Disk Scheduling
Abu Shahid
B20CS003

Disk Scheduling

Execution

`gcc disk_scheduler.c -o disk && ./disk`

- The prompt will ask you to enter c: number of cylinders and r: requests.
- The user then is supposed to enter the arrival times of the r requests.
- The initial possition of head in our implemetation is assumed to be 0 and therefore not taken as input.
- The upper limit of the requests is 1000.
- Scheduling algorithm is then selected using switch.

```
ceyxasm@pop-os:~/.../lab_9/Lab8$ gcc disk_scheduler.c -o disk && ./disk
Enter total number of cylinders : 8
enter number of requestes : 8
Enter request arrival time for requests : 176 114 79 92 60 41 34 11


Enter
1 for FCFS
2 for SCAN
3 for C_SCAN
4 for SSTF
any other to exit
1

Sequence of requsets : 176 114 79 92 60 41 34 11
Waiting time for every request : 0 62 35 13 32 19 7 23
Avg waiting time : 23.875000
TurnAround for every request : 62 35 13 32 19 7 23 0
Average TurnAround time : 23.875000
Total Cylinder Movement : 191
```

(PFA log file in the root folder for result of other scheduling algorithms)

Memory Resident File System

I. Linked List Implementation using FAT

Execution

`g++ ll_fat.cpp -o ll_fat && ./ll_fat`

- Execution procudes the user a list of command which they can use to perform various operations.
  - my_open: to open/create a new file
  - my_close: to close the current file
  - my_read: to read the contents of the current file
  - my_write: to write into our current file
  - my_mkdir: to make a directory
  - my_chdir: to change the directory
  - my_rmdir: to remove a directory

    - my_copy
    - 1 : for exit
- After choosing a command, the user is prompted with other details that need to be provided
- In our implementation, my_open is a way of selecting our file. That is, inorder to read a file or write into it, it must first be selected (opened) using my_open.
    - If we attempt to open a file with a name that does not exist, file of that name will be created.

```
_____________________________________
COMMAND_LIST :
my_open
my_close
my_read
my_write
my_mkdir
my_chdir
my_rmdir
my_copy
1 : for exit

my_open
Enter filename: file2
Enter file size in MB: 4
Enter block size in KB: 512
File opened !!
```

```
_____________________________________
COMMAND_LIST :
my_open
my_close
my_read
my_write
my_mkdir
my_chdir
my_rmdir
my_copy
1 : for exit

my_write
Enter content to write in file: "hello2"
Data written in file !!
```

- Refer to the log in root folder where following sequence of operations is demonstrated:
- Create file1
    1. File1 is opened automatically while creating it.
- Write into file1
- Close file1
- Create file2
- Write into file2
- Read file2
- Close file2
- Open file1
- Read file1

II. Indexed Implementation using i-node

Execution

`gcc inode.cpp -o inode`

- This implementation produces the exact same results as the previous implementation (to the end user)
- Refer to the codebase for deeper insight.

```
ceyxasm@pop-os:~/.../lab_9/Lab8$ g++ inode.cpp -o inode
ceyxasm@pop-os:~/.../lab_9/Lab8$ ./inode

Enter the follow :
my_open
my_close
my_read
my_write
my_mkdir
my_chdir
my_rmdir
my_copy
1 : for exit

my_open
Enter file size in MB: 10
Enter block size in KB: 512
Enter filename: file_1
File opened !!

Enter the follow :
my_open
my_close
my_read
my_write
my_mkdir
my_chdir
my_rmdir
my_copy
1 : for exit
```