Vaishak Balachandra.

021)

HW #2 of OPERATING SYSTEMS

Spring 2025

Name: VAISHAK BALACHANDRA (021) (0037831852)

1. Using Banker's Algorithm for deadlock avoidance in the following example:

5 processes P_0 through P_4;

3 resource types:

A (10 instances), B (5instances), and C (7 instances)

Snapshot at time T_0:

	Allocation	*Max*	*Available*
	A B C	*A B C*	*A B C*
P_0	0 1 0	7 5 3	3 3 2
P_1	2 0 0	3 2 2	
P_2	3 0 2	9 0 2	
P_3	2 1 1	2 2 2	
P_4	0 0 2	4 3 3	

1.1) Exam at time T_0 whether the system is in a safe state or not.

To examine, whether system is in a safe state or not, we need first Need metrix :

Need = Max - Allocation.

	Allocation			Max			Need			Available		
	A	B	C	A	B	C	A	B	C	A	B	C
P_0	0	1	0	7	5	3	7	4	3	3	3	2
P_1	2	0	0	3	2	2	1	2	2			
P_2	3	0	2	9	0	2	6	0	0			
P_3	2	1	1	2	2	2	0	1	1			
P_4	0	0	2	4	3	3	4	3	1			

Available resources,

A = 3
B = 3
C = 2

∴

Step 1 : Initialize : work = (3 3 2)

finish [i] = False [i = 0, 1, 2, 3, 4]

Step 2 & 3 : Check Request ≤ Need

Request ≤ Available

P_0 : $(7\ 4\ 3) \not\leq (3\ 3\ 2)$ ⇒ Need to wait!

finish[0] = false.

P_1 : $(1\ 2\ 2) \leq (3\ 3\ 2)$ ⇒ Update work

work = $(3\ 3\ 2) + (2\ 0\ 0)$ ⇒ $(5\ 3\ 2)$ P_1 ✓

finish[1] = true

P_2 : $(6\ 0\ 0) \not\leq (5\ 3\ 2)$ ⇒ Need to wait!

finish[2] = false

P_3 : $(0, 1, 1) \leq (5\ 3\ 2)$ ⇒ update work

work = $(5\ 3\ 2) + (2\ 1\ 1)$ ⇒ $(7\ 4\ 3)$ P_3 ✓

finish[3] = true

P_4 : $(4\ 3\ 1) \leq (7\ 4\ 3)$ ⇒ update work

work = $(7\ 4\ 3) + (0\ 0\ 2)$ ⇒ $(7\ 4\ 5)$ P_4 ✓

finish[4] = true

P_0 : $(7, 4, 3) \leq (7\ 4\ 5)$ ⇒ Update work

work = $(7\ 4\ 5) + (0\ 1\ 0)$ ⇒ $(7\ 5\ 5)$ P_0 ✓

finish[0] = true

P_2 : $(6, 0, 0) \leq (7\ 5\ 5)$ ⇒ update work

work = $(7\ 5\ 5) + (3\ 0\ 2)$ ⇒ $(10, 5, 7)$ P_2 ✓

finish[2] = true

System Sequence : $[\ P_1 \longrightarrow P_3 \longrightarrow P_4 \longrightarrow P_0 \longrightarrow P_2\ ]$

∴ The system is in Safe state !!

1.2) Can request for (1,0,2) by P_1 be granted?

To check, if P_1's request can be granted → it need to
i) check if request exceeds P_1's Need or not.
ii) check if there are enough Available resources or not.
iii) check if granting request leaves system in safe state. or not.

Here, Request = (1, 0, 2) | Available = (3, 3, 2)
for P_1 :- $Need_{P_1}$ = (1, 2, 2)

i) Here $(1,0,2) \leq (1,2,2)$ i.e Request $\leq Need_{P_1}$
Thus request is less than or equal to Need of P_1 ⇒ Good ✓

ii) Here $(1,0,2) \leq (3,3,2)$ i.e Request $\leq$ Available.
Thus request is less than or equal to Available at that time too.
⇒ Good ✓

iii) Update Data Structure

new_Available = Available − Request = (3 3 2) − (1 0 2)

new_available = (2 3 0)

$new\_Allocation_{P_1}$ = $Allocation_{P_1}$ + Request = (2 0 0) + (1 0 2)

$new\_allocation_{P_1}$ = (3 0 2)

$new\_Need_{P_1}$ = $Need_{P_1}$ − Request = (1 2 2) − (1 0 2)

$New\_Need_{P_1}$ = (0, 2, 0)

Initialize : work = new_available = (2 3 0)
finish[i] = False, i = 0, 1, 2, 3, 4.

Repeat : check request ≤ Need
request ≤ Available.

P_0 : $(7,4,3) \not\leq (2,3,0)$ ⇒ Need to wait !!
finish[0] = false.

P_1 : $(0,2,0) \leq (2,3,0)$ ⇒ update work
work = $(2,3,0) + (3,0,2)$ ⇒ $(5,3,2)$
finish[1] = true. P_1 ✓

P_2 : $(6,0,0) \not\leq (5,3,2)$ ⇒ Need to wait !!
finish[2] = false

P_3 : $(0,1,1) \leq (5,3,2)$ ⇒ update work
work = $(5,3,2) + (2,1,1)$ ⇒ $(7,4,3)$
finish[3] = true. P_3 ✓

P_4 : $(4,3,1) \leq (7,4,3)$ ⇒ update work
work = $(7,4,3) + (0,0,2)$ ⇒ $(7,4,5)$
finish[4] = true. P_4 ✓

P_0 : $(7,4,3) \leq (7,4,5)$ ⇒ update work
work = $(7,4,5) + (0,1,0)$ ⇒ $(7,5,5)$
finish[0] = true. P_0 ✓

P_2 : $(6,0,0) \leq (7,5,5)$ ⇒ update work
work = $(7,5,5) + (3,0,2)$ ⇒ $(10,5,7)$
finish[2] = true. P_2 ✓

System Sequence : [$P_1 \longrightarrow P_3 \longrightarrow P_4 \longrightarrow P_0 \longrightarrow P_2$]

Thus still system is in safe state !!

⇒ Thus satisfies all 3 cases.

∴ Request of $(1,0,2)$ for P_1 can be granted !!

2. Reference String:

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1 9 8 3

Using diagram to show the step-by-step page replacement processing of the following, indicate each page fault and count the total page fault numbers:

1) Frame number = 3; Using a queue and a pointer to implement the FIFO algorithm.

pages → / frames ↓	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1	9	8	3
f1 →	7	7	7	2	2	2 → 2	4	4	4 → 4	0	0	0	0	0 → 0	0 → 0	0	1	7	7	7 → 7	9	9	9
f2		0	0	0 → 0	0 → 0	3	3	3 → 3	2	2	2 → 2	2 → 2	2 → 2	1	1	1	1	1 → 1	0	0	0 → 0	8	8
f3			1	1	1	1 → 1	0	0	0 → 0	3	3	3	3	3 → 3	2	2	2	2	2 → 2	1	1	1 → 1	3
Hit(✓)/ miss(X)	X	X	X	X	✓	X	X	X	X	X	X	✓	✓	X	X	✓	✓	X	X	X	X	X	X

Here,

X → Page Fault/ Miss

✓ → Page Hit

Faults / Miss = 18 ⇒ Miss rate / ratio = 18/23 = 0.7826 //

Hits = 5 ⇒ Hit rate / ratio = 5/23 = 0.21739 //

Total = 23

[* Used pointer (→) to indicate next Hit.]

2) Frame number = 5; Using a stack to implement the LRU algorithm.

Pages → / frames ↓	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1	9	8	3
f1	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1	9	8	3
f2		7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1	9	8
f3			7	0	1	2	2	3	0	4	2	2	0	3	3	1	2	0	1	7	0	1	9
f4				7	7	1	1	2	3	0	4	4	4	0	0	3	3	2	2	2	7	0	1
f5						7	7	1	1	1	1	1	1	4	4	4	4	3	3	3	2	7	0
✓/X	X	X	X	X	✓	X	✓	X	✓	✓	✓	✓	✓	✓	✓	✓	✓	X	✓	✓	X	X	X

→ Recently Accessed

Acting as a Stack

→ Least recently accessed.

✓ → Hit

X → Miss/Fault

Faults/Miss = 10 ⇒ Miss rate/ratio = 10/23 = 0.4347 //

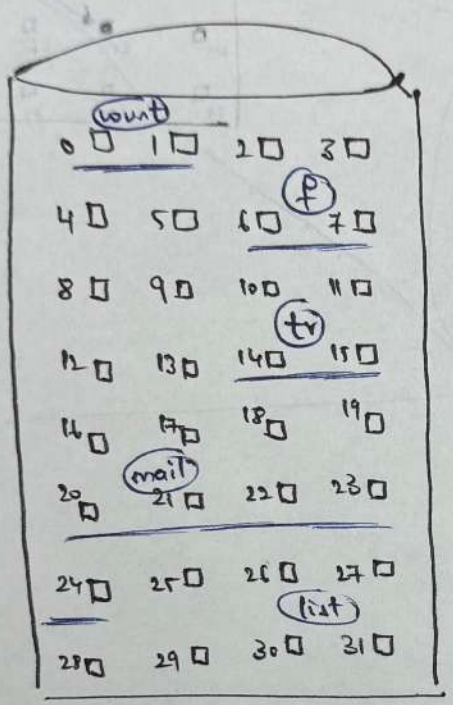
Hits = 13 ⇒ Hit rate/ratio = 13/23 = 0.56521 //

Total = 23

3. Using diagrams to show directory tables and blocks allocated in disk for the three allocation methods for files.

⇒ Contigueous Allocation : mapping from logical to physical.

Directory :

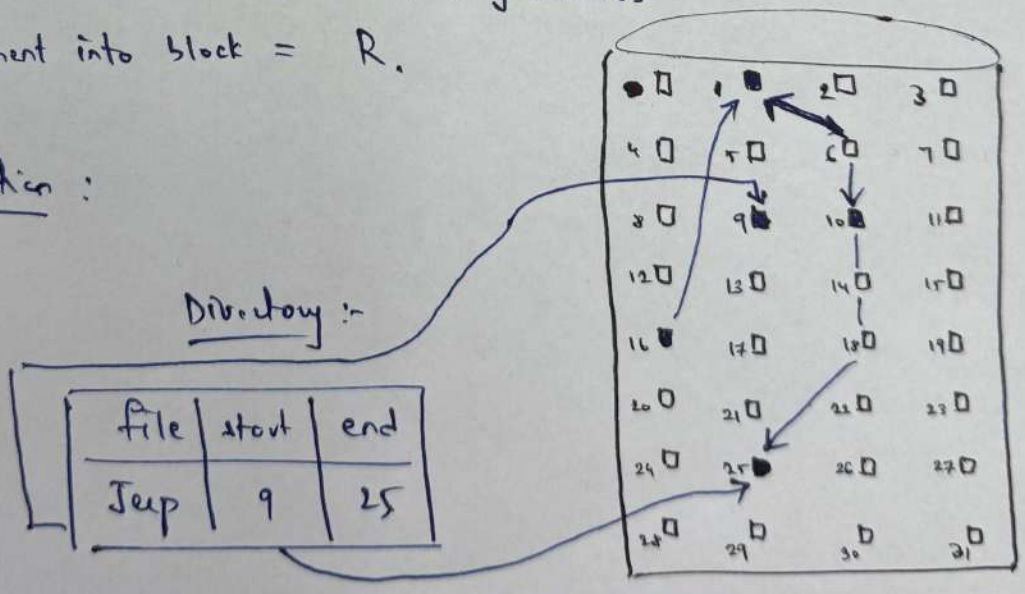


File	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

512 → Q, R

- Block to be accessed = Q + starting Address
- Displacement into block = R.

⇒ Linked Allocation :



Directory :-

file	start	end
Jeep	9	25

⇒ Indexed Allocation :

Directory :-

File	Index - Block
Jay	19