ADVANCED OPERATING SYSTEMS

CSEN383

PROJECT-4 (Page Replacement Simulator)

Group – 3

-Aman Jain, Amarnath S Kaushik, Daryl Ho, Meghana Kale, Sai Kiran Jasti

Structs:

1. **Job**
2. **Page**
3. **Memory**

Functions:

1. **void generateWorkload(JobQueue *jobQueue)**
   This function initializes a job queue and generates MAX_JOBS jobs with random attributes. Each job is assigned a unique process name (cycling through A-Z), a random process size (chosen from 5, 11, 17, or 31 pages), a random arrival time (within half of the simulation time), and a random service duration. The jobs are dynamically allocated and inserted into the queue in ascending order of arrival time, ensuring that earlier-arriving jobs are processed first. Finally, the function updates the total job count in the queue.

2. **void runSimulation(JobQueue *jobQueue, Memory *memory, int (*replacementAlgorithm)(Memory *, char, int, int), char *algorithmName, double *hitRatioSum, double *missRatioSum, int *swappedInSum)**
   The runSimulation function simulates job execution and memory management using a specified page replacement algorithm, tracking page hits, misses, and swapped-in processes. It initializes counters, maintains an active job list, and ensures jobs start only if sufficient memory is available. The simulation iterates through time steps, processing page references for active jobs while checking memory for hits or triggering page replacements

on misses. Completed jobs release their allocated memory, and results such as hit ratio, miss ratio, and process swaps are computed and displayed. The function provides insights into the efficiency of different page replacement strategies under simulated workload conditions.

3. **void* seller_thread(void* arg)**
   This function simulates a seller processing customers in a queue while tracking service metrics. Each seller runs while global time is within simulation time, serving customers whose arrival times have passed. If a seat is assigned via assign seat, the function simulates service time (sleep), updates response and turnaround times, and tracks served or turned-away customers using mutex-locked statistics. The queue is then adjusted by shifting customers forward. Successful transactions print the updated seating chart, ensuring real-time simulation updates.

Page Replacement Algorithm Averaged Output after 5 runs:

1. **FIFO**

Statistic	**Value**
Hit	**49%**
Miss	**51%**
Page Swaps	**3012**

2. **LRU**

Statistic	**Value**
Hit	**52%**
Miss	**48%**
Page Swaps	**2873**

3. **LFU**

Statistic	**Value**
Hit	**54%**
Miss	**46%**
Page Swaps	**2769**

4. **MFU**

Statistic	**Value**
Hit	**46%**
Miss	**54%**
Page Swaps	**3227**

5. **Random Pick**

Statistic	**Value**
Hit	**50%**
Miss	**50%**
Page Swaps	**2999**

Conclusion:

The simulation results provide valuable insights into the effectiveness of different **page replacement algorithms** in managing memory efficiently. Among the tested algorithms, **LFU (Least Frequently Used)** demonstrated the best performance, achieving the highest **hit ratio (54%)** and the lowest **number of page swaps (2769)**, indicating that prioritizing less frequently used pages for replacement leads to better memory utilization. **LRU (Least Recently Used)** also performed well, with a **52% hit ratio** and **2873-page swaps**, proving effective in handling memory requests by replacing the least recently accessed pages.

On the other hand, **MFU (Most Frequently Used)** performed the worst, with the lowest **hit ratio (46%)** and the highest **page swaps (3227)**, suggesting that evicting the most frequently used pages disrupts the memory access pattern, leading to higher page faults. **FIFO (First-In-First-Out)** and **Random Pick** had moderate performance, both yielding around **50% hit ratio**, but FIFO suffered slightly higher misses, likely due to its inability to consider access frequency or recency.

Overall, **LFU and LRU proved to be the most efficient**, effectively reducing page faults and minimizing memory swaps, while **MFU's poor performance makes it unsuitable for optimizing memory management**. The results emphasize that choosing an appropriate page replacement strategy significantly impacts system performance, with algorithms that consider frequency and recency of access generally providing better results.