4 CPU çekirdekli (her biri tek thread'li) bir sistemde aynı anda "running" durumunda olabilecek maksimum process sayısı nedir?

A. 1 B. 2 **C. 4** D. 8 E. 16 F. Unlimited

Solution: Her CPU çekirdeği aynı anda sadece 1 process çalıştırabilir. 4 çekirdek varsa maksimum 4 process "running" durumunda olabilir.

4. 4 puan **(İngilizce)** *Thread implementation in user space (M:1 or M:N threading) is faster because:*
(Türkçe) Thread'lerin kullanıcı uzayında gerçekleştirilmesi (M:1 veya M:N threading) neden daha hızlıdır?

4. **System calls and context switches are avoided for thread operations**

Solution: Kullanıcı uzayı thread'leri, thread operasyonları için sistem çağrısı ve context switch gerektirmez, bu da daha hızlı performans sağlar.

5. 4 puan **(İngilizce)** *One limitation of implementing threads entirely in user space is:*
(Türkçe) Thread'lerin tamamen kullanıcı uzayında gerçekleştirilmesinin bir kısıtlaması nedir?

5. **A blocking system call blocks all threads in the process**

Solution: Kullanıcı uzayı thread'lerinde, bir thread'in yaptığı blocking sistem çağrısı tüm process'teki diğer thread'leri de bloke eder.

6. 6 puan **(İngilizce)** *Provide one example each for "policy" and "mechanism" in OS design:*
(Türkçe) İşletim sistemi tasarımında "policy" (politika) ve "mechanism" (mekanizma) için birer örnek veriniz:
   (a) **Policy**:
      (a) **CPU timer allocation for specific process groups**
   (b) **Mechanism**:
      (b) **Creating a hardware timer for CPU protection**

> **Solution:** Policy: Ne yapılacağına karar verir (örn: CPU zamanının process grupları arasında nasıl dağıtılacağı)
> Mechanism: Nasıl yapılacağını sağlar (örn: CPU koruması için donanım timer'ı oluşturma)

7. 10 puan **(İngilizce)** *Match the process activities with their states:*
(Türkçe) Process aktivitelerini durumlarıyla eşleştiriniz:
   (a) Process waiting for I/O completion (processing I/O bitmesini beklemesi)
   → **Waiting**
   (b) Process just created with PCB allocated (Processin henüz PCB ayrılarak oluşturulması)
   → **New**
   (c) Process executing on CPU (CPUda çalışan mevcut process)
   → **Running**
   (d) Process finished but exit status not read (Bitmiş process ancak henüz exit durumu okunmamış)
   → **Terminated (Zombie)**
   (e) Process ready to run but waiting for CPU (CPU da çalışmak için hazırkıta bekleyen process)
   → **Ready**

> **Solution:** - I/O completion bekleyen: Waiting
> - Yeni oluşturulan: New
> - CPU'da çalışan: Running
> - Bitmiş ama exit status okunmamış: Terminated (Zombie)
> - Çalışmaya hazır ama CPU bekleyen: Ready

8. 10 puan **(İngilizce)** *Why would you choose a microkernel design for a military system requiring high security? Explain.*
(Türkçe) Yüksek güvenlik gerektiren askeri bir sistem için neden microkernel tasarımını seçersiniz? Açıklayınız.

> **Solution:** Microkernel minimizes trusted computing base, isolates components, and provides better security through process separation and minimal kernel privileges.
> Microkernel tasarımı, güvenilir bilgi işlem tabanını (TCB) minimize eder, bileşenleri izole eder ve process ayrımı ile minimal kernel ayrıcalıkları sayesinde daha iyi güvenlik sağlar. Kernel'da çalışan kod miktarı az olduğundan saldırı yüzeyi küçülür ve hata olasılığı düşer.

9. [5 puan] **(İngilizce)** *What is the difference between a zombie process and an orphan process? How does the OS handle each?*
(Türkçe) Zombie process ile orphan process arasındaki fark nedir? İşletim sistemi her birini nasıl yönetir?

9. **Zombie: Terminated but parent hasn't read exit status; Orphan: Parent terminated befor**

> **Solution:** Zombie process: Sonlanmış ama parent'ı exit durumunu okumamış process. OS PCB'yi tutar.
> Orphan process: Parent'ı kendisinden önce sonlanmış process. OS bu process'i init process'in altına alır.

10. **(İngilizce)** *Compare user-level threads vs kernel-level threads in terms of:*
(Türkçe) Kullanıcı seviyesi thread'ler ile kernel seviyesi thread'leri aşağıdaki açılardan karşılaştırınız:
   (a) [2 puan] **(İngilizce)** *Performance* **(Türkçe)** Performans

   (a) **User-level: Faster context switching; Kernel-level: Slower due to system calls**
   (b) [2 puan] **(İngilizce)** *Blocking behavior* **(Türkçe)** Bloklama davranışı

   (b) **User-level: One blocked thread blocks all; Kernel-level: Only blocked thread affected**
   (c) [2 puan] **(İngilizce)** *Multiprocessor utilization* **(Türkçe)** Çoklu işlemci kullanımı

   (c) **User-level: Cannot use multiple CPUs; Kernel-level: Can utilize multiple CPUs**
   (d) [2 puan] **(İngilizce)** *Implementation complexity* **(Türkçe)** Gerçekleştirim karmaşıklığı

   (d) **User-level: Complex to implement; Kernel-level: Simpler, handled by OS**

> **Solution:** - Performans: Kullanıcı seviyesi thread'ler daha hızlı context switch yapar
> - Bloklama: Kullanıcı seviyesinde bir thread bloke olursa tüm process bloke olur
> - Çoklu işlemci: Kernel thread'leri farklı CPU'lara dağıtılabilir
> - Karmaşıklık: Kullanıcı seviyesi thread'ler daha karmaşık implementasyon gerektirir

11. **(İngilizce)** *Calculate the maximum speedup using Amdahl's Law for an application that is 85% parallelizable running on:*
(Türkçe) %85'i paralelleştirilebilen bir uygulama için Amdahl Yasası'nı kullanarak maksimum hızlanmayı hesaplayınız:
   (a) [2 puan] 4 cores: **2.91**
   (b) [2 puan] 8 cores: **3.77**
   (c) [2 puan] 16 cores: **4.35**

Solution: Amdahl's Law: Speedup = 1 / (S + P/N)
S = 0.15 (serial), P = 0.85 (parallel)
4 cores: 1 / (0.15 + 0.85/4) = 1 / (0.15 + 0.2125) = 1/0.3625 = 2.91
8 cores: 1 / (0.15 + 0.85/8) = 1 / (0.15 + 0.10625) = 1/0.25625 = 3.77
16 cores: 1 / (0.15 + 0.85/16) = 1 / (0.15 + 0.053125) = 1/0.203125 = 4.35

12. **(İngilizce)** *Given processes with arrival times and burst times:*
(Türkçe) Aşağıda processlerin varış zamanları ve çalışma süreleri verilmiştir:

Process	Arrival Time	Burst Time	Priority (1=highest)
P1	0	8	3
P2	1	4	1
P3	2	6	2
P4	3	3	4
P5	4	2	1

(a) 5 puan **(İngilizce)** *FCFS scheduling - Draw Gantt chart and compute average waiting time*
(Türkçe) FCFS çizelgeleme - Gantt grafiği çiziniz ve ortalama bekleme süresini hesaplayınız

AWT: **9.8**

(b) 5 puan **(İngilizce)** *SJF (non-preemptive) scheduling*
(Türkçe) SJF (kesintisiz) çizelgeleme

AWT: **7.6**

(c) 5 puan **(İngilizce)** *Round Robin (q=4) scheduling*
(Türkçe) Round Robin (q=4) çizelgeleme

AWT: **10.2**

Solution:

Process Scheduling Analysis(edit after taking AI

Given processes with the following characteristics:

Process	Arrival Time	Burst Time	Priority (1=highest)
P1	0	8	3
P2	1	4	1
P3	2	6	2
P4	3	3	4
P5	4	2	1

1. FCFS (First-Come-First-Served) - Without Priorities

Execution Order: P1 → P2 → P3 → P4 → P5

Gantt Chart: P1[0-8], P2[8-12], P3[12-18], P4[18-21], P5[21-23]

$$\text{Waiting Time}_{P1} = 8 - 0 - 8 = 0$$
$$\text{Waiting Time}_{P2} = 12 - 1 - 4 = 7$$
$$\text{Waiting Time}_{P3} = 18 - 2 - 6 = 10$$
$$\text{Waiting Time}_{P4} = 21 - 3 - 3 = 15$$
$$\text{Waiting Time}_{P5} = 23 - 4 - 2 = 17$$
$$\text{Average Waiting Time} = \frac{0 + 7 + 10 + 15 + 17}{5} = \frac{49}{5} = 9.8$$

2. SJF (Shortest Job First) - Without Priorities

Execution Order: P1 → P5 → P4 → P2 → P3

Gantt Chart: P1[0-8], P5[8-10], P4[10-13], P2[13-17], P3[17-23]

$$\text{Waiting Time}_{P1} = 8 - 0 - 8 = 0$$
$$\text{Waiting Time}_{P2} = 17 - 1 - 4 = 12$$
$$\text{Waiting Time}_{P3} = 23 - 2 - 6 = 15$$
$$\text{Waiting Time}_{P4} = 13 - 3 - 3 = 7$$
$$\text{Waiting Time}_{P5} = 10 - 4 - 2 = 4$$
$$\text{Average Waiting Time} = \frac{0 + 12 + 15 + 7 + 4}{5} = \frac{38}{5} = 7.6$$

3. Round Robin (Time Quantum = 2) - Without Priorities

Execution: P1→P2→P3→P4→P5→P1→P3

Gantt Chart: P1[0-4], P2[4-8], P3[8-12], P4[12-15], P5[15-17], P1[17-21], P3[21-23]

$$\text{Finish Time}P1 = 21$$
$$\text{Finish Time}P2 = 8$$
$$\text{Finish Time}P3 = 23$$
$$\text{Finish Time}P4 = 15$$
$$\text{Finish Time}P5 = 17$$
$$\text{Waiting Time}P1 = 21 - 0 - 8 = 13$$
$$\text{Waiting Time}P2 = 8 - 1 - 4 = 3$$
$$\text{Waiting Time}P3 = 23 - 2 - 6 = 15$$
$$\text{Waiting Time}P4 = 15 - 3 - 3 = 9$$
$$\text{Waiting Time}P5 = 17 - 4 - 2 = 11$$
$$\text{Average Waiting Time} = \frac{13 + 3 + 15 + 9 + 11}{5} = \frac{51}{5} = 10.2$$

4. FCFS Priority Scheduling (1=highest) - With Priorities

Execution Order: P1 → P2 → P5 → P4 → P3
Gantt Chart: P1[0-8], P2[8-12], P5[12-14], P3[14-20], P4[20-23]

$$\text{Waiting Time}_{P1} = 8 - 0 - 8 = 0$$
$$\text{Waiting Time}_{P2} = 12 - 1 - 4 = 7$$
$$\text{Waiting Time}_{P3} = 20 - 2 - 6 = 12$$
$$\text{Waiting Time}_{P4} = 23 - 3 - 3 = 17$$
$$\text{Waiting Time}_{P5} = 14 - 4 - 2 = 8$$
$$\text{Average Waiting Time} = \frac{0 + 7 + 12 + 17 + 8}{5} = \frac{44}{5} = 8.8$$

5. SJF with Priorities

Execution Order: P1 → P5 → P2 → P3 → P4
Gantt Chart: P1[0-8], P5[8-10], P2[10-14], P3[14-20], P4[20-23]

$$\text{Waiting Time}_{P1} = 8 - 0 - 8 = 0$$
$$\text{Waiting Time}_{P2} = 14 - 1 - 4 = 9$$
$$\text{Waiting Time}_{P3} = 20 - 2 - 6 = 12$$
$$\text{Waiting Time}_{P4} = 23 - 3 - 3 = 17$$
$$\text{Waiting Time}_{P5} = 10 - 4 - 2 = 4$$
$$\text{Average Waiting Time} = \frac{0 + 9 + 12 + 17 + 4}{5} = \frac{42}{5} = 8.4$$

5. RR with Priorities

Execution Order: P1 → P2 → P5 → P3 → P3(2unit)→ P1(4unit) → P4
AWT **left as an exercise**
Summary of Results

Scheduling Algorithm	Average Waiting Time
FCFS (No Priority)	9.8
SJF (No Priority)	7.6
Round Robin (Q=4)	10.2
Priority Scheduling	7.4
SJF with Priority	8.4

13. [5 puan] **(İngilizce)** *In Round Robin scheduling, if the time quantum (q) is too large, the algorithm degenerates to* ___***FCFS***___*, and if q is too small, it increases the number of* ***context switches****.*
(Türkçe) Round Robin çizelgelemede, zaman kuantumu (q) çok büyük olursa algoritma ___**FCFS**___ algoritmasına indirgenir, ve q çok küçük olursa **context switch** sayısını artırır.

> **Solution:** Büyük q değeri FCFS gibi davranırken, küçük q değeri context switch overhead'ini artırır.

14. [8 puan] **(İngilizce)** *Given the exponential averaging formula for SJF:*

$$\tau_{n+1} = \alpha \times actual_n + (1 - \alpha) \times \tau_n$$

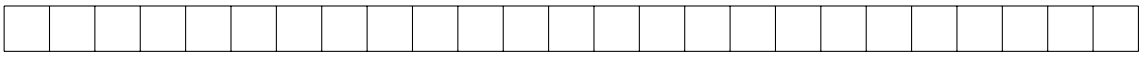
(Türkçe) SJF için üstel ortalama formülü verilmiştir: **(İngilizce)** *If* $\alpha = 0.5, \tau_0 = 10$*, and the actual burst times are: 6, 4, 6, 8, calculate* $\tau_1, \tau_2, \tau_3, \tau_4$
(Türkçe) $\alpha = 0.5, \tau_0 = 10$,ve gerçek çalışma süreleri: 6, 4, 6, 8 ise $\tau_1, \tau_2, \tau_3, \tau_4$ değerlerini hesaplayınız
τ_1: ___**8**___ τ_2: ___**6**___ τ_3: ___**6**___ τ_4: ___**7**___

> **Solution:** $\tau_1 = 0.5 \times 6 + 0.5 \times 10 = 3 + 5 = 8$
> $\tau_2 = 0.5 \times 4 + 0.5 \times 8 = 2 + 4 = 6$
> $\tau_3 = 0.5 \times 6 + 0.5 \times 6 = 3 + 3 = 6$
> $\tau_4 = 0.5 \times 8 + 0.5 \times 6 = 4 + 3 = 7$

15. **(İngilizce)** *Real-time Scheduling: Given periodic tasks, draw Gantt charts for:*
(Türkçe) Gerçek Zamanlı Çizelgeleme: Periyodik task'lar verilmiştir, Gantt grafiklerini çiziniz:

Task	Period	Execution	Deadline
T1	10	3	10
T2	6	2	6
T3	15	4	15

(a) [5 puan] **(İngilizce)** *Rate Monotonic Scheduling* **(Türkçe)** Rate Monotonic Çizelgeleme



(b) [5 puan] **(İngilizce)** *Earliest Deadline First* **(Türkçe)** En Erken Bitiş Tarihi Önce

Solution: Rate Monotonic: Priority order $T2 > T1 > T3$ (shorter period = higher priority)
Earliest Deadline First: Dynamic priorities based on current deadlines
Detailed Gantt charts would show T2 having highest priority in RMS, while EDF dynamically adjusts based on which task has earliest deadline at each scheduling point

Bonus Sorular

16. [5 puan] **(İngilizce)** *Explain how copy_to_user() and copy_from_user() prevent security issues when transferring data between kernel and user space.*
(Türkçe) copy_to_user() ve copy_from_user() fonksiyonlarının kernel ve kullanıcı uzayı arasında veri transferi yaparken güvenlik sorunlarını nasıl önlediğini **maddeler halinde** açıklayınız (iki tanesi tam puan almaya yetiyor).

Solution: Bu fonksiyonlar: 1. Pointer geçerlilik kontrolü yapar - kullanıcı alanına işaret etmeyen pointer'ları reddeder 2. Bellek erişim haklarını kontrol eder - salt okunur belleğe yazma denemelerini engeller 3. Sayfa hatalarını güvenli şekilde handle eder 4. Buffer overflow saldırılarını önlemek için boyut kontrolü yapar 5. Kernel'ın kullanıcı tarafından sağlanan verileri güvenilmez kaynak olarak işlemesini sağlar Bu önlemler kernel'ın kötü niyetli veya hatalı kullanıcı programları tarafından compromise edilmesini önler.

17. [6 puan] **(İngilizce)** *Complete the Linux kernel compilation commands:*
(Türkçe) Linux kernel derleme komutlarını tamamlayınız:
   (a) ___**make**___ Kernel source dosyasındaki makefile içerisindeki komutları çalıştırarak kernel'ı compile ve build eder
   (b) **make menuconfig** Linux kaynak kodunu compile etmeden önce gerekli konfigurasyonları yapmamızı sağlar
   (c) **make install** Build edilen kernel'ı mevcut sisteme yüklememizi sağlar

Solution: - make: Derleme ve build işlemini başlatır - make menuconfig: Kernel konfigurasyon menüsünü açar - make install: Derlenmiş kernel'ı sisteme yükler