

Labs

Neo Pi

Neo.Pi1@Marist.edu

September 15, 2024

1 LAB ONE

1.1 WHAT ARE THE ADVANTAGES AND DISADVANTAGES OF USING THE SAME SYSTEM CALL INTERFACE FOR MANIPULATING BOTH FILES AND DEVICES?

The advantage of using the same system call interface for both files and devices is so that each device can be accessed as though it was a file within the system. Since kernel already deals with devices through a file interface, adding code to support a different kind of file interface is possible. The disadvantage of using the same interface is that the functionality of some devices might not be able to be accessed. Loss of functionality or performance could be a result.

1.2 WOULD IT BE POSSIBLE FOR THE USER TO DEVELOP A NEW COMMAND INTERPRETER USING THE SYSTEM CALL INTERFACE PROVIDED BY THE OPERATING SYSTEM? HOW?

Yes, a user can develop a new command interpreter by using the system call interface provided by the operating system. You can develop it by specifically using system calls to manage processes and file-related system calls for i/o operations.

2 LAB TWO

2.1 HOW IS YOUR CONSOLE LIKE THE ANCIENT TTY SUBSYSTEM IN UNIX AS DESCRIBED IN [HTTPS://WWW.LINUXASKESSON/PROGRAMMING/TTY/](https://www.linuxaskesson.com/programming/tty/)?

Originally, the TTY system was designed for text-based input and output. Similarly, the current console we are working on operates the same way. It takes character-based commands, and handles commands by processing input as text characters. The input is buffered line-by-line, just as it is in our own OS. Another important factor, is that the TTY subsystem was the primary interface for the system, and similarly our OS provides a similar interactive interface. Lastly, the interaction is all in real time, when issuing commands and receiving feedback.

3 LAB THREE

3.1 EXPLAIN THE DIFFERENCE BETWEEN INTERNAL AND EXTERNAL FRAGMENTATION.

Internal fragmentation happens when memory is allocated in block larger than what is required. The memory that is not used inside the allocated block becomes useless if the process does not require the whole block. External fragmentation happens when there is enough free memory, but the free memory is scattered in different blocks. Even though there is enough free memory, there are no single blocks large enough for the request. Overall, internal fragmentation contains wasted space within memory blocks, while external fragmentation has wasted space between memory blocks.

3.2 GIVEN FIVE MEMORY PARTITIONS OF 100KB, 500KB, 200KB, 300KB, AND 600KB (IN THAT ORDER), HOW WOULD OPTIMAL, FIRST-FIT BEST-FIT, AND WORST-FIT ALGORITHMS PLACE PROCESSES OF 212KB, 417KB, 112KB, AND 426KB (IN THAT ORDER)?

For optimal placement, it would work like this: 212KB : 300KB 417KB : 500KB 112KB : 200KB 426KB : 600KB

First-Fit: 212KB : 500KB 417KB : 600KB 112KB : 200KB 426KB :

Best-Fit: 212KB : 300KB, 417KB : 500KB, 112KB : 200KB, 426KB : 600KB

Worst-Fit: 212KB : 600KB, 417KB : 500KB, 112KB : 300KB, 426KB :

4 PROBLEM THREE

REFERENCES

5 APPENDIX

5.1 SOME JAVASCRIPT SOURCE CODE LISTINGS

```
1 var A = [5,0,5,6,6,8,45,50];
2
3 function solve(A) {
4     // Base case to stop the recursion.
5     if (A.length == 1) {
6         if (A[0] % 5 == 0) {
7             var retVal = 1;
8         } else {
9             var retVal = 0;
10        }
11        return retVal;
12    } else {
13        // Divide.
14        var divPoint = Math.floor( A.length / 2);
15        var left  = A.slice(0, divPoint);
16        var right = A.slice(divPoint, A.length);
17
18        // Conquer.
19        var left5s  = solve(left, level+1);
20        var center5s = straddle(left, right);
21        var right5s = solve(right, level+1);
22
23        // Combine.
24        return Math.max(left5s, Math.max(center5s, right5s));
25    }
26 }
27
28 function straddle(left, right) {
29     var retVal = 0;
30     if ((left[left.length-1] % 5 == 0) && (right[0] % 5 == 0)) {
31         // Count back the 5's on the left going from right to left.
32         var leftCount = 0;
33         var index = left.length-1;
34         while ( (index >= 0) && (left[index] % 5 == 0) ) {
35             index--;
36             leftCount++;
37         }
38         // Count forward the 5's on the right going from left to right.
39         var rightCount = 0;
40         while ( (rightCount < right.length) && (right[rightCount] % 5 == 0) ) {
41             rightCount++;
42         }
43         // Return the sum of the straddling 5s on the left and right.
44         retVal = leftCount + rightCount;
45     }
46     return retVal;
47 }
```