Text

Description automatically generated with medium confidence

**CPEN 452 Operating Systems**

Simulation of Operating System Algorithms for Process Management, Memory Page Replacement, and Disk Scheduling using C++ Program

School of Engineering

Department of Computer Engineering

Carlos Roque

Faustino R. Fuertes Millán

Introduction

Simulating operating system algorithms for process management, memory page replacement, and disk scheduling using C++ program is essential to understand the behavior of these algorithms under different conditions. This understanding can help system designers to make informed decisions about selecting the appropriate algorithms for specific system requirements. Moreover, simulating these algorithms can also aid in the identification of potential performance bottlenecks, allowing for the development of more efficient algorithms to overcome these limitations.

The simulation of OS algorithms has numerous applications. For example, it can be used to evaluate the performance of an OS in a virtualized environment, where multiple operating systems are running on a single physical machine. It can also be used to simulate the behavior of different scheduling algorithms in real-time operating systems, where the timeliness of response is critical. Additionally, it can be used to evaluate the effectiveness of memory page replacement algorithms in a virtual memory system, where the available memory is limited.

Summary

Project Design

1. Data dictionary diagram

|  |  |
| --- | --- |
| Data | Description |
| calculateNonPreemptive | Calculate Waiting Time with Non-preemption.This function will caculate the waiting time for each process sequentially in the order that they are passed.Will return the waiting time |
| findWaitingTimeRoundRobin | Function to calculate waiting times of processes. |
| Process | Struct for each node in a linked list |
| generateProcessList | Generate Process List: This method will generate a LinkedList by first asking the user how many Nodes to add (i.e Processes to add) and then for each, ask the user for the burst time It first creates a head node as the starting point.Will return head. |
| printSubMenu | Print the sub menu.After the user has selected the option from the first menu, this method will be called and it will hold a string array that contains the names of the algorithms that can be used. Return selection. |
| printMenu | Print Main Menu. This is the first menu that will be shown to the user for selection. It will hold 3 options where each has their own correspondent algorithm for processing. Return NULL |
| dFCFS | Function to calculate the total number of head movements and print the sequence of movements. Return total\_values |
| sstf | Selects the next request to be serviced based on the shortest seek time from the current head position. |
| cscan | Takes an array of disk requests as input and returns the total number of disk movements required to service all the requests. Starting from the head position and moving towards the smallest request. Return movements. |
| scan | The algorithm works by first setting the initial head position as the first request in the array, and then sorting the array in ascending order.Starting from the head position and moving towards the largest request. Return movements. |
| generateNewList | to create lists of processes |
| deleteLinkedList | to delete lists of processes |
| firstComeFirstServe | This method will receive a linked list in order as presented and return an array, or rather a pointer to the first element in the array that contains the waiting time for each process. |
| shortestJobFirst | This function will calculate the waiting time of each process in a linkedlist: It will first take the arrival time and the burst time to decide which processwill go first and at what time (ms) the dispatcher will step over to the second process considering the size of its burst time |
| priorityScheduling | This function will receive a Sorted linked list of process in the order based off the lowest priority among them |
| roundRobin | This function will calculate the waiting time and turnaround time of a process it receives from a linked list and will schedule them using Round Robin considering the Quantum time |
| sortByPriority | ort Processes by Priority This method will receive an unsorted linked list and reorder it based on their priority. The algorithm used for sorting is Selection Sort, which will take the smallest number in the linked list and will swap it with the first element in the linked list. Best and worst case for it is O(n^2), unless the list is already sorted in which case the time complexity is O(n) |
| sortByburstTime | Sort a linked list of processes based on their burst time in ascending order.Return head; |
| processReconstruct | This function takes a parameter of type process which is a singly linked list by default. The function will then reconstruct it so that it can become a Circular Doubly Linked List of processes.Return head. |
| calculatePageFaultsFIFO | This function takes the number of frames and a linked list of page references as input and returns the number of page faults that occurred using the FIFO page replacement algorithm. |
| calculatePageFaultsLRU | This function takes the number of frames and a linked list of page references as input and returns the number of page faults that occurred using the LRU page replacement algorithm. |
| findOptimalPageToReplace | This function takes a map of page access positions and a set of pages currently in frames and returns the page that should be replaced based on the Optimal algorithm. |
| calculatePageFaultsOptimal | Calculate the number of page faults using the Optimal Page Replacement algorithm.This function takes the number of frames and a linked list of page references as input.and returns the number of page faults that occurred using the Optimal page replacement algorithm |

1. UML(OOP) diagrams and data flow diagrams (structured programming)

UML DIAGRAM:

Diagram, schematic

Description automatically generated

Development

Links for replit files of the code:

This main function wich will be excecuting and getting other header files.

* <https://replit.com/@Talio_/ProyectoOS#main.cpp>

This is the calculate.h header file wich have the function calculateNonPreemptive and findWaitingTimeRoundRobin

* <https://replit.com/@Talio_/ProyectoOS#calculate.h>

The disk.h header file will have all the functions needed to implement for the disk request management

* <https://replit.com/@Talio_/ProyectoOS#disk.h>

The memory.h header file will have implemented the functions for the calculations of the page replacements and calculates the page faults.

* <https://replit.com/@Talio_/ProyectoOS#memory.h>

The menu.h will have the menus that we need to be printed for the user to choose.

* <https://replit.com/@Talio_/ProyectoOS#menu.h>

This scheduling.h file will have functions for the process management scheduling

* https://replit.com/@Talio\_/ProyectoOS#scheduling.h

The sortProcesses.h file has thee function that implements the sorting using the priority or the burst Time for the process scheduling.

* https://replit.com/@Talio\_/ProyectoOS#sortProcesses.h

Results discussion

Processes:

FSFS:

The FCFS algorithm simply runs processes in the order in which they arrive. The waiting time for each process is the sum of the burst times of all previous processes, and the turnaround time is the sum of the burst time and waiting time. The function first calculates the burst times for each process and passes them to a separate function, calculateNonPreemptive, which returns an array of waiting times. The waiting times and turnaround times are then stored in the Process struct. Finally, the function outputs the results, including the average waiting time and turnaround time for all processes.

Text

Description automatically generated

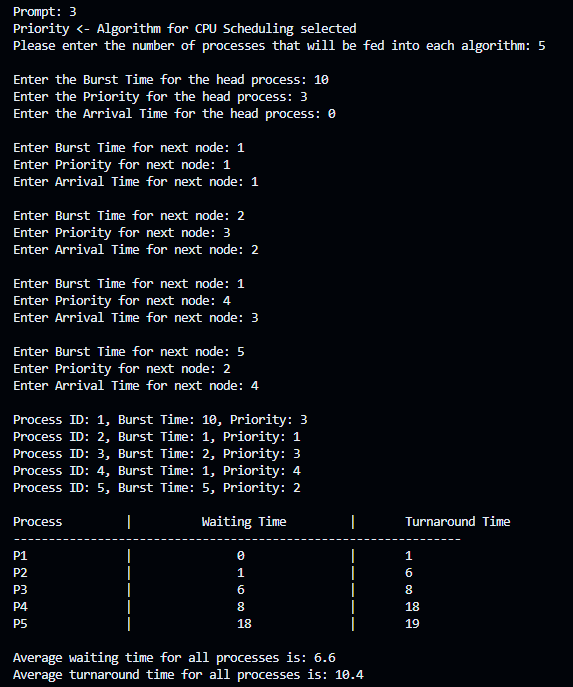
SJF:

The SJF algorithm implemented in the provided code takes in a linked list of processes, sorts them by arrival time, and then iterates through the list of processes. It finds the process with the shortest remaining burst time that has arrived, executes it for one unit of time, and updates its remaining burst time. If a process has completed execution, its waiting time and turnaround time are calculated and stored.The function outputs a table of each process's waiting time and turnaround time, as well as the total and average waiting time and turnaround time for all processes.Text

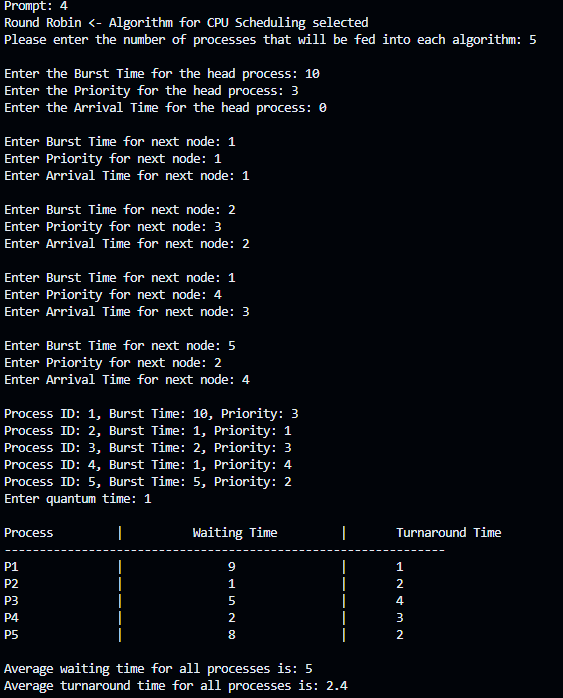
Description automatically generated with low confidence

Priority:

This implements priority scheduling for a linked list of processes. It first sorts the list by priority, then calculates the waiting times.The waiting times and turnaround times are stored in the process struct. Finally, the processes are sorted by their process ID and the results are outputted, including the average waiting and turnaround times for all processes.



RoundRobbin:

This algorithm implements the round-robin scheduling algorithm for a linked list of processes. It prompts the user to enter the quantum time and then calculates the waiting time for each process using the round-robin algorithm. It also calculates the total waiting time and turnaround time for all processes, then displays the waiting time and turnaround time for each process and the average waiting time and turnaround time for all processes.

Memory

FIFO:

This algorithm calculates the number of page faults that occur using the First-In-First-Out (FIFO) page replacement algorithm. It takes in the number of frames available for storing pages, and a linked list of page references.

Text

Description automatically generated

OPTIMAL:

This algorithm implements the Optimal page replacement algorithm, which attempts to minimize the number of page faults by replacing the page that will not be used for the longest period of time in the future.

Text

Description automatically generated

LRU:

This is an implementation of the Least Recently Used (LRU) page replacement algorithm.The function takes in the number of available memory frames and a linked list

of page references, and returns the number of page faults that occurred during the simulation.

Text

Description automatically generated

Disk:

FIFO:

This code implements the disk First-Come-First-Served (dFCFS) algorithm, which serves disk I/O requests in the order they are received.The algorithm takes an array of requests as input, where the first element is the current position of the disk head and the remaining elements are the positions of the disk sectors that need to be accessed.

Text

Description automatically generated

SSTF:

This is an implementation of the SSTF (Shortest Seek Time First) disk scheduling algorithm. It takes an array of disk requests as input verifying the most its near the head number and returns the total number of disk movements required to service all requests.

Text

Description automatically generated

SCAN: This is an implementation of the SSTF (Shortest Seek Time First) disk scheduling algorithm. It takes an array of disk requests as input and moving in ascending order from the head and returns the total number of disk movements required to service all requests.

Text

Description automatically generated

CSCAN:

This implementation takes an array of request and sorts it in ascending order divide it in two so it can take the edges in countAnd calculate the movements.

Text

Description automatically generated

Conclusion

In conclusion, the C++ project successfully simulates the algorithms used by the operating system to manage processes, memory (page replacement), and disk scheduling. The program presents a clear and consistent menu for selecting between different types of algorithms and displaying the results obtained. Moreover, a submenu is provided for each category:

a. Processes: FCFS, SJF (Shortest Job First Scheduling, Preemptive), Priority (Preemptive, user-defined priorities), Round Robin (RR, user-defined quantum). The program requests the number of processes (maximum of 10), the information for each process (burst time, arrival time), and allows the user to choose whether to run all algorithms or one in particular. The results clearly display the waiting time and turnaround time for each process, as well as the average waiting time and average turnaround time. The program uses linked lists as the required data structure.

b. Memory: LRU (Least Recently Used), Optimal, FIFO (First In - First Out). The program requests the number of frames and the reference string (maximum of 20 pages). For all algorithms, the program displays the number of page faults, assuming demand paging. Linked lists are used as the required data structure.

c. Disk Scheduling: FCFS (First Come, First Served), SSF (Shortest Seek First), Scan (elevator algorithm), C-Scan. The program requests the initial position and the queue of requests (10 requests). For all algorithms, the program clearly displays the movement chain and the number of movements made. There is no specific requirement for the use of data structures in this category.

References