Grau en Enginyeria Informàtica (GEI)

Sistemes Operatius

2025-2026 (1er Semestre)

Examen Segon Parcial Gener 2026

Solució

Jordi Mateo jordi.mateo@udl.cat

— Informació important —

Temps: 2 h

- Aquest examen avalua els coneixements teòrics i pràctics de l'assignatura.
- Només es permet l'ús d'una fulla d'apunts en format A4; preparada prèviament per l'estudiant. Altres materials de consulta no estan permesos.
- L'examen s'ha de respondre directament en el mateix document proporcionat i amb bolígraf (blau o negre); les respostes escrites amb llapis no seran vàlides.
- Qualsevol sospita de plagi o còpia comportarà sancions d'acord amb la normativa acadèmica.

Happy coding!!

— Omplir per l'estudiant —

Nom: .....

Cognoms: .....

DNI: .....

- Declaro que jo he entès les instruccions.

Signatura: .....

— Omplir pel professor —

Exercisis	1	2	3	4	5	
Punts	2	2	2	2	2	10
Correcció						

1: Planificació de processos — 2 Punts

Considereu la taula següent de processos, amb els seus temps d'arribada i d'execució associats:

Procés	Temps d'arribada	Temps d'execució a la CPU
A	0	2
B	1	6
C	4	1
D	7	4
E	8	3

Mostreu l'ordre de planificació d'aquests processos sota **3 polítiques**:

(a) *First Come First Serve (FCFS).* **[0.3 punt]**
(b) *Shortest Remaining Time First (SRTF).* **[0.3 punts]**
(c) *Round-Robin (RR)* amb *quantum* de temps = 1. **[0.4 punts]**

Assumiu que:

- El cost del canvi de context és 0.
- Els nous processos estan disponibles per a la planificació tan bon punt arriben.
- Els nous processos s'afegeixen a l'**inici de la cua**, excepte en **FCFS**, on s'afegeixen al **final de la cua**.

Ompliu la taula amb les lletres (**A, B, C, D, E**), indicant quin procés s'executa en cada instant de temps segons l'ordre de planificació de cada política.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FCFS																
SRTF																
RR																

Solution:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FCFS	A	A	B	B	B	B	B	B	C	D	D	D	D	E	E	E
SRTF	A	A	B	B	C	B	B	B	B	E	E	E	D	D	D	D
RR	A	B	A	B	C	B	B	D	E	B	D	E	B	D	E	D

Suposeu que estem implementant **EEVDF (Earliest Eligible Virtual Deadline First)** com a algorisme de planificació. El nostre sistema té les restriccions següents.

- La longitud de cada petició es mesura en temps físic real, expressat en *ticks*.
- El temps de CPU s'assigna en increments d'un quantum fix de $\mathbf{q = 1}$ **tick**.
- En cas d'empat en la planificació, es prioritza el **procés amb l'identificador més alt**.
- El **procés 1** esdevé actiu a $\mathbf{t = 0}$, amb longitud de petició $r_1 = 3$, i pes $w_1 = 1$.
- El **procés 2** esdevé actiu a $\mathbf{t = 1}$, amb longitud de petició $r_2 = 1$, i pes $w_2 = 1$.

- El **procés 3** esdevé actiu a $\mathbf{t = 2}$, amb longitud de petició $r_3 = 3$, i pes $w_3 = 3$.

Assumiu que cada procés consumeix exactament el temps de servei sol · licitat per a cada petició, i que abandona el sistema després de completar **3 peticions** de la longitud especificada.

(a) Indica com evoluciona la **taxa de temps virtual** $\frac{dV}{dt}$ per a $t \in \{0, 1, 2\}$. **[0.3 punts]**

- de t_0 a $t_1 \Rightarrow dV/d_0 =$ _____ *[0.1 punt]*
- de t_1 a $t_2 \Rightarrow dV/d_1 =$ _____ *[0.1 punt]*
- de t_2 a $t_3 \Rightarrow dV/d_2 =$ _____ *[0.1 punt]*

(b) Completeu la taula següent. La columna *Procés Planificat* fa referència al procés que s'està executant en aquell instant. **[0.7 punt]**

Temps real (ticks)	**Temps virtual**	**T1** v_e	**T1** D	**T2** v_e	**T2** D	**T3** v_e	**T3** D	**Procés planificat**
0								
1								
2								
3								
4								
5								
6								

Solution:

- de t_0 a $t_1 \Rightarrow dV/d_0 = \frac{1}{\sum w_i} = \frac{1}{1} = 1$
- de t_1 a $t_2 \Rightarrow dV/d_1 = \frac{1}{\sum w_i} = \frac{1}{1+1} = 0.5$
- de t_2 a $t_3 \Rightarrow dV/d_2 = \frac{1}{\sum w_i} = \frac{1}{1+1+3} = 0.2$

Temps real (ticks)	**Temps virtual**	**T1** v_e	**T1** D	**T2** v_e	**T2** D	**T3** v_e	**T3** D	**Procés planificat**
0	0	0	3	-	-	-	-	T1
1	1	0	3	1	2	-	-	T2
2	1.5	0	3	2	3	1.5	2.5	T3
3	1.7	0	3	2	3	1.5	2.5	T3
4	1.9	0	3	2	3	1.5	2.5	T3
5	2.1	0	3	2	3	2.5	3.5	T2
6	2.3	0	3	3	4	2.5	3.5	T1

Solution:
Explicació pas a pas:

- **t=0**:
  - $V(t) = 0$, $v_{e_1} = 0$.
  - $D_1 = 0 + 3/1 = 3$.
  - S'executa P1.
- **t=1**:
  - $V(t) = 1$
  - *P1 actiu* el seu $D_1 = 3$, el temps virtual d'arribada no ha canviat ni tampoc r ni w.
  - P2, $v_{e_2} = 1$ ja que acaba d'arribar.
  - $D_2 = 1 + 1/1 = 2$.
  - Com a $D_2 < D_1$, P2 té més prioritat.
  - S'executa P2.
- **t=2**:
  - $V(t) = 1.5$, P1 actiu el seu $D_1 = 3$, ja que no ha canviat cap valor.
  - P2 ha completat la seva petició surt i torna a entrar al sistema.
  - T3, $v_{e_3} = 1.5$ ja que acaba d'arribar i per tant el seu Deadline $D_3 = 1.5 + 3/3 = 2.5$.
  - P2 torna a entrar al sistema i $v_{e_2} = max(1.5, 2) = 2$, per tant el seu Deadline $D_2 = 2 + 1/1 = 3$. Com a $D_3 < D_2 < D_1$, P3 té més prioritat.
- **t=3,t=4**:
  - $V(t) = 1.7, 1.9$ no hi ha canvis en els valors del Deadline ni del temps virtual d'arribada.
  - S'executa P3.
- **t=5**:
  - $V(t) = 2.1$, P3 ha completat la seva petició i surt del sistema.
  - P1,P2 segueixen amb els mateixos valors de Deadline i temps virtual d'arribada.
  - P3 torna a entrar al sistema i $v_{e_3} = max(2.1, 2.5) = 2.5$, per tant el seu Deadline $D_3 = 2.5 + 3/3 = 3.5$.
  - D(1)=3, D(2)=3, D(3)=3.5. Com a $D_1 = D_2 < D_3$, P2 té més prioritat ja que té l'identificador més alt.
- **t=6**:
  - $V(t) = 2.3$, P2 ha completat la seva petició i surt del sistema.
  - P1,P3 segueixen amb els mateixos valors de Deadline i temps virtual d'arribada.
  - P2 torna a entrar al sistema i $v_{e_2} = max(2.3, 3) = 3$, per tant el seu Deadline $D_2 = 3 + 1/1 = 4$.
  - D(1)=3, D(3)=3.5, D(2)=4. Com a $D_1 < D_3 < D_2$, P1 té més prioritat.

Recordeu que:

- $D_i = v_{e_i} + \frac{r_i}{w_i}$
- $v_{e_i} = max(V(t), D_i)$
- $S_i(t_x, t_y) = \int_{t_x}^{t_y} f_i(t)dt$
- $L_i(t) = S_i(t_0, t) - s_i(t_0, t)$

2: Interbloquejos — 2 Punts

Suposeu que tenim els recursos següents: **A, B, C** i els processos **T1, T2, T3, T4**. El nombre total de cada recurs és conegut.

Recurs	A	B	C
Total	12	9	12

A més, assumiu que els processos tenen les **assignacions actuals** i els **requeriments màxims** següents:

	Assignació actual				**Requeriments màxims**		
	A	B	C		A	B	C
T1	2	1	3	T1	4	9	4
T2	1	2	3	T2	5	3	3
T3	5	4	3	T3	6	4	3
T4	2	1	2	T4	4	8	2

El sistema es troba en un **estat segur**?

- Si la resposta és **sí**, mostreu una **seqüència d'execució no bloquejant** dels fils.
- En cas contrari, proporcioneu una **demostració que el sistema es troba en un estat insegur**.

(a) Mostreu tot el raonament i **justifiqueu cada pas** de la vostra resposta. *Heu de mostrar l'estat de totes les matrius i vectors després de cada pas.* [**1 punt**]

Solution:

	A	B	C
TOTALS	12	9	12

Assignats	A	B	C
T1	2	1	3
T2	1	2	3
T3	5	4	3
T4	2	1	2
Disponibles	2	1	1

TOTALS - Assignats

Màxims	A	B	C
T1	4	9	4
T2	5	3	3
T3	6	4	3
T4	4	8	2

Màxims - Assignats

Necessitat	A	B	C
T1	2	8	1
T2	4	1	0
T3	1	0	0
T4	2	7	0

Amb Disponibles (2,1,1), l'única necessitat que es pot satisfer és
Un cop satisfeta T3, els disponibles són (7,5,4).
Amb Disponibles(7,5,4) , l'única necessitat que es pot satisfer és
Un cop satisfeta T2, els disponibles són (8,7,7).
Amb Disponibles(8,7,7) , l'única necessitat que es pot satisfer és
Un cop satisfeta T4, els disponibles són (10,8,9).
Amb Disponibles(10,8,9) , es pot satisfer a T1(2,8,1).
Un cop satisfeta T1, els disponibles són (12,9,12).

T3-T2-T4-T1 és una seqüencia segura.

Considereu un centre d'entrenament on diversos entrenadors Pokémon comparteixen recursos. Tots els entrenadors tenen el mateix nombre de Pokémons, `numPokemons`. Per completar una sessió d'entrenament, cada entrenador necessita exactament:

- `numPokemons / 2` ⇒ PokeBalls i `numPokemons / 2` ⇒ Pocions.

El centre manté:

- el nombre de recursos disponibles.
- el nombre de recursos actualment assignats a cada entrenador.

- Per evitar estats insegurs, el centre utilitza una versió de l'algorisme del banquer.

(b) Indiqueu quantes vegades cal executar l'algorisme del banquer per determinar si una petició d'un entrenador és segura o no. *Justifiqueu la vostra resposta.* [**0.25 punts**]

Solution:
Si l'algorisme del banquer pot determinar que un sol entrenador pot completar la sessió i alliberar els seus recursos, aleshores aquests recursos seran suficients perquè qualsevol altre entrenador també pugui completar. Per això, n'hi ha prou amb una sola passada.

Implementeu la funció `TrainerCheck()`. Aquesta funció ha de retornar **true** si l'entrenador amb identificador *TrainerID* pot rebre `numBalls` Poké Balls i `numPotions` Pocions sense portar el sistema a un estat insegur, i **false** en cas contrari.

Podeu assumir que:

- Un entrenador mai no sol · licitarà més recursos dels que necessita.
- La funció es crida amb el mecanisme de sincronització corresponent ja adquirit.
- L'estat del centre no pot quedar modificat de manera permanent.

(c) Completeu el codi següent omplint els espais en blanc. Només es permet una expressió per línia (*no s'admeten expressions amb comes ni punts i coma addicionals*). [**0.75 punt**]

```
1  typedef struct {
2      int balls; int potions;
3  } Trainer_t;
4
5  typedef struct {
6      int numTrainers;  int numPokemons; int idleBalls; int idlePotions;
7      Trainer_t trainers[MAX_TRAINERS];
8  } Center_t;
9
10 bool TrainerCheck(Center_t *myCenter, int TrainerID, int numBalls, int numPotions){
11     if (numBalls > myCenter->idleBalls || numPotions > myCenter->idlePotions)
12         return false;
13
14     if (_______________________________(1) && _________________________________(2) {
15         return ___________________(3);
16     }
17
18     for (__________(4); ________________________(5); __________(6)) {
19         if (_______________________________(7) && ____________________________(8)) {
20             return ___________________(9);
21         }
22     }
23     return ____________________(10);}
```

Solution:

```
1  bool TrainerCheck(Center_t *myCenter, int TrainerID, int numBalls, int
       numPotions) {
2
3      // Tenim prou recursos disponibles?
4      if (numBalls > myCenter->idleBalls || numPotions > myCenter->idlePotions)
5          return false;
6
7      // Pot l'entrenador sol·licitant completar després de la petició?
8      if (myCenter->numPokemons / 2 - myCenter->trainers[TrainerID].balls <=
           myCenter->idleBalls &&
9          myCenter->numPokemons / 2 - myCenter->trainers[TrainerID].potions <=
               myCenter->idlePotions) {
10         return true;
11     }
12
13     // Pot algun altre entrenador completar després de la petició?
14     for (int i = 0; i < myCenter->numTrainers; i++) {
15         if (myCenter->numPokemons / 2 - myCenter->trainers[i].balls <= myCenter
               ->idleBalls - numBalls &&
16             myCenter->numPokemons / 2 - myCenter->trainers[i].potions <=
                   myCenter->idlePotions - numPotions) {
17             return true;
18         }
19     }
20
21     return false;
22 }
```

3: Gestió de Memòria — 2 Punts

Disposem d'un sistema de gestió de la memòria del tipus **segmentació paginada** amb les característiques següents:

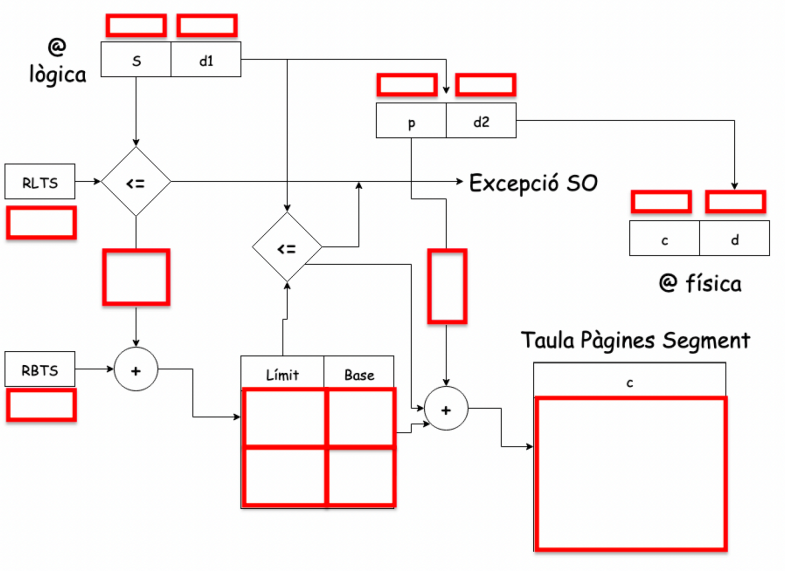
- La mida d'una pàgina (cel · la) és de **64 Bytes**.
- La mida d'una paraula és de **1 Byte**.
- Cada procés consta de **com a màxim 4 segments**.
- Cada segment té una mida màxima de **256 Bytes**.
- El contingut ocupat pel sistema operatiu no es té en compte.
- La **cel · la 0** conté tota la informació relacionada amb el sistema de memòria (taules de pàgines, segments, etc.).
- Memòria principal formada per **32 cel · les**.

A la memòria principal hi ha carregats dos processos, P_1 i P_2. El contingut de la memòria principal (en pàgines) és el següent:

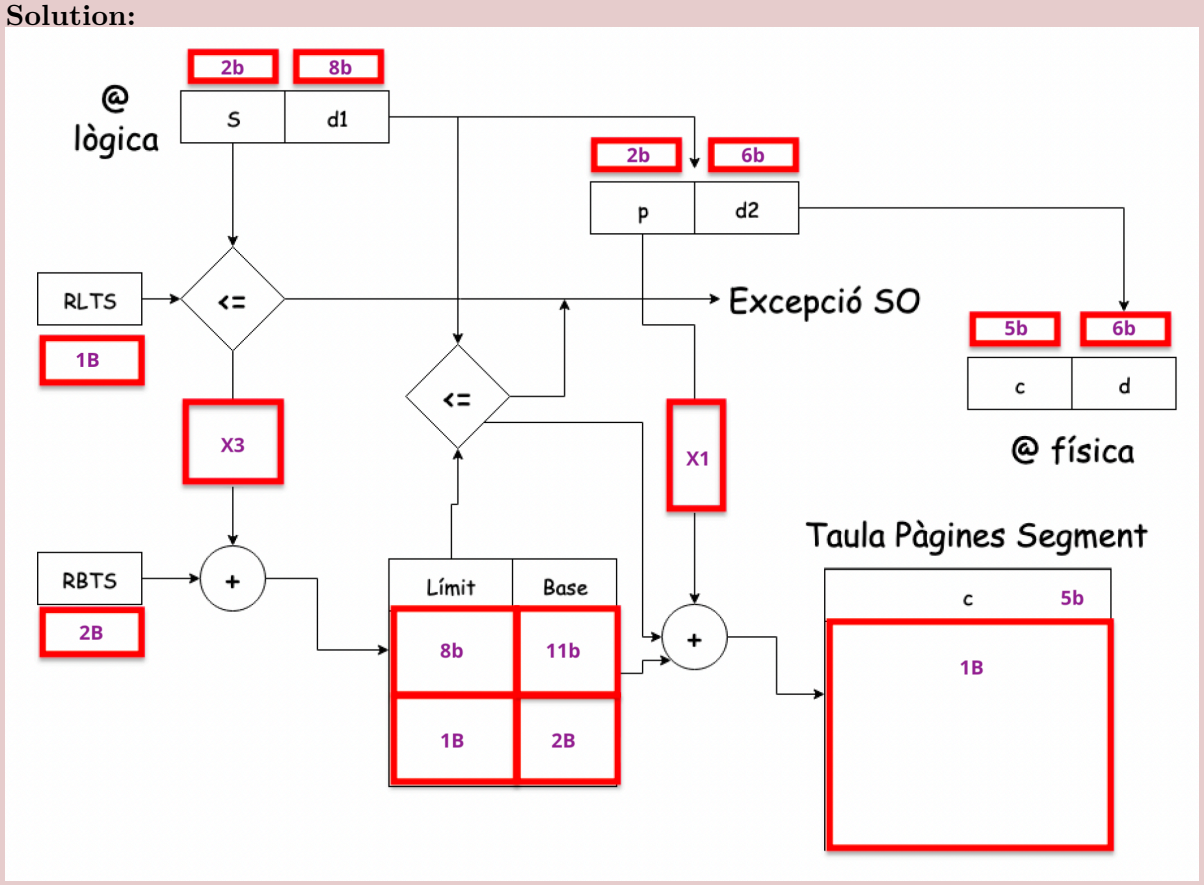
0	Gestió Mem	8		16	x	24	codi(p1,p2)
1	codi(p1,p2)	9	dades(p1)	17	dades(p2)	25	
2	codi(p1)	10	dades(p1)	18		26	x
3	codi(p2)	11	x	19	x	27	
4		12		20		28	codi(p1,p2)
5	x	13	codi(p2)	21	dades(p1)	29	
6	dades(p1)	14		22	x	30	
7	dades(p1)	15	x	23	dades(p2)	31	x

Les pàgines marcades com *codi(p1,p2)* contenen codi compartit entre els processos P_1 i P_2. Les pàgines de dades són privades de cada procés. Les x indiquen cel · les ocupades.

(a) Doneu l'**esquema de la traducció d'adreces lògiques a físiques**. **[0.5 punts]**



Solution:



(b) Doneu **el rang d'adreçament lògic** per al procés P_1. **[0.25 punt]**.

Solution:

s	p	d	
00	00	xxx xxx	codi(p1,p2)
00	01	xxx xxx	
00	10	xxx xxx	
01	00	xxx xxx	codi(p1)
10	00	xxx xxx	dades(p1)
10	01	xxx xxx	
10	10	xxx xxx	
10	11	xxx xxx	
11	00	xxx xxx	

(c) Seria possible carregar un nou procés P_3 amb les característiques següents: Codi (256 Bytes), Dades (512 Bytes) i Pila (64 Bytes)? Assumeix el cas que 256 Bytes de dades són compartides amb les dades de les cel · les x i el cas en que totes les dades són privades. *Justifiqueu la vostra resposta.* **[0.4 punt]**.

Solution:
Si les dades són privades, no hi ha prou memòria per carregar el procés P_3. Ja que necessitem 13 cel · les lliures (4 de codi, 8 de dades i 1 de pila), però només en tenim 10.
Si 256 Bytes de dades són compartides amb el procés X, llavors només necessitem 9 cel · les lliures (4 de codi, 4 de dades i 1 de pila). En aquest cas, sí que hi ha prou memòria per carregar el procés P_3. Una possible assignació seria la següent:

(d) Com quedaria el contingut de la **cel · la 0** si el procés **P1** finalitza l'execució i només el procés **P2** roman actiu? **[0.6 punt]**

Solution:

Cel·la 0				
2^6 + 0	RLTS	2	1B	
2^6 + 1	RBTS	2^6 + 3	2B	
2^6 + 3	192B (L)	2^6 + 12 (B)	3B	Segment 1
2^6 + 6	128B (L)	2^6 + 15 (B)	3B	Segment 2
2^6 + 9	128B (L)	2^6 + 17 (B)	3B	Segment 3
2^6 + 12	1		1B	Taula Pàgines codi(p1,p2)
2^6 + 13	24		1B	
2^6 + 14	28		1B	
2^6 + 15	3		1B	Taula Pàgines (codi p2)
2^6 + 16	13		1B	
2^6 + 17	17		1B	Taula Pàgines dades(p2)
2^6 + 18	23		1B	

(d) Pot existir en aquest sistema una adreça lògica vàlida que provoqui una excepció del sistema operatiu? En cas afirmatiu, doneu-ne un exemple per al procés **P2**; en cas contrari, *justifiqueu la resposta.* **[0.25 punt]**

Solution:
Sí. Una adreça lògica pot ser formalment vàlida però provocar una excepció si la pàgina corresponent no està present en memòria. Per exemple, per al procés P2, 01 10 001000, el segment 1, únicament té la pàgina 00, si intentem accedir a la pàgina 01 del segment 1 es producirà una excepció.

4: Memòria Virtual — 2 Punts

Disposem d'un sistema de gestió del tipus segmentació paginada i de memòria virtual amb pàginació sota demanda amb les característiques següents:

- Assignació local de cel · les amb 6 cel · les per procés.
- Algorisme de reemplaçament de pàgines **LRU (Least Recently Used)**.
- La mida d'una pàgina i una cel · la és de 16 Bytes.
- La mida d'una paraula és de 1 Byte.
- La mida d'un enter és de 1 Byte.

Donat el programa següent que s'executa en aquest sistema:

@log	Codi
002Bh	Reg0 ← 0
002Ch	Comparar Reg0,40
002Dh	Branch_equal to @0035h
002Eh	Load Reg1 ← Mem(@0600h + Reg0)
002Fh	Load Reg2 ← Mem(@0700h + Reg0+1)
0030h	Load Reg3 ← Mem(@0800h + (Reg0%16))
0031h	Reg4 ← Reg1+Reg2+Reg3
0032h	Store Mem(@0500h + Reg0) ← Reg4
0033h	Reg0 ← Reg0+1
0034h	Branch to @002Ch

(a) Indiqueu el **nombre total de fallades de pàgina** que es produeixen durant l'execució del programa anterior, suposant que la memòria principal està buida al començament de l'execució. Feu l'esquema de com es van omplint les cel · les de memòria principal al llarg de l'execució. **[0.5 punts]**

Solution:

- Fallada 1: Pagina 2 corresponent al codi
- Fallada 2: Pagina 60 corresponent a dades de l'adreça 0600h
- Fallada 3: Pagina 70 corresponent a dades de l'adreça 0700h
- Fallada 4: Pagina 3 corresponent al codi
- Fallada 5: Pagina 80 corresponent a dades de l'adreça 0800h
- Fallada 6: Pagina 5 corresponent a dades de l'adreça 0500h

Les cel · les estan plenes en aquest moment.

- Quan Reg0 = 15 -> Fallada 7 -> Pàgina 71 es necessaria es substitueix per la pàgina 70.
- Quan Reg0 = 16 -> Fallada 8 i 9 -> Pàgines 61,51 es necessiten, es substitueixen per les pàgines 60 i 50.
- Quan Reg0 = 31 -> Fallada 10 -> Pàgina 72, es substitueix per la pàgina 71.
- Quan Reg0 = 32 -> Fallada 11 i 12 -> Pàgines 62,52 es necessiten, es substitueixen per les pàgines 61 i 51.

En total hi ha 12 fallades de pàgina.

(b) Quin seria el nombre de fallada amb 5 cel · les assignades per procés? Justifiqueu la vostra resposta. **[0.5 punts]**

Solution:

Amb 5 cel · les per procés, dues queden ocupades permanentment per les pàgines de codi, deixant només tres cel · les per a dades. El bucle necessita accedir a quatre pàgines de dades diferents en cada iteració, de manera que el conjunt de treball no cap en memòria. Amb l'algorisme LRU, cap pàgina de dades es pot mantenir entre iteracions i totes es tornen a carregar a cada volta del bucle. Després de les 6 fallades inicials, cada iteració produeix 4 fallades de pàgina. Com que hi ha 39 iteracions $6 + 39 \cdot 4 = 162$.

(c) Justifiqueu quin dels algorismes de reemplaçament, **FIFO** o **LRU**, és més eficient en aquest cas, i expliqueu-ne el motiu. **[0.5 punts]**

Solution:
L'algorisme LRU és més eficient en aquest cas perquè sempre reemplaça la pàgina que no s'ha utilitzat durant el període de temps més llarg. En aquest programa, les pàgines de codi es reutilitzen freqüentment, mentre que les pàgines de dades es canvien constantment. LRU manté les pàgines de codi a la memòria, reduint així el nombre de fallades de pàgina. En canvi, FIFO podria reemplaçar pàgines de codi que encara són necessàries, augmentant el nombre de fallades de pàgina.

(d) Quina optimització es podria aplicar per reduir el nombre de fallades de pàgina amb l'algorisme LRU sense augmentar el nombre de cel · les assignades (5 cel · les)? *Justifiqueu la resposta.* **[0.5 punts]**

Solution:
Es podria evitar el reemplaçament de la pàgina corresponent a l'adreça @0800h (pàgina 80), fixant-la en memòria principal. Aquesta pàgina s'utilitza en totes les iteracions del bucle, ja que conté les dades accedides mitjançant l'expressió Reg0 mod 16. Mantenir aquesta pàgina resident en memòria garanteix que no sigui expulsada per l'algorisme LRU, evitant així una fallada de pàgina en cada iteració del bucle.

5: Shell Scripting — 2 Punts

Assumeix que estem desenvolupant la funció `update_life()` en Bash:

- Disposem d'un array global `pokemons` que emmagatzema la vida actual dels Pokémons.
- Volem que la funció `update_life()` resti el dany a la vida del Pokémon defensor i asseguri que la vida no sigui negativa. Al final s'ha d'actualitzar l'array `pokemons` amb la nova vida.
- L'array `pokemons` ja està inicialitzat i els identificadors dels Pokémons són vàlids.

(a) Completeu el codi següent omplint els espais en blanc. **[0.8 punts]**

```
1  declare -a pokemons
2  update_life() {
3      pokemon_attacker_id=$1
4      pokemon_defender_id=$2
5      damage=$3
6      current_life=_____________________________________________(1)        # 0.2pts
7      new_life=_________________________________________________(2)        # 0.2pts
8      if _______________________________________________________(3); then # 0.2pts
9          new_life=0
10     fi
11     ___________________________________________________________(4)       # 0.2pts
12 }
```

Solution:

```
update_life() {
    pokemon_attacker_id=$1
    pokemon_defender_id=$2
    damage=$3
    current_life=${pokemons[$pokemon_defender_id]}
    new_life=$((current_life - damage))
    if [ $new_life -lt 0 ]; then
        new_life=0
    fi
    pokemons[$pokemon_defender_id]=$new_life
}
```

Ara disposem del següent fitxer:

```
01,100
02,070
03,200
```

(b) Indiqueu un codi Bash que permeti crear aquest fitxer utilitzant `cat` amb redireccions d'entrada i soritda. El fitxer resultant s'anomenarà `pokemons.txt`. **[0.4 punt]**

Solution:

```
cat > pokemons.txt << EOL
01,100
02,070
03,200
EOL
```

(c) Completeu el codi següent perquè [**0.8 punts**]:

- Llegeixi el fitxer `pokemons.txt` línia a línia.
- Mostri el PID del shell actual i el de l'últim procés en segon pla.

```
total_life=0
count=0
_____________________________________________________(1)      # 0.2 pts
while read id life;  do
    ...
_____________________________________________________(2)      # 0.2 pts
sleep 1 &
echo "PID shell: _____________________________________"(3)     # 0.2 pts
echo "PID background: ________________________________"(4)     # 0.2 pts
```

Solution:

```
total_life=0
count=0

IFS=','

while read id life; do
...
done < pokemons.txt

sleep 1 &

echo "PID shell: $$"
echo "PID background: $!"
```