מערכות הפעלה- מטלה 3

היבה אבו-כף: 323980441

גיל פסי: 206500936

שאלה 1 (60 נקודות)

נתונה המערכת הבאה של 4 תהליכים ו-5 סוגים של משאבים:

Allocation

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

Claim / Max Demand

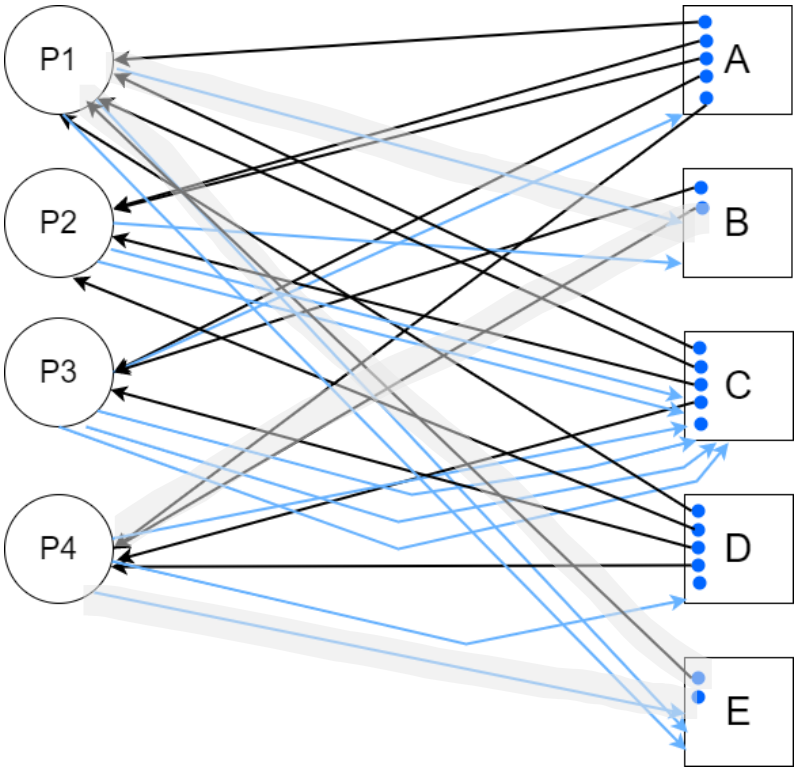
	A	B	C	D	E
P1	1	1	2	1	3
P2	2	2	2	1	0
P3	2	1	3	1	0
P4	1	1	2	2	1

Available

A	B	C	D	E
0	0	1	1	X

א) (5 נק') נתון ש X=1, צייר את תמונת המצב של המערכת באמצעות Resource Allocation Graph.

א)



ב) (5 נק') האם המצב הנוכחי בטוח? הסבר באמצעות הגרף והאלגוריתם הבנקאי.

ב)

נתבונן בגרף ונזהה מעגל :

P1->B -> P4 ->E -> P1

כלומר קיים חשד ל- deadlock וחייבים להשתמש באלגוריתם הבנקאי בכדי לוודא שהמצב בטוח.

אלגוריתם הבנקאי:

1. אתחול: work := available
2. חפש P_i לא מסומן כך ש- $N_i \leq work$ — אם אין, לך לצעד 4
3. אם יש כזה: סמן את P_i "מטופל"
   - $work += A_i$
   - חזור לצעד 2
4. אם כל התהליכים מסומנים, המצב בטוח

נריץ את האלגוריתם:

נתחיל ממשאב E:

קיימים שני תהליכים הדורשים משאבים מ- E , P1 ו- P4.

ננסה להתחיל מ – P1 .

P1.Demand – P1.Allocation = 3 – 1 = 2

כלומר דרושים עוד שני משאבים מסוג E .

נתון כי X = 1 ולכן ל - E יש רק משאב חופשי אחד. לא נוכל למלא את הדרישות של P1 .

ננסה להתחיל מהאופציה השנייה- P4 .

P4.Demand – P4.Allocation = 1-0 = 1

יש ברשותנו את המשאב ולכן נקצה אותו ל- P4.

בתום השימוש , P4 משחרר משאב אחד מסוג E . נמשיך לתהליך הבא.

שוב ננסה להקצות ל – P1 ושוב ניתקל באותה הבעיה. מסקנה : לא קיימת סדרת הקצאות שמסמנת את כל התהליכים. לכן המצב אינו בטוח.

Allocation

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

Claim / Max Demand

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

Available

A	B	C	D	E
0	0	1	1	X

כמו כן ניתן לראות אישור נוסף מהגרף: עבור E קיימות 2 נקודות וישנם שלושה חצים המובילים לתהליך P1 .

ג) (5 נק') מה הוא הערך של ה-X המינימאלי שעבורו המצב הוא בטוח?

ג)

ראינו בסעיף הקודם שמקור הבעיה הוא במשאב E אל מול הדרישה של P4 ל – 3 משאבים.

לכן עלינו השיג מספר משאבים שיכול להגיע ל – 3 . מאחר ומשאב אחד כבר מוקצה , חסרים לנו עוד 2 .

לכן עבור X=2 המצב יהיה בטוח .

ד) (40 נק') כתבו תכנית ב-Java אשר תקלוט מהמשתמש את כמות התהלכים ומספר המשאבים במערכת. לאחר מכאן, התוכנית תקלוט את המטריצות Allocation, Claim ו-Available. על התוכנית לבדוק את הדברים הבאים:

1. האם המערכת נמצאת במצב בטוח. אם כן התוכנית תדפיס שכן, ואם לא התוכנית תדפיס את התהליכים המעורבים ב-deadlock.
2. התוכנית תקלוט מהמשתמש את סוג המשאב (A, B, C, D או E) ותדפיס את כמות העותקים המינימאלית (במערך Available) מהסוג הנ"ל, כך שהמערכת תהיה במצב בטוח.
3. התוכנית תקלות מהמשתמש את מספר התהליך ואת הבקשה של התהליך (כמה משאבים מכל סוג שתהליך מבקש) ותדפיס האם יש להיענות בחיוב לבקשה.

ד)

מצורף בקובץ דחוס.

ה) (5 נק') במערכת יש 3 סורקים ו-2 תהליכים. כל תהליך צריך מקסימום 2 סורקים. האם יתכן deadlock? נמק.

ה)

המצב בטוח , אין סיכוי ל – deadlock .

בשביל deadlock יש לחלק את כל המשאבים. במצב המתואר ישנן 3 משאבים בסה"כ ושני סורקים.

מסיבה זו החלוקה חייבת להיות אחת מבין השתיים הבאות : שני סורקים לתהליך 1 וסורק אחד לתהליך 2 או להיפך, סורק אחד לתהליך 1 ושני סורקים לתהליך 2 .

כלומר כך או אחרת לפחות אחד מהתהליכים יקבל שני סורקים וישלים בהצלחה את פעולתו .כך יפנה את המקום לתהליך השני.

שאלה 2 (10 נקודות)

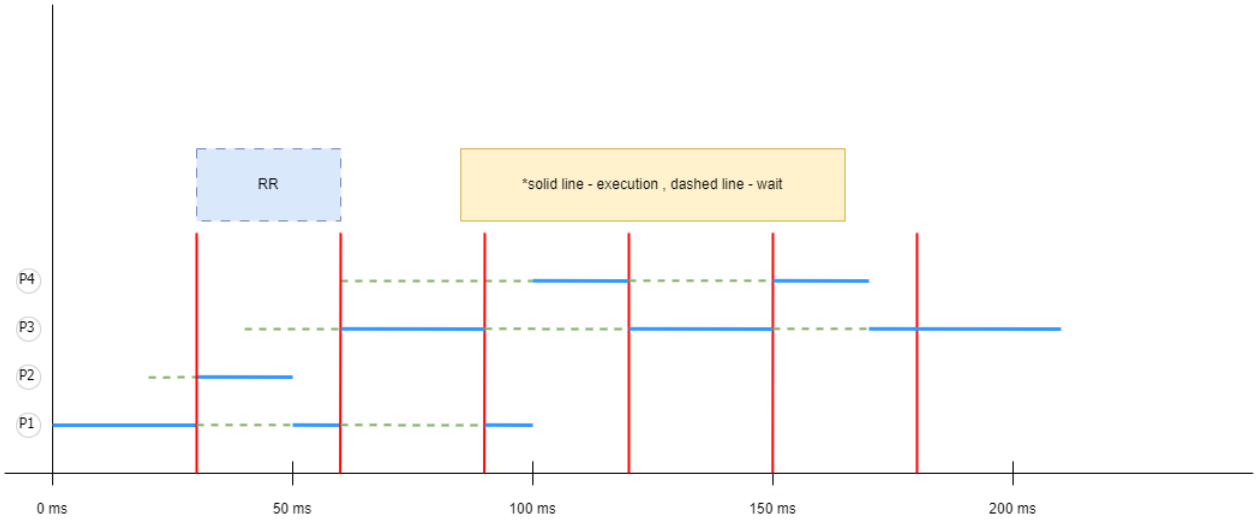
ארבע משימות P1, P2, P3, P4 מגיעות למערכת. זמני ההגעה, זמני ה-CPU burst ועדיפויות נתונים בטבלה (1 היא העדיפות הגבוהה ביותר, 4 היא העדיפות הנמוכה ביותר):

Process	Burst Time	Priority	Arrival Time
P1	50 ms	4	0 ms
P2	20 ms	1	20 ms
P3	100 ms	3	40 ms
P4	40 ms	2	60 ms

עבור כל אחד מאלגוריתמי התזמון הבאים ציירו תרשים Gantt ומלאו את הטבלת ההרצה וחשבו את זמן הסבב הממוצע (turnaround time).

א. Round Robin (time quantum = 30 ms)
ב. Nonpreemptive Priority Scheduling
ג. First-Come, First-Served
ד. Shortest Job First

.א



Process	Arrival	Execution(ms)	Finish(ms)
P1	0	0,50,90	100
P2	20	30	50
P3	40	60,120,170	210
P4	60	100,150	170

<u>Turnarounds:</u>

P1 : 100 – 0 = 100

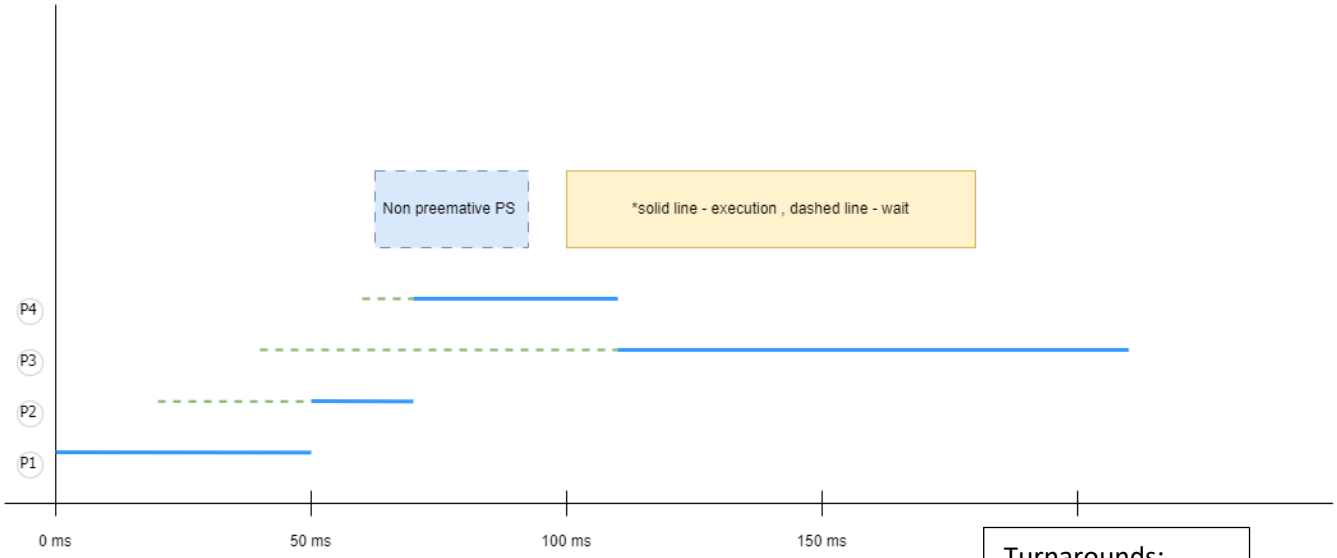
P2: 50 – 20 = 30

P3 : 210 – 40 = 170

P4 : 170 – 60 = 110

ממוצע: (100 + 30 + 170 + 110) / 4 = 102.5ms

ב.



Process	Arrival	Execution(ms)	Finish(ms)
P1	0	0	50
P2	20	50	70
P3	40	110	210
P4	60	70	110

Turnarounds:

P1 : 50 – 0 = 50

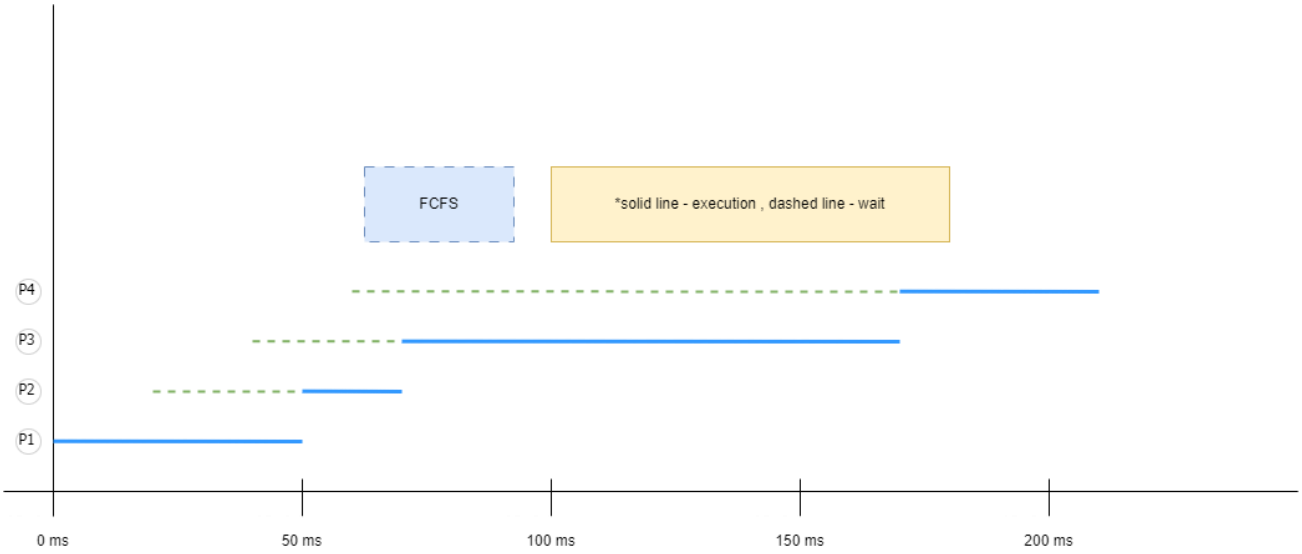
P2: 70 – 20 = 50

P3 : 210 – 40 = 170

P4 : 110 – 60 = 50

ממוצע: (50 + 50 + 170 + 50) / 4 = 80ms

ג.



Process	Arrival	Execution(ms)	Finish(ms)
P1	0	0	50
P2	20	50	70
P3	40	70	170
P4	60	170	210

Turnarounds:

P1 : 50 – 0 = 50

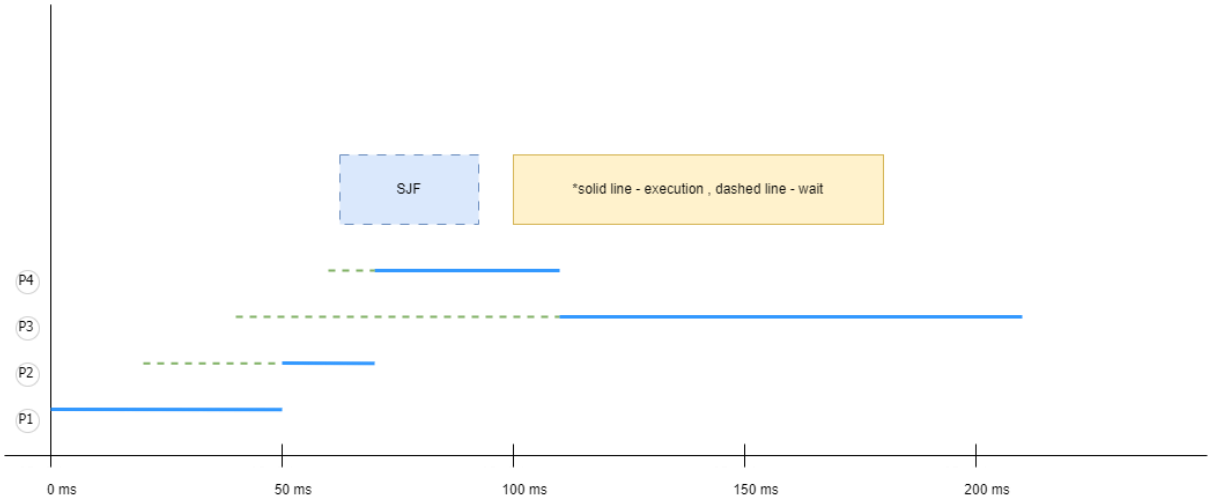
P2: 70 – 20 = 50

P3 : 170 – 40 = 130

P4 : 210 – 60 = 150

ממוצע: (50 + 50 + 130 + 150) / 4 = 95ms

ד.



Process	Arrival	Execution(ms)	Finish(ms)
P1	0	0	50
P2	20	50	70
P3	40	110	210
P4	60	70	110

Turnarounds:

P1 : 50 – 0 = 50

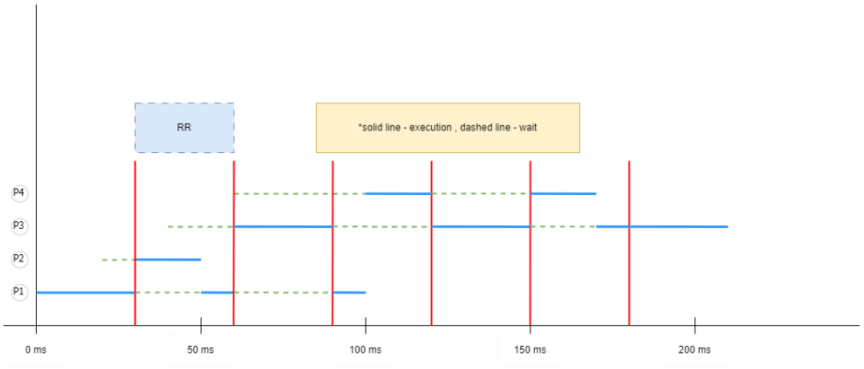
P2: 70 – 20 = 50

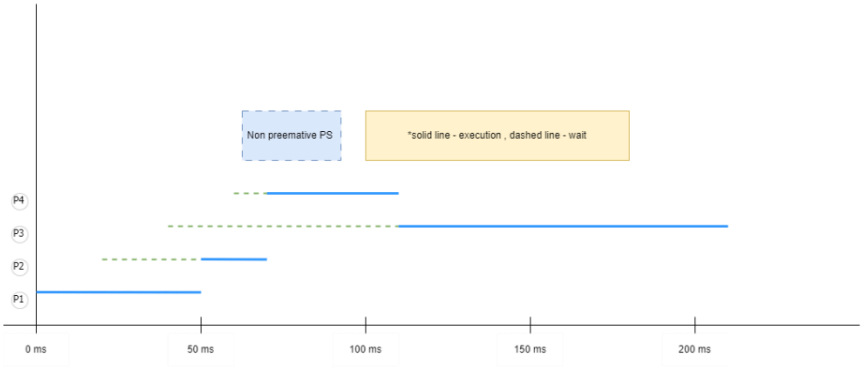
P3 : 210 – 40 = 170

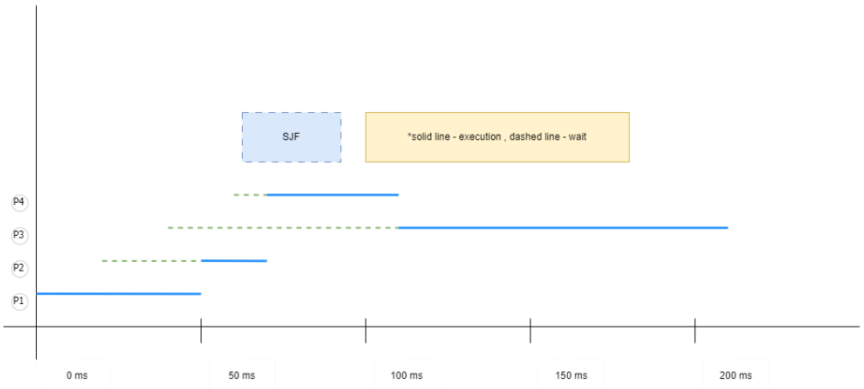
P4 : 110 – 60 = 50

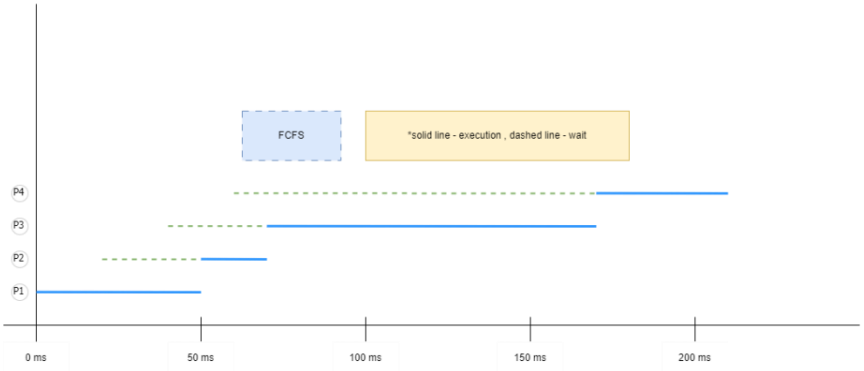
ממוצע: (50 + 50 + 170 + 50) / 4 = 80ms

כל הגאנטים ביחד להשוואה:









שאלה 3 (6 נקודות)

פעולת test-and-set האטומית מוגדרת כדלהלן:

```
boolean test_and_set (int i)
{
  if (i == 0)
   { i= 1;
   return true;
  }
  else return false;
}
```

הסבר כיצד משתמשים בפעולה זו על מנת ליצור מניעה הדדית.

האם פתרון זה מכיל busy-waiting?

ניתן ליצור מניעה הדדית באופן הבא:

Global int i=0;	

Process 1:
while(true){
if(test_and_set(i))
Process job …
i=0;
break;
}
	Process 2:
	while(true){
	if(test_and_set(i))
	Process job …
	i=0;
	break;
	}

כלומר, מאחר והפעולה test_and_set אטומית, ניתן להשתמש בה ע"מ לדמות נעילה של מנעול כמו בסנכרון רגיל. יחד עם זאת המימוש לא מאפשר 'להעיר' תהליך עם notify או פונקציה דומה אחרת. לכן יש צורך להשתמש בבדיקה מתמשכת – **Busy-waiting** .

שאלה 4 (6 נקודות)

בטבלה שלהלן כתובים המשאבים שכל תהליך מחזיק, וכן משאב שהוא מבקש. הנח כי אם לא מצוין אחרת, מכל משאב ישנו עותק אחד.

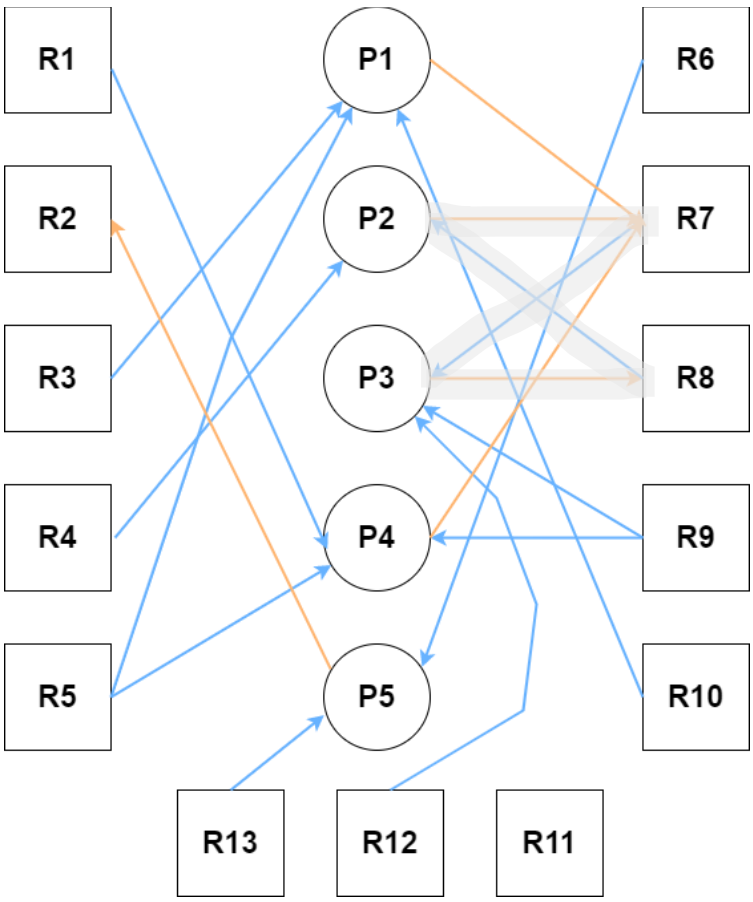
Process תהליך	Have resources משאבים מוחזקים	Requests resources בקשות למשאבים
P1	R3, R10, R5	R7
P2	R4, R8	R7
P3	R7, R9, R12	R8
P4	R5, R9, R1	R7
P5	R6, R13	R2

האם קיים deadlock?

a. אם לא, תאר בתסריט את סדר נטילת המשאבים וריצת התהליכים עד לסיום.
b. אם כן, מי מעורב בו? הסבר. שרטט את גרף הקצאת המשאבים כפי שלמדנו, עבור התהליכים המעורבים בדדלוק.

במידה ו P3 אינו קיים, האם יש שינוי בתשובתך?

- נתבונן ב Resource Allocation Graph :



ניתן לראות את המעגל P2 -> R8 -> P3 -> R7 -> P2

מאחר והוגדר כי בברירת המחדל כי לכל משאב יש העתק אחד, ניתן לדעת כי מתקיים deadlock .

- P3 מחזיק את המשאב R7 . משאב זה הוא חלק מהמעגל שציינו קודם. מלבד זאת ניתן לראות בגרף שלא קיימים עוד מעגלים כלל. אי לכך ביטול P3 יבטל את ה -deadlock .

שאלה 5 (6 נקודות)

במערכת מסוימת ישנם ארבעה סוגי משאבים (כאשר מכל משאב יש מספר נתון של יחידות) וחמישה תהליכים שמשתמשים במשאבים .להלן פירוט של ההקצאות.

Allocation

	R1	R2	R3	R4
P1	0	0	1	2
P2	2	0	0	0
P3	0	0	3	4
P4	2	3	5	4
P5	0	3	3	2

Claim / Max Demand

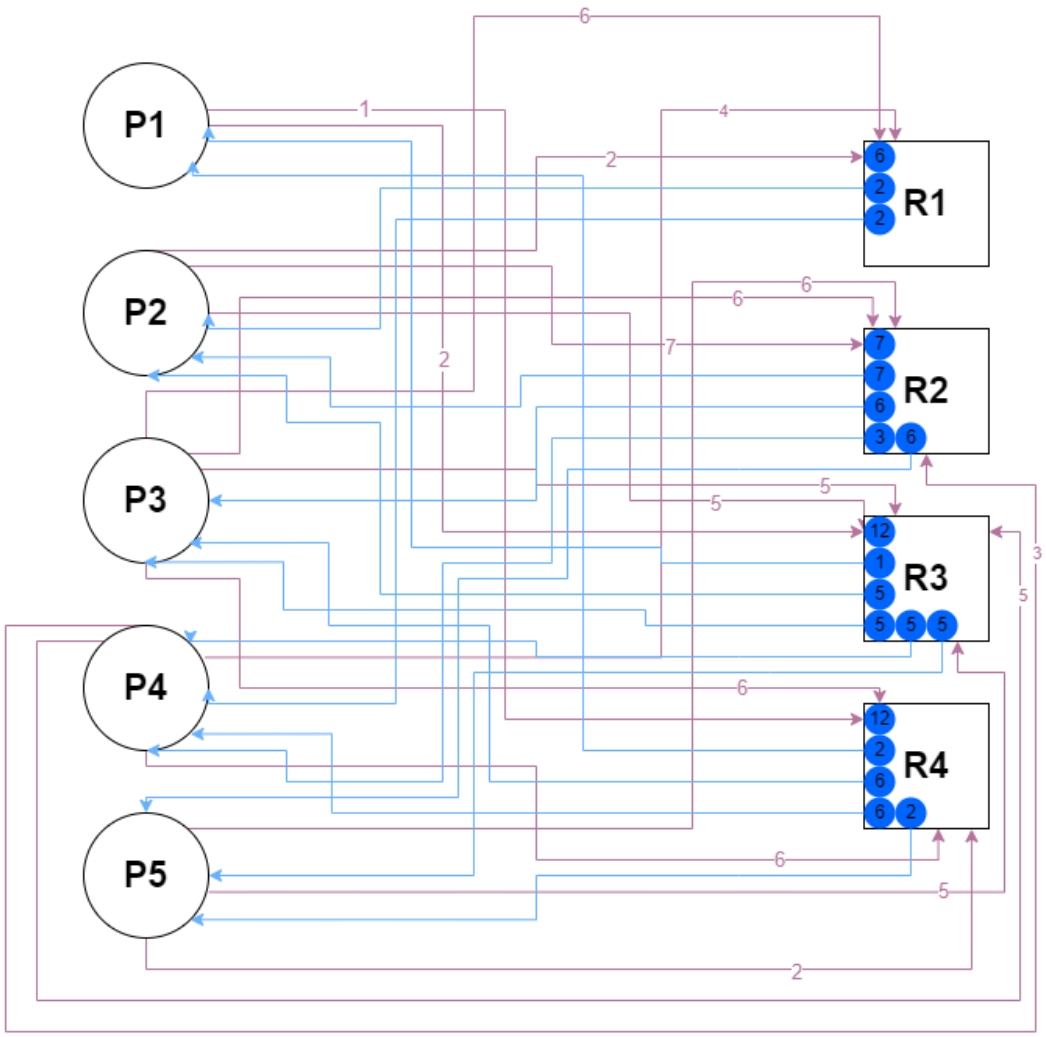
	R1	R2	R3	R4
P1	0	0	1	2
P2	2	7	5	0
P3	6	6	5	6
P4	4	3	5	6
P5	0	6	5	2

Resources

R1	R2	R3	R4
6	7	12	12

א. צייר את גרף הקצאת המשאבים המתאים.

א.



ב. האם המצב הנוכחי הוא מצב בטוח ?נמק.

ב.

בכדי לגלות האם המצב בטוח, נשתמש באלגוריתם הבנקאי ונוסיף לו "שדרוג" .

אלגוריתם הבנקאי בודק האם ישנה סדרת הקצאות שתשחרר את כלל התהליכים.

כמו כן, ניתן להסיק שאם הדרישה הגדולה ביותר למשאב מסוים קטנה או שווה לכמות המשאב החופשית, ניתן לשחרר אותה.

לאחר השחרור , הדרישה השניה בגודלה הפכה כעת להיות הדרישה הגדולה ביותר. על בסיס העקרון ניתן לשחרר גם אותה.

על השיטה הזו ניתן לחזור עבור כל הדרישות.

לפיכך שאם כמות המשאבים הגדולה ביותר שנדרשת קטנה או שווה מכמות החופשית, ניתן להשיג בוודאות סדרת הקצאות תקינה. כלומר המצב בטוח.

תרחיש זה מתקיים עבור כל משאב בטבלה שלנו ולכן במקרה הזה המצב בטוח.

ג. אם במצב הנוכחי מגיעה בקשה של P3 להקצאה של יחידה נוספת ממשאב R2 האם יש להיענות בחיוב לבקשה? (לפי אלגוריתם הבנקאי)

ג.

ניתן לראות שלאחר הגדלת הדרישה של P3 למשאב נוסף מסוג R2 הדרישה תהיה כעת ל – 7 יחידות מסוג R2 .

כמות היחידות החופשית היא 7 לפי הטבלה.

על בסיס ההסבר של הסעיף הקודם (שגם מבוסס על אלגוריתם הבנקאי) , $7 \leq 7$ ולכן אין בעיה עם הגדלת ההקצאה ביחידה אחת .

יש לאשר את הבקשה.

שאלה 6 (12 נקודות)

נתונה התוכנית הבאה לפתרון בעיית ה"פילוסופים האוכלים" (עבור N=5).

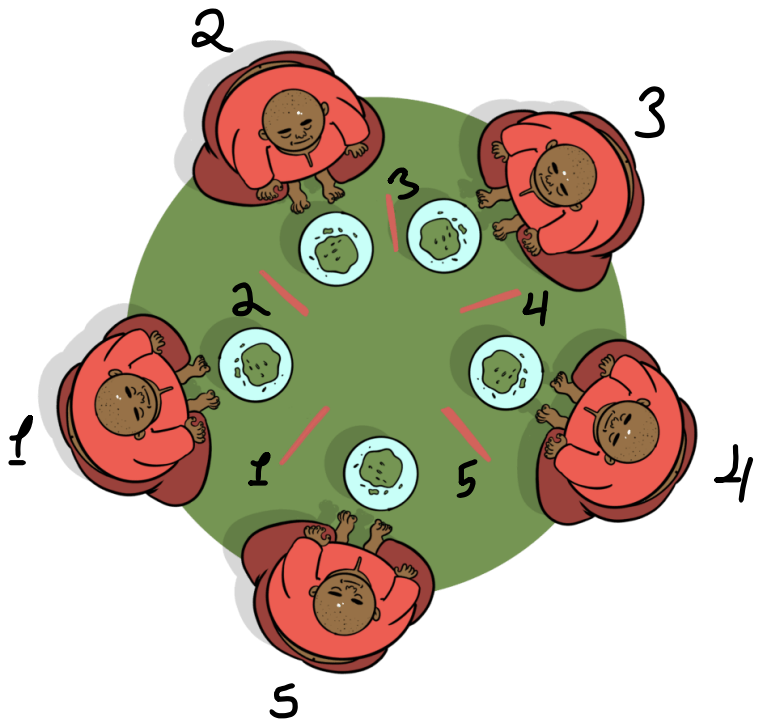
```
void philospher(int i)
{
      while(TRUE)
      {
            think();
            take_fork(i);
            take_fork((i+1) % 5);
            eat();
            put_fork(i);
            put_fork((i+1) % 5);
      }
}
```

ענו על השאלות הבאות:

א. האם הפתרון הוא Deadlock-free? אם כן, הוכיחו, אחרת הראו תסריט שייצור Deadlock.

א.

אפתור את השאלה בהינתן ש – i הוא ערך חד-חד ערכי לפילוסוף המשקף את מיקומו באופן המתואר בסרטוט:



נתבונן בתסריט הבא:

1. פילוסוף 1 תופס את מזלג 1 .
2. פילוסוף 2 תופס את מזלג 2 .
3. פילוסוף 3 תופס את מזלג 3 .
4. פילוסוף 4 תופס את מזלג 4 .
5. פילוסוף 5 תופס את מזלג 5 .

6. כעת לכל מזלג יש בעלים אבל אף פילוסוף לא סיים לאכול.

מסקנה : הפתרון אינו deadlock free .

ב. הקוד בסעיף א' הוא Uniform כלומר זהה עבור כל אחד מהפילוסופים. אם היינו יכולים לכתוב קוד שונה לכל פילוסוף, האם פתרון שבו יש לכל מזלג semaphore בינארי המייצג אותו היה אפשרי ? הוכיחו שלא, או רשמו את הקוד עבור הפילוסופים.

ב.

כתיבת קוד שונה עבור כל פילוסוף יכולה לפתור את הבעיה :

Philosopher 5

```
void philospher(int i)
{
      while(TRUE)
      {
            think();
            take_fork((i+1) % 5);
            take_fork(i);
            eat();
            put_fork((i+1) % 5);
            put_fork(i);
      }
}
```

Philosophers 1- 4

```
void philospher(int i)
{
      while(TRUE)
      {
            think();
            take_fork(i);
            take_fork((i+1) % 5);
            eat();
            put_fork(i);
            put_fork((i+1) % 5);
      }
}
```

הפכנו את סדר המזלגות עבור הפילוסוף האחרון ובכך יצרנו חוסר סימטריה

שתשבור את ה – deadlock .

ג.

מאלגוריתם הבנקאי בבעיית הפילוסופים עולה כי הבעיה הבטוחה למרות שידוע לנו כי הבעיה עלולה

ליצור deadlock .

הפערים נובעים מהעובדה שאלגוריתם הבנקאי מסתמך על היכולת של מערכת ההפעלה לסרב לתת משאבים. הדבר שקול לכך שאם פילוסוף מנסה לאחוז במזלג, המזלג יכול להתנגד ולבטל את פעולת האחיזה. שינוי זה בבעיה יאפשר למשאבים לנהל את התהליכים ולא שהתהליכים ינהלו את המשאבים.

ד.

פתרון זה לא הוצע עד היום כיוון שבבעיית הפילוסופים מנסים למצוא פתרון כללי שיהיה נכון לכל מקרה ולא רק למקרים שבהם ניתן לנהל את הקצאת המשאבים.