Kathmandu University

Department of Computer Science and Engineering

Dhulikhel, Kavre

A Report on

"Memory Management"

[Code No.: COMP 307]

(CE-III/I)

Submitted by

Puspa Hari Ghimire (21)

Group members

Suyog Dhungana (18)

Swochhanda Jangam (25)

Submitted to

Mr. Santosh Khanal

Assistant Professor

Department of Computer Science and Engineering

Submission Date:

June 11, 2024

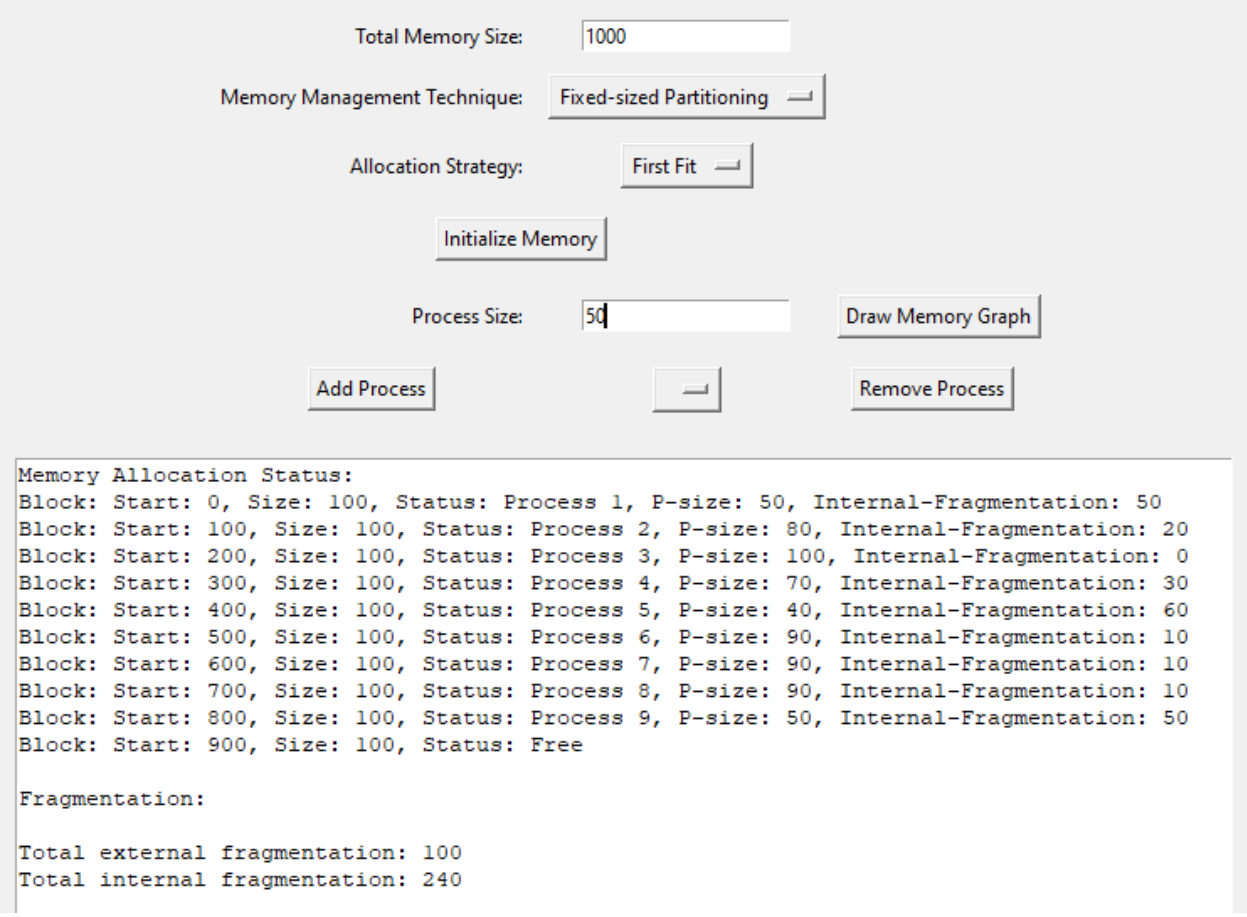
Memory Management:

Memory management is a critical component of an operating system that handles the allocation and deallocation of memory resources to various processes. It ensures efficient utilization of memory, maximizes performance, and maintains system stability. Effective memory management enables the system to run multiple processes simultaneously without interference, optimizing both speed and resource use.

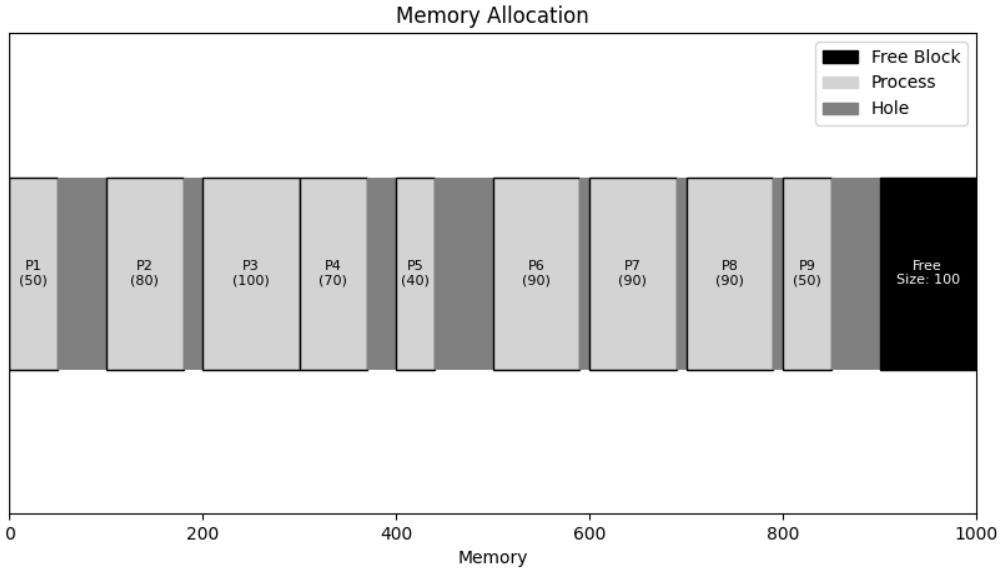
Fixed-sized Memory Partitioning:

Fixed-sized memory partitioning, also known as static partitioning, involves dividing the main memory into fixed-size partitions at system initialization. Each partition is called a block and can hold exactly one process, with no flexibility for resizing. This method is simple to implement but can lead to internal fragmentation, where unused memory within a partition is wasted if the process size is smaller than the partition size.

Use of first fit:



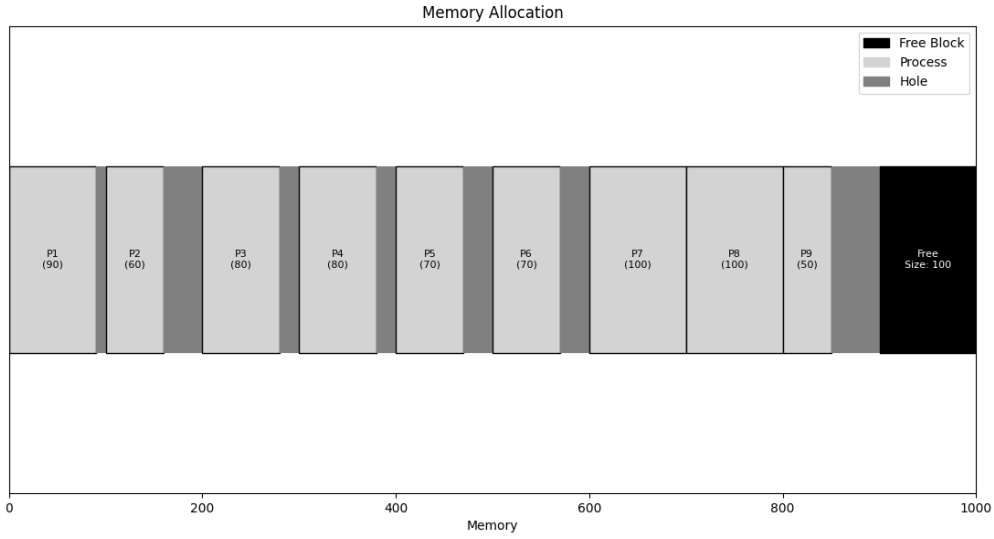
Source Code: https://github.com/puspah-ghimire/Memory-management

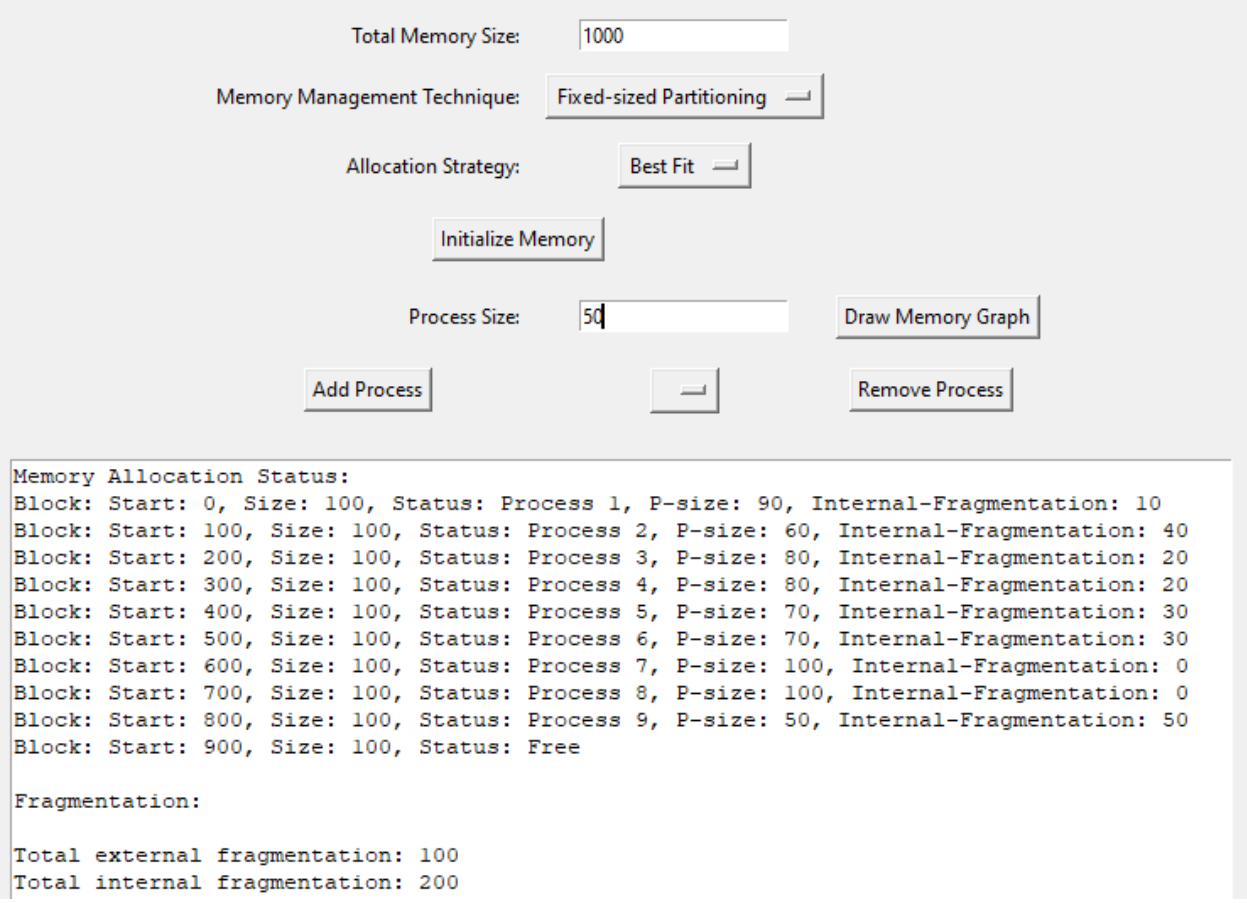


Processes P1, P2, P3…. arrive sequentially with their own size. In case of first-fit, the processes are placed into the fixed sized block serially. From the above chart, we see that the processes having unequal sizes are placed in the blocks. It creates the holes, which are also known as internal fragmentations. If the block doesn't contain any process, then it is referred as external fragmentation.

Use of best fit:

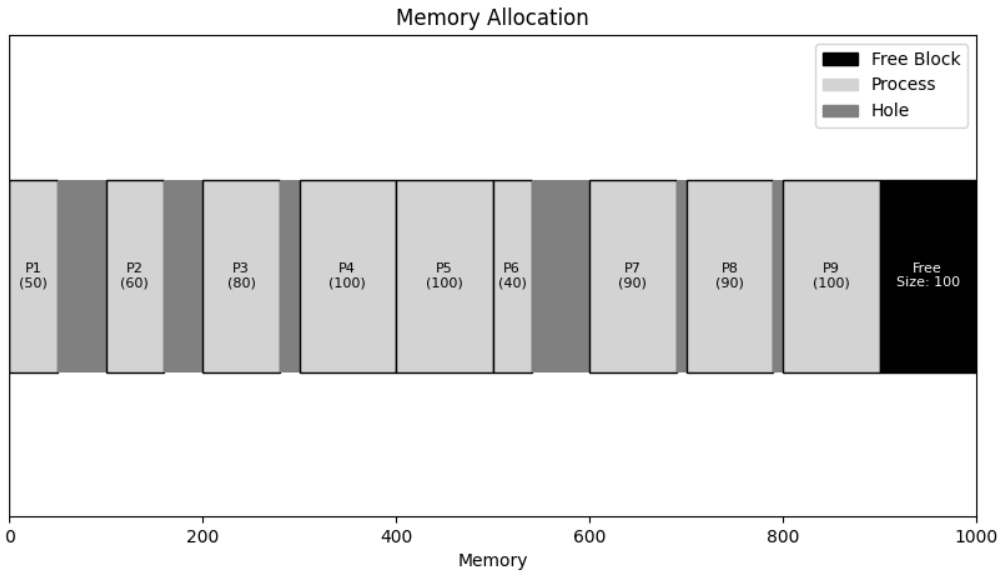
The use of best-fit algorithm is as same as the first-fit algorithm as the blocks are of equal sizes.

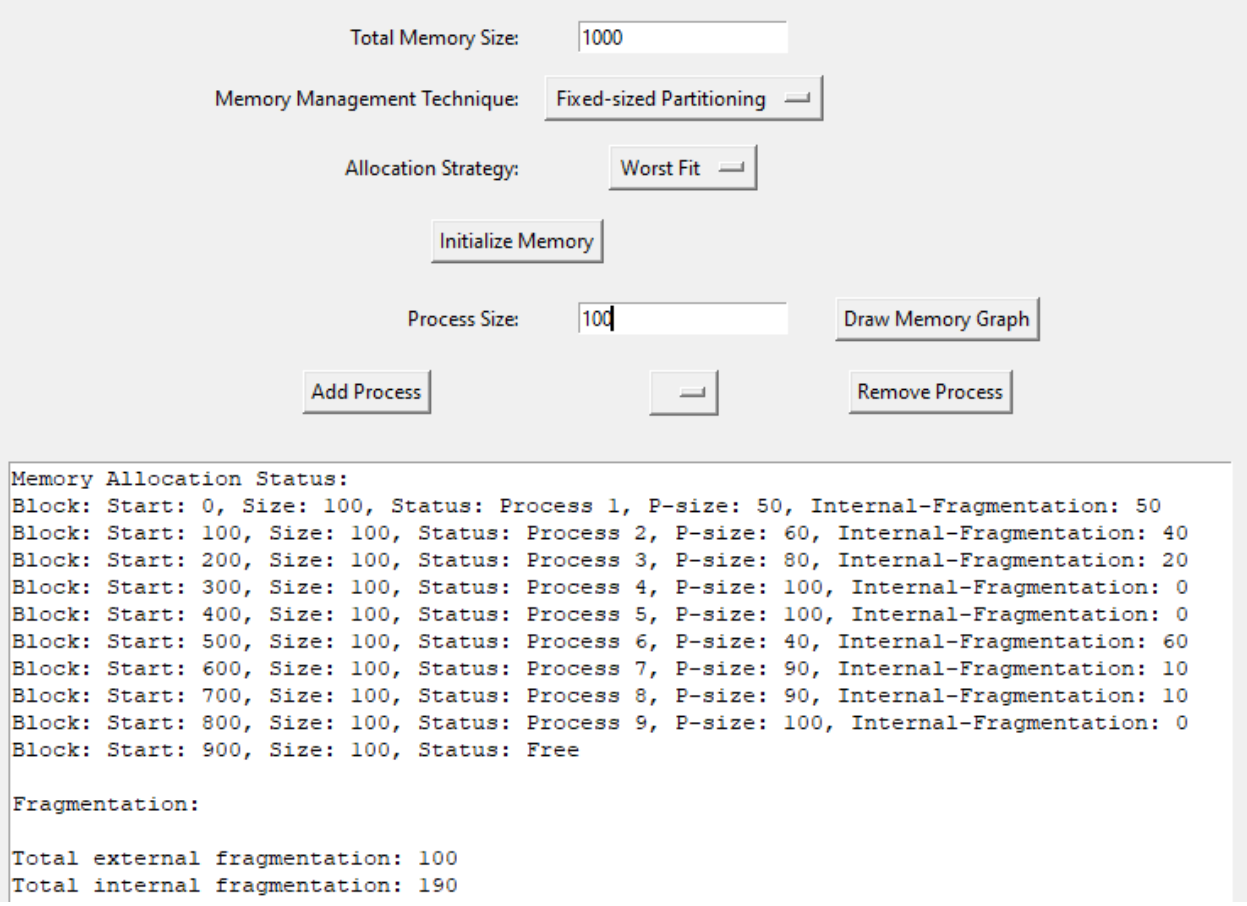




Use of worst-fit:

There is no difference in fixed sized partitioning when using worst-fit strategy.



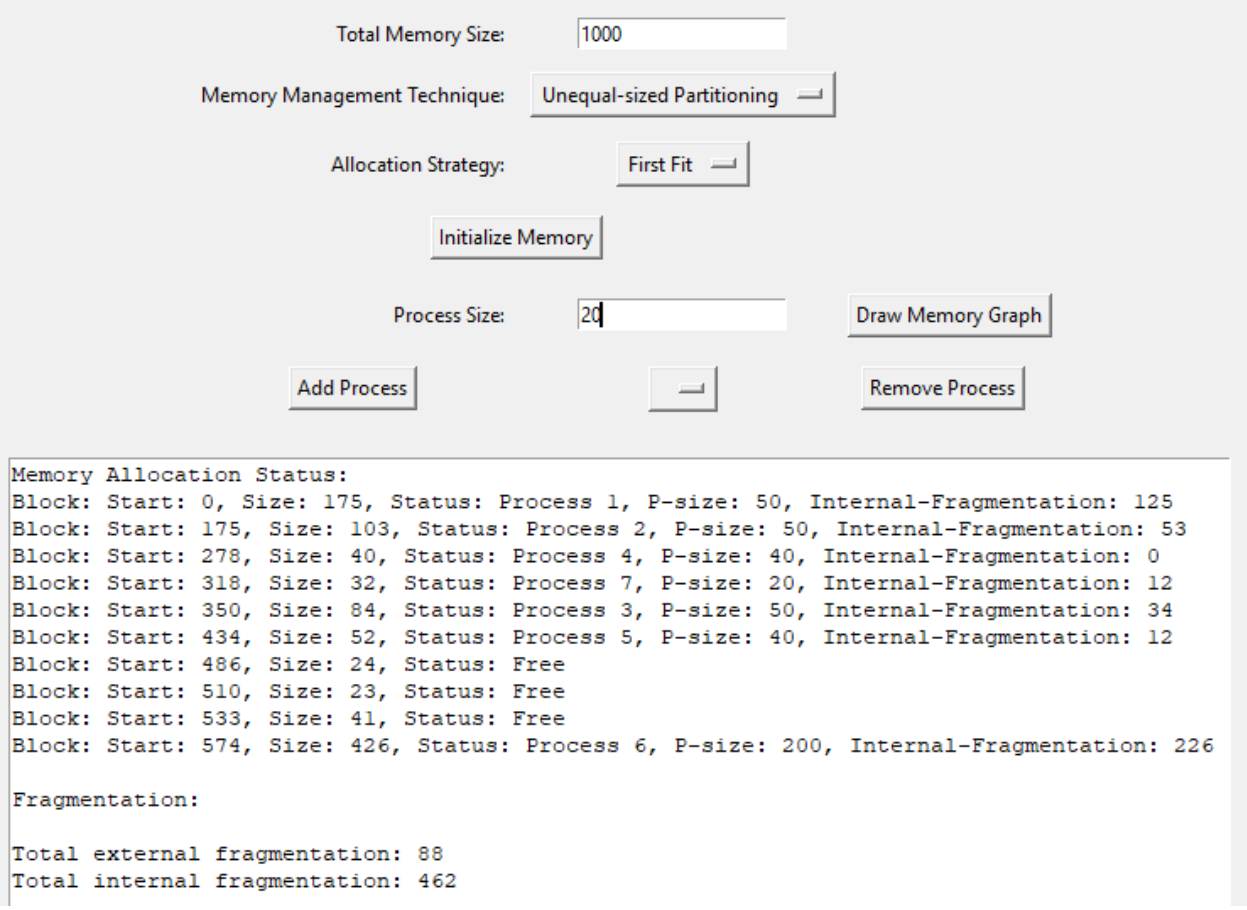


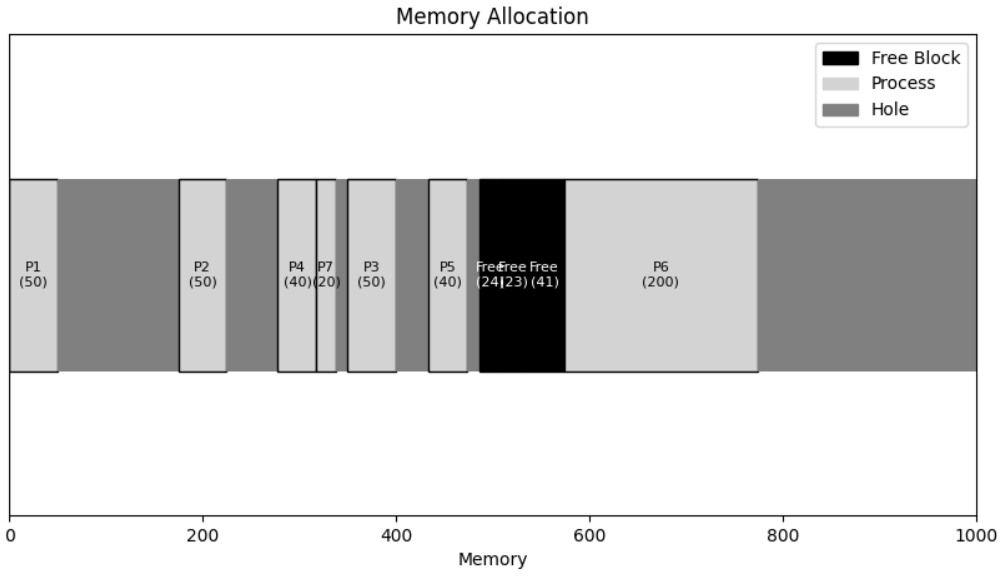
Unequal-sized Fixed Partitioning:

Unequal-sized fixed partitioning improves upon fixed-sized partitioning by allowing partitions of different sizes. This setup can reduce internal fragmentation by better matching partition sizes to process requirements. However, it still suffers from external fragmentation, as there might be unused spaces between partitions that cannot be used by new processes if no single partition is adequately sized.

Use of first-fit:

Whatever the size of the process, the first-fit algorithm allocates the processes into the first block it fits. If the size of the block is inadequate to the process, it finds the next block which can accommodate the incoming process. It creates large internal fragmentations as we can see from the chart given below.

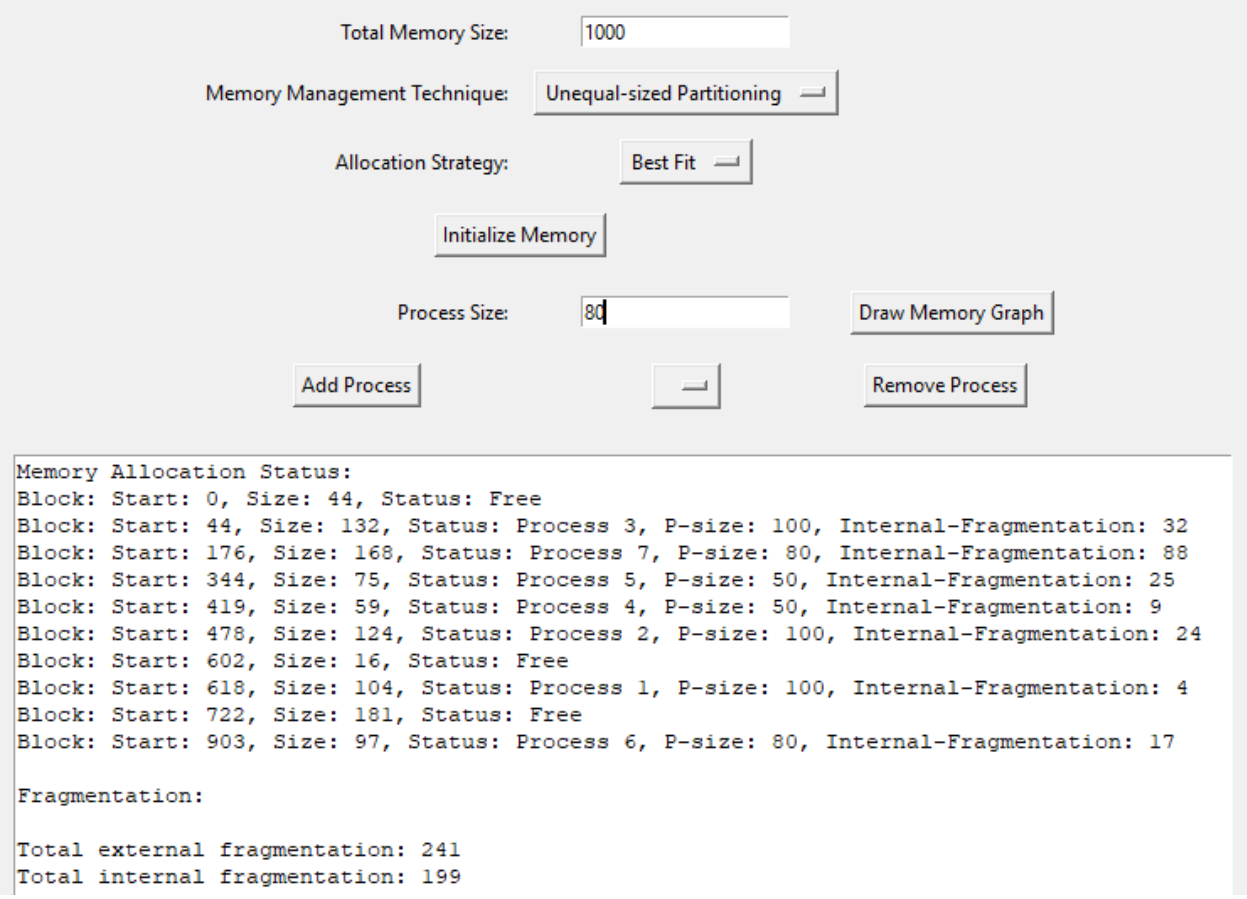


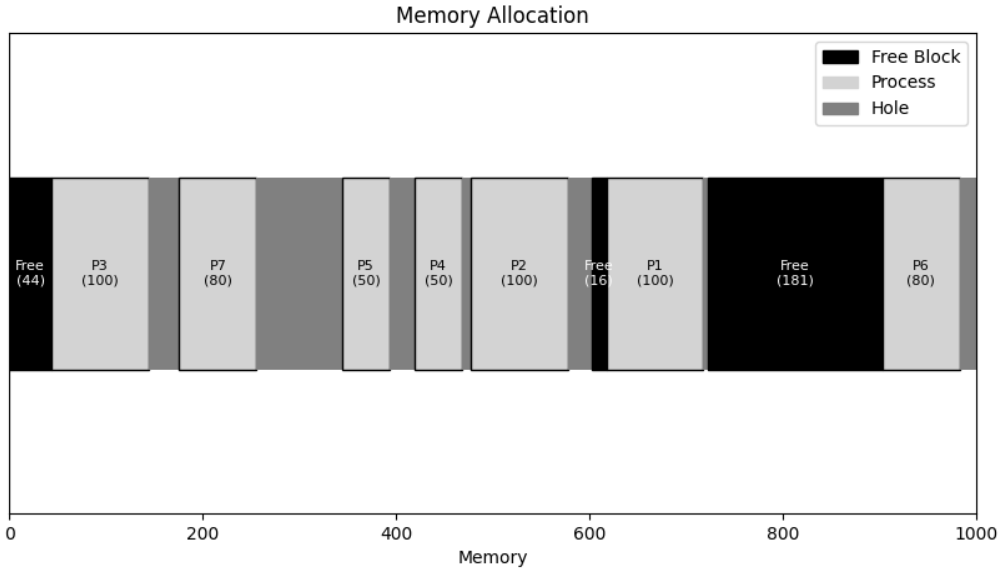


We can see that even though process 1 has size 50 which can be accommodated into other similar sizes, the first-fit algorithm puts it into the first block.

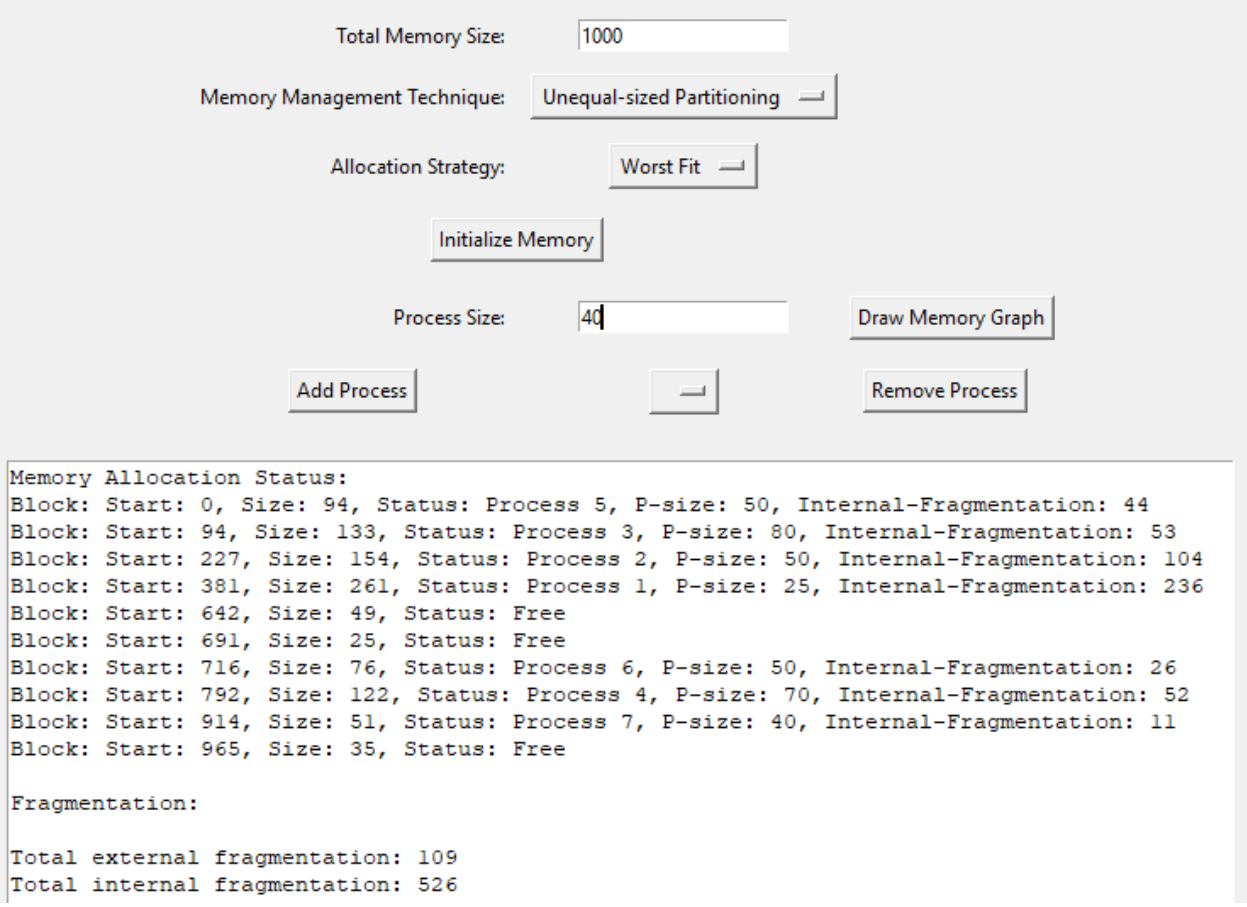
Use of best-fit:

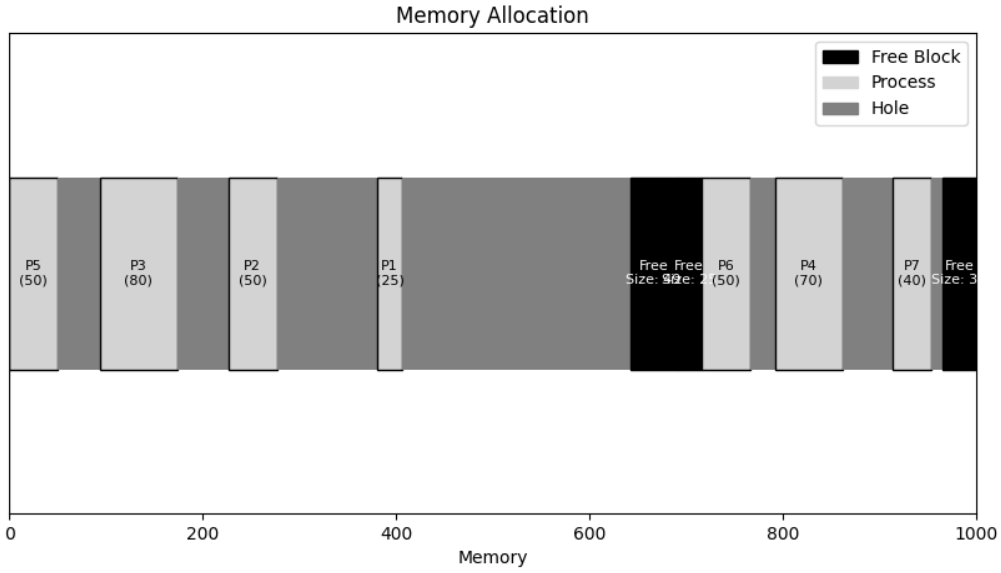
The below output and graph give the idea on how the best-fit algorithm works in the unequal fixed-sized partitioning. Here the process P1 has a size of 100 and hence, it is placed in the block of size 104 which is the best fit for this process. Similarly, other processes are also addressed in the similar manner.





Use of worst-fit:





Worst-fit algorithm places the process with the smallest size into the largest block. In the above graph, we see that the worst-fit strategy in unequal-sized fixed partitioning creates highest internal fragmentation. Process 1 having size 25 is the smallest of all processes and is placed in the biggest block of the memory.

Dynamic Memory Allocation:

Dynamic memory allocation assigns memory to processes at runtime, based on their needs. This method allows for efficient use of memory, as allocations can vary in size and can be resized or freed as needed. Techniques such as first fit, best fit, and worst fit are used to manage free memory blocks. Although dynamic allocation can reduce fragmentation, it can suffer from external fragmentation over time.

Allocation:

Below, we can clearly see that, there is no internal fragmentation in dynamic memory allocation process. Whatever the process size it is given the memory in runtime.

Total Memory Size: 1000

Memory Management Technique: Dynamic Allocation

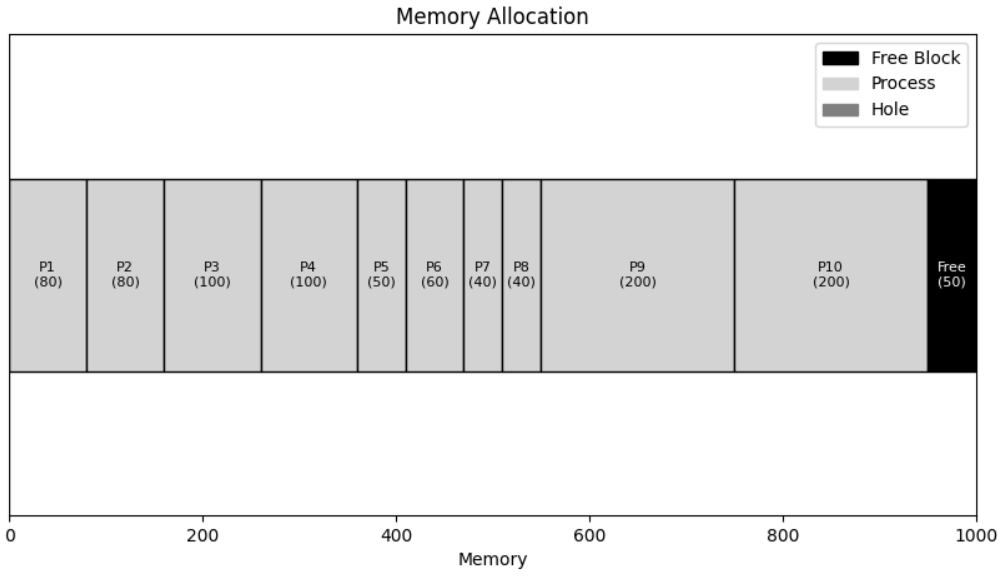
Allocation Strategy: Best Fit

Initialize Memory | Compact Memory

Process Size: 200 | Draw Memory Graph

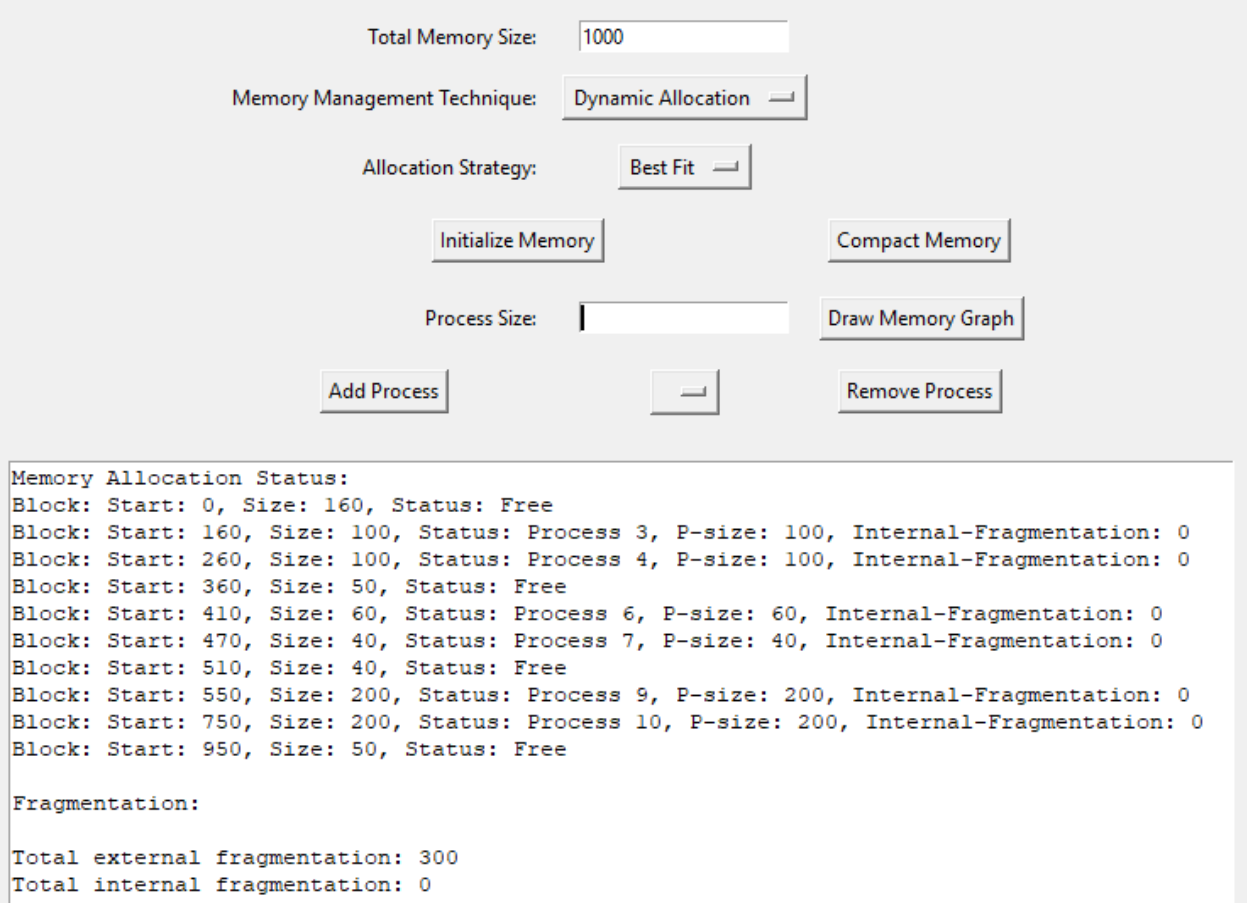
Add Process | | Remove Process

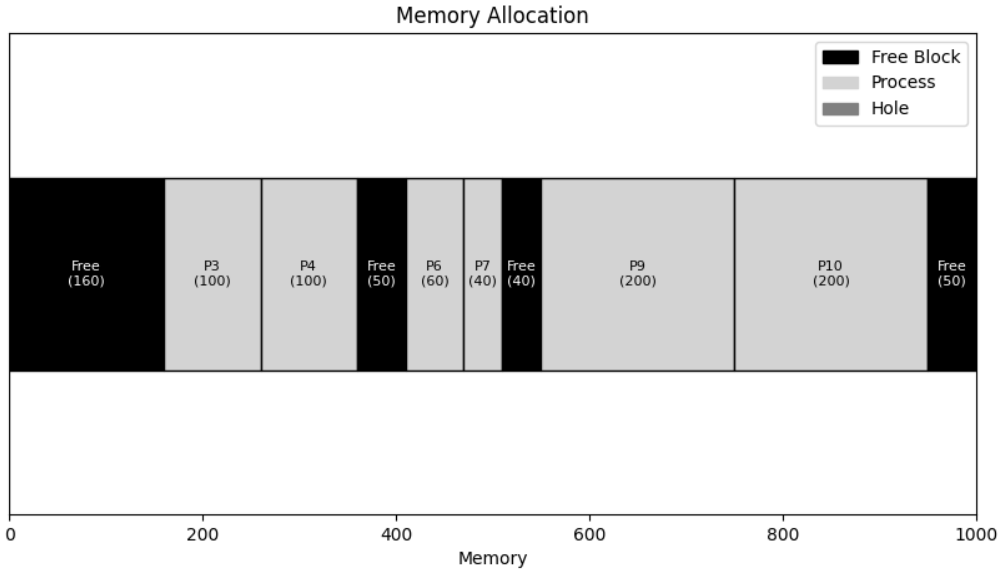
```
Memory Allocation Status:
Block: Start: 0, Size: 80, Status: Process 1, P-size: 80, Internal-Fragmentation: 0
Block: Start: 80, Size: 80, Status: Process 2, P-size: 80, Internal-Fragmentation: 0
Block: Start: 160, Size: 100, Status: Process 3, P-size: 100, Internal-Fragmentation: 0
Block: Start: 260, Size: 100, Status: Process 4, P-size: 100, Internal-Fragmentation: 0
Block: Start: 360, Size: 50, Status: Process 5, P-size: 50, Internal-Fragmentation: 0
Block: Start: 410, Size: 60, Status: Process 6, P-size: 60, Internal-Fragmentation: 0
Block: Start: 470, Size: 40, Status: Process 7, P-size: 40, Internal-Fragmentation: 0
Block: Start: 510, Size: 40, Status: Process 8, P-size: 40, Internal-Fragmentation: 0
Block: Start: 550, Size: 200, Status: Process 9, P-size: 200, Internal-Fragmentation: 0
Block: Start: 750, Size: 200, Status: Process 10, P-size: 200, Internal-Fragmentation: 0
Block: Start: 950, Size: 50, Status: Free

Fragmentation:

Total external fragmentation: 50
Total internal fragmentation: 0
```



Deallocation:

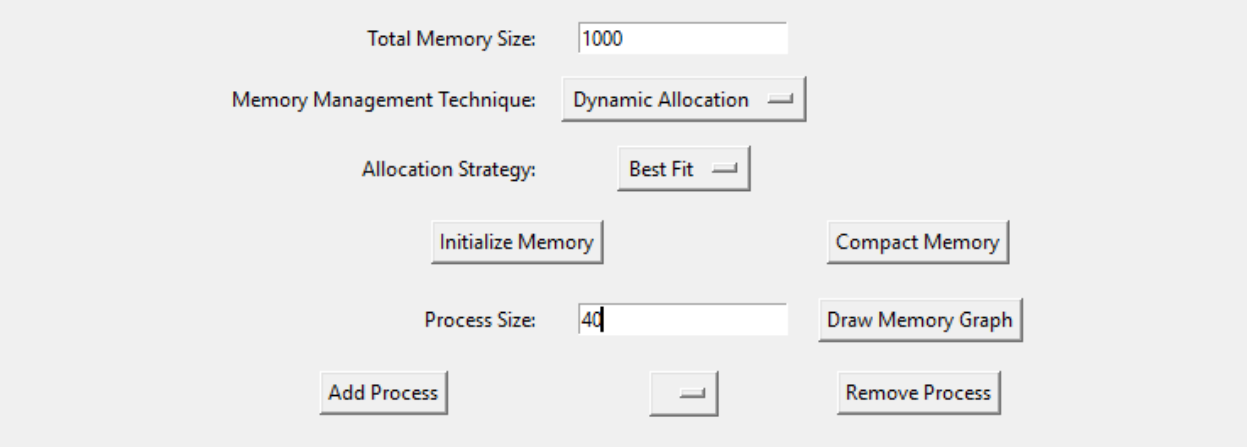
Processes P1, P2, P5, P8 are deallocated.



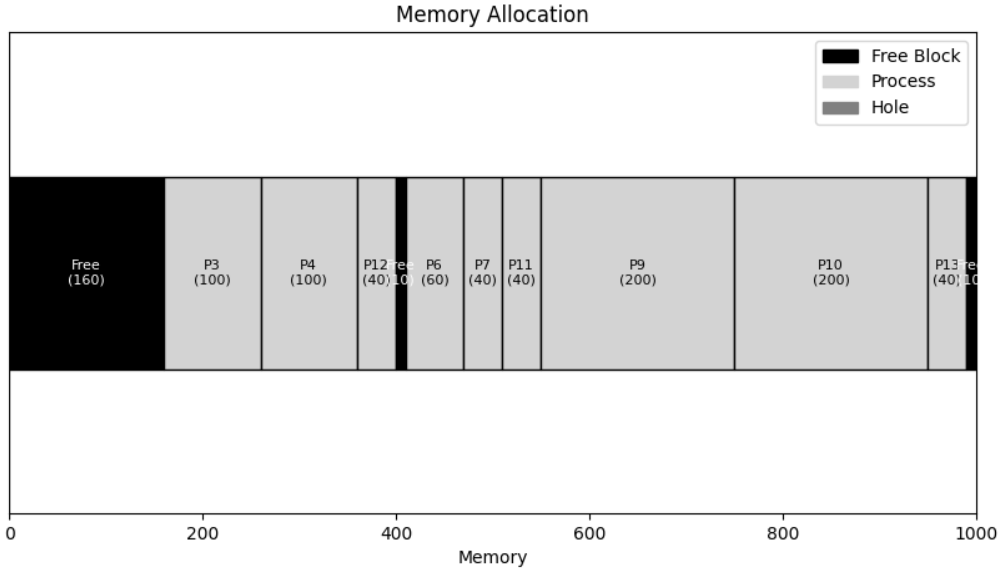


After deallocating the processes, there is the formation of external fragmentation as shown in above figure.

Use of best-fit:




```
Memory Allocation Status:
Block: Start: 0, Size: 160, Status: Free
Block: Start: 160, Size: 100, Status: Process 3, P-size: 100, Internal-Fragmentation: 0
Block: Start: 260, Size: 100, Status: Process 4, P-size: 100, Internal-Fragmentation: 0
Block: Start: 360, Size: 40, Status: Process 12, P-size: 40, Internal-Fragmentation: 0
Block: Start: 400, Size: 10, Status: Free
Block: Start: 410, Size: 60, Status: Process 6, P-size: 60, Internal-Fragmentation: 0
Block: Start: 470, Size: 40, Status: Process 7, P-size: 40, Internal-Fragmentation: 0
Block: Start: 510, Size: 40, Status: Process 11, P-size: 40, Internal-Fragmentation: 0
Block: Start: 550, Size: 200, Status: Process 9, P-size: 200, Internal-Fragmentation: 0
Block: Start: 750, Size: 200, Status: Process 10, P-size: 200, Internal-Fragmentation: 0
Block: Start: 950, Size: 40, Status: Process 13, P-size: 40, Internal-Fragmentation: 0
Block: Start: 990, Size: 10, Status: Free

Fragmentation:

Total external fragmentation: 180
Total internal fragmentation: 0
```



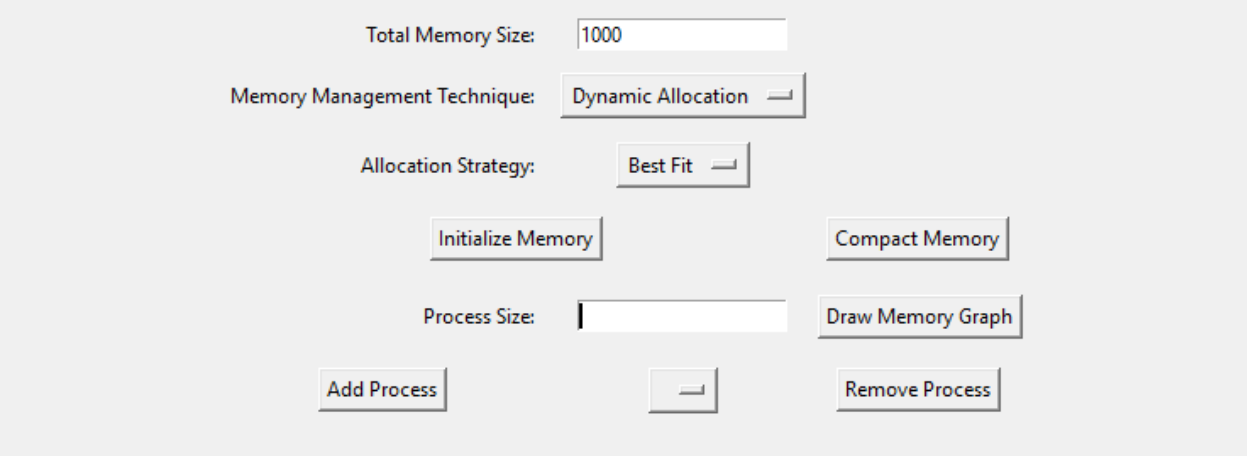
Process 11, 12, 13 serially enter in the memory. Here, the processes are placed according to their sizes that best fit in the fragmented memory. Process 11 has size 40, so it is placed into the block starting from 510, that has exactly 40 free-space. Similarly, process 12 also has size 40, and it is placed in the block starting from 360 which has 50 free-space. And process 13 having size 40 is accommodated in the block starting from 950 also having 50 free-space.

Hence, here is the formation of two small external fragmentation of size 10 which should be removed using compaction algorithm.

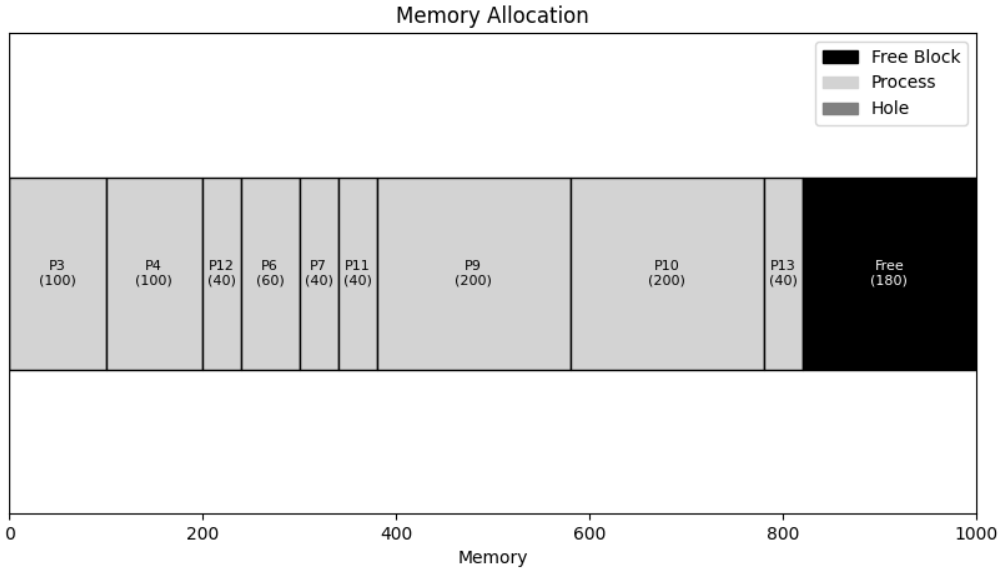
Compaction algorithm:

The compaction algorithm in dynamic memory allocation reduces external fragmentation by moving processes to consolidate free memory into a single contiguous block. This improves memory utilization and simplifies future allocations but is resource-intensive and can disrupt running processes. Despite its overhead, compaction is essential for efficient memory use in dynamic allocation systems.

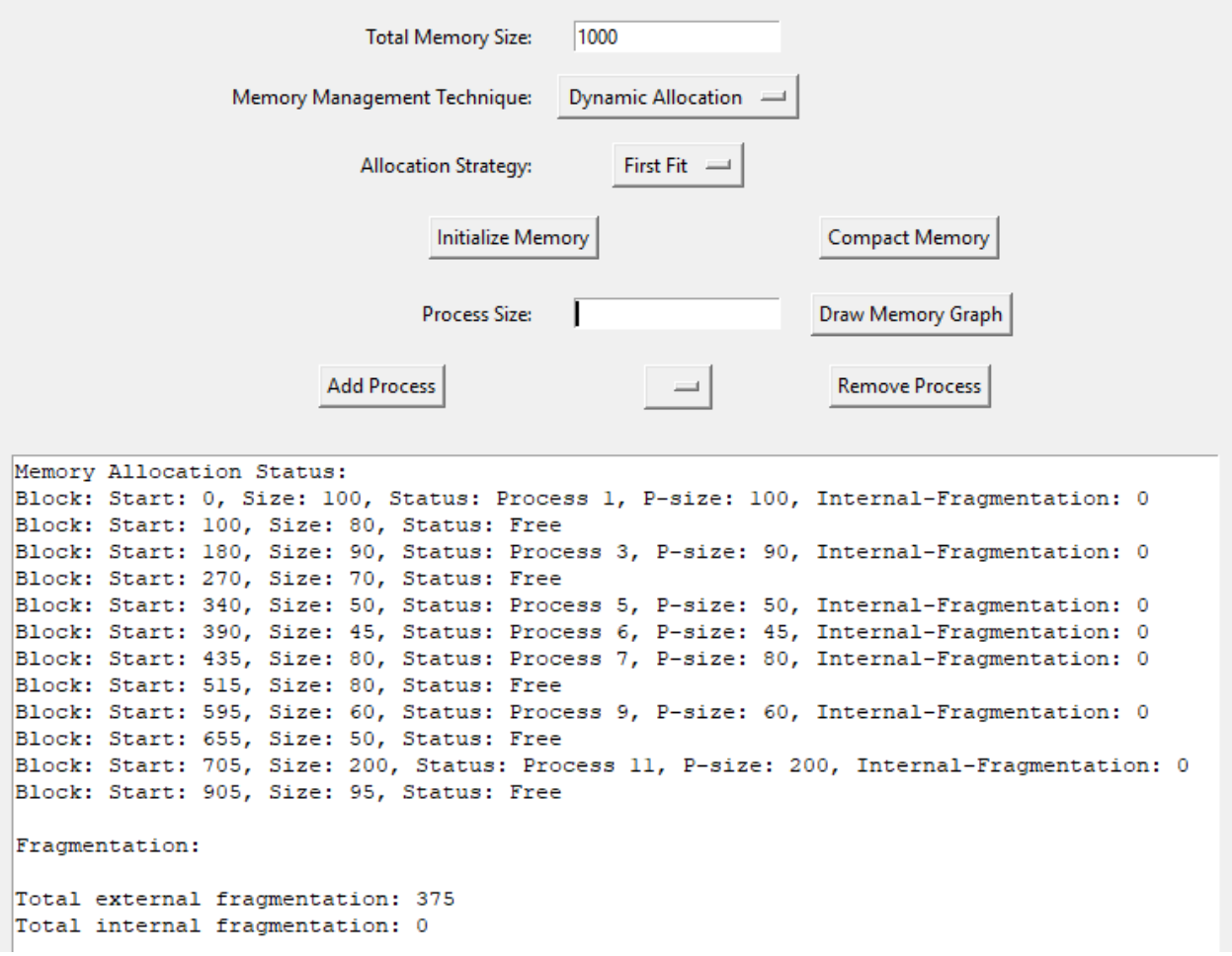
The below figures show us the output after compaction algorithm is applied.



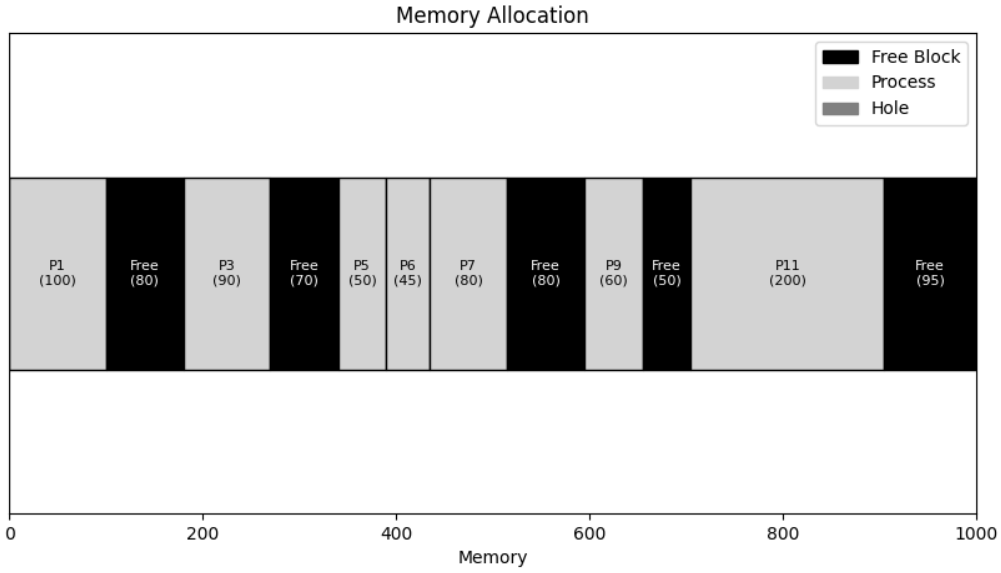

```
Memory Allocation Status:
Block: Start: 0, Size: 100, Status: Process 3, P-size: 100, Internal-Fragmentation: 0
Block: Start: 100, Size: 100, Status: Process 4, P-size: 100, Internal-Fragmentation: 0
Block: Start: 200, Size: 40, Status: Process 12, P-size: 40, Internal-Fragmentation: 0
Block: Start: 240, Size: 60, Status: Process 6, P-size: 60, Internal-Fragmentation: 0
Block: Start: 300, Size: 40, Status: Process 7, P-size: 40, Internal-Fragmentation: 0
Block: Start: 340, Size: 40, Status: Process 11, P-size: 40, Internal-Fragmentation: 0
Block: Start: 380, Size: 200, Status: Process 9, P-size: 200, Internal-Fragmentation: 0
Block: Start: 580, Size: 200, Status: Process 10, P-size: 200, Internal-Fragmentation: 0
Block: Start: 780, Size: 40, Status: Process 13, P-size: 40, Internal-Fragmentation: 0
Block: Start: 820, Size: 180, Status: Free

Fragmentation:

Total external fragmentation: 180
Total internal fragmentation: 0
```



Use of first-fit: (Before Allocation)




```
Memory Allocation Status:
Block: Start: 0, Size: 100, Status: Process 1, P-size: 100, Internal-Fragmentation: 0
Block: Start: 100, Size: 80, Status: Free
Block: Start: 180, Size: 90, Status: Process 3, P-size: 90, Internal-Fragmentation: 0
Block: Start: 270, Size: 70, Status: Free
Block: Start: 340, Size: 50, Status: Process 5, P-size: 50, Internal-Fragmentation: 0
Block: Start: 390, Size: 45, Status: Process 6, P-size: 45, Internal-Fragmentation: 0
Block: Start: 435, Size: 80, Status: Process 7, P-size: 80, Internal-Fragmentation: 0
Block: Start: 515, Size: 80, Status: Free
Block: Start: 595, Size: 60, Status: Process 9, P-size: 60, Internal-Fragmentation: 0
Block: Start: 655, Size: 50, Status: Free
Block: Start: 705, Size: 200, Status: Process 11, P-size: 200, Internal-Fragmentation: 0
Block: Start: 905, Size: 95, Status: Free

Fragmentation:

Total external fragmentation: 375
Total internal fragmentation: 0
```



Total Memory Size: 1000

Memory Management Technique: Dynamic Allocation

Allocation Strategy: First Fit

Initialize Memory

Compact Memory

Process Size: 50

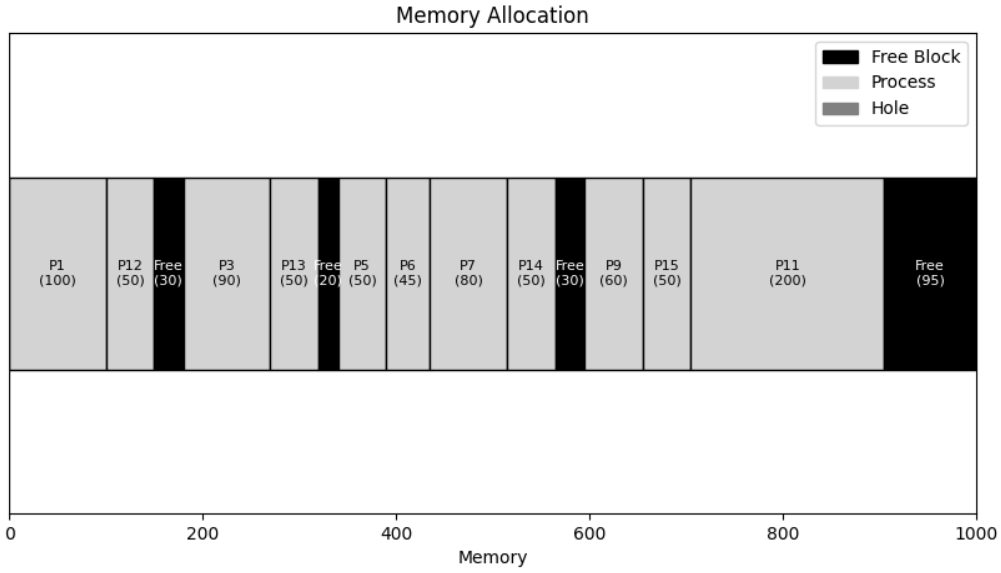
Draw Memory Graph

Add Process

Remove Process

After allocation

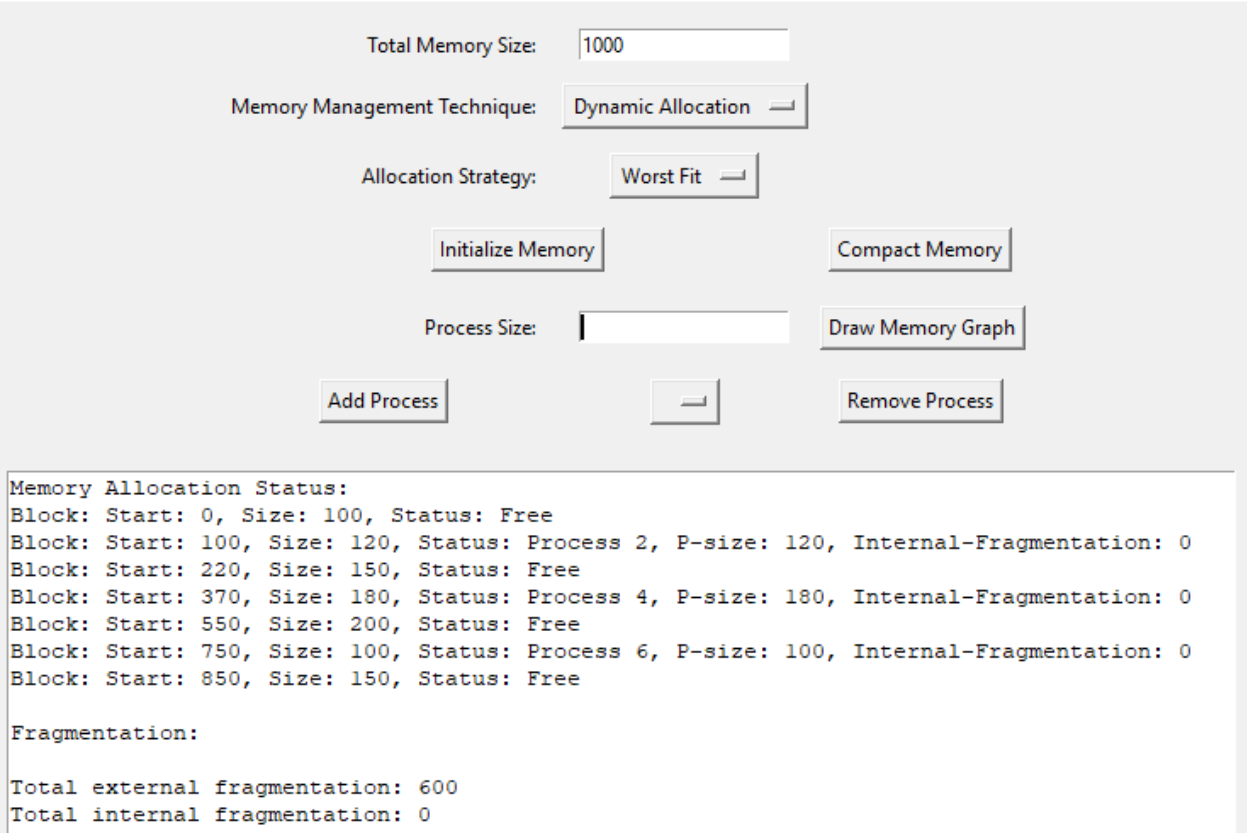
```
Memory Allocation Status:
Block: Start: 0, Size: 100, Status: Process 1, P-size: 100, Internal-Fragmentation: 0
Block: Start: 100, Size: 50, Status: Process 12, P-size: 50, Internal-Fragmentation: 0
Block: Start: 150, Size: 30, Status: Free
Block: Start: 180, Size: 90, Status: Process 3, P-size: 90, Internal-Fragmentation: 0
Block: Start: 270, Size: 50, Status: Process 13, P-size: 50, Internal-Fragmentation: 0
Block: Start: 320, Size: 20, Status: Free
Block: Start: 340, Size: 50, Status: Process 5, P-size: 50, Internal-Fragmentation: 0
Block: Start: 390, Size: 45, Status: Process 6, P-size: 45, Internal-Fragmentation: 0
Block: Start: 435, Size: 80, Status: Process 7, P-size: 80, Internal-Fragmentation: 0
Block: Start: 515, Size: 50, Status: Process 14, P-size: 50, Internal-Fragmentation: 0
Block: Start: 565, Size: 30, Status: Free
Block: Start: 595, Size: 60, Status: Process 9, P-size: 60, Internal-Fragmentation: 0
Block: Start: 655, Size: 50, Status: Process 15, P-size: 50, Internal-Fragmentation: 0
Block: Start: 705, Size: 200, Status: Process 11, P-size: 200, Internal-Fragmentation: 0
Block: Start: 905, Size: 95, Status: Free

Fragmentation:

Total external fragmentation: 175
Total internal fragmentation: 0
```

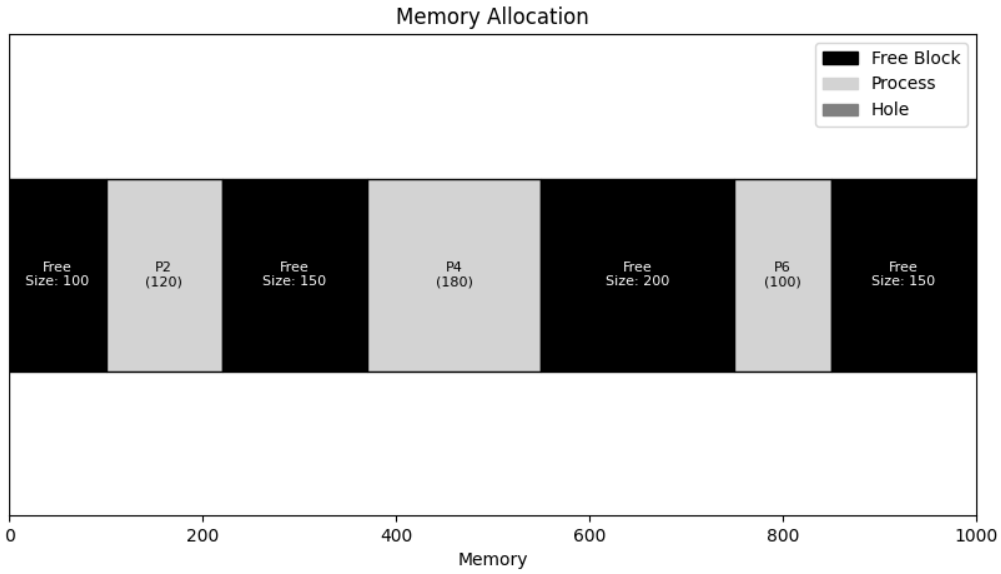


Processes 12, 13, 14 and 15 arrive sequentially and are placed into the first block that can accommodate each process.

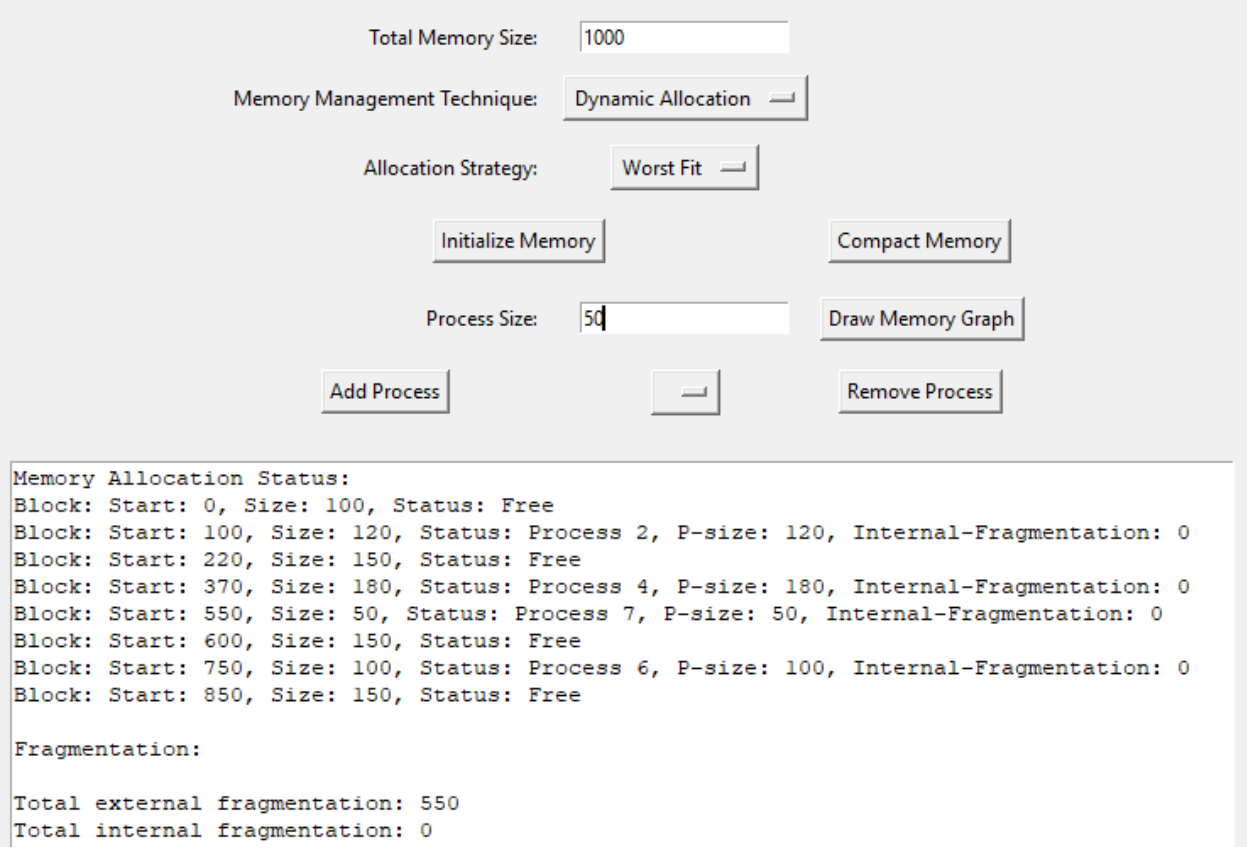
Use of worst-fit:

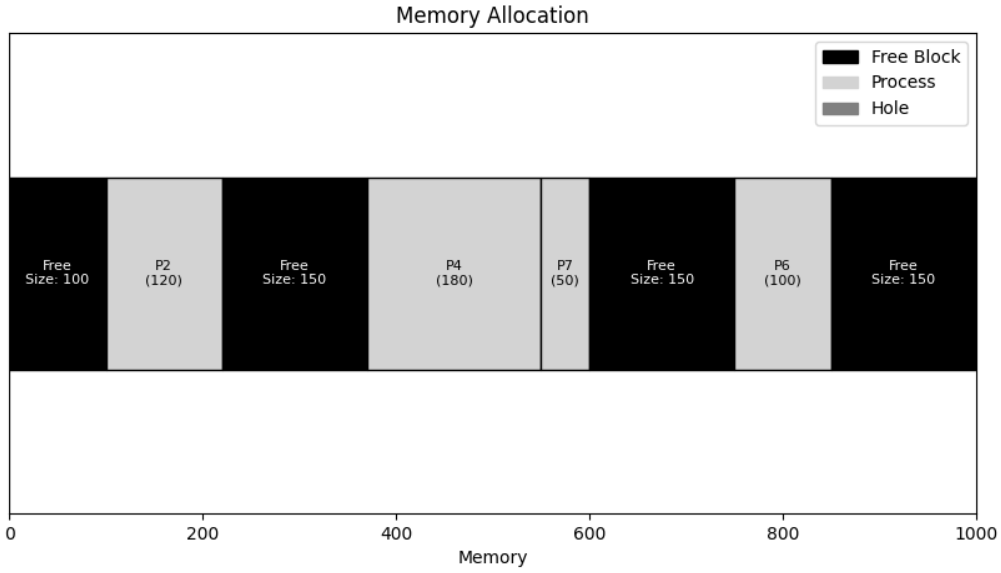
(Before Allocation)





(After Allocation)





In dynamic allocation, while using the worst-fit strategy, the process 7 of size 50 is placed in the biggest fragmented memory in the total memory-space.

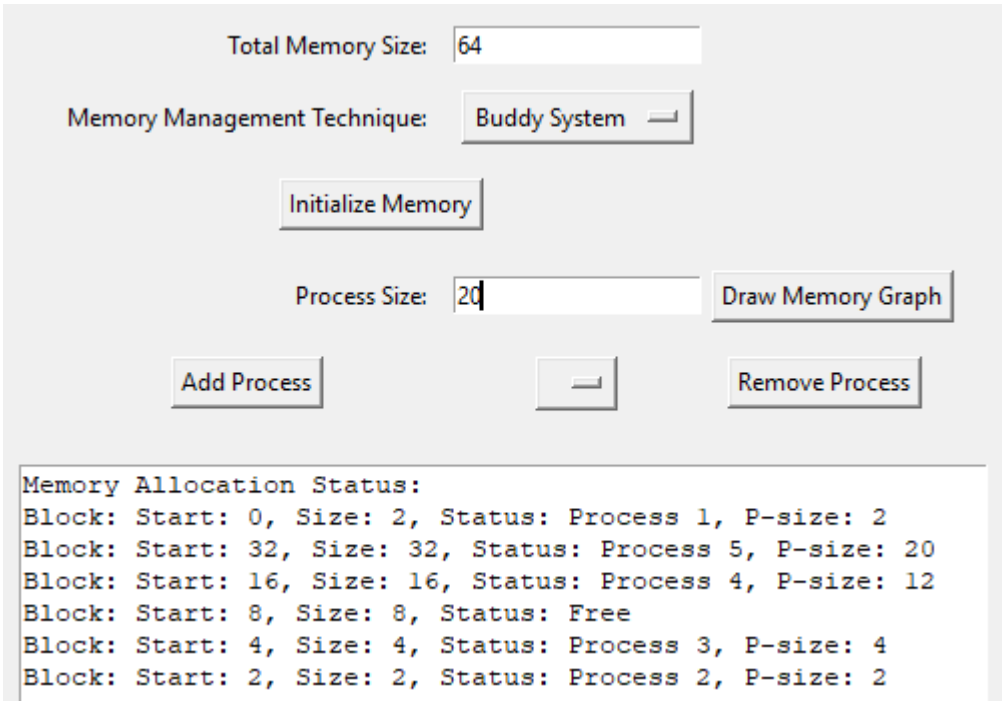
Buddy System:

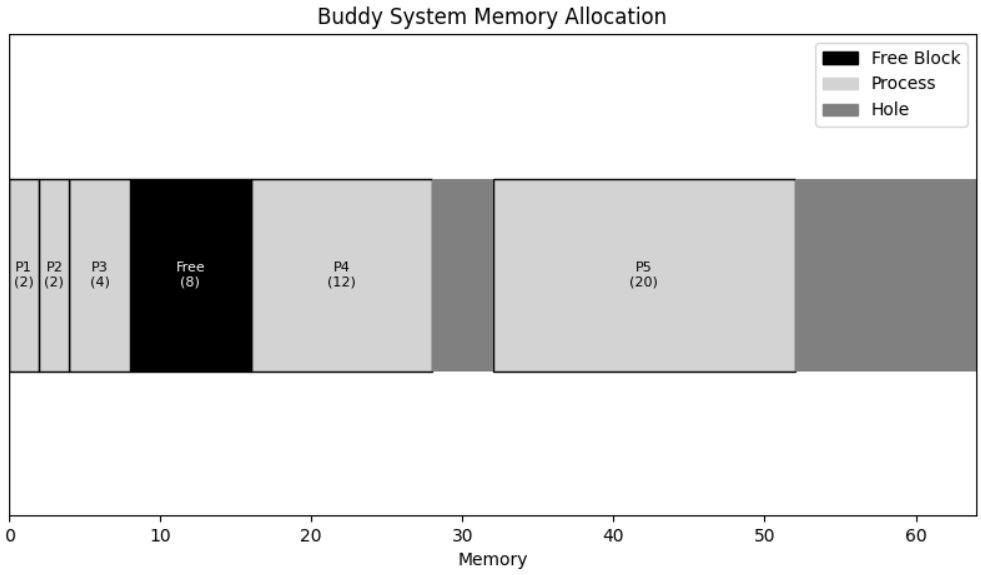
The buddy system is a memory allocation technique that manages free memory by dividing it into blocks of sizes that are powers of two.

When a memory request is made, the system finds the smallest available block that fits the request. If the smallest block is too large, it splits the block into two smaller "buddy" blocks, continuing to split until it finds an appropriately sized block.

Each split creates pairs of blocks, or "buddies," which can be merged back together when they are freed, reducing fragmentation.

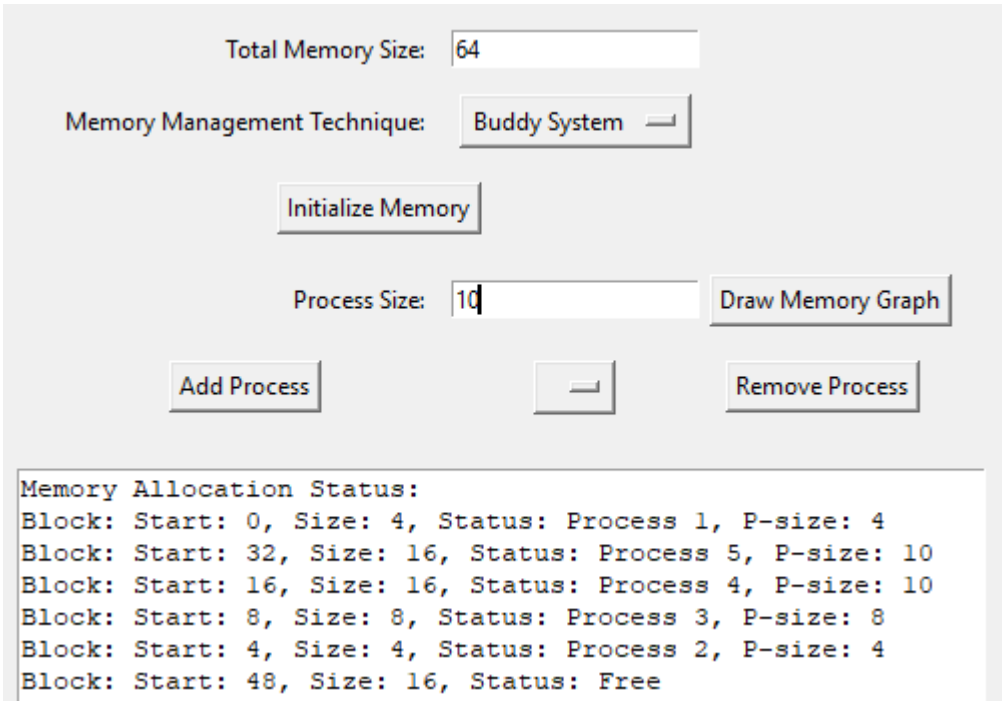
Splitting of blocks:

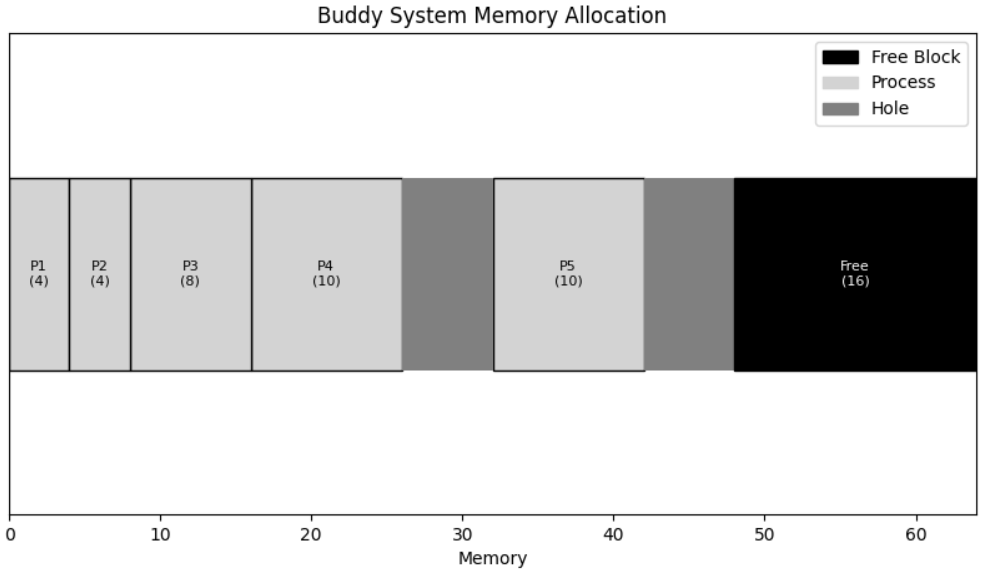




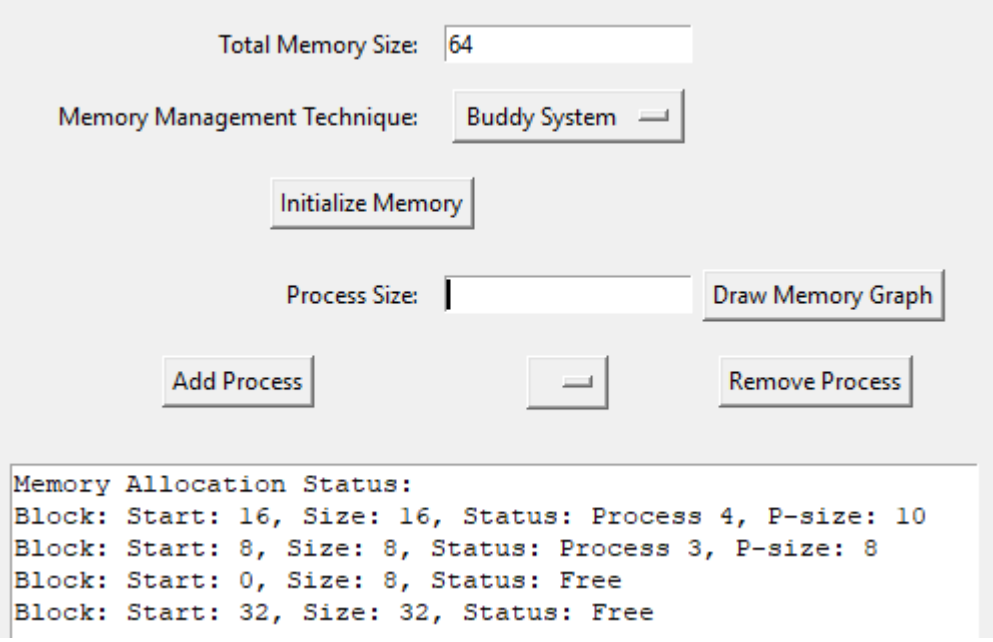
Process 1 of size 2 is the smallest so when the request is made the total block of size 64 is split into two blocks of 32. Again, one of the blocks of 32 into two blocks of 16. Similarly, 8-4 and finally 2, where process 1 is accommodated.

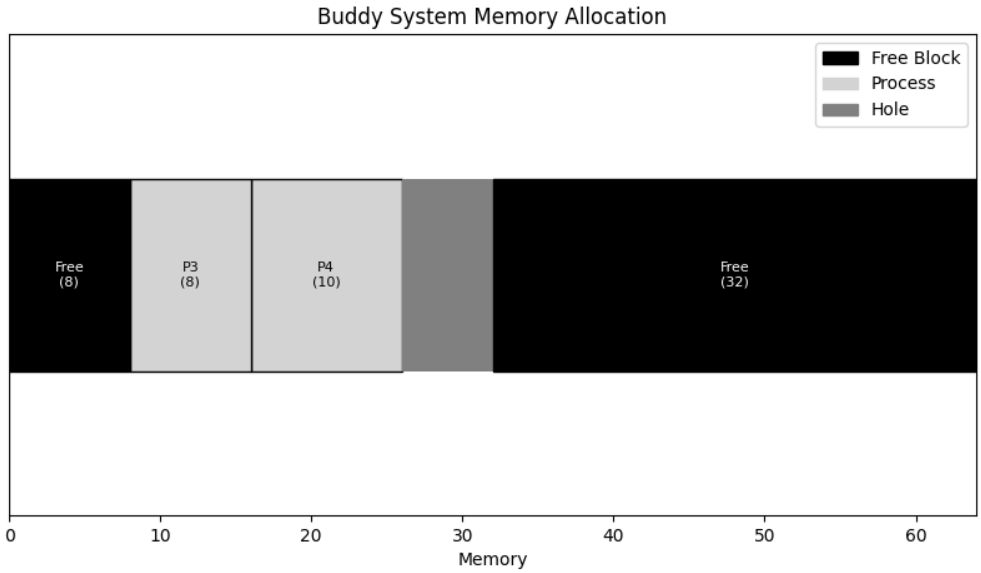
Before Deallocation:





After Deallocation:





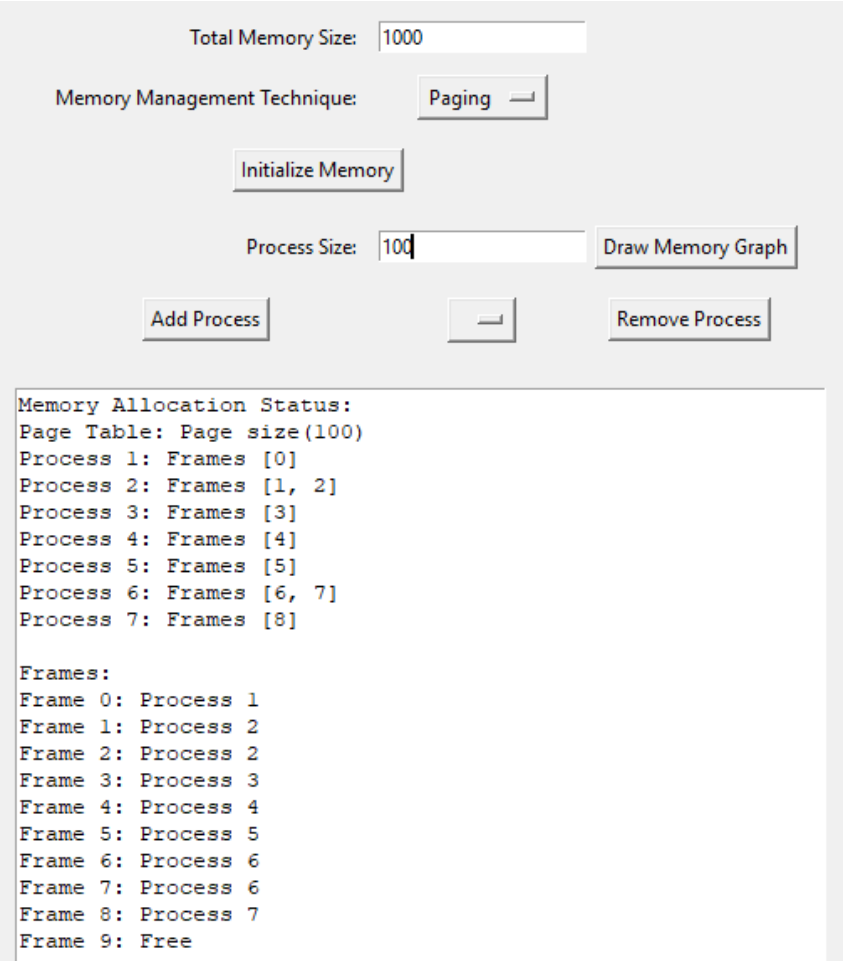
When processes 1 and 3 of size 4 are deallocated, then the free memory of both buddies are merged into the size of 8. Similarly, when process 5 of size 10 is deallocated, then the two buddies of size 16 are merged into the block of size 32.

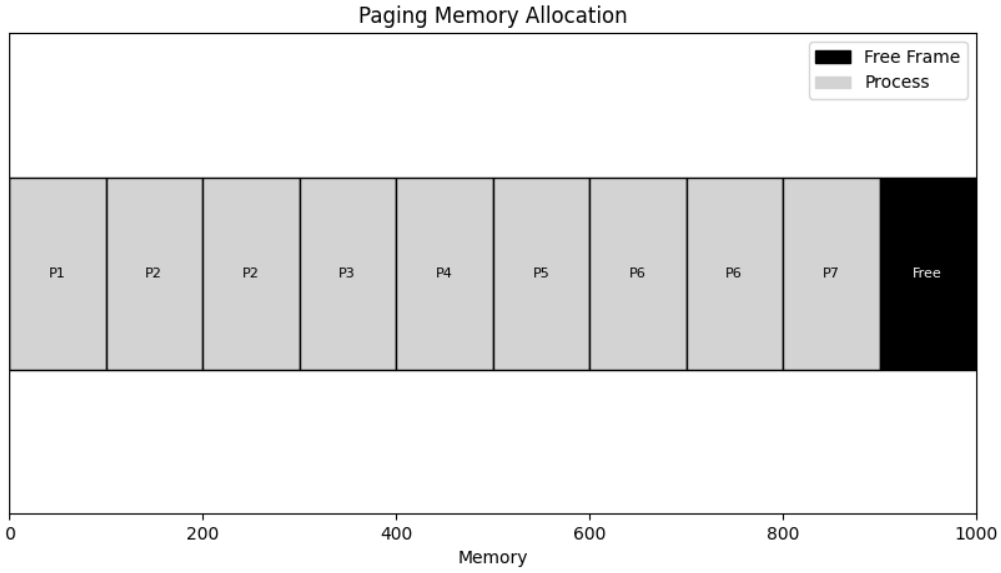
Paging:

Paging is a memory management scheme that eliminates the need for contiguous memory allocation by dividing both physical memory and logical address spaces into fixed-size pages and frames, respectively. When a process runs, its pages are loaded into available memory frames, which can be scattered throughout physical memory. Paging simplifies memory allocation and reduces fragmentation, as pages can be mapped flexibly to any available frame, but it requires a page table to keep track of the mappings, adding overhead.

Allocation:

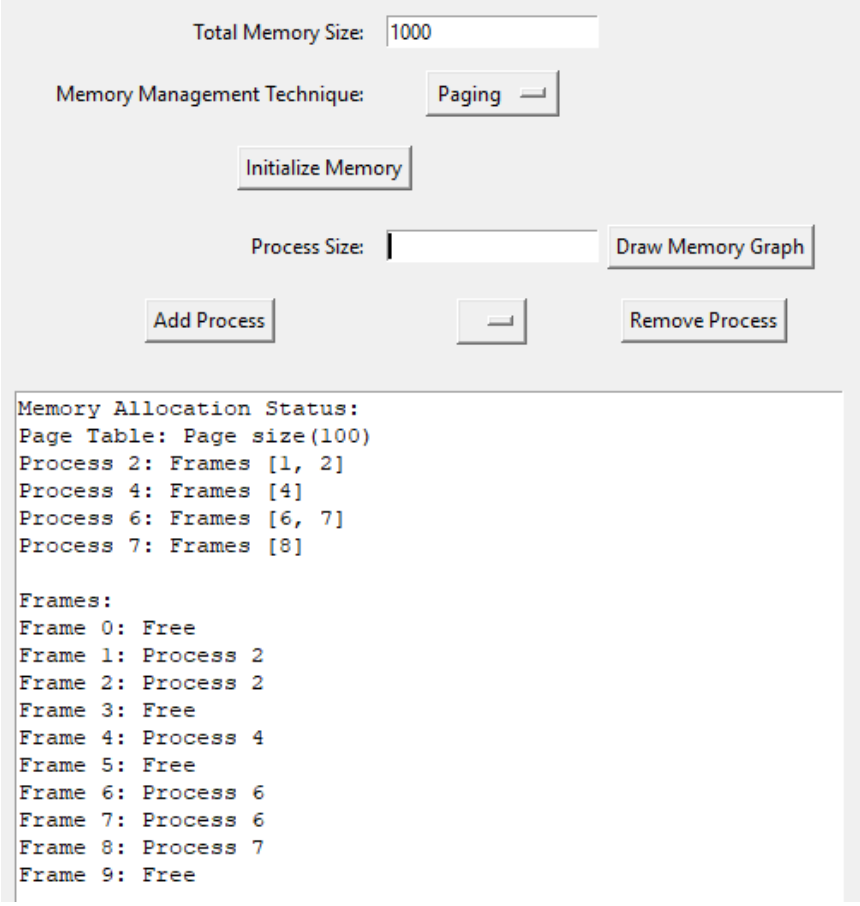
Here, the page size and frame size are of size 100. The processes are placed inside the frame of suitable sizes. There are 10 frames starting from 0-9. The page table shown in the first figure gives the clear idea of the process within the frames.

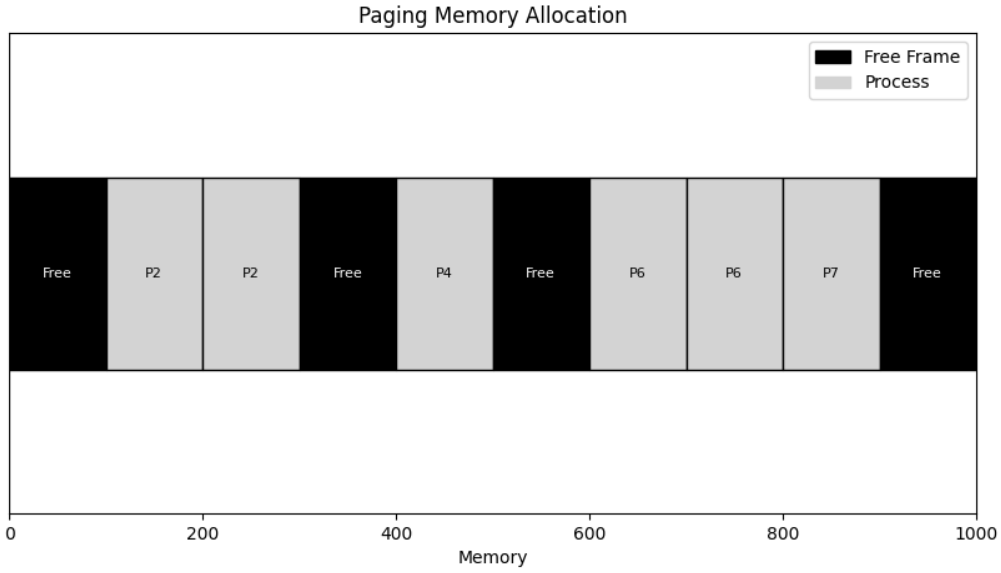




Deallocation:

The process 1, 3 and 5 are deallocated.





The above deallocation of the processes has freed frames 0, 3, 5 and 9. Now if the process of size more than 100 can be allocated to two frames or more. Doing so requires correct implementation of the page table. We have properly implemented the page table for mapping of the processes.

Allocation of the process in scattered frames:

We allocate the processes 8 and 9 a size of 200 so that they are properly loaded into available memory frames, which is scattered throughout physical memory.

In the below figures, we can see that frame 0 and 3 is occupied by process 8 of size 200 and similarly, process 9 has occupied the frames 5 and 9. This shows the implementation of paging even though the frames are scattered across the main memory.

Total Memory Size: 1000

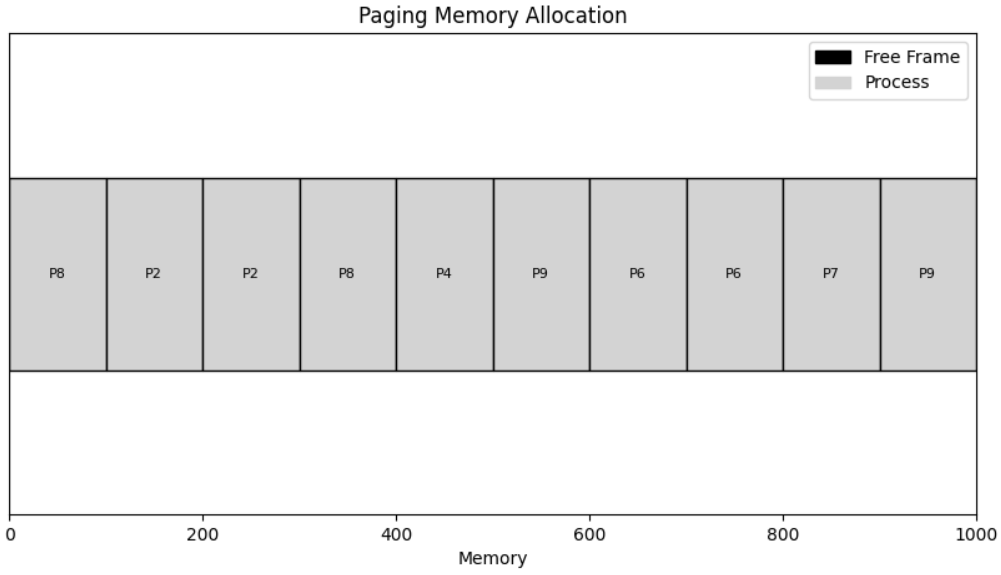
Memory Management Technique: Paging

Initialize Memory

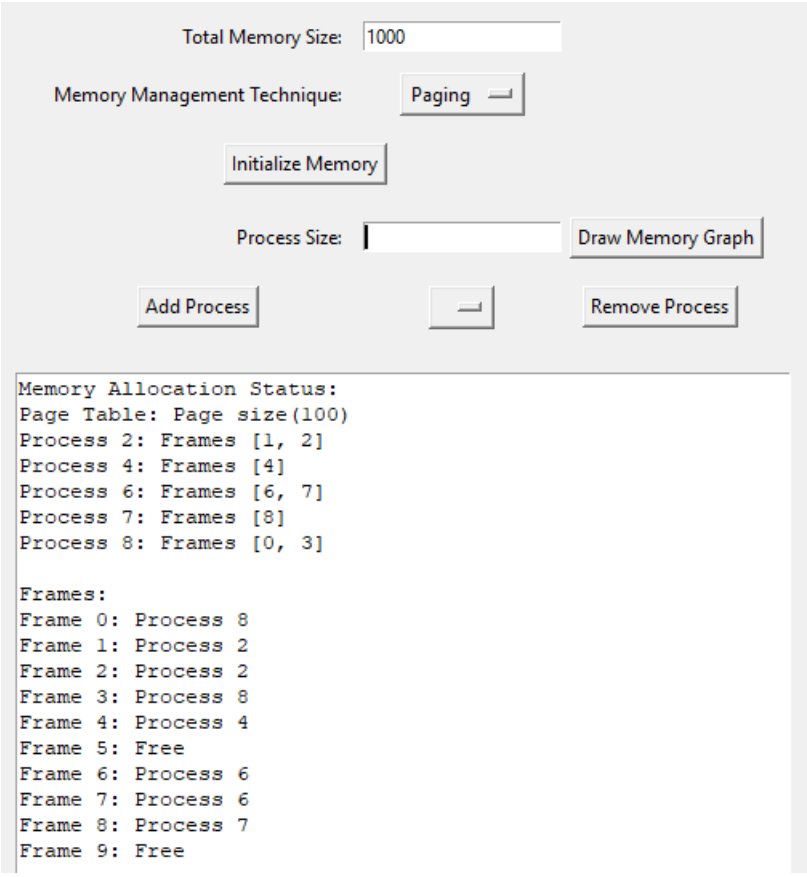
Process Size: 200 Draw Memory Graph

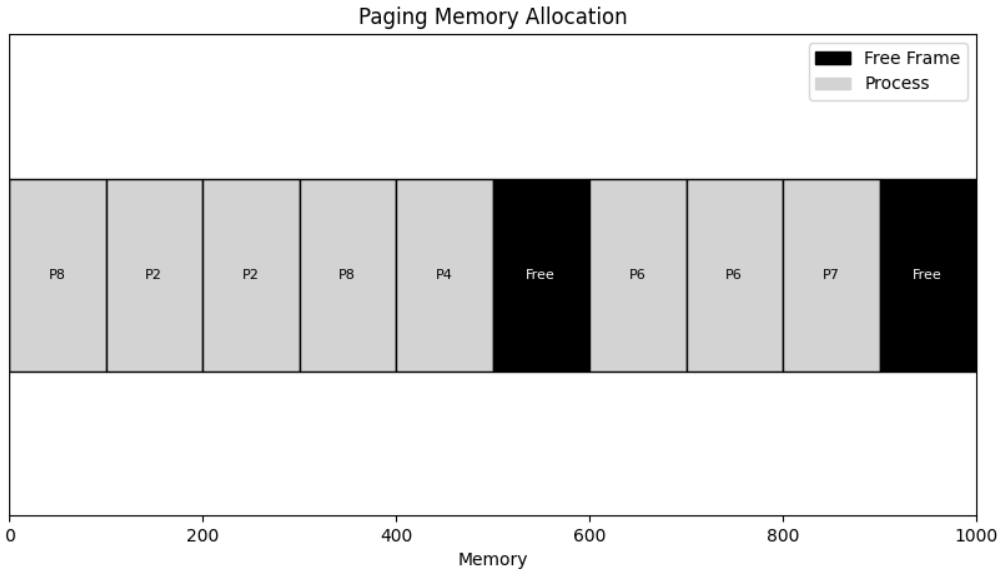
Add Process Remove Process

```
Memory Allocation Status:
Page Table: Page size(100)
Process 2: Frames [1, 2]
Process 4: Frames [4]
Process 6: Frames [6, 7]
Process 7: Frames [8]
Process 8: Frames [0, 3]
Process 9: Frames [5, 9]

Frames:
Frame 0: Process 8
Frame 1: Process 2
Frame 2: Process 2
Frame 3: Process 8
Frame 4: Process 4
Frame 5: Process 9
Frame 6: Process 6
Frame 7: Process 6
Frame 8: Process 7
Frame 9: Process 9
```



Deallocation of the process in scattered frames:





In the above figure, we can see that even if the frame is distributed across the main memory, the process 9 is deallocated correctly which occupied the frames 5 and 9. This is possible due to correct mapping of page table.

Comparison Chart:

Technique	Key Characteristics	Pros	Cons	Best Use Case
Fixed-sized Partitioning	Divides memory into fixed-size partitions	Simple to implement	Internal fragmentation, inflexible	Systems with predictable, uniform workloads
Unequal-sized Fixed Partitioning	Partitions of different sizes	Reduces internal fragmentation compared to fixed-sized	External fragmentation, still somewhat inflexible	Systems with varying process sizes
Dynamic Memory Allocation	Allocates memory dynamically at runtime	Highly flexible, efficient use of memory	Complex management, can lead to fragmentation	Systems with highly variable workloads
Buddy System	Divides memory into power-of-two sized blocks	Reduces fragmentation, efficient merging	May still suffer from fragmentation, complexity in management	Systems with frequent allocation/deallocation
Paging	Divides memory into fixed-size pages and frames	Eliminates external fragmentation, allows non-contiguous allocation	Page table overhead, complexity in management	Modern operating systems, multitasking environments

Conclusion:

Each technique has its advantages and challenges, addressing different aspects of memory management such as fragmentation, allocation speed, and complexity. Overall, effective memory management is essential for maintaining system stability, maximizing performance, and enabling the smooth execution of multiple processes concurrently. Ultimately, the best approach depends on the specific needs and constraints of the system being designed.