CST 334: Operating Systems

Dr. Glenn Bruns

# AWK programming

**Purpose**. AWK is a little language that is powerful and handy for command line scripting. The purpose of this assignment is to build your AWK programming skills.

**Instructions**. You will create 5 small AWK programs. On mlc104, the directory /home/CLASSES/brunsglenn/cst334/hw/hw5 contains a tar file hw6.tar. Copy this to your own directory and untar it. You will see a folder 'awk\_problem'. Underneath this directory is one sub-directory for each of the five small awk programs you need to write. In each of these subdirectories is the file for the awk program you need to write, a Makefile, and a test script test1.sh.

You do not need advanced features of AWK for these scripts. Please do not use 'getline' or 'next', and stick to basic AWK programming.

Your awk programs will be run on the output of an OSTEP simulator. Here’s an example.

ptr[2] = Alloc(5) returned 1001 (searched 3 elements)

Free List [ Size 3 ]: [ addr:1000 sz:1 ] [ addr:1006 sz:2 ] [ addr:1008 sz:92 ]

ptr[3] = Alloc(8) returned 1008 (searched 3 elements)

Free List [ Size 3 ]: [ addr:1000 sz:1 ] [ addr:1006 sz:2 ] [ addr:1016 sz:84 ]

Free(ptr[3]) returned 0

Free List [ Size 4 ]: [ addr:1000 sz:1 ] [ addr:1006 sz:2 ] [ addr:1008 sz:8 ] [ addr:1016 sz:84 ]

You need to understand this output a little. The idea is that if a program needs memory (for example, to build a data structure) it makes an **Alloc()** call, and when the program is done with the memory, it makes a **Free()** call. For example, in the first line above a program calls Alloc(5) to get 5 bytes of memory. The Alloc() call is successful, so the return value (shown as ptr[2]) is a pointer to the allocated chunk of 5 bytes of memory.

The operating system keeps track of memory that is available to allocate to processes by using a "free list". Look at the second line in the example above. This shows that, after the Alloc(5) call, the operating system has a free list containing three "chunks" of memory. The first chunk is at address 1000 and is only 1 byte long. The second chunk is at address 1006 and is 2 bytes long. The third chunk is at address 1008 and is 92 bytes long.

Look at line 4. After the Alloc(8) call, the third chunk of memory is now 84 bytes, not 92 bytes. That's because 8 bytes of the third chunk were made available to the program that called Alloc(8). It was a successful Alloc() call. If an Alloc(100) call were made at this point, the value -1 would be returned, indicating that the Alloc() call failed. It failed because no chunk in the free list had at least 100 bytes.

1. Write an awk script freesize.awk that gets the size of the free list after every “Free” or “Alloc” operation. Your program should act like this:

$ awk -f freesize.awk malloc-out.txt | wc -l

1000

$ awk -f freesize.awk malloc-out.txt | tail -5

41

42

41

42

43

1. Write an awk script count\_allocs.awk that counts the number of successful allocs and the number of failed alloc calls. Your program should act like this:

$ awk -f count\_allocs.awk malloc-out.txt

num successes: 448; num failures: 106

1. Write an awk script num\_bytes.awk that records the number of bytes requested in each alloc call. Your program should act like this:

$ awk -f num\_bytes.awk malloc-out.txt | wc -l

554

$ awk -f num\_bytes.awk malloc-out.txt | head -200 | tail -5

1

10

3

10

6

1. Write an awk script succ\_reqs.awk that prints the number of bytes requested, and then a 1 or a 0 depending on whether the request was successful (1 means success). Your program should act like this:

$ awk -f succ\_reqs.awk malloc-out.txt | tail -5

8 0

3 1

5 1

10 0

6 1

1. Write an awk script list\_sizes.awk that prints the size of every element in the free list, in order, after each Free or Alloc operation. Your program should act like this:

$ awk -f list\_sizes.awk malloc-out.txt | head

99

1 99

1 92

1 7 92

1 2 92

1 2 84

1 2 8 84

1 5 2 8 84

5 2 8 84

2 8 84

Hint: you can use ‘printf’ and loops in awk programs. In both cases the syntax is similar to C. Here is an example of an awk program with a loop:

{ n = $1

for (i=0; i < n; i++) {

printf(“%s ”, $(2+i))

}

}

This program assigns the value in field 1 to variable n, then prints fields 2, 3, 4 up to field 2 + n - 1.

**Testing your code**. You know how to use the test scripts to test your code. Read the test scripts so that you understand how your code is tested. Read the Makefile in each subdirectory and use it if you like.

**Submitting.** Submit your 5 scripts as separate files on iLearn. Do not submit a zip file or other archive file. Do not change the names of your scripts -- my grading software will not find them if you do.

**Grading**. Each program is worth 10 points. Some of the tests I use to grade your awk scripts will differ from the scripts I have provided you. I will deduct 3 points per script if your code is untidy.