Name: Amit Nijjar Pid: A11489111 email: a2nijjar@ucsd.edu

CSE 120 Homework #4

Spring 2016 (Kesden)

**1. Disk Scheduling**

Please consider the disk specification and trace of track-requests given below. Please determine the mean and median service time for each policy: FIFO, SCAN, C-SCAN, and SSTF. (As an aside, the median is interesting, because it is less sensitive to outliers than the mean).

Also determine the *average unfairness* of each approach, where the average unfairness is the average number of requests that a request is displaced from its position in FIFO ordering, e.g. on average, how many requests have gotten ahead of or fallen behind a request. Please note that the displacement should be in absolute values: negative displacements should not compensate for posititive displacements, since each movement is equally unfair.

**Disk Specification:**

Please assume the disk has 1000 tracks and a starting position of 499. Furthermore, please assume that the track-to-track seek is 1mS and the maximum (full stroke) seek time is 10mS.

Please also assume that the first seek for SCAN and C-SCAN will be in the direction of increasing track number and that the head, given two equal choices, prefers to continue in the direction of the current sweep.

**Request Trace:** 700, 200, 225, 450, 650, 300, 200, 500, 750, 650

FIFO – avg seek 5.5ms; avg unfairness 0

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Track | 700 | 200 | 225 | 450 | 650 | 300 | 200 | 500 | 750 | 650 |
| Seek time | 5 | 11 | 1.5 | 5.5 | 5 | 8 | 4 | 7 | 6 | 3 |
| Displacement | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Scan – avg seek 2.4 ms; avg unfairness 4.1

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Track | 500 | 650 | 650 | 700 | 750 | 450 | 300 | 225 | 200 | 200 |
| Seek time | 1 | 4 | 0 | 2 | 2 | 7 | 4 | 2.5 | 1.5 | 0 |
| Displacement | 7 | 3 | 7 | 2 | 4 | 2 | 1 | 5 | 7 | 3 |

C-Scan – avg seek 3.7; avg unfairness 4.1

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Track | 500 | 650 | 650 | 700 | 750 | 200 | 200 | 225 | 300 | 450 |
| Seek time | 1 | 4 | 0 | 2 | 2 | 10 | 0 | 1.5 | 2.5 | 4 |
| Displacement | 7 | 3 | 7 | 2 | 4 | 2 | 1 | 5 | 7 | 3 |

Sstf – avg seek 2.5; avg unfairness 2.9

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Track | 500 | 450 | 300 | 225 | 200 | 200 | 650 | 650 | 700 | 750 |
| Seek time | 1 | 2 | 4 | 2.5 | 1.5 | 0 | 10 | 0 | 2 | 2 |
| Displacement | 7 | 3 | 7 | 2 | 4 | 4 | 0 | 5 | 3 | 6 |

**2. File Size**

Consider a UNIX-style inode with 14 direct pointers, one single- indirect pointer, and one double-indirect pointer only. Assume that the block size is 4K bytes, and that the size of a pointer is 4 bytes. How large a file can be indexed using such an inode?

Pointers per block = 4K/4 = 1024

(14 x 4K) + (1024 x 4k) + (1024 x 1024 x 4K)

4K (14 + 1024 + (1024 x 1024) )

2^12 ( 14 + 2^10 + (2^20) )

= 4299218944

Name: Amit Nijjar Pid: A11489111 email: a2nijjar@ucsd.edu

3. **Caching Schemes**

In class we discussed *unified caches* in which the demand paging system and file system share are common cache, as well as a *segregated caches* in which one pool of buffers is used to cache pages for the virtual memory system and another blocks for the file system.

1. What high-level constraint is imposed by a unified cache that doesn’t exist under a segregated cache?

Unified caches have a higher cost to provide bandwidth enough for instruction and data operations every clock cycle. There is more overhead with unified caches.

1. What are the advantages of a unified caching scheme in general purpose environments

A unified cache can better determine if a program needs more data references than instruction references and vice versa. Because of this, unified caches better perform load balancing.

1. Under what circumstances, when a unified cache is possible, might a segregated cache perform better?

When programs use equal amounts of instruction references and data references, split caches are better because a split cache can perform two references at the same time.

Name: Amit Nijjar Pid: A11489111 email: a2nijjar@ucsd.edu

4. **File System Data Structures**

(a) Suppose that the disk file “foobar.txt” consists of the six ASCII characters “CSE 120 is my most favorite class!”. What is the output of the following program?

Output: “CSE 12CSE 120”

int main() {

char buffer[256];

int bytes;

int fd = open ('foobar.txt', O\_RDONLY);

if (!fork()) { /\* child \*/

bytes = read(fd, buf, 3);

write (1, buf, bytes);

} else { /\* parent \*/

wait (NULL);

bytes = read(fd, buf, 4));

write (1, buf, bytes);

}

bytes = read(fd, buf, 3);

write (1, buf, bytes);

return 0;

}

Parent returns: “CSE” “ 12”

Child writes out “CSE ” “120”

Output : “CSE 12CSE 120”

Name: Amit Nijjar Pid: A11489111 email: a2nijjar@ucsd.edu

5. **File System Data Structures**

(a) Suppose that the disk file “foobar.txt” consists of the six ASCII characters “foobar”. What is the output of the following program?

Output: buf = “CSE 1200000000000000”

/\* any necessary includes \*/

char buf[20] = {0}; /\* init to all zeroes \*/

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

int fd1 = open("foobar.txt’, O\_RDONLY);

int fd2 = open(‘foobar.txt", O\_RDONLY);

/\* Google this. It copies the fd table entry indexed by fd1 (right

argument)into the fd table entry indexed by fd2 (left arg) \*/

dup2(fd2, fd1)); //fd1 now refers to the same value as fd2

read(fd1, buf, 3);

close(fd1);

read(fd2, buf[3], 3)

close(fd2); printf("buf = %s\n"; buf);

return 0;

}

(b) Now consider the identical program, except that dup2 is commented out. What is the output?

Output: buf = “CSECSE00000000000000”

/\* any necessary includes \*/

char buf[20] = {0}; /\* init to all zeroes \*/

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

int fd1 = open("foobar.txt’, O\_RDONLY);

int fd2 = open(‘foobar.txt", O\_RDONLY);

/\* Google this. It copies the fd table entry indexed by fd1 (right

argument)into the fd table entry indexed by fd2 (left arg) \*/

/\* COMMENTED OUT: dup2(fd2, fd1)); \*/

read(fd1, buf, 3);

close(fd1),

read(fd2; buf[3], 3)

close(fd2); printf("buf = %s\n"; buf);

return 0;

}

Name: Amit Nijjar Pid: A11489111 email: a2nijjar@ucsd.edu

6. **File I/O**

Consider the code below. Assume that the disk file file.txt contains the string (as characters, without the quotes) of characters “120120” . What will be the output when this code is compiled and run?

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

char buf[3] = "ab";

int r = open("file.txt", O\_RDONLY);

int r1, r2, pid;

/\* Google this. It copies the r*th* file descriptor into a new file

\* descriptor table entry and returns the new file descriptor

\*/

r1 = dup(r); //r1 is a copy of r

read(r, buf, 1);

if((pid=fork())==0) {

r1 = open("file.txt’, O\_RDONLY);

} else{

waitpid(pid, NULL, 0);

}

read(r1, buf+1, 1)

printf("%s", buf);

return 0;

}

Child returns 11

Parent returns 12

Output = 1112