Selfish Round Robin Scheduling Algorithm

7SENG012W Software Development Environments

Ben Hogg
George Charalambous
December 2, 2023

Requirements

Prioritized Requirement Table

Requirement	Details	Priority Level
Data Structuring	Data file must be parsed and mapped to its respective arrays, creating a mechanism to access, track, and modify process values.	High
Process Queing	Processes must be allocated, removed and reallocated to appropriate queues when arrival and priority criteria are met. Processes must leave CPU service when they finish or moved to the back of the queue if their quanta expires.	High
Process Prioritization	Process must increment their priority based on the queue they reside in.	High
Process Service	Processes in service must decrement their service for each time slice that they hold the CPU.	High
Process Status	Process status must reflect the state of the process, based its arrival and position in the queues. Processes not in the system must be marked with a "-", processes waiting to use the CPU must be marked with a "W", processes running must be marked with an "R", and processes that have completed must be marked with an "F".	High
Program Termination	Program must terminate when all processes have finished and no longer require service.	High

Requirement	Details	Priority Level
Inputs	Users must be able to submit data files and choose settings such as quanta, queue priorities and output configuration.	Medium
Output	Output must be determined by and match user choice. Users may choose to display output on screen, save to a file, or both.	Medium
Displays	Displays such as error reports, program settings, and program output must be clear and organized.	Medium
Validate Inputs	User inputs such as parameters, data file, and output choice must be validated to avoid errors. Data file must be a regular file. Number of parameters must be three, with an option for a fourth. Output choice must be a valid option.	Low

Design

Parameter Validation and Storage

This module serves the critical function of validating user inputs to ensure programmatic integrity and avoiding potential anomalies and instability. It also organizes and labels user inputs, including the data file and parameters, into more intuitive variable names.

i. Check for the correct number of parameters; error if more than 4 parameters.
ii. Ensure the data file is a regular file; display an error if not.
iii. Verify inputs as valid integers; show an error if not.
iv. Store parameter 1 as $dataFile, parameter 2 as $newIncrement, parameter 3 as $acceptedIncrement
v. If number of parameters is 3, store default value of 1 as $quanta. If number of parameters is 4, store input as $quanta.

Array and Data Storage Design

This module organizes data from the file into arrays for data processing. The vital $referenceIndex array stores elements for queue allocation, acting as both a dynamic representation of processes in the queues, as well as a key index to access, display, and modify process variables across arrays. Within these arrays, all sizes are consistent, aligning with the number of processes in the system (n). Notably, $newQueue is designated for processes waiting to be serviced, while $acceptedQueue represents processes in line to undergo service.

i. Create array [n] $name: allocate process names from data file.

ii. Create array [n] $service: allocate NUT value from data file.

iii. Create array [n] $arrival: allocate arrival time value from data file.

iv. Create array [n] $priority: default to 0.

v. Create array [n] $status: default to '-'.

vi. Create array [n] $referenceIndex: Integers 0 to n.

vii. Create array [n] $newQueue: leave empty.

viii. Create array [n] $acceptedQueue: leave empty.

ix. Create array [n] $quantaArray: $quanta.

Display Settings

This (optional) module enhances the user interface by presenting input values and data file content systematically for user review before program execution.

i. Display the content of $dataFile, $newIncrement, $acceptedIncrement, and $quanta.

ii. Display concatenation of $dataFile.

Handling Output Choice

This module allows users to choose their preferred output mechanism (on screen, saved to file, or both) and validates it.

i. Validate $choice as a number between 1 and 3.

ii. If 2 or 3 is chosen, user names the file and store in $fileName.

iii. Wrap in a while loop with error and retry message.

Main Loop Conditions

Representing the program's primary control structure, this loop iterates until all processes conclude, driven by the $time variable and the status of processes stored in the $status array.

i. Initialize $time to 0 outside loop.
ii. Run loop until all $status elements are “F”.

Removing Finished Processes

This module systematically removes completed processes from active arrays, preventing concluded processes from affecting ongoing computations and cleaning the array of empty elements.

i. Loop through entire acceptedQueue
ii. If service[element] is 0; Set status to “F” and remove the element.

Match for Arrival Time

This module assigns arriving processes to either an immediate position in $acceptedQueue or a waiting state in $newQueue.

i. For loop over $referenceIndex array.
ii. If process arrival equals current time or if the $acceptedQueue[*] is empty;
iii. If $acceptedQueue[*] is empty; Allocate to $acceptedQueue and set status to “R”.
iv. Else; Allocate to $newQueueUpdate[n-1] and update to “W”.

Incrementing Priorities

This module augments process priorities in $newQueue and $acceptedQueue.

i. Create two independent for loops; $newQueue and $acceptedQueue. Logic will be the same for both.
ii. If $element is an integer value; *(ensures program integrity)*
iii. Access $priority[$element] and increment by $newIncrement or $acceptedIncrement respectively.

Matching Priorities

This module facilitates migration of processes from the $newQueue to the $acceptedQueue based on priority level.

i. If $newQueue and acceptedQueue are not empty; create a for loop and a nested for loop. The outer for loop iterates the $newQueue and the inner iterates the $acceptedQueue.
ii. If processes in $newQueue has equal or greater priority than any process in the $acceptedQueue; add process to the $acceptedQueue and remove from $newQueue.
iii. Create an independent if statement: If $acceptedQueue is empty and $newQueue is not empty; add $newQueue[0] to $acceptedQueue and remove from $newQueue. *(for edge cases where there are no processes in the accepted queue to evaluate)*

Servicing the Leading Process

Servicing the foremost process within $acceptedQueue, this module manages alterations to process status, quanta allocation, and service time.

i. If $acceptedQueue is not empty;
ii. Decrement the process $service and $quantaArray values.
iii. Update the process status to “R”.

Handling Output

This module discerns between on-screen presentation and file storage depending on user’s choice.

i. If $time equals 0; Echo a banner with “T” followed by the $name array
ii. Echo $time follow by $status array on all.
iii. Use if statements to send output to console or save to $fileName.

Completing a Time Slice

At the end of each time slice, this module creates the movement of the leading process to the back of the $acceptedQueue, contingent on quanta allocation.

i. If acceptedQueue is not empty and the $quantaArray[element] equals 0;
ii. Update $quantaArray[element] with the value of $quanta.
iii. Move acceptedQueue[0] to acceptedQueue[n-1].
iv. Set status to "W" for the moved element.
v. Increment time by 1.

Program Termination

This section handles the conclusion of the program, providing user notifications and ensuring a graceful exit.

i. Indicate to user that all processes have finished and (if $choice is 1 or 2) that file has been saved.
ii. Exit 0 to end the program.

Alternative Designs

Associative Arrays

Use Bash's associative arrays to directly link process attributes, eliminating the need for multiple arrays. This approach simplifies data organization and access by associating each process's unique key with its attributes. This design choice allows for greater readability and reduces redundancy in updating process information. However, associative arrays are a new feature and require bash version 4.

File Storage

Adopt a file-based approach by storing process information in individual files. Each file represents a process, containing its attributes. This method has a clean data management style and is easily readable and scalable. It encourages a better separation of data and storage when dealing with a large number of processes.

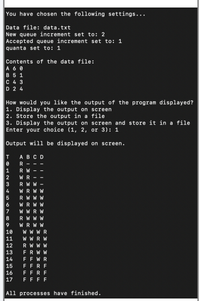
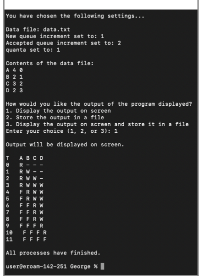
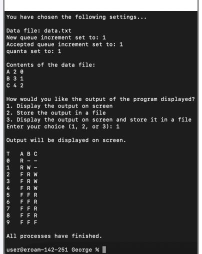
Functions

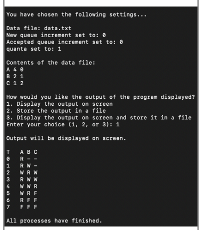
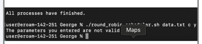
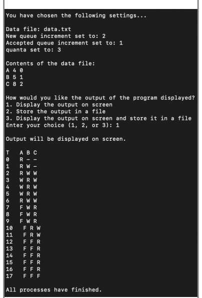
Enhance code organization and reusability by encapsulating process operations within separate Bash functions. This modular design isolates specific operations, making the codebase more maintainable, readable, and easier to build upon safely.

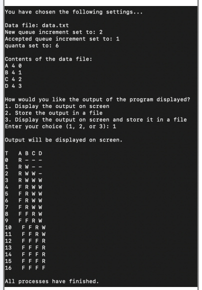
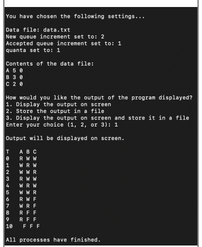
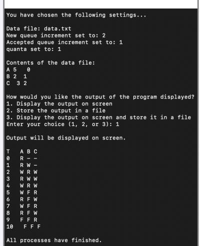
Extended Single-String Elements

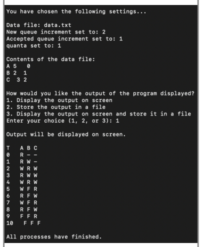
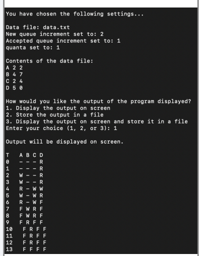
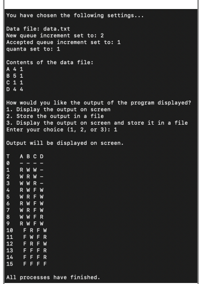
Opt for storing process information in single string elements, condensing data into a unified format. This design may require extensive string manipulation code, but will minimizes the number of variables. This design necessitates careful handling of string parsing, potentially increasing code complexity.

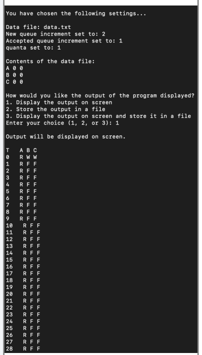
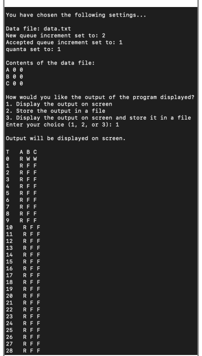
Testing

Description	Input Data	Expected Output	Actual Output	Pass/Fail	Screenshot
Testing Input: High new queue priority increment, low accepted queue priority increment	DATA 2 1 A 6 0 B 5 1 C 4 3 D 2 4	T A B C D 0 R – – – 1 R W – – 2 W R – – 3 R W W – 4 W R W W 5 R W W W 6 W R W W 7 W W R W 8 R W W W 9 W R W W 10 W W W R 11 W W R W 12 R W W W 13 F R W W 14 F F W R 15 F F R F 16 F F R F 17 F F F F	T A B C D 0 R – – – 1 R W – – 2 W R – – 3 R W W – 4 W R W W 5 R W W W 6 W R W W 7 W W R W 8 R W W W 9 W R W W 10 W W W R 11 W W R W 12 R W W W 13 F R W W 14 F F W R 15 F F R F 16 F F R F 17 F F F F	Pass	
Testing Input: Low new queue priority increment, high accepted queue priority increment	DATA 1 2 A 4 0 B 2 1 C 3 2 D 2 3	T A B C D 0 R – – – 1 R W – – 2 R W W – 3 R W W W 4 F R W W 5 F R W W 6 F F R W 7 F F R W 8 F F R W 9 F F F R 10 F F F R 11 F F F F	T A B C D 0 R – – – 1 R W – – 2 R W W – 3 R W W W 4 F R W W 5 F R W W 6 F F R W 7 F F R W 8 F F R W 9 F F F R 10 F F F R 11 F F F F	Pass	
Testing Input: Even priority increments	DATA 1 1 A 2 0 B 3 1 C 4 2	T A B C 0 R – – 1 R W – 2 F R W 3 F R W 4 F R W 5 F F R 6 F F R 7 F F R 8 F F R 9 F F F	T A B C 0 R – – 1 R W – 2 F R W 3 F R W 4 F R W 5 F F R 6 F F R 7 F F R 8 F F R 9 F F F	Pass	

Testing Input: 0 value priority increments	DATA 0 0 A 4 0 B 2 1 C 1 2	T A B C 0 R – – 1 R W – 2 W R W 3 R W W 4 W W R 5 W R F 6 R F F 7 F F F	T A B C 0 R – – 1 R W – 2 W R W 3 R W W 4 W W R 5 W R F 6 R F F 7 F F F	Pass	 You have chosen the following settings... Data file: data.txt New queue increment set to: 0 Accepted queue increment set to: 0 quanta set to: 1 Contents of the data file: A 4 0 B 2 1 C 1 2 How would you like the output of the program displayed? 1. Display the output on screen 2. Store the output in a file 3. Display the output on screen and store it in a file Enter your choice (1, 2, or 3): 1 Output will be displayed on screen. T A B C 0 R – – 1 R W – 2 W R W 3 R W W 4 W W R 5 W R F 6 R F F 7 F F F All processes have finished.
Testing Input: Invalid parameters	DATA c y	Display: The parameters you entered are not valid	Display: The parameters you entered are not valid	Pass	 All processes have finished. user@eroam-142-251 George % ./round_robin_scheduler.sh data.txt c y The parameters you entered are not valid user@eroam-142-251 George %
Testing Input: User quanta setting	DATA 2 1 3 A 4 0 B 5 1 C 8 2	T A B C 0 R – – 1 R W – 2 R W W 3 W R W 4 W R W 5 W R W 6 R W W 7 F W R 8 F W R 9 F W R 10 F R W 11 F R W 12 F F R 13 F F R 14 F F R 15 F F R 16 F F R 17 F F F	T A B C 0 R – – 1 R W – 2 R W W 3 W R W 4 W R W 5 W R W 6 R W W 7 F W R 8 F W R 9 F W R 10 F R W 11 F R W 12 F F R 13 F F R 14 F F R 15 F F R 16 F F R 17 F F F	Pass	 You have chosen the following settings... Data file: data.txt New queue increment set to: 2 Accepted queue increment set to: 1 quanta set to: 3 Contents of the data file: A 4 0 B 5 1 C 8 2 How would you like the output of the program displayed? 1. Display the output on screen 2. Store the output in a file 3. Display the output on screen and store it in a file Enter your choice (1, 2, or 3): 1 Output will be displayed on screen. T A B C 0 R – – 1 R W – 2 R W W 3 W R W 4 W R W 5 W R W 6 R W W 7 F W R 8 F W R 9 F W R 10 F R W 11 F R W 12 F F R 13 F F R 14 F F R 15 F F R 16 F F R 17 F F F All processes have finished.

Testing Input: User quanta setting exceeds process NUT values	DATA 2 1 6	T A B C D 0 R – – – 1 R W – – 2 R W W – 3 R W W W 4 F R W W 5 F R W W 6 F R W W 7 F R W W 8 F F R W 9 F F R W 10 F F R W 11 F F R W 12 F F F R 13 F F F R 14 F F F R 15 F F F R 16 F F F F	T A B C D 0 R – – – 1 R W – – 2 R W W – 3 R W W W 4 F R W W 5 F R W W 6 F R W W 7 F R W W 8 F F R W 9 F F R W 10 F F R W 11 F F R W 12 F F F R 13 F F F R 14 F F F R 15 F F F R 16 F F F F	Pass	 You have chosen the following settings... Data file: data.txt New queue increment set to: 2 Accepted queue increment set to: 1 quanta set to: 6 Contents of the data file: A 4 0 B 4 1 C 4 2 D 4 3 How would you like the output of the program displayed? 1. Display the output on screen 2. Store the output in a file 3. Display the output on screen and store it in a file Enter your choice (1, 2, or 3): 1 Output will be displayed on screen. T A B C D 0 R – – – 1 R W – – 2 R W W – 3 R W W W 4 F R W W 5 F R W W 6 F R W W 7 F R W W 8 F F R W 9 F F R W 10 F F R W 11 F F R W 12 F F F R 13 F F F R 14 F F F R 15 F F F R 16 F F F F All processes have finished.
Testing Data File: All processes start from 0 arrival time	DATA 2 1 A 5 0 B 3 0 C 2 0	T A B C 0 R W W 1 W R W 2 W W R 3 R W W 4 W R W 5 W W R 6 R W F 7 W R F 8 R F F 9 R F F 10 F F F	T A B C 0 R W W 1 W R W 2 W W R 3 R W W 4 W R W 5 W W R 6 R W F 7 W R F 8 R F F 9 R F F 10 F F F	Pass	 You have chosen the following settings... Data file: data.txt New queue increment set to: 2 Accepted queue increment set to: 1 quanta set to: 1 Contents of the data file: A 5 0 B 3 0 C 2 0 How would you like the output of the program displayed? 1. Display the output on screen 2. Store the output in a file 3. Display the output on screen and store it in a file Enter your choice (1, 2, or 3): 1 Output will be displayed on screen. T A B C 0 R W W 1 W R W 2 W W R 3 R W W 4 W R W 5 W W R 6 R W F 7 W R F 8 R F F 9 R F F 10 F F F All processes have finished.
Testing Data File: Unformatted process strings	DATA 2 1 A 5 0 B 2 1 C 3 2	T A B C 0 R – – 1 R W – 2 W R W 3 R W W 4 W R W 5 W F R 6 R F W 7 W F R 8 R F W 9 F F R 10 F F F	T A B C 0 R – – 1 R W – 2 W R W 3 R W W 4 W R W 5 W F R 6 R F W 7 W F R 8 R F W 9 F F R 10 F F F	Pass	 You have chosen the following settings... Data file: data.txt New queue increment set to: 2 Accepted queue increment set to: 1 quanta set to: 1 Contents of the data file: A 5 0 B 2 1 C 3 2 How would you like the output of the program displayed? 1. Display the output on screen 2. Store the output in a file 3. Display the output on screen and store it in a file Enter your choice (1, 2, or 3): 1 Output will be displayed on screen. T A B C 0 R – – 1 R W – 2 W R W 3 R W W 4 W R W 5 W F R 6 R F W 7 W F R 8 R F W 9 F F R 10 F F F All processes have finished.

Testing Data File: Gaps between arrival times	DATA 2 1 A 2 2 B 2 5 C 2 9	T A B C 0 – – – 1 – – – 2 R – – 3 R – – 4 F – – 5 F R – 6 F R – 7 F F – 8 F F – 9 F F R 10 F F R 11 F F F	T A B C 0 – – – 1 – – – 2 R – – 3 R – – 4 F – – 5 F R – 6 F R – 7 F F – 8 F F – 9 F F R 10 F F R 11 F F F	Pass	 You have chosen the following settings... Data file: data.txt New queue increment set to: 2 Accepted queue increment set to: 1 quanta set to: 1 Contents of the data file: A 5 0 B 2 1 C 3 2 How would you like the output of the program displayed? 1. Display the output on screen 2. Store the output in a file 3. Display the output on screen and store it in a file Enter your choice (1, 2, or 3): 1 Output will be displayed on screen. T A B C 0 R – – 1 R W – 2 W R W 3 R W W 4 W R W 5 W F R 6 R F W 7 W F R 8 R F W 9 F F R 10 F F F All processes have finished.
Testing Data File: Unsorted arrival times	DATA 2 1 A 2 2 B 4 7 C 2 4 D 5 0	T A B C D 0 – – – R 1 – – – R 2 W – – R 3 W – – R 4 R – W W 5 W – W R 6 R – W F 7 F W R F 8 F W R F 9 F R F F 10 F R F F 11 F R F F 12 F R F F 13 F F F F	T A B C D 0 – – – R 1 – – – R 2 W – – R 3 W – – R 4 R – W W 5 W – W R 6 R – W F 7 F W R F 8 F W R F 9 F R F F 10 F R F F 11 F R F F 12 F R F F 13 F F F F	Pass	 You have chosen the following settings... Data file: data.txt New queue increment set to: 2 Accepted queue increment set to: 1 quanta set to: 1 Contents of the data file: A 2 2 B 4 7 C 2 4 D 5 0 How would you like the output of the program displayed? 1. Display the output on screen 2. Store the output in a file 3. Display the output on screen and store it in a file Enter your choice (1, 2, or 3): 1 Output will be displayed on screen. T A B C D 0 – – – R 1 – – – R 2 W – – R 3 W – – R 4 R – W W 5 W – W R 6 R – W F 7 F W R F 8 F W R F 9 F R F F 10 F R F F 11 F R F F 12 F R F F 13 F F F F
Testing Data File: Multiple arrivals at the same time	DATA 2 1 A 4 1 B 5 1 C 1 1 D 4 4	T A B C D 0 – – – – 1 R W W – 2 W R W – 3 W W R – 4 R W F W 5 W R F W 6 R W F W 7 W R F W 8 W W F R 9 R W F W 10 F R F W 11 F W F R 12 F R F W 13 F F F R 14 F F F R 15 F F F F	T A B C D 0 – – – – 1 R W W – 2 W R W – 3 W W R – 4 R W F W 5 W R F W 6 R W F W 7 W R F W 8 W W F R 9 R W F W 10 F R F W 11 F W F R 12 F R F W 13 F F F R 14 F F F R 15 F F F F	Pass	 You have chosen the following settings... Data file: data.txt New queue increment set to: 2 Accepted queue increment set to: 1 quanta set to: 1 Contents of the data file: A 4 1 B 5 1 C 1 1 D 4 4 How would you like the output of the program displayed? 1. Display the output on screen 2. Store the output in a file 3. Display the output on screen and store it in a file Enter your choice (1, 2, or 3): 1 Output will be displayed on screen. T A B C D 0 – – – – 1 R W W – 2 W R W – 3 W W R – 4 R W F W 5 W R F W 6 R W F W 7 W R F W 8 W W F R 9 R W F W 10 F R F W 11 F W F R 12 F R F W 13 F F F R 14 F F F R 15 F F F F All processes have finished.

Testing Data File: All process values at 0 NUT and 0 arrival	DATA 2 1 A 0 0 B 0 0 C 0 0	All processes have finished	T A B C 0 R W W 1 R F F 2 R F F 3 R F F 4 R F F 5 R F F 6 R F F 7 R F F 8 R F F 9 R F F 10 R F F (Infinite loop)	Fail	 You have chosen the following settings... Data file: data.txt New queue increment set to: 2 Accepted queue increment set to: 1 quanta set to: 1 Contents of the data file: A 0 0 B 0 0 C 0 0 How would you like the output of the program displayed? 1. Display the output on screen 2. Store the output in a file 3. Display the output on screen and store it in a file Enter your choice (1, 2, or 3): 1 Output will be displayed on screen. T A B C 0 R W W 1 R F F 2 R F F 3 R F F 4 R F F 5 R F F 6 R F F 7 R F F 8 R F F 9 R F F 10 R F F 11 R F F 12 R F F 13 R F F 14 R F F 15 R F F 16 R F F 17 R F F 18 R F F 19 R F F 20 R F F 21 R F F 22 R F F 23 R F F 24 R F F 25 R F F 26 R F F 27 R F F 28 R F F
Testing Output: Data is stored in designated file name	DATA 2 1 A 3 0 B 2 1 C 4 4 D 1 1	T A B C D 0 R – – – 1 R W – W 2 W R – W 3 W W – R 4 R W W F 5 F R W F 6 F F R F 7 F F R F 8 F F R F 9 F F R F 10 F F F F (Stored in test.txt)	T A B C D 0 R – – – 1 R W – W 2 W R – W 3 W W – R 4 R W W F 5 F R W F 6 F F R F 7 F F R F 8 F F R F 9 F F R F 10 F F F F (Stored in test.txt)	Pass	 You have chosen the following settings... Data file: data.txt New queue increment set to: 2 Accepted queue increment set to: 1 quanta set to: 1 Contents of the data file: A 0 0 B 0 0 C 0 0 How would you like the output of the program displayed? 1. Display the output on screen 2. Store the output in a file 3. Display the output on screen and store it in a file Enter your choice (1, 2, or 3): 1 Output will be displayed on screen. T A B C 0 R W W 1 R F F 2 R F F 3 R F F 4 R F F 5 R F F 6 R F F 7 R F F 8 R F F 9 R F F 10 R F F 11 R F F 12 R F F 13 R F F 14 R F F 15 R F F 16 R F F 17 R F F 18 R F F 19 R F F 20 R F F 21 R F F 22 R F F 23 R F F 24 R F F 25 R F F 26 R F F 27 R F F 28 R F F

Evaluation of Built System

Specification Table

Specification	Code Line
Tests for correct number of parameters	44
Tests that filename is a regular file	49
Read data and store in appropriate data structure	60-75
Loop over Time	221
Add/Set processes that AT == T to the end of the existing list initially from index 0 & set status to W	141-151
Test queues if not empty add job to appropriate queue	169-181
Set top process in accepted queue to R decrement NUT	190-195
Print out process status in order of header	198-210
Test if all process completed — exit loop	129
Test top process if NUT == - then set status to F	132-138
Increment of priority values of each queue	154-166
Move process from new to accepted queue	169-181
Loop and move or set index so that top process is moved to back of queue; STOP when top process NUT > 0	213-219
Correct header on output based on order provided by user	198-210
Display correct symbolism; -: not on system; W: waiting; R: running; F: finished	198-210
Display time	198-210
Display process states correctly under header	198-210
Outputs also written to stdout and named file	198-210
Read 3rd parameter to define quanta level and validate	43, 54-57
Set process to sit at the top of accepted queue to set quanta level	213
Move process if completes before quanta level expires	132-138

Each requirement is marked by the corresponding line of code that performs said function.

Specification Compliance

The code meets fundamental requirements and addresses edge cases such as unformatted data files, simultaneous arrivals, unsorted arrival times, and priorities settings with a value of 0. This approach ensures smooth execution across various scenarios and considers different input situations.

Documentation

The comments and documentation in the code generally fulfill a useful purpose. The functionality of each module is given without stating the obvious and additional comments are made for more ambiguous lines of code. Code readability is largely provided through appropriate variable names, minimizing the need for excessive comments.

Structure and Modularity

The code exhibits an organized structure, primarily adhering to a procedural design. While the modules are well-defined, there exists an opportunity for enhancement through the incorporation of functions to further improve modularity. Nonetheless, readability is maintained through intuitive variable naming and clearly defined modules. It follows CamelCase naming conventions for improved readability and maintains a procedural coding structure. Suggestions for improvement include making certain variable names, such as $element, more explicit for enhanced understanding.

Efficiency

The algorithms and overall program flow demonstrate efficiency. The static reference index contributes to a clean design, resulting in a quick and concise codebase. The script has few duplications and the minimalistic design, featuring loops concluding after a single iteration, facilitates efficient access and modification of process data. The streamlined implementation requires a relatively small amount of code, ensuring effective processing.

Error Handling

Error handling has been implemented for invalid data file types, parameters, output settings, and unformatted data files. Existing validations are clear, but additional checks, such as handling invalid data within the data file, could bolster the overall robustness of error handling. This represents an area where further refinement could contribute to a more resilient codebase.

Limitations and Expansion

This script successfully emulates a round robin algorithm to allocate processes to a CPU, but it comes with certain limitations. Firstly, it exclusively models one algorithm among various alternatives that can fulfill this function. Notably, it overlooks alternative algorithms like First Come First Serve (FCFS) or Shortest Job Firsts (SJF) which play a crucial role in process allocation strategies. Furthermore, it operates strictly as a simulation, and its efficacy in replicating real-world scenarios is questionable.

While the script does simulate fundamental aspects such as process allocation, state preservation, and context switching, it falls short in handling dynamic scenarios where new processes emerge during execution. Additionally, it lacks consideration for the intricacies of how CPUs leverage multiple threads for process allocation or how an operating system might prioritize certain processes over others. In essence, this script operates within a controlled environment and neglects the inherent unpredictability of real-world operating systems in their need to allocate processing resources effectively. To enhance its realism and applicability, the script could benefit from incorporating a broader spectrum of algorithms and accounting for the dynamic nature of process allocation in a genuine operating system environment.

Program Quantities Table

Metric	Results
Execution Time for User	0.03s
Execution Time for System	0.04s
Total Elapsed Time	2.155
Percentage of CPU utilization	3%
Runtime Complexity	O(n^2 * t)
Average Cyclomatic Complexity	34.00
Non-Comment Lines of Code	158
Logical Lines of Code	125
Total Lines	234
Total Comment Lines	37
Blank Lines	39

Metrics were recorded in an environment devoid of as many ongoing processes as possible with the BASH time function. Runtime complexity is O(n^2 * t) where 'n' is the number of processes and 't' is time. Cyclomatic complexity was generated through https://github.com/shellspec/shellmetrics.

References

Charalambous, G. (2023) 'Bash Part 1.' Software Development Environments, Westminster University, London, 13 November 2023.

Charalambous, G. (2023) 'Bash Part 2.' Software Development Environments, Westminster University, London, 20 November 2023.

How do I test if a variable is a number in Bash? (no date) *Stack Overflow*. Available at: https://stackoverflow.com/questions/806906/how-do-i-test-if-a-variable-is-a-number-in-bash (Accessed: 28 November 2023).

Bash Tutorial => Array Modification (no date) *riptutorial.com*. Available at: https://riptutorial.com/bash/example/3380/array-modification (Accessed: 2 December 2023).

command line - How do I save terminal output to a file? (no date) *Ask Ubuntu*. Available at: https://askubuntu.com/questions/420981/how-do-i-save-terminal-output-to-a-file (Accessed: 2 December 2023).

Check if a Bash array contains a value (no date) *Stack Overflow*. Available at: https://stackoverflow.com/questions/3685970/check-if-a-bash-array-contains-a-value (Accessed: 29 November 2023).

How to add/remove an element to/from the array in bash? (no date) *Unix & Linux Stack Exchange*. Available at: https://unix.stackexchange.com/questions/328882/how-to-add-remove-an-element-to-from-the-array-in-bash (Accessed: 29 November 2023).

Operating Systems: Processes (2019) *Uic.edu*. Available at: https://www.cs.uic.edu/~jbell/CourseNotes/OperatingSystems/3_Processes.html (Accessed: 11 December 2023).

10th ed. chapter 05 (no date) www.cs.csustan.edu. Available at: https://www.cs.csustan.edu/~john/Classes/CS3750/Notes/Chap05/05_CPU_Scheduling.html (Accessed: 11 December 2023).

Script

#!/bin/bash

Author: Ben Hogg

Version: 1.0

Description:
This script implements the round-robin scheduling algorithm, simulating how processes are fairly allocated to a CPU without conflicts. It takes as input a data file containing process names, NUT values, and arrival times.
The design involves mapping each process's details to a single index, facilitating efficient tracking and modification. Processes initially enter the new queue based on their arrival times, matched with a time variable that increments with each time slice.
Processes transition from the new queue to the accepted queue when their priority is equal to or greater than those in the accepted queue. The algorithm iterates until all processes have completed their CPU servicing time.
Users can customize quanta and priority values through the parameters.
Output options include displaying results on screen or saving them to a file.

Terminology:
- "NUT" refers to the Normalized Utilization Time. This is the maximum amount of turns a process can use the CPU consecutively.
- "Time slice" refers to the unit of time in the scheduling algorithm.
- "Priority" determines the order in which processes move from the new queue to the accepted queue.

File Format:
The data file should be a plain text file with columns specifying process names, NUT values, and arrival times.

Parameters:
$1 Regular data file
$2 New queue priority increment setting

```
# $3 Accepted queue priority increment setting

# Example:
# ./SDEproject.sh input_data.txt 2 5



# Program starts here...

# Validates and assigns inputs
if [ "$#" -eq 3 ]; then
   dataFile="$1"
   newIncrement="$2"
   acceptedIncrement="$3"
   quanta=1
elif [ "$#" -eq 4 ]; then
   dataFile="$1"
   newIncrement="$2"
   acceptedIncrement="$3"
   quanta="$4" # (Charalambous, 2023)
else
   echo "The number of parameters is invalid. Please enter a file name with 2 or 3
parameters"
   exit 1
fi

if [ ! -f "$dataFile" ]; then
   echo "The data file you entered is not a regular file. Please try again with the name of a
regular file"
   exit 1
fi

if [[ ! "$newIncrement" =~ ^[0-9]+$ || ! "$acceptedIncrement" =~ ^[0-9]+$ || ! "$quanta" =~
^[0-9]+$ ]]; then # Stack Overflow (n.d.)
   echo "The parameters you entered are not valid"
   exit 1
fi
```

```
# Initiate arrays and assign data
newQueue=()
acceptedQueue=()

while read -r name_var service_var arrival_var # (Charalambous, 2023)
do
   quantaArray+=("$quanta")
   priority+=("0")
   status+=("-")
   name+=("$name_var")
   service+=("$service_var")
   arrival+=("$arrival_var")
done < "$dataFile"

for ((i=0; i<${#name[@]}; i++)); do
   referenceIndex+=("$i")
done

# Displays input settings
echo
echo "You have chosen the following settings..."
echo
echo "Data file: $dataFile"
echo "New queue increment set to: $newIncrement"
echo "Accepted queue increment set to: $acceptedIncrement"
echo "quanta set to: $quanta"
echo

# Displays data file
echo "Contents of the data file:"
cat "$dataFile"
echo

# Reads and validates output settings
while true; do
```

```
    echo "How would you like the output of the program displayed?"
    echo "1. Display the output on screen"
    echo "2. Store the output in a file"
    echo "3. Display the output on screen and store it in a file"

    read -p "Enter your choice (1, 2, or 3): " choice

    if [ "$choice" == "1" ]; then
        echo
        echo "Output will be displayed on screen."
        echo
        break

    elif [ "$choice" == "2" ]; then
        echo
        read -p "Enter the file name to store the output: " fileName
        echo
        break

    elif [ "$choice" == "3" ]; then
        echo
        read -p "Enter the file name to store the output: " fileName
        echo "Output will be displayed on screen and stored in the file: $fileName."
        echo
        break

    else
        echo "Invalid choice. Please enter 1, 2, or 3."
        echo
    fi
done

time=0

# Main loop
while [[ $(IFS=; echo "${status[*]}") =~ [^F]+ ]]; do # (Stack Overflow, n.d.)
```

```
  # Removes finished elements and updates status to F
  for ((i = ${#acceptedQueue[@]} - 1; i >= 0; i--)); do
    element=${acceptedQueue[i]}
    if [ "${service[$element]}" -eq 0 ]; then
      status[$element]="F"
      acceptedQueue=("${acceptedQueue[@]:0:i}" "${acceptedQueue[@]:i+1}") # (Stack
Exchange, n.d.)
    fi
  done

  # Moves the first process to arrive directly into the accepted queue and all other arriving
processes into the new queue
  for ((i=0; i<${#referenceIndex[@]}; i++)); do
    if [ "$time" -eq "${arrival[$i]}" ]; then
      if [[ ${#acceptedQueue[@]} == 0 ]] && [[ ${#newQueue[@]} == 0 ]]; then
        acceptedQueue+=("${referenceIndex[i]}")
        status[i]="R"
      else
        newQueue+=("${referenceIndex[i]}")
        status[i]="W"
      fi
    fi
  done

  # Increments priorities
  for ((i=0; i<${#newQueue[@]}; i++)); do
    relevantIndex=${newQueue[i]}
    if [[ -n $relevantIndex && $relevantIndex =~ ^[0-9]+$ ]]; then # =~ ^[0-9]+$ to filter out
invalid elements # Stack Overflow (n.d.)
      ((priority[$relevantIndex] += $newIncrement))
    fi
  done

  for ((i=0; i<${#acceptedQueue[@]}; i++)); do
    relevantIndex=${acceptedQueue[i]}
```

```
    if [[ -n $relevantIndex && $relevantIndex =~ ^[0-9]+$ ]]; then
      ((priority[$relevantIndex] += $acceptedIncrement))
    fi
  done

  # Introduces processes from the new queue into the accepted queue
  if [[ ${#acceptedQueue[@]} != 0 ]] && [[ ${#newQueue[@]} != 0 ]]; then
    for ((i=0; i<${#newQueue[@]}; i++)); do
      for ((j=0; j<${#acceptedQueue[@]}; j++)); do
        if [[ ${priority[${newQueue[i]}]} -ge ${priority[${acceptedQueue[j]}]} ]]; then
          element=("${newQueue[i]}")
          if [[ "$element" =~ ^[0-9]+$ ]]; then
            acceptedQueue=("${acceptedQueue[@]}" "$element")
            newQueue=("${newQueue[@]:0:i}" "${newQueue[@]:i+1}")
          fi
        fi
      done
    done
  fi

  # Moves process from new to accepted queue; For cases when the accepted queue is
empty
  if [[ ${#acceptedQueue[@]} = 0 ]] && [[ ${#newQueue[@]} != 0 ]]; then
    acceptedQueue=("${newQueue[0]}")
    newQueue=("${newQueue[@]:1}")
  fi

  # Adjusts values values for leading process
  if [ "${#acceptedQueue[@]}" -gt 0 ]; then
    relevantIndex="${acceptedQueue[0]}"
    ((service[$relevantIndex]--))
    ((quantaArray[$relevantIndex]--))
    status[$relevantIndex]="R"
  fi

  # Displays and stores output
```

```
    if [ "$choice" -eq 1 ] || [ "$choice" -eq 3 ]; then
        if [ "$time" -eq 0 ]; then
            echo "T   ${name[*]}" | tr ' ' '   '
        fi
        echo "$time   ${status[*]}"
    fi

    if [ "$choice" -eq 2 ] || [ "$choice" -eq 3 ]; then
        if [ "$time" -eq 0 ]; then
            echo "T   ${name[*]}" | tr ' ' '   ' >> "$fileName" # (Ask Ubuntu, n.d.)
        fi
        echo "$time   ${status[*]}" >> "$fileName"
    fi

    # Sends leading process to the back of the accepted queue
    if [ "${#acceptedQueue[@]}" -gt 0 ] && [ "${quantaArray[${acceptedQueue[0]}]}" -eq 0 ];
then
        element="${acceptedQueue[0]}"
        quantaArray[$element]=$quanta
        acceptedQueue=("${acceptedQueue[@]:1}")
        acceptedQueue+=("$element")
        status[$element]="W"
    fi

    ((time++))

done

# Exit display
echo
echo "All processes have finished."
echo
    if [ "$choice" -eq 2 ] || [ "$choice" -eq 3 ]; then
        echo "The results have been saved to: $fileName"
        echo
    fi
```

exit 0

Video Link

Click this link!