TOP SECRET

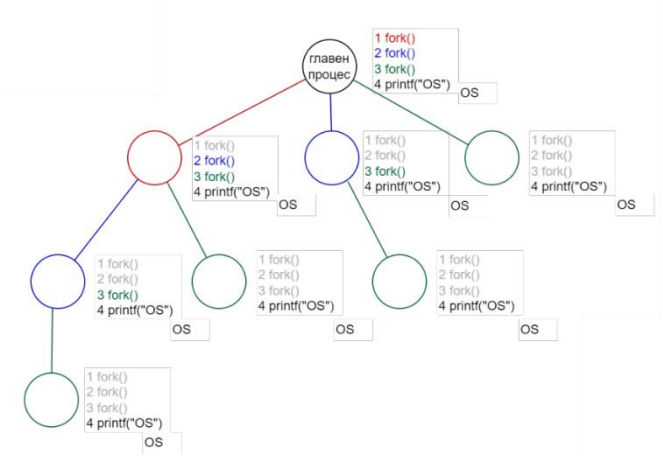
CLASSIFIED DOCUMENT

OS Theory Exercises

FINKI FEMBOYS

Колку пати ќе се испечати **OS** по извршувањето на кодот?

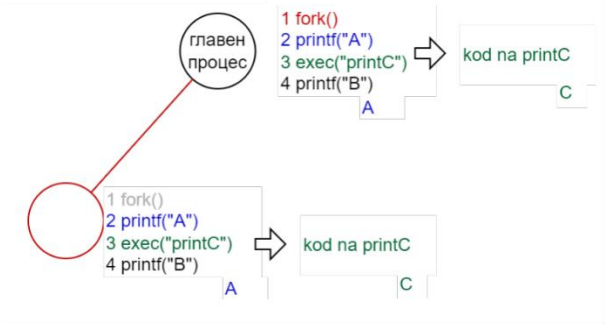
```
fork();
fork();
fork();
printf("OS");
```



OS ќе се испечати 8 пати.

Колку пати ќе се испречати **B** по извршувањето на кодот?

```
fork();
printf("A");
exec("printC");
printf("B");
```



B нема да се испечати.

Претпоставете дека имате три конкурентни нишки во рамки на еден процес кои ги извршуваат процедурите **A**, **B**, и **C**. Наведете ја вредноста на глобалната променлива **x** како и соодветната секвенца на извршување откако ќе завршат трите нишки.

```
#include <prototypes.h>
typedef int semaphore;
semaphore sA = 1, sB = 0, sC = 0;
int x = 0;

void A() {
    Down(&sA);
    x += 2;
    Up(&sB);
}

void B() {
    Down(&sB);
    x -= 1;
    Up(&sC);
}

void C() {
    Down(&sC);
    x += 2;
    Up(&sA);
}
```

Секвенцата на извршување е: A, B, C.

Вредноста на x е: 3.

Со која **најмала** вредност треба да се иницијализира семафорот **s** во кодот за да **нема deadlock**, ако две различни нишки **T1** и **T2** вршат повикување на функциите дефинирани подолу и тоа во следниот редослед:

T1 повикува A, B, C, A, B
T2 повикува C, B, A, A, B

```
#include <prototypes.h>
typedef int semaphore;
semaphore s = ____ ;
int x = 0;


void A() {
    wait(&s);
    ...
}


void B() {
    signal(&s);
    ...
}


void C() {
    wait(&s);
    ...
}
```

Почетната вредност на семафорот е 3.

Вредноста на семафорот по извршување на двете нишки е 1.

За s = 0 :
A1 : s = -1 (T1 спие)
C2 : s = -2 (T2 спие)
: нема кој да ги разбуди – deadlock

За s = 1 :
A1 : s = 0
C2 : s = -1 (T2 спие)
B1 : s = 0 (T2 е разбуден)
B2 : s = 1
C1 : s = 0
A2 : s = -1 (T2 спие)
A1 : s = -2 (T1 спие)
: нема кој да ги разбуди – deadlock

За s = 2 :
A1 : s = 1
C2 : s = 0
B1 : s = 1
B2 : s = 2
C1 : s = 1
A2 : s = 0
A1 : s = -1 (T1 спие)
A2 : s = -2 (T2 спие)
: нема кој да ги разбуди – deadlock

За s = 3 :
A1 : s = 2
C2 : s = 1
B1 : s = 2
B2 : s = 3
C1 : s = 2
A2 : s = 1
A1 : s = 0
A2 : s = -1 (T2 спие)
B1 : s = 0 (T2 е разбуден)
B2 : s = 1

Со која **најголема** вредност треба да се иницијализира семафорот **s** во кодот за да **има deadlock**, ако две различни нишки **T1** и **T2** вршат повикување на функциите дефинирани подолу и тоа во следниот редослед:

T1 повикува A, A, B, C, C
T2 повикува A, A, B, C, C, B

```
#include <prototypes.h>
typedef int semaphore;
semaphore s = ____ ;
int x = 0;


void A() {
    wait(&s);
    ...
}


void B() {
    signal(&s);
    ...
}


void C() {
    wait(&s);
    ...
}
```

Почетната вредност на семафорот е 5.

Вредноста на семафорот кога ќе дојде до deadlock ќе биде -1.

За s = 0 :
A1 : s = -1 (T1 спие)
A2 : s = -2 (T2 спие)
: нема кој да ги разбуди – deadlock

...

За s = 5 :
A1 : s = 4
A2 : s = 3
A1 : s = 2
A2 : s = 1
B1 : s = 2
B2 : s = 3
C1 : s = 2
C2 : s = 1
C1 : s = 0 (T1 завршил)
C2 : s = -1 (T2 спие)
: нема кој да го разбуди T2 – deadlock

...

За s = 6 нема да има deadlock – двете нишки ќе завршат па одговорот ќе биде s = 5.

Даден е систем со 5 процеси (**P0, P1, P2, P3, P4**) и три типа на ресурси (**A, B, C**). Системот користи распределување според приоритети, каде помал број на процес означува поголем приоритет.
Пополнете ги матриците **Need** и **Available** и одговорете дали секој од процесите ќе заврши.

Available **A: 3** **B: 2** **C: 1**

Allocation (дадено)			
	A	**B**	**C**
P0	1	2	2
P1	1	3	1
P2	0	3	3
P3	1	3	1
P4	1	0	3

Max (дадено)			
	A	**B**	**C**
P0	3	3	5
P1	6	6	7
P2	3	5	4
P3	4	7	3
P4	7	7	4

Need			
	A	**B**	**C**
P0	2	1	3
P1	5	3	6
P2	3	2	1
P3	3	4	2
P4	6	7	1

Need = Max – Alloc

Available (по завршување)				
	A	**B**	**C**	
P0	4	7	6	2nd
P1	6	13	8	4th
P2	3	5	4	1st
P3	5	10	7	3rd
P4	7	13	11	5th

Available += Process Allocation
За следниот процес се гледа новодобиениот Available.

Дали процесот ќе заврши?	
P0	Да
P1	Да
P2	Да
P3	Да
P4	Да

Даден е систем со следните карактеристики:

- 4 процеси: P0, P1, P2, P3
- 2 типа на ресурси: A (6 инстанци), B (3 инстанци)
- Во време T:

	Allocation A B	**Max** A B
P0	0 1	3 1
P1	1 1	4 2
P2	1 0	5 2
P3	2 0	2 1

Available:
- A: 6 – 1 – 1 – 2 = 2
- B: 3 – 1 – 1 = 1

Needed:
- P0: 3, 0
- P1: 3, 1
- P2: 4, 2
- P3: 0, 1

Available (по завршување)
- P3: 4, 1
- P0: 4, 2
- P1: 5, 3
- P2: 6, 3

Во системот има достапни (available) 2 инстанца/и од ресурсот A и 1 инстанца/и од ресурсот B.
На процесот P3 му требаат (needed) уште 0 инстанца/и од ресурот A и 1 инстанца/и од ресурсот B.
Системот е во безбедна состојба бидејќи ресурсите можат да се доделат на процесите по следниот редослед P3, P0, P1, P2.

Нека е даден еден систем во реално време со три периодични настани. Процесите кои што ги отсликуваат овие настани се со периоди во 100, 200, и 500 ms. Ако овие процеси побаруваат 50, 30, и 100 ms од времето на CPU-то, дали системот ќе може да управува со овие процеси, односно дали системот ќе биде распределив?

$\frac{50}{100} + \frac{30}{200} + \frac{100}{500} = 0.85$ Да, системот е распределив, бидејќи $0.85 \leq 1$.

Ако се додаде 4-ти процес со периода од 1 sec, колкаво треба да биде неговото максимално време за извршување за да може системот да биде распределив?

$1\,\text{sec} \rightarrow 1000\,ms$ $0.85 + \frac{x}{1000} \leq 1$ $\frac{x}{1000} \leq 1 - 0.85$ $x \leq 0.15 * 1000$
$x = 150\,ms$

За предвидување на времетраење на процеси се користи алгоритам на стареење со **a = ½**. Ако времетраењата на предходните 4 извршувања биле 40, 20, 40, и 15 ms, кое е предвидувањето за следното време на извршување?

$a = \frac{1}{2} = 0.5$ $F_n = a * T_{n-1} + (1 - a) * F_{n-1}$

T1 = 40 $F_1 = 40$ (нема F_0)

T2 = 20 $F_2 = 0.5 * 40 + 0.5 * 40 = 40$

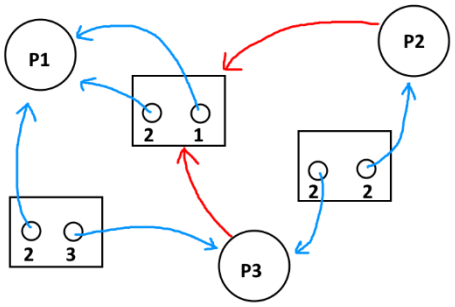
T3 = 40 $F_3 = 0.5 * 20 + 0.5 * 40 = 30$

T4 = 15 $F_4 = 0.5 * 40 + 0.5 * 30 = 35$

Во кој редослед ќе се извршат процесите и во која состојба е секој од нив на почетокот?

Сини стрелки – процесот ги поседува тие ресурси.
Црвени (или испрекинати) стрелки – на процесот му требаат тие ресурси.

P1 (Running) → P2 (Waiting) → P3 (Waiting)

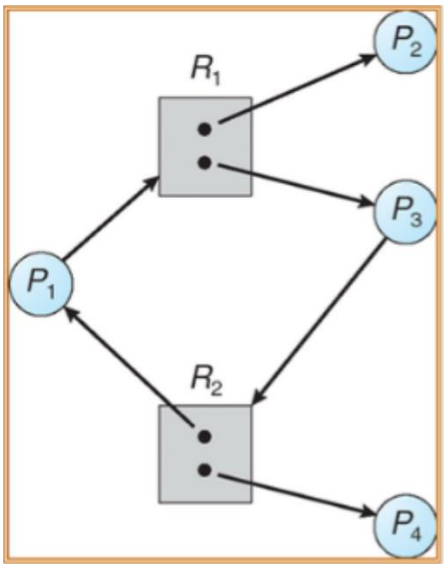


Даденият граф на алокација на ресурси прикажува систем со 4 процеси и 2 ресурси со вкупно 4 инстанци.

Според графот на 2 процеси им се доделени ресурси, а 2 процеси бараат ресурси да им бидат доделени кога ќе бидат достапни.

Графот има јамка. Во системот нема блокада.

Блокада нема дека има доволно инстанци за секој од процесите.



Нека се дадени 5 процеси: A, B, C, D, и E, и соодветно нивните времиња на извршување: 25, 22, 33, 35, 28. Процесите пристапуваат (скоро) истовремено. Да се пополни табелата за секој од распределувачките алгоритми.

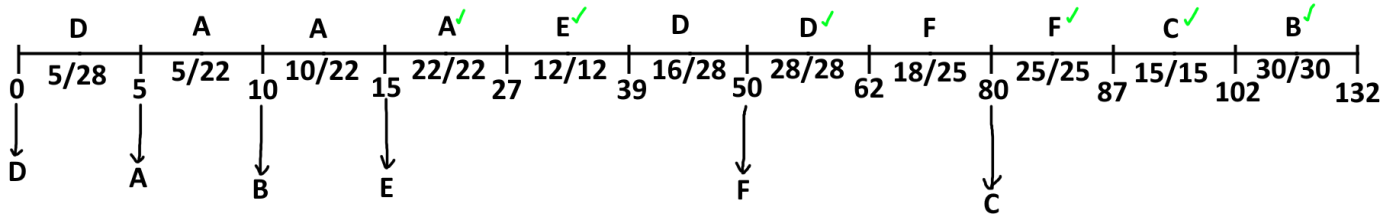
			Алгоритми за распределување					
			FCFS		SJF		Round Robin со квантум 10 ms	
Процес	Work Time	Arrive Time	wait time	sys time	wait time	sys time	wait time	sys time
A	25	0	0	25	22	47	80	105
B	22	0	25	47	0	22	85	107
C	33	0	47	80	75	108	105	138
D	35	0	80	115	108	143	108	143
E	28	0	115	143	47	75	107	135

За Round Robin:

A 10/25 **+0**	A 20/25 **+40**	A 25/25 **+ 40**	**= 80**
B 10/22 **+10**	B 20/22 **+40**	B 22/22 **+35**	**= 85**
C 10/33 **+20**	C 20/33 **+40**	C 30/33 **+27**	C 33/33 **+18** **= 105**
D 10/35 **+30**	D 20/35 **+40**	D 30/35 **+27**	D 35/35 **+11** **= 108**
E 10/28 **+40**	E 20/28 **+40**	E 28/28 **+27**	**= 107**

Со помош на алгоритамот SRTN да се распределат 6 процеси A, B, C, D, E, и F кои што имаат соодветно 22, 30, 15, 28, 12, и 25 времиња на извршување. Меѓутоа, процесите не се придружуваат истовремено на редицата на спремни процеси. Нивните времиња на придружување се 5, 10, 80, 0, 15m и 50 ms соодветно.

Процес	Време на извршување	Време на пристигнување	Време на чекање	Вкупно време во систем
A	22	5	0	22
B	30	10	92	122
C	15	80	7	22
D	28	0	34	62
E	12	15	12	24
F	25	50	12	37

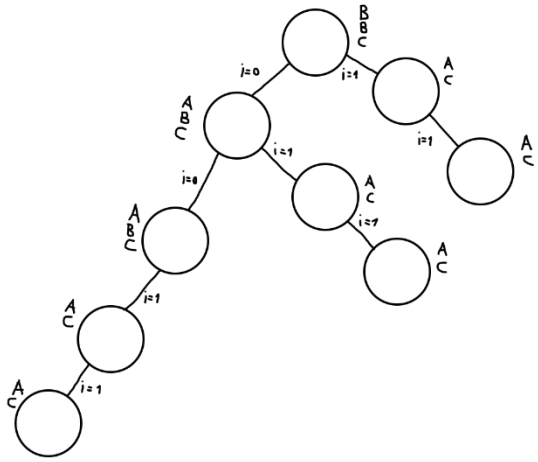


Доколку се исврши следнава програма:

```c
#include <stdio.h>
int main() {
    int i;
    for (i = 0; i < 2; ++) {
        int pid = fork();
        if (pid == 0) {
            fork();
            printf("A\n");

        }else {
            printf("B\n");
        }
    }//for
    printf("C\n");
    return 0;
}//main
```

Тогаш A ќе се испечати 8 пати, B ќе се испечати 4 пати, а C ќе се ипсечати 9 пати.



Даден е кодот:

```c
#include <stdio.h>
#include <unistd.h>
int main() {
    int a = 0;
    if (!fork()) {
        a = 1;
        if (!fork()) {
            printf("Вредноста на а е %d, мојот PID е %d, а PID-от на мојот родител е %d\n", a, pid(),ppid());

        }else {
            wait(NULL);
            a = 2;
            printf("Вредноста на а е %d, мојот PID е %d, а PID-от на мојот родител е %d\n", a, pid(),ppid());
            a = 3;
            execlp("pecati", "pecati", a, pid(), ppid(), NULL);
            a = 4;
            printf("Вредноста на а е %d, мојот PID е %d, а PID-от на мојот родител е %d\n", a, pid(),ppid());
        }

    }else {
        wait(NULL);
        a = 5;
        printf("Вредноста на а е %d, мојот PID е %d, а PID-от на мојот родител е %d\n", a, pid(), ppid());
    }
    return 0;
}
```

Познато е дека програмата се стартува од **shell** процес кој има **PID = 9**, а процесот кој започнува со извршување на истата има **PID = 10**. Сите процеси дополнително креирани од програмата имаат инкрементален PID.

Системскиот повид **execlp** го извршува кодот на функцијата која ја печати вредноста на **а, PID-то на процесот, и PID-то на неговиот родител процес** во ист формат како и printf().

Системскиот повик **wait(NULL)** го блокира процесот се додека не заврши барем едно од неговите деца процеси. Функцијата **pid()** го враќа PID-то на тековниот процес, додека **ppid()** го враќа PID-то на родителот процес.

За дадениот код и погоренаведените информации, одговорете на следните прашања. Доколку сметате дека во дадениот ред програмата не печати ништо внесете -1 во сите полиња од соодветниот ред.

Во првиот ред програмата ќе испечати: Вредноста на а е 1 мојот PID е 12, а PID-от на мојот родител е 11.
Во вториот ред програмата ќе испечати: Вредноста на а е 2 мојот PID е 11, а PID-от на мојот родител е 10.
Во третиот ред програмата ќе испечати: Вредноста на а е 3, мојот PID е 11, а PID-от на мојот родител е 10.
Во четвртиот ред програмата ќе испечати: Вредноста на а е 5, мојот PID е 10, а PID-от на мојот родител е 9.
Во петтиот ред програмата ќе испечати: Вредноста на а е -1, мојот PID е -1, а PID-от на мојот родител е -1.

Дополнително во моментот кога процесот го повикува системскиот повик wait(NULL) во **ред 9 од кодот**, одговорете:
Процесот со PID = 11 преминува од состојба Running во состојба Blocked. Додека од кога ќе заврши неговиот дете-процес преминува од состојба Blocked во состојба Ready.
По извршување на системскиот повик exit() (не сум сигурен за ова) од страна на процесот со PID = 10 извршувањето се враќа на процесот со **PID = 9**.

Времиња на пристигнување и извршување на процесите A, B, C, D, E се дадени во следната табела:

Процес	Пристигнување	Извршување
A	3	15 ms
B	2	8 ms
C	8	10 ms
D	13	17 ms
E	4	13 ms

Ако се користи **Round Robin алгоритмот за распределување со квантум 5ms**, тогаш времето на одзив, вкупното време во системот, и времето за чекање за процесите изнесува:

Процес	Одзив	Систем	Чекање
A	7 – 3 = 4	50 – 2 = 48	32
B	2 – 2 = 0	20 – 2 = 18	10
C	20 – 8 = 12	45 – 8 = 37	27
D	30 – 13 = 17	65 – 13 = 52	35
E	12 – 4 = 8	58 – 4 = 54	41

Во следната табела наведете ги временските интервали во кои се извршува секој од процесите согласно гантограмот за извршување на процесите:

Процес	Од	До
B	2	7
A	7	12
E	12	17
B	17	20
C	20	25
A	25	30
D	30	35
итн...		

Задачава е решена со queue – работи на ист принцип како таа погоре само треба да се пази кој процес доваѓа следен дека сите пристигнуваат во различни времиња.

Време	Извршување	Queue
2 – 7	B – 5/8	A, E, B
	3: Стигнува A	
	4: Стигнува E	
7 – 12	A – 5/15	E, B, C, A
	8: Стигнува C	
12 – 17	E – 5/13	B, C, A, D, E
	13: Стигнува D	
17 – 20	B – 8/8	C, A, D, E
20 – 25	C – 5/10	A, D, E, C
25 – 30	A – 10/15	D, E, C, A
30 – 35	D – 5/17	E, C, A, D
35 – 40	E – 10/13	C, A, D, E
40 – 45	C – 10/10	A, D, E
45 – 50	A – 15/15	D, E
50 – 55	D – 10/17	E, D
55 – 58	E – 13/13	D
58 – 63	D – 15/17	D
63 – 65	D – 17/17	

Времиња на пристигнување и извршување на процесите A, B, C, D, E се дадени во следната табела:

Процес	Пристигнување	Извршување
A	9	17 ms
B	5	14 ms
C	2	10 ms
D	1	11 ms
E	6	13 ms

Ако се користи **SRTN алгоритмот за распределување**, тогаш времето на одзив, вкупното време во системот и времето за чекање за процесите изнесува:

Процес	Одзив	Систем	Чекање
A	49 – 9 = 40	66 – 9 = 57	40
B	35 – 5 = 30	49 – 5 = 44	30
C	12 – 2 = 10	22 – 2 = 20	10
D	1 – 1 = 0	12 – 1 = 11	0
E	22 – 6 = 16	35 – 6 = 29	16

Во следната табела наведете ги временските интервали во кои се извршува секој од процесите согласно гантограмот за извршување на процесите:

Процес	Од	До
D	1	2
D	2	5
D	5	6
D	6	9
D	9	12
C	12	22
E	22	35
B	35	49
A	49	66

Пристигнување: D ➞ C ➞ B ➞ E ➞ A

Време	**Извршување**
1 – 2	D – 1/11
	2: Стигнува C
2 – 5	D – 4/11 (D == C, си продолжува D)
	5: Стигнува B
5 – 6	D – 5/11
	6: Стигнува E
6 – 9	D – 8/11
	9: Стигнува A
9 – 12	D – 11/11
12 – 22	C – 10/10
22 – 35	E – 13/13
35 – 49	B – 14/14
49 – 66	A – 17/17

Времиња на пристигнување и извршување на процесите A, B, C, D, E се дадени во следната табела:

Процес	Пристигнување	Извршување
A	1	13 ms
B	8	13 ms
C	8	13 ms
D	1	16 ms
E	12	14 ms

Ако се користи **Multilevel Feedback Queues** каде **Q0 користи Round Robin со квантум 2**, **Q1 користи Round Robin со квантум 4**, и **Q2 користи FCFS**, тогаш времето на одзив, вкупното време во системот, и времето за чекање за процесите изнесува:

Процес	Одзив	Систем	Чекање
A	0	37	24
B	1	47	34
C	3	54	41
D	2	47	31
E	1	58	44

Сите queues се non-preemptive.

Во овие задачи би требало да е кажано кои queues се preemptive а кој не. Ако нема јбг брат нема што да се прави.

Пристигнување: A + D ➞ B + C ➞ E

A ➞ D ➞ B ➞ C ➞ **E**

Q0 (RR со квантум 2)

Време	**Извршување**
1 – 3	A – 2/13 – се префрла во Q1
	1: Стигнува D
3 – 5	D – 2/16 – се префрла во Q1
5 – 9	си chilla братот, досадно му е
	8: Стигнува B
	8: Стигнува C
9 – 11	B – 2/13 – се префрла во Q1
11 – 13	C – 2/13 – се префрла во Q1
	12: Стигнува E
13 – 15	E – 2/14 – се префрла во Q1

Q1 (RR со квантум 4)

Време	**Извршување**
5 – 9	A – 6/13 – се префрла во Q2
11 – 15	Ништо не се прави. D чека да заврши Q0 со работа, а во меѓувреме се местат и B, C, E во овој queue.
15 – 19	D – 6/16 – се префрла во Q2
19 – 23	B – 6/13 – се префрла во Q2
23 – 27	C – 6/13 – се префрла во Q2
27 – 31	E – 6/14 – се префрла во Q2

Q2 (FSCF)

Време	**Извршување**
31 – 38	A – 13/13
38 – 48	D – 16/16
48 – 55	B – 13/13
55 – 62	C – 13/13
62 – 70	E – 14/13

Исто како предходната само Q3 користи SRTN.

Q0 – non-preemptive **Q1 – non-preemptive** **Q2 - preemptive**

Процес	Пристигнување	Извршување
A	10	12 ms
B	0	10 ms
C	8	12 ms
D	7	8 ms
E	9	17 ms

Процес	Одзив	Систем	Чекање
A	3	38	26
B	0	36	26
C	1	34	22
D	0	26	18
E	2	50	33

Пристигнување: B ➞ D ➞ C ➞ E ➞ A

Време	Q0 (RR со квантум 2)	Q1 (RR со квантум 4)	Q2 (SRTN)
0 – 2	B: 2/10 ➞ Q1		
2 – 6		B: 6/10 ➞ Q2	
6 – 7	7: Стигнува D		B: 7/10
7 – 9	D: 2/8 ➞ Q1 8: Стигнува C 9: Стигнува E		
9 – 11	C: 2/12 ➞ Q1 10: Стигнува A		
11 – 13	E: 2/17 ➞ Q1		
13 – 15	A: 2/12 ➞ Q1		
15 – 19		D: 6/8 ➞ Q2	
19 – 23		C: 6/12 ➞ Q2	
23 – 27		E: 6/17 ➞ Q2	
27 – 31		A: 6/12 ➞ Q2	
31 – 33			D: 8/8
33 – 36			B: 10/10
36 – 42			C: 12/12
42 – 48			A: 12/12
48 – 59			E: 17/17

Даден е кодот на сликата:

```c
int main() {
    int i, p;
    i = 1;

    p = fork();
    i++;
    p = fork();
    i--;
    if (p > 0) {
        i++;
        wait(NULL);
        printf("i=%d, pid=%d, ppid=%d\n", i, getpid(), getppid());

    } else {
        i--;
        printf("i=%d, pid=%d, ppid=%d\n", i, getpid(), getppid());
        exit();
    }

    i++;
    printf("i=%d, pid=%d, ppid=%d\n", i, getpid(), getppid());
}
```

Познато е дека програмата се стартува од shell процес кој има PID = 5, а процесот кој започнува со извршување на истата има PID = 7. Сите процеси дополнително креирани од програмата имаат инкрементален PID.

Системскиот повик wait(NULL) го блокира процесот се додека не заврши дете процесот. Функцијата getpid() го враќа PID-то на тековниот процес, додека getppid() го враќа PID-то на родителот процес.

За дадениот код и погоренаведените информации одговорете на прашањата:

Колку вкупно процеси (вклучувајќи го и процесот со PID = 7) ќе бидат стартувани? 4
Колку вкупно процеси ќе ја извршат wait(NULL) функцијата? 2
Колку вкупно процеси ќе ја извршат exit() функцијата? 2
Колку **различни финални вредности** на i ќе бидат испечатени од процесите? 2

Hey. You made it to the bottom. Good job!