**ASSIGNMENT 3 - Part1 Simulator**

SYSC 4001 - Operating Systems

December 06, 2024

Ali Al-Ghaib 101211245

Jaydon Haghighi Saed 101206884

Analyses on FCFS, EP and RR:  
  
Input\_data:  
15, 10, 0, 25, 11, 3

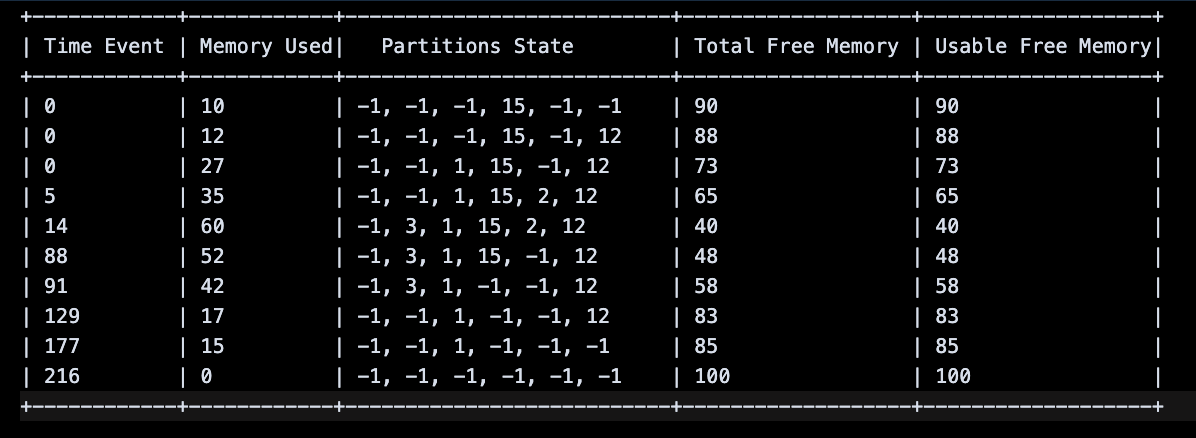
12, 1, 0, 50, 10, 5

1, 10, 0, 100, 25, 25

2, 1, 5, 20, 10, 2

3, 7, 14, 5, 2, 10

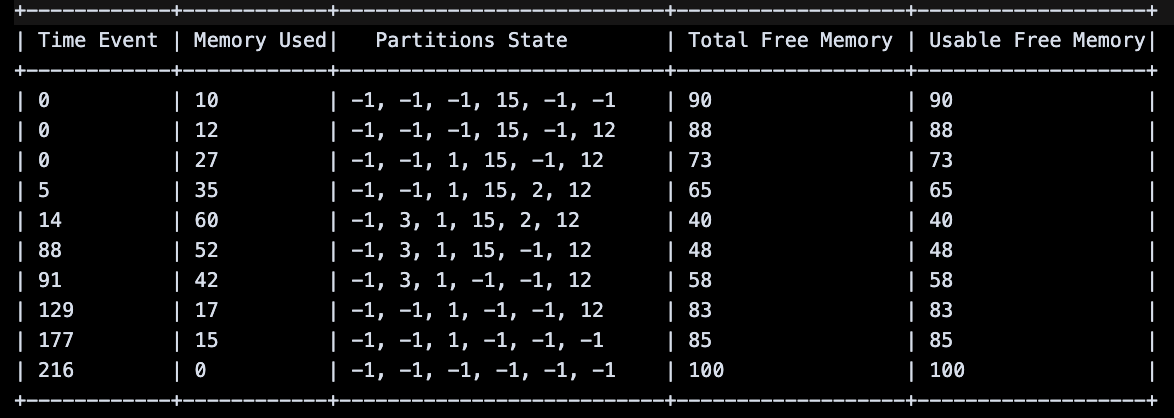
**FCFS Scheduler**



1. Initial State (Time = 0):  
   * Total Memory: 100 units.
   * Memory Used: Starts at 10 units.
   * Partitions State: Indicates several partitions (-1 implies free), with a few partitions (15 and 12) allocated early.
   * Total Free Memory: 90 units, matching the initial free memory.
2. Gradual Increase in Memory Usage:  
   * By Time = 14, the memory usage increases to 60 units, with significant partitions allocated (3, 15, 12).
   * Usable Free Memory decreases due to fragmentation, even though free memory remains (48 units by this point).
3. Memory Release and Reuse:  
   * At Time = 91, memory usage drops to 42 units, and free memory increases to 58 units.
   * This shows that the FCFS scheduler is releasing memory when processes complete but struggles to efficiently reuse free memory due to fragmentation.
4. End State (Time = 216):  
   Memory usage returns to 0, and all partitions are marked as free (-1).
   * Total Free Memory returns to 100 units, showing complete memory recovery.
5. Fragmentation Issues:
   * While total free memory fluctuates, the usable free memory matches the total free memory, indicating minimal internal fragmentation.
   * However, external fragmentation becomes apparent as partitions are not merged when memory is released.
6. Low Memory Utilization:
   * During periods of high memory demand (Time = 14, Memory Used = 60), 40% of memory remains free, but not all of it is usable for large processes due to fragmentation.
7. FCFS Limitations:
   * FCFS allocates memory in the order requests arrive, without considering if a more optimal partition (a better fit) could reduce fragmentation.
   * Memory release timing (Time = 91) shows that earlier processes limit flexibility for later allocations.

The FCFS scheduler does an adequate job of handling memory requests, but it is prone to external fragmentation and inefficient memory utilization. While the table shows that all memory is eventually recovered, significant improvements could be made by adopting more advanced allocation strategies or memory management techniques.

**External Priority Scheduling**



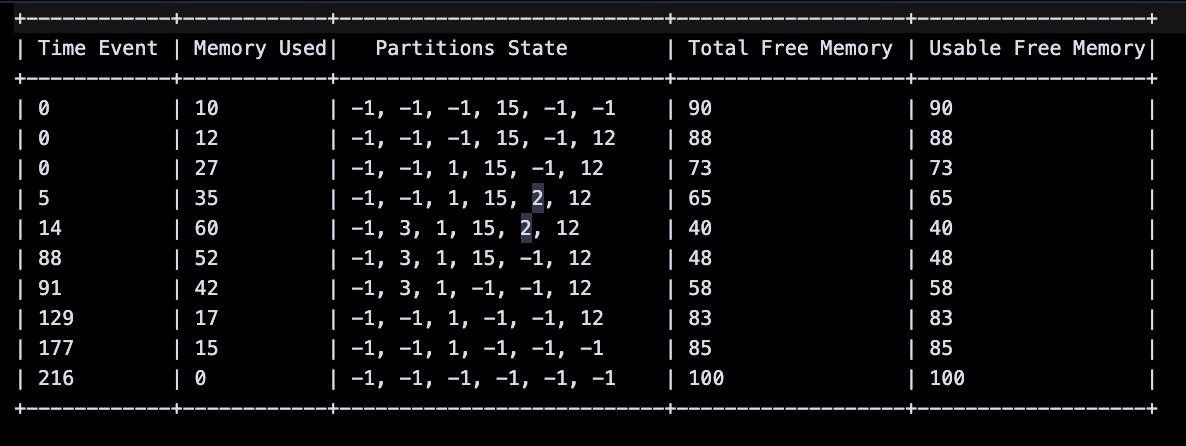
Initial State (Time = 0):

* Memory Used: Starts at 10 units.
* Partitions State: Mostly free (-1), with only a small allocation.
* Total Free Memory: 90 units, reflecting minimal usage.

Memory Usage Over Time:

1. Time = 14:
   * Memory usage increases to 60 units, with priority processes allocated (3, 1, 15, 2, 12).
   * Total Free Memory drops to 40 units, but usable memory is still efficient due to prioritization.
2. Time = 88:
   * Memory usage decreases to 52 units, reflecting the completion or removal of some processes.
   * Total Free Memory rises to 48 units, but the fragmentation of free partitions becomes more noticeable, as scattered partitions limit usability.
3. Time = 91:
   * Memory usage further decreases to 42 units as additional processes are completed.
   * Total Free Memory increases to 58 units, but fragmentation persists.
4. Time = 129:
   * Memory usage drops significantly to 17 units, with most processes completed or deallocated.
   * Partitions State shows only a few active allocations, and free memory increases to 83 units.
5. Time = 216 (Final State):
   * Memory usage returns to 0 units, with all partitions cleared (-1).
   * Total Free Memory and Usable Free Memory return to 100 units, indicating full memory recovery.
6. Efficient High-Priority Handling:  
   * High-priority processes are allocated memory promptly, ensuring critical tasks are prioritized and completed first.
   * This results in efficient utilization of available memory during peak demand.
7. Fragmentation:  
   * As memory is released over time, external fragmentation becomes evident. Free memory is available but scattered, limiting its usability for larger processes.
8. Recovery of Memory:  
   * Memory is gradually freed as processes complete. By the end, the memory is fully recovered, showing effective deallocation.
9. Impact on Lower-Priority Processes:  
   * Lower-priority processes are delayed or preempted in favor of higher-priority tasks. While this ensures fairness for critical processes, it reduces throughput for less important tasks.

External Priority Scheduling efficiently allocates memory to high-priority processes, ensuring critical tasks are addressed first. However, fragmentation and delays for lower-priority processes reduce overall memory efficiency. Implementing compaction and balancing priority adjustments could improve performance and memory utilization.

**Round Robin Scheduling**  


Round Robin is a preemptive scheduling algorithm where each process is assigned a fixed time quantum (100 in this case). If a process does not finish within its allocated quantum, it is preempted and placed at the back of the queue, ensuring fairness. This algorithm emphasizes equal allocation of CPU time but can impact memory usage depending on process behavior.

Initial State (Time = 0):

* Memory Used: Starts at 10 units.
* Partitions State: Mostly free partitions (-1), with only a small allocation.
* Total Free Memory: 90 units, reflecting the initial memory availability.

Memory Usage Over Time:

1. Time = 14:
   * Memory usage increases to 60 units, reflecting multiple process allocations (3, 1, 15, 2, 12).
   * Total Free Memory drops to 40 units, showing significant memory usage.
2. Time = 88:
   * Memory usage reduces to 52 units, with some partitions freed.
   * Processes preempted by the quantum return to memory in subsequent cycles, maintaining active memory use.
3. Time = 91:
   * Memory usage drops further to 42 units, as more processes complete or are preempted and do not return to memory.
   * Total Free Memory increases to 58 units, indicating better memory recovery.
4. Time = 129:
   * Memory usage reduces significantly to 17 units, with most processes completing their cycles.
   * Partitions State shows most partitions freed except for a few small allocations.
5. Time = 216 (Final State):
   * Memory usage returns to 0 units, with all partitions marked as free (-1).
   * Total Free Memory and Usable Free Memory return to 100 units, reflecting full recovery.
6. Fair Allocation:
   * Round Robin ensures fairness in process execution by cycling processes through memory. This results in consistent memory usage without favoring specific processes.
7. Fragmentation:
   * External fragmentation is evident, as shown by the scattered free partitions in the Partitions State column. While total free memory increases over time, fragmentation limits usable memory for larger processes.
8. Memory Recovery:
   * Processes are preempted and removed as their cycles complete, ensuring that memory is gradually freed.
   * By the final time (216), memory is fully recovered, showing effective deallocation.
9. Impact of Time Quantum:
   * The large quantum (100) allows processes to complete significant work before being preempted, reducing the frequency of memory swaps.
   * However, this could delay lower-priority processes if higher-priority ones dominate early cycles.

Round Robin scheduling ensures fair memory usage by cycling processes through fixed time quanta. While memory is well-recovered by the end, fragmentation limits efficiency during peak usage. Compaction and adaptive quantum adjustments could enhance overall performance and memory utilization.

**Comparison between FCFS, EP and RR:**

**First-Come, First-Served (FCFS):**

FCFS processes memory requests in the order of arrival, making it simple to implement. However, it does not prioritize based on process importance or memory efficiency, often leading to external fragmentation and suboptimal memory utilization. Low-priority or long-running processes can block others, reducing overall throughput.

**External Priority (EP):**

EP prioritizes high-priority processes, ensuring critical tasks are handled promptly. While this improves response times for essential tasks, it often delays lower-priority processes, potentially causing starvation. Additionally, EP suffers from fragmentation due to the prioritization of smaller partitions for high-priority tasks.

**Round Robin (RR):**

RR assigns a fixed time quantum (100 here) to each process, ensuring fairness by cycling through all processes. This avoids starvation and balances CPU time among processes. However, frequent context switching can lead to overhead, and fragmentation may occur if processes are preempted before fully completing their tasks.

Conclusion:

* FCFS is fair in order but inefficient due to fragmentation and lack of prioritization.
* EP ensures high-priority tasks are completed quickly but delays others and struggles with fragmentation.
* RR provides fairness and prevents starvation but introduces overhead and may suffer from fragmentation if memory allocation is not compacted effectively.