Disk-scheduling Algorithms

Aaron Alphonsus
Version 1.0
DATE: 28 April 2017

File Index

File List

Here is a list of all files with brief descriptions:

File Documentation

c_look.cpp File Reference

Simulates the C-LOOK scheduling algorithm.

```
#include <algorithm>
#include <vector>
#include "disk_sched.h"
```

Functions

- int **c_look** (int initial_pos, int request[**REQUESTS**])
  C-LOOK (Circular-LOOK)

Detailed Description

Simulates the C-LOOK scheduling algorithm.

Simulates the C-LOOK algorithm and returns total amount of head movement required by the algorithm.

Author:
Aaron Alphonsus

Date:
28 April 2017

Function Documentation

int c_look (int initial_*pos*, int request[REQUESTS])

C-LOOK (Circular-LOOK)

Simulates the C-LOOK algorithm. Functionality of the C-LOOK was unclear regarding direction switching. Used description from here:
`http://courses.teresco.org/cs432_f02/lectures/17-files/17-files.html`
The direction switches once there are no pending requests in that direction.

The function creates two arrays: for requests less than the initial head position, and for requests greater. The order in which they are processed depends on the initial direction of the disk head.

Parameters:

in	*initial_pos*	Initial disk head position
in	*request*	The random request array generated

Returns:
Total amount of head movement

Place requests into an 'up' and 'down' vector which will be sorted

Assumes that == are processed first, therefore 0 head movement

If the head is moving left initially

Sort both up and down in descending order

Service the 'down' array
Head position wraps around to the highest request in 'up' array
Service 'up' array
Sort both up and down in ascending order
Service 'up' array
Head position wraps around to the lowest request in 'down' array
Service the 'down' array

c_scan.cpp File Reference

Simulates the C-SCAN scheduling algorithm.

```
#include <algorithm>
#include <vector>
#include "disk_sched.h"
```

Functions

- int **c_scan** (int initial_pos, int request[**REQUESTS**])
  C-SCAN (Circular-SCAN)

Detailed Description

Simulates the C-SCAN scheduling algorithm.

Simulates the C-SCAN algorithm and returns total amount of head movement required by the algorithm.

Author:

Aaron Alphonsus

Date:

28 April 2017

Function Documentation

int c_scan (int initial_*pos*, int request[REQUESTS])

C-SCAN (Circular-SCAN)

Simulates the C-SCAN algorithm. Function creates two arrays: for requests less than the initial head position, and for requests greater. The order in which they are processed depends on the initial direction of the disk head. In between processing each, we make sure that the position gets 'wrapped' around to simulate the circular list. The algorithm makes sure that the C-SCAN goes to the end by appending a 0 or CYLINDER-1 as the case may be.

Parameters:

in	*initial_pos*	Initial disk head position
in	*request*	The random request array generated

Returns:

Total amount of head movement

Place requests into an 'up' and 'down' vector which will be sorted

Assumes that == are processed first, therefore 0 head movement

If the head is moving left initially

Sort both up and down in descending order

C-SCAN goes down to 0 (as long as the up array is not empty)

Service the 'down' array

Head position wraps around to the last cylinder

Service 'up' array
Sort both up and down in ascending order
C-SCAN goes up to last cylinder (as long as down array is not empty)
Service 'up' array
Head position wraps around to the first cylinder
Service the 'down' array

disk_sched.cpp File Reference

Main file for the Disk-Scheduling Algorithms.

```
#include <iostream>
#include <stdlib.h>
#include "disk_sched.h"
```

Functions

- int **main** (int argc, char *argv[])

Detailed Description

Main file for the Disk-Scheduling Algorithms.

Creates request array and reads in initial disk head position. Passes these values into each function.

Compilation Instructions: make

Run: ./disk_sched initial_position

Author:

Aaron Alphonsus

Date:

28 April 2017

Function Documentation

int main (int argc, char * *argv*[])

Creates an array of random requests, and takes in initial disk head position from the command line. These are passed to each of the disk scheduling functions. The head movement for each function is printed out.

Parameters:

in	*argc*	Integer count of the command-line arguments
in	*argv*	Vector of the command-line arguments

Returns:

0 Indicates normal termination of main.

Check for correct number of command-line arguments

Read in initial disk head position

Array to hold random cylinder requests

Seed random number generator

Fill with rand mod number of cylinders

Print results

disk_sched.h File Reference

Disk scheduler header file. Contains function prototypes for each disk scheduling algorithm. Also defines number of requests and cylinders.

Macros

- #define **REQUESTS** 1000
- #define **CYLINDERS** 5000

Functions

- int **fcfs** (int initial_pos, int request[**REQUESTS**])
  First Come First Serve.
- int **sstf** (int initial_pos, int request[**REQUESTS**])
  Shortest Seek Time First.
- int **scan** (int initial_pos, int request[**REQUESTS**])
  SCAN (aka elevator)
- int **c_scan** (int initial_pos, int request[**REQUESTS**])
  C-SCAN (Circular-SCAN)
- int **look** (int initial_pos, int request[**REQUESTS**])
  LOOK.
- int **c_look** (int initial_pos, int request[**REQUESTS**])
  C-LOOK (Circular-LOOK)

Detailed Description

Disk scheduler header file. Contains function prototypes for each disk scheduling algorithm. Also defines number of requests and cylinders.

Author:
Aaron Alphonsus

Date:
28 April 2017

Macro Definition Documentation

#define CYLINDERS 5000

#define REQUESTS 1000

Function Documentation

int c_look (int initial_*pos*, int request [REQUESTS])

C-LOOK (Circular-LOOK)

Simulates the C-LOOK algorithm. Functionality of the C-LOOK was unclear regarding direction switching. Used description from here:
`http://courses.teresco.org/cs432_f02/lectures/17-files/17-files.html`
The direction switches once there are no pending requests in that direction.

The function creates two arrays: for requests less than the initial head position, and for requests greater. The order in which they are processed depends on the initial direction of the disk head.

Parameters:

in	*initial_pos*	Initial disk head position
in	*request*	The random request array generated

Returns:

Total amount of head movement

Place requests into an 'up' and 'down' vector which will be sorted

Assumes that == are processed first, therefore 0 head movement

If the head is moving left initially

Sort both up and down in descending order

Service the 'down' array

Head position wraps around to the highest request in 'up' array

Service 'up' array

Sort both up and down in ascending order

Service 'up' array

Head position wraps around to the lowest request in 'down' array

Service the 'down' array

int c_scan (int initial_*pos*, int request[REQUESTS])

C-SCAN (Circular-SCAN)

Simulates the C-SCAN algorithm. Function creates two arrays: for requests less than the initial head position, and for requests greater. The order in which they are processed depends on the initial direction of the disk head. In between processing each, we make sure that the position gets 'wrapped' around to simulate the circular list. The algorithm makes sure that the C-SCAN goes to the end by appending a 0 or CYLINDER-1 as the case may be.

Parameters:

in	*initial_pos*	Initial disk head position
in	*request*	The random request array generated

Returns:

Total amount of head movement

Place requests into an 'up' and 'down' vector which will be sorted

Assumes that == are processed first, therefore 0 head movement

If the head is moving left initially

Sort both up and down in descending order

C-SCAN goes down to 0 (as long as the up array is not empty)

Service the 'down' array

Head position wraps around to the last cylinder

Service 'up' array

Sort both up and down in ascending order

C-SCAN goes up to last cylinder (as long as down array is not empty)

Service 'up' array

Head position wraps around to the first cylinder

Service the 'down' array

int fcfs (int initial_*pos*, int request[REQUESTS])

First Come First Serve.

Simulates the first-come, first-served algorithm. Function loops through requests summing up each consecutive head movement.

Parameters:

in	*initial_pos*	Initial disk head position
in	*request*	The random request array generated

Returns:

Total amount of head movement

Loop through requests array summing up differences in head positions

int look (int initial_*pos*, int request[REQUESTS])

LOOK.

Simulates the LOOK algorithm. Functionality of the LOOK was unclear regarding direction switching. Used description from here:
`http://courses.teresco.org/cs432_f02/lectures/17-files/17-files.html`
The direction switches once there are no pending requests in that direction.

The function creates two arrays: for requests less than the initial head position, and for requests greater. The order in which they are processed depends on the initial direction of the disk head.

Parameters:

in	*initial_pos*	Initial disk head position
in	*request*	The random request array generated

Returns:

Total amount of head movement

Place requests into an 'up' and 'down' vector which will be sorted

Assumes that == are processed first, therefore 0 head movement

Sort 'down' in descending order and 'up' in ascending

If the head is moving left initially

Service the 'down' array

Service 'up' array

Service 'up' array

Service the 'down' array

int scan (int initial_*pos*, int request[REQUESTS])

SCAN (aka elevator)

Simulates the SCAN algorithm. Function creates two arrays: for requests less than the initial head position, and for requests greater. The order in which they are processed depends on the initial direction of the disk head. The algorithm makes sure the SCAN changes directions at each end by appending a 0 or CYLINDER-1 as the case may be.

Parameters:

in	*initial_pos*	Initial disk head position
in	*request*	The random request array generated

Returns:

Total amount of head movement

Place requests into an 'up' and 'down' vector which will be sorted

Assumes that == are processed first, therefore 0 head movement

Sort 'down' in descending order and 'up' in ascending

If the head is moving left initially

SCAN goes down to 0 (as long as the up array is not empty)

Service the 'down' array

Service 'up' array

SCAN goes up to last cylinder (as long as down array is not empty)

Service 'up' array

Service the 'down' array

int sstf (int initial_*pos*, int request[REQUESTS])

Shortest Seek Time First.

Simulates the shortest-seek-time-first algorithm. Function loops through requests looking for the request with the shortest seek time. The head movement is added and the next shortest seek time is found until the request array is exhausted.

Parameters:

in	*initial_pos*	Initial disk head position
in	*request*	The random request array generated

Returns:

Total amount of head movement

Set shortest seek time to largest value

Duplicate request queue

Look for request with shortest seek time in remaining request array

Swap element with shortest request time to the front

Add head movement

Move current position down

Reset shortest seek time

fcfs.cpp File Reference

Simulates the FCFS scheduling algorithm.

```
#include <cmath>
#include "disk_sched.h"
```

Functions

- int **fcfs** (int initial_pos, int request[**REQUESTS**])
  First Come First Serve.

Detailed Description

Simulates the FCFS scheduling algorithm.

Simulates the first-come, first-served algorithm and returns total amount of head movement required by the algorithm.

Author:

Aaron Alphonsus

Date:

28 April 2017

Function Documentation

int fcfs (int initial_*pos*, int request[REQUESTS])

First Come First Serve.

Simulates the first-come, first-served algorithm. Function loops through requests summing up each consecutive head movement.

Parameters:

in	*initial_pos*	Initial disk head position
in	*request*	The random request array generated

Returns:

Total amount of head movement

Loop through requests array summing up differences in head positions

look.cpp File Reference

Simulates the LOOK scheduling algorithm.

```
#include <algorithm>
#include <vector>
#include "disk_sched.h"
```

Functions

- int **look** (int initial_pos, int request[**REQUESTS**])
  LOOK.

Detailed Description

Simulates the LOOK scheduling algorithm.

Simulates the LOOK algorithm and returns total amount of head movement required by the algorithm.

Author:
Aaron Alphonsus

Date:
28 April 2017

Function Documentation

int look (int initial_*pos*, int request[REQUESTS])

LOOK.

Simulates the LOOK algorithm. Functionality of the LOOK was unclear regarding direction switching. Used description from here:
`http://courses.teresco.org/cs432_f02/lectures/17-files/17-files.html`
The direction switches once there are no pending requests in that direction.

The function creates two arrays: for requests less than the initial head position, and for requests greater. The order in which they are processed depends on the initial direction of the disk head.

Parameters:

in	*initial_pos*	Initial disk head position
in	*request*	The random request array generated

Returns:
Total amount of head movement

Place requests into an 'up' and 'down' vector which will be sorted

Assumes that == are processed first, therefore 0 head movement

Sort 'down' in descending order and 'up' in ascending

If the head is moving left initially

Service the 'down' array

Service 'up' array

Service 'up' array
Service the 'down' array

scan.cpp File Reference

Simulates the SCAN scheduling algorithm.

```
#include <algorithm>
#include <vector>
#include "disk_sched.h"
```

Functions

- int **scan** (int initial_pos, int request[**REQUESTS**])
  SCAN (aka elevator)

Detailed Description

Simulates the SCAN scheduling algorithm.

Simulates the SCAN algorithm and returns total amount of head movement required by the algorithm.

Author:
Aaron Alphonsus

Date:
28 April 2017

Function Documentation

int scan (int initial_*pos*, int request[REQUESTS])

SCAN (aka elevator)

Simulates the SCAN algorithm. Function creates two arrays: for requests less than the initial head position, and for requests greater. The order in which they are processed depends on the initial direction of the disk head. The algorithm makes sure the SCAN changes directions at each end by appending a 0 or CYLINDER-1 as the case may be.

Parameters:

in	*initial_pos*	Initial disk head position
in	*request*	The random request array generated

Returns:
Total amount of head movement

Place requests into an 'up' and 'down' vector which will be sorted

Assumes that == are processed first, therefore 0 head movement

Sort 'down' in descending order and 'up' in ascending

If the head is moving left initially

SCAN goes down to 0 (as long as the up array is not empty)

Service the 'down' array

Service 'up' array

SCAN goes up to last cylinder (as long as down array is not empty)
Service 'up' array
Service the 'down' array

sstf.cpp File Reference

Simulates the SSTF scheduling algorithm.

```
#include <algorithm>
#include <cmath>
#include "disk_sched.h"
```

Functions

- int **sstf** (int initial_pos, int request[**REQUESTS**])
  Shortest Seek Time First.

Detailed Description

Simulates the SSTF scheduling algorithm.

Simulates the shortest-seek-time-first algorithm and returns total amount of head movement required by the algorithm.

Author:

Aaron Alphonsus

Date:

28 April 2017

Function Documentation

int sstf (int initial_*pos*, int request[REQUESTS])

Shortest Seek Time First.

Simulates the shortest-seek-time-first algorithm. Function loops through requests looking for the request with the shortest seek time. The head movement is added and the next shortest seek time is found until the request array is exhausted.

Parameters:

in	*initial_pos*	Initial disk head position
in	*request*	The random request array generated

Returns:

Total amount of head movement

Set shortest seek time to largest value

Duplicate request queue

Look for request with shortest seek time in remaining request array

Swap element with shortest request time to the front

Add head movement

Move current position down

Reset shortest seek time