Prepared by: E. Bedalli & I. Draçi

Deadline: 04.07.2021 23:30

CEN308 – Operating Systems

Homework 2

Problem 1 (30 pts) – Scheduling Simulation

In this problem you are required to prepare a simulation program (in C, Java or Python) about the Round-Robin scheduling algorithm testing on various lengths of the time quantum. You are going to randomly generate sequences of tasks (of random lengths and random arrival times) and then evaluate the waiting time and response time according to the time quantum. Thus in your results you should describe how the average waiting time and average response time vary while the time quantum changes.

Note: Waiting time is the total time spent by the process in the ready state waiting for CPU. Response time is the time spent when the process is in the ready state until it gets the CPU for the first time.

Submission: you should submit the source file and a report (in pdf format) describing various assumptions that you have made about the simulation scenario and an interpretation of the results of your experimental work (usage of descriptive charts is highly recommended).

Solution:

Using Java, I have created a program that simulates the Round Robin Task Scheduling Policy. It generates some tasks and tries to run the algorithm with difference time quantum. It uses an Array List in a circular fashion to give each task some time in the CPU. When all tasks have completed execution, it prints the task number, arrival time, duration, start time and end time for each task and calculates the average waiting time and average turnaround time for each execution. The solution is heavy structured in the OOP paradigm to simplify the main execution class and make it as abstract as possible. From the sample execution, it is clear that the RR algorithm helps with the average waiting time or responsiveness with a low time quantum. However, the average turnaround time suffers a lot from the short time quantum. With the increase of the time quantum, the average waiting time increases, but the average turnaround time decreases.

Edison Ponari
CEN III – A

Deadline: 04.07.2021 23:30

Sample Execution:

```
Enter the number of tasks that are going to be generated: 5
Time quantum = 2
Task 1:  Arrival Time: 0 Duration: 4 Start Time: 0 End Time: 10
Task 2:  Arrival Time: 2 Duration: 2 Start Time: 2 End Time: 4
Task 3:  Arrival Time: 5 Duration: 5 Start Time: 10 End Time: 22
Task 4:  Arrival Time: 4 Duration: 5 Start Time: 4 End Time: 19
Task 5:  Arrival Time: 4 Duration: 10 Start Time: 6 End Time: 26
Average Waiting Time: 1.4
Average Turnaround Time 13.2
```

```
Time quantum = 3
Task 1:  Arrival Time: 0 Duration: 4 Start Time: 0 End Time: 15
Task 2:  Arrival Time: 2 Duration: 2 Start Time: 3 End Time: 5
Task 3:  Arrival Time: 5 Duration: 5 Start Time: 5 End Time: 17
Task 4:  Arrival Time: 4 Duration: 5 Start Time: 8 End Time: 19
Task 5:  Arrival Time: 4 Duration: 10 Start Time: 11 End Time: 26
Average Waiting Time: 2.4
Average Turnaround Time 13.4


Time quantum = 4
Task 1:  Arrival Time: 0 Duration: 4 Start Time: 0 End Time: 4
Task 2:  Arrival Time: 2 Duration: 2 Start Time: 4 End Time: 6
Task 3:  Arrival Time: 5 Duration: 5 Start Time: 6 End Time: 19
Task 4:  Arrival Time: 4 Duration: 5 Start Time: 10 End Time: 20
Task 5:  Arrival Time: 4 Duration: 10 Start Time: 14 End Time: 26
Average Waiting Time: 3.8
Average Turnaround Time 12.0


Time quantum = 5
Task 1:  Arrival Time: 0 Duration: 4 Start Time: 0 End Time: 4
Task 2:  Arrival Time: 2 Duration: 2 Start Time: 4 End Time: 6
Task 3:  Arrival Time: 5 Duration: 5 Start Time: 6 End Time: 11
Task 4:  Arrival Time: 4 Duration: 5 Start Time: 11 End Time: 16
Task 5:  Arrival Time: 4 Duration: 10 Start Time: 16 End Time: 26
Average Waiting Time: 4.4
Average Turnaround Time 9.6
```

Prepared by: E. Bedalli & I. Draçi

Deadline: 04.07.2021 23:30

Problem 2 (25 pts) – Page Replacement Policies Simulation

Write a simulation program (in C, Java or Python) that will compare the usage of the FIFO, LRU, OPTIMAL and SECOND CHANCE page replacement policies. You will generate randomly a page reference string and you will check the progress by keeping track of the page faults number. Submission: you should submit the source file and a report (in pdf format) describing various assumptions that you have made about the simulation scenario and an interpretation of the results of your experimental work (usage of descriptive charts is highly recommended).

Solution:

Using Java, I created a program to simulate different Page Replacement Policies, them being FIFO, LRU, OPTIMAL and SECOND CHANCE. In the FIFO approach, an Array List with FIFO replacement policy is used to keep track of pages in memory. In the LRU and OPTIMAL approaches, an Array List of not-visited pages that are in memory and the last to remain in this List, is removed in the list of pages in memory. The only difference is that LRU looks in the previous pages, while OPTIMAL looks in the future pages. In the SECOND CHANCE approach, a HashSet is used to keep track of pages in memory that have a second chance. If they are to be removed but have a second chance, the second chance is removed from the HashSet and the next page that doesn't have a second chance is removed. As demonstrated in the sample execution, the best performing is the Optimal algorithm but in reality, the computer can't predict future requests for pages. The most realistic approach is the LRU algorithm that is fair regarding the pages with most priority.

Sample Execution:

```
Order: 7012037012030423030321201701042303701203042303032120170103212017 01
Size: 5
Algorithm: FIFO ; Page Faults: 16 ; Hits: 50
Algorithm: Least Recently Used ; Page Faults: 13 ; Hits: 53
Algorithm: Optimal ; Page Faults: 10 ; Hits: 56
Algorithm: Second Change ; Page Faults: 16 ; Hits: 50
```

Edison Ponari
CEN III – A

Prepared by: E. Bedalli & I. Draçi

Deadline: 04.07.2021 23:30

Problem 3 (30 pts) – Contiguous Memory Allocation

This problem will involve managing a contiguous region of memory of size MAX where addresses may range from 0 ... MAX − 1. Your program must respond to four different requests:

1. Request for a contiguous block of memory
2. Release of a contiguous block of memory
3. Compact unused holes of memory into one single block
4. Report the regions of free and allocated memory

Your program will be passed the initial amount of memory at startup. For example, the following initializes the program with 1 MB (1,048,576 bytes) of memory:

```
./allocator 1048576
```

Once your program has started, it will present the user with the following prompt:

```
allocator>
```

It will then respond to the following commands: RQ (request), RL (release), C (compact), STAT (status report), and X (exit). A request for 40,000 bytes will appear as follows:

```
allocator>RQ P0 40000 W
```

The first parameter to the RQ command is the new process that requires the memory, followed by the amount of memory being requested, and finally the strategy. (In this situation, “W” refers to worst ft.)

Similarly, a release will appear as:

```
allocator>RL P0
```

This command will release the memory that has been allocated to process P0. The command for compaction is entered as:

```
allocator>C
```

This command will compact unused holes of memory into one region. Finally, the STAT command for reporting the status of memory is entered as:

```
allocator>STAT
```

Given this command, your program will report the regions of memory that are allocated and the regions that are unused. For example, one possible arrangement of memory allocation would be as follows:

Edison Ponari
CEN III – A

Prepared by: E. Bedalli & I. Draçi

Deadline: 04.07.2021 23:30

Addresses [0:315000] Process P1

Addresses [315001: 512500] Process P3

Addresses [512501:625575] Unused

Addresses [625575:725100] Process P6

In the RQ command you have 3 approaches: F (first-fit), B (best-fit) and W (worst-fit)

Solution:

Using Java, I have created a program that simulates the Contiguous Memory Allocation using a Linked List data structure. Each element of the data structure contains a starting point, it's length, the process name and the next process in memory. The free memory is represented as the differences in the start + length of an element to the starting point of the next process. The memory limit is given as an argument in execution and the limits of memory are

0 and the input. RQ (request memory) request is done in 3 fashions:

- first-fit: find the first free memory space that has enough space for this process.
- best-fit: find the shortest free memory space that has enough space for this process.
- worst-fit: find the longest free memory space that has enough space for this process.

The RL (release memory) request is done using a linear search in the linked list for the process and removing it from the list. The C (compact) request makes the first memory allocation start at 0 and each process starts at starting point + length of the previous memory allocation in the linked list. The STAT (statistics) request prints every block of memory in the linked list and the differences between them are printed as unused memory.

Edison Ponari
CEN III – A

Prepared by: E. Bedalli & I. Draçi

Deadline: 04.07.2021 23:30

Sample Execution:

```
PS C:\Users\User\OneDrive\Desktop\Operating Systems> java src/ContiguousMemoryAllocation.java 1000000
allocator>RQ p1 20000 W
allocator>RQ p2 30000 W
allocator>RQ p3 10000 B
allocator>RQ p4 10000 F
allocator>stat
Addresses[0:19999] Process p1
Addresses[20000:49999] Process p2
Addresses[50000:59999] Process p3
Addresses[60000:69999] Process p4
Addresses[70001:1000000] Unused
allocator>RL p2
allocator>stat
Addresses[0:19999] Process p1
Addresses[20001:49999] Unused
Addresses[50000:59999] Process p3
Addresses[60000:69999] Process p4
Addresses[70001:1000000] Unused
allocator>RQ p5 10000 B
allocator>stat
Addresses[0:19999] Process p1
Addresses[20000:29999] Process p5
Addresses[30001:49999] Unused
Addresses[50000:59999] Process p3
Addresses[60000:69999] Process p4
Addresses[70001:1000000] Unused
allocator>RQ p6 5000 W
allocator>stat
Addresses[0:19999] Process p1
Addresses[20000:29999] Process p5
Addresses[30001:49999] Unused
Addresses[50000:59999] Process p3
Addresses[60000:69999] Process p4
allocator>C
allocator>stat
Addresses[0:9999] Process p1
Addresses[10000:19999] Process p3
Addresses[20000:29999] Process p4
Addresses[30001:1000000] Unused
```

Edison Ponari
CEN III – A

Prepared by: E. Bedalli & I. Draçi

Deadline: 04.07.2021 23:30

Problem 4 (15 pts) – Shell Scripting

Prepare a shell script (in BASH) named allFiles.sh that receives two or three command line arguments: the first and second argument are the names of two subdirectories and the third command line argument (which is optional) is the extension (like pdf, txt, py etc.).

If the script is executed with two command line arguments, then it will provide a list in alphabetical order of all files contained in any of these directories, and also printing [1] if it belongs to the first directory or [2] if it belongs to the second directory.

If the script is executed with three command line arguments, then it will provide a list of all files of the given extension which exist in both directories (i.e. having the same name, the contents may not be the same).

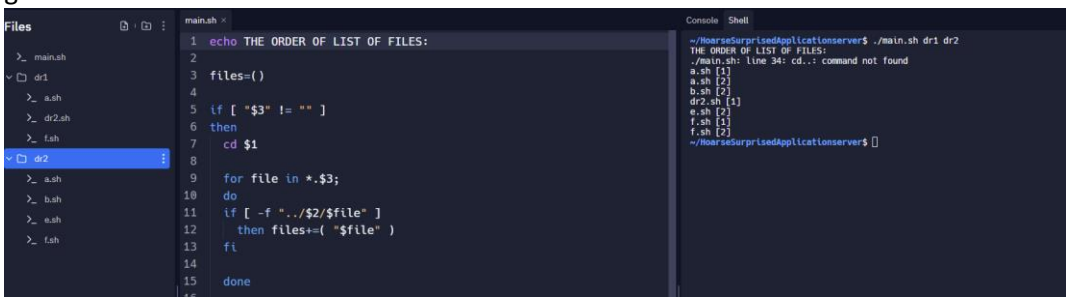
Solution:

In this problem, we have to implement 2 solutions for the following inputs:

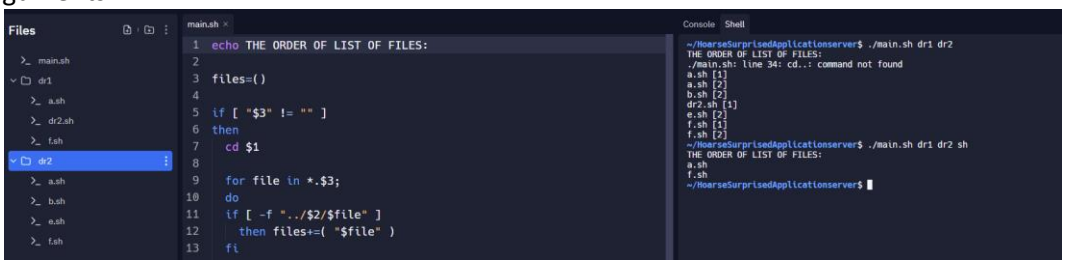
- 2 inputs (directory1, directory2): Enter each directory and save in an array each file and [1] for the first directory and [2] for the directory. In the end, use the pipe operator and redirect the printing of the array into the sort command.
- 3 inputs (directory1, directory2, extension): Enter each directory and save in an array each file ending with the given extension that appear in both directories. In the end, use the pipe operator and redirect the printing of the array into the sort command.

Sample Executions:

- 2 arguments:



- 3 arguments:



Edison Ponari
CEN III – A