Final 1. soru (15p) (process sync)

Writer Process

```
wait(wrt);
        …
        writing is performed
        …
signal(mutex);
```

Reader Process

```
wait(mutex);
        readcount = readcount +1;
        if readcount = 1 then wait(wrt);
signal(mutex);
        …
        reading is performed
        …
wait(mutex);
        readcount = readcount - 1;
        if readcount = 0 then signal(wrt);
signal(mutex);
```

Considering the readers and writers problem, assume that rules have changed. The rule "Multiple readers can read at the same time" is now changed to "Up to 5 readers can read at the same time".
Above is the pseudocode for the original readers-writers problem. Rewrite the code to meet the conditions for the new rule. Rules: 1. Only one writer writes at a time 2. While writing reading is not allowed 3. While reading writing is not allowed 4. (NEW) Up to 5 readers can read at the same time

Semaphore (Semafor): Semaphore, işlemler arasında paylaşılan kaynakların kullanımını koordine etmek için kullanılan bir araçtır. Semaforlar, işlemlerin birbirlerinin çalışmasını engelleyebilir veya izin verebilir. Örneğin, bir yazar işleminin paylaşılan bir dosyayı yazarken diğer işlemlerin dosyaya erişimini engeller.
Critical section (Kritik Bölge): Paylaşılan bir kaynağın bir işlem tarafından etkilendiği bölümdür. Örneğin, bir yazar işleminin paylaşılan bir dosyayı yazdığı bölüm bir kritik bölgedir. Kritik bölge, sadece bir işlem tarafından kullanılabilir ve diğer işlemler tarafından etkilenmemesi için korunur.
Interprocess access restrictions (İşlemler Arası Erişim Kısıtlamaları): İşlemler arasında paylaşılan kaynakların kullanımını koordine etmek için kullanılan mekanizmalardır. Örneğin, işlemler arası erişim kısıtlamaları bir yazar işleminin paylaşılan kaynağı yazarken diğer işlemlerin kaynağa erişimini engeller.
Wait/signal (Bekle/İşaret): Semaforların kullanımını koordine etmek için kullanılan bir araçtır. Örneğin, bir işlemin semafora bekleme işlemi gerçekleştirilirken diğer işlemlerin kaynağa erişim izni alamayacaktır. Bekleme işlemi sona erdikten sonra ise diğer işlemlerin kaynağa erişim izni alabilecektir.

Kod, bir yazar işleminin paylaşılan kaynağı yazarken diğer işlemlerin kaynağa erişimini engeller ve aynı zamanda, birden fazla okuyucu işleminin aynı anda paylaşılan kaynağı okumasına izin verir. Bu amaçla, "wrt" ve "mutex" semaforları kullanılmaktadır.

cevap:
Writer Process

```
# Bekleme sırasında yazar işlemi "wrt" semaforuna bekler
wait(wrt);
        # Yazma işlemi gerçekleştirilir
        …
writing is performed
        # Semafor serbest bırakılır
        …
signal(mutex);
```

Reader Process

```
# Bekleme sırasında okuyucu işlemi "mutex" semaforuna bekler
wait(mutex);
        # Okuyucu sayısı 1 arttırılır
        readcount = readcount +1;
# Eğer okuyucu sayısı 5'ten fazlaysa, yazma işlemine izin vermemek için "wrt" semaforuna bekler
if readcount > 5 then wait(wrt);
# Semafor serbest bırakılır
signal(mutex);
        # Okuma işlemi gerçekleştirilir
        …
reading is performed
        # Semafor tekrar beklenir
        …
wait(mutex);
        # Okuyucu sayısı 1 azaltılır
        readcount = readcount - 1;
# Eğer okuyucu sayısı 0'a eşitse, yazma işlemine izin vermek için "wrt" semaforu serbest bırakılır
if readcount = 0 then signal(wrt);
# Semafor serbest bırakılır
signal(mutex);
```

Yazar işleminin çalışması sırasında, "wrt" semaforuna bekler ve paylaşılan kaynağı yazar. Daha sonra "mutex" semaforunu serbest bırakır. Okuyucu işlemleri ise, "mutex" semaforuna bekler ve okuyucu sayısını 1 arttırır. Eğer okuyucu sayısı 5'ten fazlaysa, yazma işlemine izin vermemek için "wrt" semaforuna bekler. Daha sonra "mutex" semaforunu serbest bırakır ve paylaşılan kaynağı okur. Okuma işlemi bittikten sonra, "mutex" semaforuna tekrar bekler ve okuyucu sayısını 1 azaltır. Eğer okuyucu sayısı 0'a eşitse, yazma işlemine izin vermek için "wrt" semaforunu serbest bırakır ve "mutex" semaforunu da serbest bırakır. Bu şekilde, modifiye edilmiş kod ile bir yazar işleminin paylaşılan kaynağı yazarken diğer işlemlerin kaynağa erişimini engellenir ve aynı zamanda birden fazla okuyucu işleminin aynı anda paylaşılan kaynağı okumasına izin verilir.

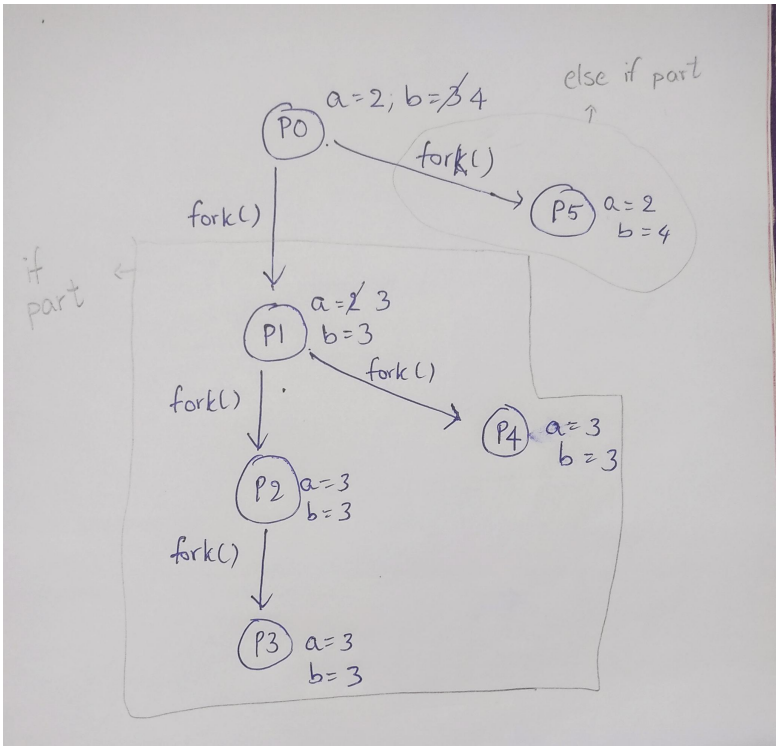
Final 2.soru (10p) (fork)

```
int main()
{
 pid_t smith;
 int a = 2;
 int b = 3;

 smith = fork();

 if (smith == 0)
 {
   fork();
   a++;
   fork();
   // BEWARE
 }
 else if (smith > 0)
 {
   b++;
   fork();
   // BEWARE
 }

 // Print the values of the variables a and b
 printf("%d %d", a, b);
}
```



Here is the code for a program named Agent_Smith.c. Including the initial parent process,
A) How many Agent_Smith processes are created? Assume there are no errors.
B) Draw the process tree showing the up-to-date variables a and b.

a- A total of 6 Agent_Smith processes exist. Creation of 5 processes starts at one initial root process. (So 6 processes exist in total

Final 3.soru (10p) (process sync)

if the semaphore operations Wait and Signal are not executed atomically, then mutual exclusion may be violated. Assume that Wait and Signal are implemented as below:
LX represents the line numbers

```
L1 void Wait (Semaphore S) {
L2  while(S.count <= 0) {:}
L3 S.count = S.count - 1;
L4 }
L5 void Signal (Semaphore S) {
L6 S.count = S.count +1;
L7 }
```

Describe a scenario of context switches (CS) where two threads, T1 and T2, can both enter a critical section guarded by a single mutex semaphore a result of a lack of atomicity. In your scenario clearly describe the initial value of the semaphore S, and sequence of operations using the line numbers Lx.
For example: S=5;T1-L1:L3,CS,T2-L5,CS,T1-L4,CS,T2-L6:L7 means that Semaphore is initialized with value of 5. Then T1 executes lines from 1 to 3. A context switch occurs, then T2 executes line 5. Another context switch occurs.

Cevap 3:
Consider the following execution of two threads T1 and T2:

- **Initially, the semaphore S is initialized to a value of 1.**
- **T1 starts and calls the Wait(S) function (L1). It then executes L2 and finds that S.count > 0.**
- **Next, a context switch (CS) occurs, and control is transferred to thread T2.**
- **Thread T2 calls the Wait(S) function (L1). It then executes L2 and finds that S.count > 0.**
- **Next, T2 executes lines L3 and L4 and sets S.count = 0. *T2 then enters the critical section.***
- **At this point, again a context switch (CS) occurs, and control is transferred back to T1.**
- **T1 executes the lines L3 and L4, and sets S.count = 0 - 1 = -1. *T1 then enters the critical section.***

Thus, both T1 and T2 enter the critical section in the following scenario:

S = 1;T1-L1:L2,CS,T2-L1:L4,CS,T1:L3-L4

Sorunuz, semafor işlemlerinin atomik olmaması durumunda, iki iş parçacığının aynı anda kritik bölüme girebileceğini gösteren bir örnek istiyor. Örnekte, semaforun başlangıç değeri 1 olarak verilmiş ve T1 ve T2'nin yaptığı işlemler, context switch'ler dahil, belirtilmiştir.

Semafor işlemleri, Wait ve Signal olmak üzere iki tane işlemden oluşur. Wait işlemi, semaforun değerini azaltır ve kritik bölüme girişi sağlar. Signal işlemi ise, semaforun değerini arttırır ve kritik bölümden çıkışı sağlar. Bu iki işlem atomik olmalıdır, yani bir iş parçacığı veya işlem bu işlemleri yürütürken, bir context switch olmamalıdır. Eğer context switch olursa, semaforun değeri doğru bir şekilde güncellenemez ve mutüel esklüzyon ihlal edilebilir.

Başlangıçta, semafor S'nin değeri 1 olarak ayarlanır.
T1 başlar ve Wait(S) fonksiyonunu (L1) çağırır. Daha sonra L2'yi çalıştırır ve S.count > 0 olduğunu bulur.
Sonra, bir koşullu işlem (CS) gerçekleşir ve kontrol T2 iş parçacığına verilir.
T2 de Wait(S) fonksiyonunu (L1) çağırır. Daha sonra L2'yi çalıştırır ve S.count > 0 olduğunu bulur. T2 sonra L3 ve L4'ü çalıştırır ve S.count = 0 yapar. T2 daha sonra kritik bölgeye girer.
Bu noktada, yine bir koşullu işlem (CS) gerçekleşir ve kontrol T1'e verilir.
T1 L3 ve L4'ü çalıştırır ve S.count = 0 - 1 = -1 yapar. T1 daha sonra kritik bölgeye girer.. Bu nedenle, T1 ve T2 aynı anda kritik bölüme girmiş ve mutüel esklüzyon ihlal edilmiştir.

S'nin başlangıç değeri 5 olsun ve aşağıdaki senaryo gerçekleşsin:
S = 5;T1-L1:L2,CS,T2-L1:L4,CS,T1-L3:L4,CS,T2-L3:L4
Bu senaryoda, T1 ve T2 iş parçacıkları arasında koşullu işlemler gerçekleşir ve T1 ve T2 L3-L4 arasındaki kod bloğunu çalıştırır. Bu noktada, iki iş parçacığı da kritik bölgeye girer.
? emin değilim: S'nin başlangıç değeri ne olursa olsun, T1 ve T2 iş parçacıkları arasında koşullu işlemler gerçekleştiğinde müşterek dışlama ihlali oluşabilir. Bu nedenle, semafor operasyonları Wait ve Signal atomik olarak çalıştırılması gerekir. Atomik olarak çalıştırılması, semafor operasyonlarının birbirlerine bağlı olarak çalıştırılması anlamına gelir. Bu sayede, müşterek dışlama ihlali önlenebilir ve kritik bölgeye girdiğinizden emin olunabilir.

Final 4.soru (15p) (concurrent execution)

Select the correct option for the outputs of the concurrent execution of process A and process B to be (15 Puan)

Initialization int x=2; int y=3;

```
Process A
while (x==2) {do-nothing};
printf("S");
y=x-y;
printf("3");
y=1;
```

```
Process B
printf("C");
x=y*x
printf("E");
while (y==3) {do-nothing};
printf("0");
x=x-y;
printf("3");
```

I. CS3E03 II. CES303 III. CSE303 IV. CE03S3 V. S3CE03

bana göre cevap : 1, 2, 3

Final 5.soru (10p) (cpu time)

A doubling scheduler uses a prioritized round-robin scheduling policy. New processes are assigned an initial quantum of length R. Whenever a process uses its entire quantum without blocking, its new quantum is set to twice its current quantum (2*R). If a process blocks before its quantum expires, its new quantum is reset to R. For the purposes of this question, assume that every process requires a finite total amount of CPU time.

"doubling scheduler" olarak adlandırılan bir sistem anlatılmaktadır. Bu sistem, bir işlemin "prioritized round-robin scheduling policy" olarak adlandırılan bir sıralama politikası kullanarak çalışmasını sağlar. Yeni işlemler, R uzunluğunda bir başlangıç "quantum" adı verilen bir zaman dilimi alır. Herhangi bir işlem, quantum'unun tamamını bloklamadan kullandığında, yeni quantum'u, mevcut quantum'unun iki katına (2*R) ayarlanır. Eğer bir işlem, quantum'u doldurmadan önce bloklar, yeni quantum'u R'ye geri döndürülür. Bu sorunun amaçları doğrultusunda, her işlem için CPU zamanının bir sonucu olduğu varsayılır.

(a) (5) Suppose the scheduler gives higher priority to processes that have larger quanta. Is starvation possible in this system? Why or why not?

No, starvation is not possible. Because we assume that a process will terminate, the worst that can happen is that a CPU bound process will continue to execute until it completes. When it finishes, one of the lower priority processes will execute. Because the I/O bound 1 processes will sit on the low priority queue, they will eventually make it to the head of the queue and will not starve.

Bu sistemde, işlemlerin quantum'larının büyüklüğüne göre öncelik sırası belirlenmiştir. Bu durumda, sistemde "starvation" (açlık) olasılığının olup olmadığı sorulmuştur.
Cevap olarak, işlemlerin tamamlanacağı varsayılmıştır. Bu durumda, en kötü ihtimal, CPU yüklü bir işlemin tamamlanana kadar çalışmaya devam etmesidir. Bu işlem, çalışma sırasında, quantum'unun tamamını kullanır ve yeni quantum'u, mevcut quantum'unun iki katına (2*R) ayarlanır. Ancak, bu işlem tamamlandığında, düşük öncelikli bir işlem çalışmaya başlar. Bu durumda, açlık olmaz.
Aynı şekilde, I/O yüklü bir işlemin sisteme girişi durumu da düşünülmüştür. Bu işlemler, düşük öncelikli bir kuyrukta bekler ve quantum'ları daha küçüktür. Bu nedenle, I/O yüklü işlemler, düşük öncelikli kuyruktaki bir işlem olana kadar beklerler. Bu işlemler, zamanının sonunda, düşük öncelikli kuyruktaki bir işlem çalışmaya başlar. Bu nedenle, sistemde açlık olmayacağı ifade edilmiştir.

(b) (5) Suppose instead that the scheduler gives higher priority to processes that have smaller quanta. Is starvation possible in this system? Why or why not?

Yes, starvation is possible. Suppose a CPU bound process runs on the processor, uses its entire quantum, and has its quantum doubled. Suppose a steady stream of I/O bound processes enter the system. Since they will always have a lower quantum and will be selected for execution before the process with the doubled quantum, they will starve the original process.

Bu soruda, "doubling scheduler" olarak adlandırılan bir sistem hakkında bilgi verilmektedir. Bu sistemde, işlemlerin çalışma sırasını belirlemek için, işlemlerin quantum'larının büyüklüğüne göre tersine öncelik sırası belirlenmiştir. Yani, quantum'ları daha küçük olan işlemler daha yüksek öncelik alır. "Quantum", bir işlemin çalışma süresini belirleyen bir zaman dilimidir. Yeni işlemler, R uzunluğunda bir başlangıç quantum'u alır. Eğer bir işlem, quantum'unun tamamını bloklamadan kullandıysa, yeni quantum'u, mevcut quantum'unun iki katına (2*R) ayarlanır. Eğer bir işlem, quantum'unu doldurmadan önce bloklar, yeni quantum'u R'ye geri döndürülür.
Bu sistemde, sistemde "starvation" (açlık) olasılığının olup olmadığı sorulmuştur. Cevap olarak, sistemde CPU yüklü bir işlemin çalıştığı, quantum'unun tamamını kullandığı ve yeni quantum'unun mevcut quantum'unun iki katına çıktığı varsayılmıştır. Aynı zamanda, I/O yüklü bir işlemin sisteme düzenli olarak girdiği de varsayılmıştır. Bu I/O yüklü işlemler, düşük quantum'ları nedeniyle, daha yüksek öncelik alır ve CPU yüklü işlemin önünde çalışmaya başlar. Bu nedenle, CPU yüklü işlemin quantum'unu kullanmasına izin verilmez ve bu işlem "starve" edilir (açlık yaşar). Bu nedenle, sistemde açlık olasılığının olduğu ifade edilmiştir.

A doubling scheduler, bir işlemi aralık arttıran bir planlama algoritmasıdır. Bu, işlemler arası geçişlerin yükünü azaltmaya yardımcı olabilir, çünkü işlem bir kez daha kesintiye uğradıktan sonra daha uzun bir süre çalışabilecektir.
Starvation, bir işlemin tamamlanması için gereken kaynaklara erişememesi durumudur, çoğunlukla diğer işlemler tarafından sürekli olarak kesintiye uğratıldığı için. Bu, performansın kötü olmasına ve verimliliğin azalmasına neden olabilir, çünkü işlem tamamlanması uzun sürebilir veya asla tamamlanmayabilir.
Round-robin scheduling, işlemlere sabit bir zaman dilimini (aşırı işlem süresi olarak da bilinir) atayan ve daire şeklinde yürütülen bir planlama politikasıdır. Bu politika, tüm işlemlerin CPU zamanının adil bir payını alacağını garanti etmeye yardımcı olur ve tek bir işlemin CPU'yu monopolize etmesini önler.

Final 6.soru (20p) (bankers algorithm)

	Allocation					Max					Available				
	A	B	C	D	E	A	B	C	D	E	A	B	C	D	E
P0	2	0	0	1	0	4	2	1	2	3	3	3	3	1	2
P1	0	1	2	1	1	3	2	5	2	3					
P2	2	1	0	3	1	2	3	1	7	3					
P3	1	3	1	0	2	1	4	2	4	4					
P4	1	4	3	2	2	3	6	6	5	4					

Consider the given snapshot of a system: Answer the following questions using the banker's algorithm:
a) How many resources are there for each resource type in this system?
b) Is this system safe? If yes provide a safe sequence. If not explain why?
c) If a request from P4 arrives for (0,0,2,0,1) can the request be granted immediately? Explain why?

Step: 1
Solution Approach:

To compute the number of resources we have to sum up the allocatd resources for each process and the avaiable resource.

In order to check for safe state the first step is to create the need matrix. The need matrix can be created byt subtracted the allocation matrix from the Max matrix. Then we need to find the safe sequence in which the need of the process can be satisfied with the available amount of resources. If the need of any of the process cannot be satisfied then the system is nt in safe state.

Solution:

a)
A=2+0+2+1+1+3=9
B=0+1+1+3+4+3=12
C=0+2+0+1+3+3=9
D=1+1+3+0+2+1=8
E=0+1+1+2+2+2=8

b)
Need Matrix: Max'tan Allocation matrix'i çıkarılır.

	A	B	C	D	E
P0	2	2	1	1	3
P1	3	1	3	1	2
P2	0	2	1	4	2
P3	0	1	1	4	2
P4	2	2	3	3	2

Available:

A	B	C	D	E
3	3	3	1	2

(Önce a şıkkındaki toplamlara bakıp available matrisine yeten en büyük değerdeki satırın allocation matrisindeki satır available matrisin'e eklenir.)
Need of P1 can be satisfied. P1 executes and deallocated the resource

Allocation

P1	0	1	2	1	1

Available:

A	B	C	D	E
3	4	5	2	3

Need of P0 can be satisfied. P0 executes and deallocated the resource

Available:

A	B	C	D	E
5	4	5	3	3

Need of P4 can be satisfied. P4 executes and deallocated the resource

Available:

A	B	C	D	E
6	8	8	5	5

Need of P2 can be satisfied. P2 executes and deallocated the resource

Available:
A B C D E
8 9 8 8 6

Need of P3 can be satisfied. P3 executes and deallocated the resource

Available:
A B C D E
9 12 9 8 8

Since all the process got executed the system is in safe state.
Safe sequence: P1,P0,P4,P2,P3

c) Suppose P4 is granted (0,0,2,0,1). The available resources will be

Available:
A B C D E
3 3 1 1 1

Each process has a requirement of 2 or more instance of E but there is only 1 instance of E that is available.
Hence P4 cannot be granted the request immediately.

Explanation:Please refer to solution in this step.

Answer:

Answer Summary:

a) The total number of instance of each resource in the system:

A B C D E
9 12 9 8 8

b) The system is in safe state and the safe sequence is: P1, P0, P4, P2, P3

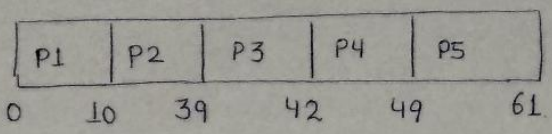
c) P4 cannot be granted (0,0,2,0,1) imediately else the system will not be safe.
(ÇÜNKÜ P4 öyle olursa max-allocation farkı büyür ve ilk P4 ten başlanmak zorunda kalınır bu şekilde de başlayamaz.) anlamak için şu linkteki 7.3 olan soruya bakılabilr:
https://www.os-book.com/OS9/practice-exer-dir/7-web.pdf

Final 7.soru (20p) (Cpu scheduling Burst algorithm)

Process- burst time p1=10 p2=29 p3=3 p4=7 p5=12 considering the given processes and their CPU burst times, compare the fcfs and RR scheduling algorithms using the average waiting time metric.Quanta q=5. (fcfs ilk giren ilk çalışıyo, round robinde verilen quarter neyse her birisi o süre bitesiye çalışıyor)

FCFS :→ In FCFS, processes are given to CPU on first come first serve basis.

Gantt Chart :-



P1's waiting time = 0

P2's waiting time = 10

P3's waiting time = 39

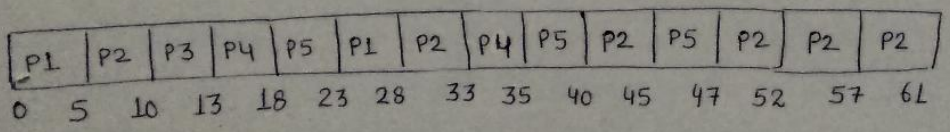
P4's waiting time = 42

P5's waiting time = 49

Average waiting time = $\frac{0 + 10 + 39 + 42 + 49}{5} = \frac{140}{5} = 28$.

Round Robin Scheduling :→ Each process is assigned a fixed time slot

Gantt chart :-



P1's waiting time = 0 + (23 - 5) = 0 + 18 = 18

P2's waiting time = (5-0) + (28-10) + (40 - 33) + (47 - 45) = 32

P3's waiting time = (10 - 0) = 10

P4's waiting time = (13-0) + (33-18) = 28

P5's waiting time = (18 - 0) + (35 - 23) + (45-40) = 35

Average waiting time = $\frac{18 + 32 + 10 + 28 + 35}{5} = \frac{123}{5} = 24.6$.