**Memory Management: First-Fit and Best-Fit Algorithms**

Arun Saxena

Colorado State University Global

CSC507-1: Foundations of Operating Systems

Dr. Joseph Issa

December 15, 2024

**Introduction**

Memory management is a crucial aspect of operating systems, ensuring efficient allocation and utilization of memory resources. This paper explores the First-Fit and Best-Fit algorithms for memory allocation, providing a comparative analysis of their performance.

**First-Fit Algorithm**

**How It Works**  
 The First-Fit algorithm allocates the first available memory block that can accommodate a process. It is simple and fast but can lead to fragmentation as it leaves small unusable memory blocks.

**Code Implementation**

def first\_fit(memory\_blocks, process\_sizes):

    allocation = [-1] \* len(process\_sizes)

    for i in range(len(process\_sizes)):

        for j in range(len(memory\_blocks)):

            if memory\_blocks[j] >= process\_sizes[i]:

                allocation[i] = j

                memory\_blocks[j] -= process\_sizes[i]

                break

    return allocation

memory\_blocks = [100, 500, 200, 300, 600]

process\_sizes = [212, 417, 112, 426]

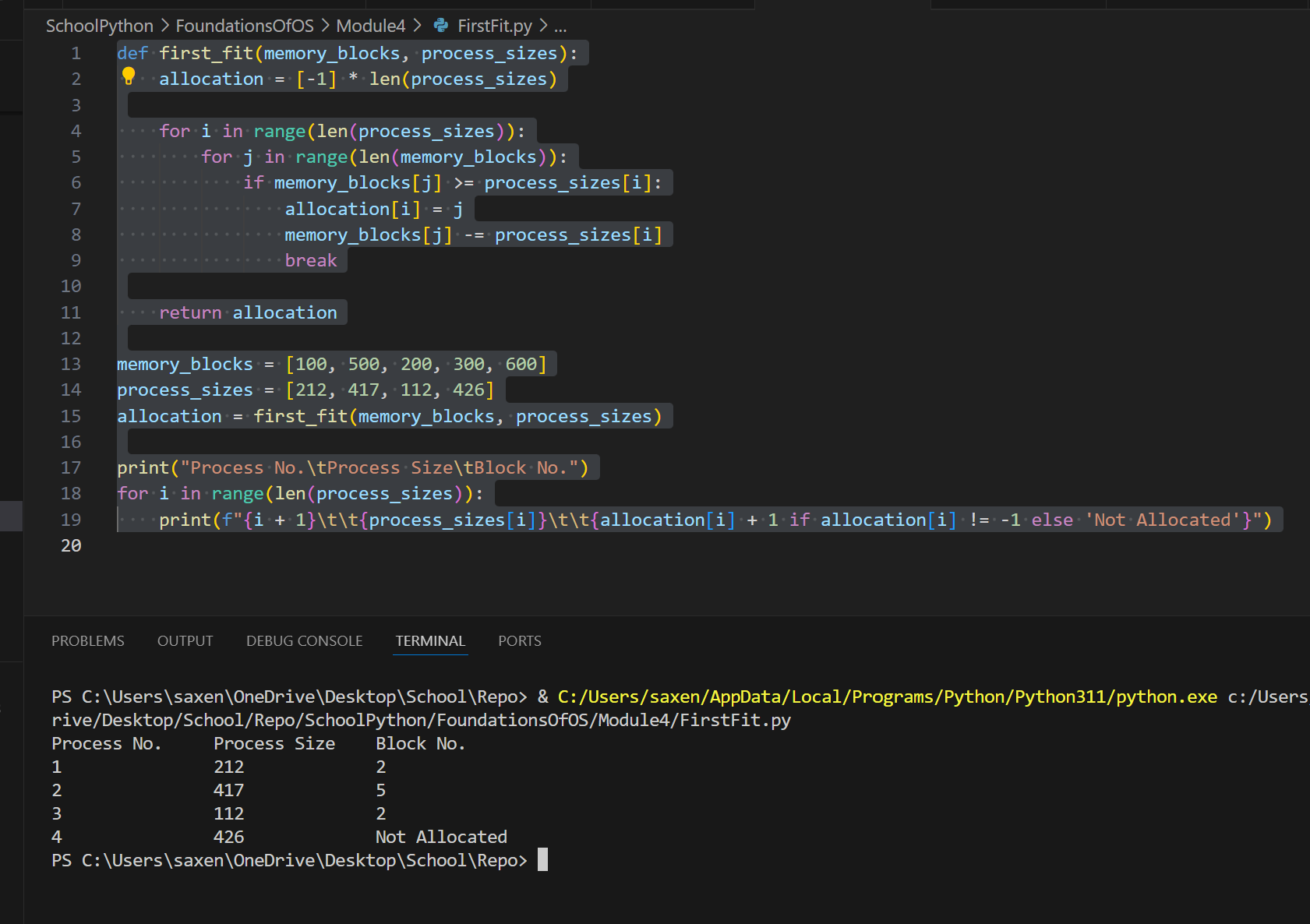
allocation = first\_fit(memory\_blocks, process\_sizes)

print("Process No.\tProcess Size\tBlock No.")

for i in range(len(process\_sizes)):

    print(f"{i + 1}\t\t{process\_sizes[i]}\t\t{allocation[i] + 1 if allocation[i] != -1 else 'Not Allocated'}")

**Execution**



**Best-Fit Algorithm**

**How It Works**

The Best-Fit algorithm allocates the smallest available memory block that can accommodate a process. It reduces fragmentation by minimizing wasted space but is slower than First-Fit as it requires searching for the best block.

**Code Implementation**

def best\_fit(memory\_blocks, process\_sizes):

allocation = [-1] \* len(process\_sizes)

for i in range(len(process\_sizes)):

best\_idx = -1

for j in range(len(memory\_blocks)):

if memory\_blocks[j] >= process\_sizes[i]:

if best\_idx == -1 or memory\_blocks[best\_idx] > memory\_blocks[j]:

best\_idx = j

if best\_idx != -1:

allocation[i] = best\_idx

memory\_blocks[best\_idx] -= process\_sizes[i]

return allocation

memory\_blocks = [100, 500, 200, 300, 600]

process\_sizes = [212, 417, 112, 426]

allocation = best\_fit(memory\_blocks, process\_sizes)

print("Process No.\tProcess Size\tBlock No.")

for i in range(len(process\_sizes)):

print(f"{i + 1}\t\t{process\_sizes[i]}\t\t{allocation[i] + 1 if allocation[i] != -1 else 'Not Allocated'}")

**Execution**

A screenshot of a computer program

Description automatically generated

**Comparison**

* **First-Fit Algorithm**:
  + Simple and fast.
  + Can lead to fragmentation as it leaves small unusable memory blocks.
* **Best-Fit Algorithm**:
  + Reduces fragmentation by minimizing wasted space.
  + Slower than First-Fit as it requires searching for the best block.

**Situations Where Best-Fit is Preferable**

The Best-Fit algorithm is preferable in scenarios where memory fragmentation is a significant concern. For example, in systems with limited memory resources or where processes frequently request and release memory, the Best-Fit algorithm can help reduce wasted space and improve overall memory utilization.

**Conclusion**

Both the First-Fit and Best-Fit algorithms have their advantages and disadvantages. The choice between them depends on the specific requirements and constraints of the system.

**References**

Silberschatz, Galvin and Gagne - https://codex.cs.yale.edu/avi/os-book/OS8/os8j/slide-dir/PDF-dir/ch8.pdf