L28- TUTORIAL ON MEMORY MANAGEMENT TECHNIQUES-1

Dynamic Memory Allocation

Q1: Consider 4 holes H1, H2, H3, H4 in memory as shown below along with active process P1 to P5. A new process P6 with a size 5K is arriving. Which hole will be assigned for P6?

Find out the new holes and its size after the termination of P3 and P1 (active process are P2, P4, P5 and P6)

Solve the above two questions as per (a) First Fit (b) Best Fit (c) Worst fit memory allocation strategy.

P1	H1	P2	P3	H2	P4	H3	P5	H4
3K	8K	4K	5K	6K	20K	12K	9K	3K

Dynamic Memory Allocation

Case A: First Fit memory allocation strategy.

Which hole will be assigned for P6 with a size 5K?

Find out the new holes and its size after the termination of P3 and P1 (active process are P2, P4, P5 and P6)

P1	H1	P2	P3	H2	P4	H3	P5	H4
3K	8K	4K	5K	6K	20K	12K	9K	3K

Dynamic Memory Allocation

Case B: Best Fit memory allocation strategy.

Which hole will be assigned for P6 with a size 5K?

Find out the new holes and its size after the termination of P3 and P1 (active process are P2, P4, P5 and P6)

P1	H1	P2	P3	H2	P4	H3	P5	H4
3K	8K	4K	5K	6K	20K	12K	9K	3K

Dynamic Memory Allocation

Case C: Worst Fit memory allocation strategy.

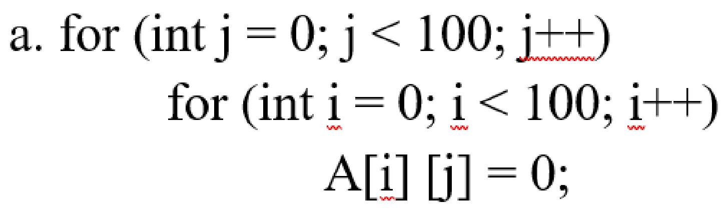
Which hole will be assigned for P6 with a size 5K?

Find out the new holes and its size after the termination of P3 and P1 (active process are P2, P4, P5 and P6)

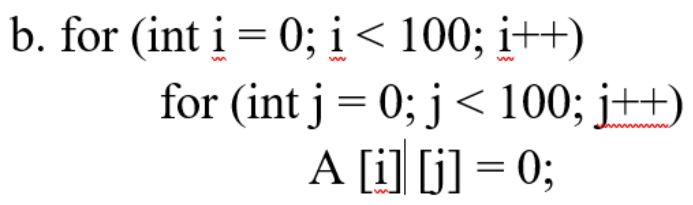
P1	H1	P2	P3	H2	P4	H3	P5	H4
3K	8K	4K	5K	6K	20K	12K	9K	3K

Page Replacement Policy

Q2: Consider the two-dimensional integer array A [100] [100] ; where A [0] [0] is at byte number 200 in a paged memory system with pages of size 200B. A small process that manipulates the matrix resides in page 0 (byte 0 to 199). Thus, every instruction fetch will be from page 0. For a three page frame main memory, how many page faults are generated by the following array-initialization loops (a and b are two different program), using LRU replacement and assuming that page frame 1 contains the process and the other two are initially empty? Assume that an integer is assigned 2 bytes in memory.



```
a. for (int j = 0; j < 100; j++)
       for (int i = 0; i < 100; i++)
           A[i] [j] = 0;
```



```
b. for (int i = 0; i < 100; i++)
       for (int j = 0; j < 100; j++)
           A [i] [j] = 0;
```

Page Replacement Policy

Q2: A [100] [100] ; where A [0] [0] is at byte number 200. 200B pages. For a three page frame main memory, frame 0 is for the program and frames 1 and 2 are for storing A[][]. One element of A is 2B.

```
a. for (int j = 0; j < 100; j++)
        for (int i = 0; i < 100; i++)
                A[i] [j] = 0;
```

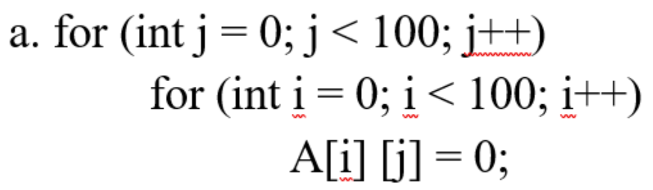
```
b. for (int i = 0; i < 100; i++)
        for (int j = 0; j < 100; j++)
                A [i] [j] = 0;
```

A[][] stored in row major form.

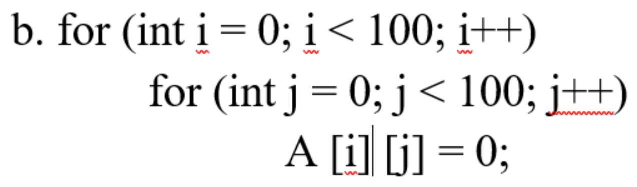
A[0][0]
A[0][1]
A[0][2]
..
A[0][99]
A[1][0]
A[1][1]
…
A[1][99]
A[2][0]
A[2][1]
..
..
A[99][0]
..
A[99][98]
A[99][99]

Page Replacement Policy

Q2: A [100] [100] ; where A [0] [0] is at byte number 200. 200B pages. For a three page frame main memory, frame 0 is for the program and frames 1 and 2 are for storing A[][]. One element of A is 2B.



```
a. for (int j = 0; j < 100; j++)
       for (int i = 0; i < 100; i++)
           A[i] [j] = 0;
```



```
b. for (int i = 0; i < 100; i++)
       for (int j = 0; j < 100; j++)
           A [i] [j] = 0;
```

A[][] stored in row major form.

Case a: order of access A[0][0], A[1][0], A[2][0],… A[0][1], A[1][1]…A[0][2], A[1][2]….

During page fault at A[0][0] → A[0][0] to A[0][99] is brought, but others not used.

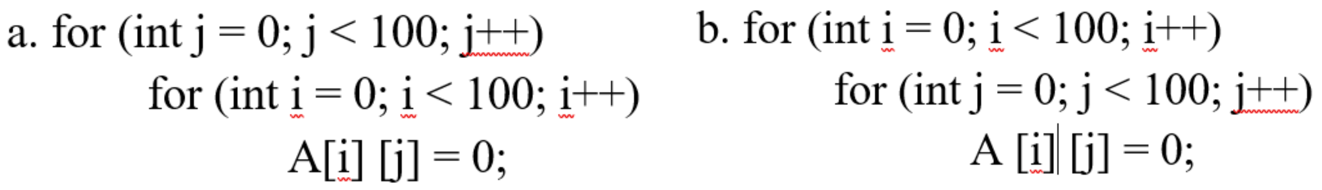
During page fault at A[1][0] → A[1][0] to A[1][99] is brought, but others not used. ..

During page fault at A[0][1] → A[0][0] to A[0][99] is brought, but others not used…

Hence access of each A[][] results in page faults → 10,000 page faults.

Page Replacement Policy

Q2: A [100] [100] ; where A [0] [0] is at byte number 200. 200B pages. For a three page frame main memory, frame 0 is for the program and frames 1 and 2 are for storing A[][]. One element of A is 2B.



```
a. for (int j = 0; j < 100; j++)
       for (int i = 0; i < 100; i++)
           A[i] [j] = 0;

b. for (int i = 0; i < 100; i++)
       for (int j = 0; j < 100; j++)
           A [i] [j] = 0;
```

A[][] stored in row major form.

Case b: order of access A[0][0], A[0][1], A[0][2],…A[1][0], A[1][1],...A[2][0], A[2][1]….

During page fault at A[0][0] → A[0][0] to A[0][99] is brought, all are accessed.

During page fault at A[1][0] → A[1][0] to A[1][99] is brought, all are accessed.

During page fault at A[2][1] → A[2][0] to A[2][99] is brought, all are accessed.

Hence access of each new i A[i][j] only results in page faults → 100 page faults.

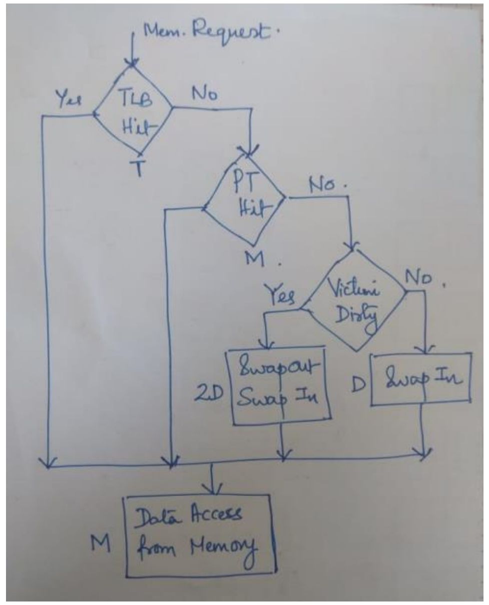
TLB and Page Table Access Mechanism

Q3: A paging system uses a page table residing in main memory and a TLB for address translation. The TLB has a lookup time of 20 ns and miss rate of 10%. The page table that resides in main memory has a page hit rate of 95%. Each main memory access takes 400 ns and each page transfer to/from the disk takes 6000 ns. During page replacement, 15% victim frames are found to be dirty which have to be written back to disk before the required page is read in from the disk. Assume that page table update, subsequent TLB update and referring to updated entry take negligible time. What is the average memory access time?

- ❖ TLB access time: T=20ns : Main Memory Access time: M=400ns
- ❖ Disk access time: D =6000ns
- ❖ TLB hit rate: hr=90% : Page fault rate: pf=5% :
- ❖ Dirty page rate: dr =15%

TLB and Page Table Access Mechanism

- TLB access time: T=20ns : Main Memory Access time: M=400ns
- Disk access time: D =6000ns
- TLB hit rate: hr=90% : Page fault rate: pf=5% :
- Dirty page rate: dr =15%
- After PT/ TLB update it takes M more to access memory.
- AMAT = hr.(T+M) + (1- hr).{(1-pf).(T+M+M) + dr).(T+M+D+M) + dr.(T+M+D+D+M)]}]



- Substituting the value in the above equation, **AMAT=494.5ns**.

Thank You