CSD 204 Lab

Lab 7: Implementing Banker's & Memory Management Unit (MMU)

[Dr. Sweta Kumari, Assistant Professor, SNIoE]

Deadline – 13th April 2025, 11:59PM

Goal: - The goal of this assignment is to implement Banker's algorithm & address translation via MMU unit studied in the class. Implement these algorithms in C/C++.

Question-1

Details: Banker's algorithm is a deadlock avoidance algorithm. It is named so because this algorithm is used in banking systems to determine whether a loan can be granted or not. Consider there are n account holders in a bank and the sum of the money in all of their accounts is S. Every time a loan has to be granted by the bank; it subtracts the loan amount from the total money the bank has. Then it checks if that difference is greater than S. It is done because, only then, the bank would have enough money even if all the n account holders draw all their money at once.

Data Structures used to implement the Banker's Algorithm

Some data structures **we have studied in our class that** are used to implement the banker's algorithm are as follows:

1. Available

It is an array of length m. It represents the number of available resources of each type. If Available[j]= k, then there are k instances available, of resource type Rj.

2. Max

It is an n x m matrix which represents the maximum number of instances of each resource that a process can request. If Max[i][j] = k, then the process Pi can

request atmost k instances of resource type Rj.

3. Allocation

It is an n x m matrix which represents the number of resources of each type currently allocated to each process. If Allocation[i][j] = k, then process Pi is currently allocated k instances of resource type Rj.

4. Need

It is a two-dimensional array. It is an n x m matrix which indicates the remaining resource needs of each process. If Need[i][j] = k, then process Pi may need k more instances of resource type Rj to complete its task.

Need[i][j] = Max[i][j] - Allocation [i][j]

Banker's algorithm comprises of two algorithms:

1.Safety algorithm
2. Resource request algorithm

Safety Algorithm

A safety algorithm is an algorithm used to find whether or not a system is in its safe state. The algorithm is as follows:

1.Let Work and Finish be vectors of length m and n, respectively. Initially,

Work = Available
Finish[i] =false for i = 0, 1, ... , n - 1.

This means, initially, no process has finished and the number of available resources is represented by the Available array.

2.Find an index i such
that both Finish[i]
==false
Needi <= Work
If there is no such i present, then proceed to step 4.

It means, we need to find an unfinished process whose needs can be satisfied by the available resources. If no such process exists, just go to step 4.

3. Perform the following:
Work = Work + $Allocation_i$
Finish[i] = true
Go to step 2.

When an unfinished process is found, then the resources are allocated, and the process is marked finished. And then, the loop is repeated to check the same for all other processes.

4.If Finish[i] == true for all i, then the system is in a safe state.
That means if all processes are finished, then the system is in safe state.

This algorithm may require an order of mxn^2 operations in order to determine whether a state is safe or not.

Resource Request Algorithm

Now the next algorithm is a resource-request algorithm and it is mainly used to determine whether requests can be safely granted or not.

Let $Request_i$ be the request vector for the process Pi. If Requesti[j]==k, then process Pi wants k instance of Resource type Rj.When a request for resources is made by the process Pi, the following are the actions that will be taken:

1.If $Request_i$ <= $Need_i$, then go to step 2;else raise an error condition, since the process has exceeded its maximum claim.

2.If $Request_i$ <= $Available_i$ then go to step 3; else Pi must have to wait as resources are not available.

3.Now we will assume that resources are assigned to process Pi and thus perform the following steps:

Available= Available-$Request_i$;

$Allocation_i$=$Allocation_i$ +$Request_i$;
$Need_i$ =$Need_i$ - $Request_i$;

If the resulting resource allocation state comes out to be safe, then the transaction

is completed and, process Pi is allocated its resources. But in this case, if the new state is unsafe, then Pi waits for Requesti, and the old resource-allocation state is restored.

Write a C/C++ program for Banker's algorithm for finding out the safe sequence. Bankers algorithm is used to schedule processes according to the resources they need. It is very helpful in Deadlock Handling. **Bankers algorithm produce a safe sequence as a output if all the resources can be executed and return error if no safe sequence of the processes available.**

Input: There are 5 processes P1, P2, P3, P4, P5, and 3 types of resources X, Y, Z.

Total resources X=7, Y=5, Z=7

Process	Maximum Need M[i][j]			Allocation A[i][j]		
	X	Y	Z	X	Y	Z
P1	6	5	3	1	0	2
P2	4	2	1	3	1	0
P3	5	1	2	0	1	1
P4	1	0	2	1	0	1
P5	5	2	3	1	0	2

Output: The safe sequence of execution is P2 -> P4 -> P5 -> P3 -> P1

Question- 2

Details:- Write a C/C++ program for address translation via MMU unit. Assumes memory is a byte addressable memory. You will be give Logical address space size, Main memory size and Page size as an input to your program. Your program should be interactive with switch cases. On providing the program size It will give number of pages of that particular process followed by perform random allocation inside main memory. On requesting for any address it should return frame number in the main memory.

Your interactive input should be:

1. process size (bytes)
2. Logical address (bits)
3. Main Memory Size
4. Page Size

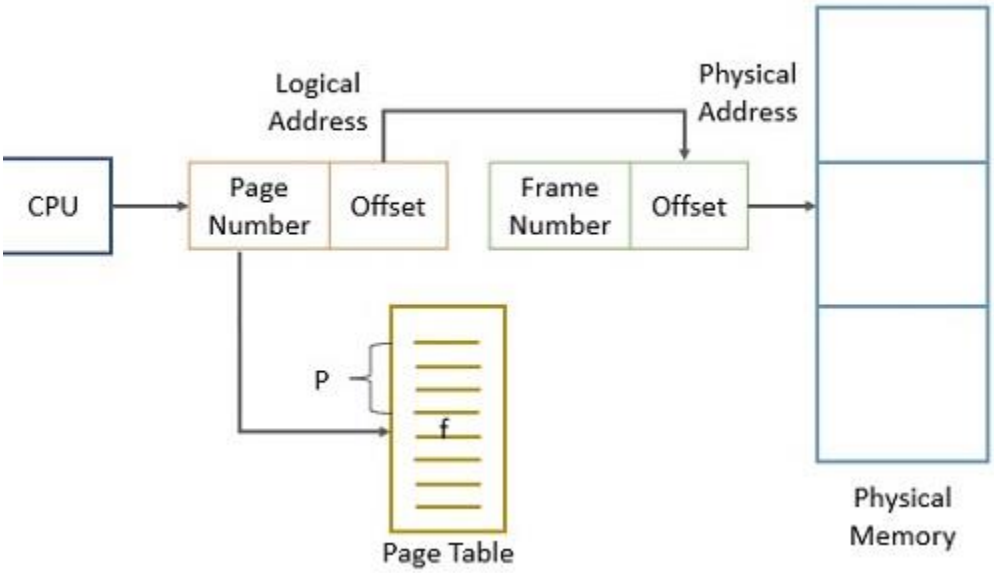
Your output should be as follows:

1. Logical address Bits =
2. Page Number Bits =
3. Page offset bits =
4. Physical Address Bits =
5. frame number bits =
6. frame offset bits =
7. Total number of pages in the Logical Address Space
8. Total Number of frames can be allocated in Main memory (MM)

Test case:

- **Input:**
  - Process Size = 4096 bytes (4KB)
  - Logical Address Bits = 16
  - Main Memory Size = 65536 bytes (64KB)
  - Page Size = 1024 bytes (1KB)
- **Expected Output:**
  - Logical Address Bits = 16
  - Page Number Bits = 6 (since 16 - log2(1024) = 16 - 10 = 6)
  - Page Offset Bits = 10
  - Physical Address Bits = 16 (since 6 (frame bits) + 10 (offset) = 16)
  - Frame Number Bits = 6 (since 65536 / 1024 = 64 frames → log2(64) = 6)
  - Frame Offset Bits = 10
  - Total number of pages in the Logical Address Space = 64 (2^6)
  - Total Number of frames can be allocated in MM= 64 (65536 / 1024)

Hint: Frame size is always equal to Page Size.



Submission Format:- You have to upload: (1) The source code in the following format: Assgn7Src<Name_Q1>.c & Assgn7Src<Name_Q2>.c (2) Report: Assgn7Report-<Name>.pdf. Don't zip any of your documents. Upload all the documents on the blackboard.
Note: Please follow this naming convention mentioned above.

Grading Policy: - The policy for grading this assignment will be - (1) Design as described in the report and analysis of the results: 50% (2) Execution of the tasks based on the description in the readme: 40% (3) Code documentation and indentation: 10%.

Please note:

- All assignments for this course have a late submission policy of a penalty of 10% each day after the deadline of six days. After that, it will not be evaluated.

- **All submissions are subject to plagiarism checks.** Any case of plagiarism will be dealt with severely.