Name: Dayu Liu Career Account ID (email): liu1589@purdue.edu

Answers are on page 3 to page 5

Assignment 6

Due: December 6th (Friday), 11:59 pm on Blackboard.

The complexity of operating systems code is difficult to assess. A software engineer may spend many weeks designing a module, even though the resulting program only contains a few hundred lines of code. Another module may require thousands of lines of code, but can be written quickly without spending time thinking.

Independent of the intellectual effort, the size of source code can be useful in estimating the amount of detail involved in a particular module. This homework asks you to measure the lines of source code in various modules of Xinu, plot a pie chart of the results, and draw conclusions about the level of detail.

1. (5 pts) Consider the following modules in Xinu:

- Process management
- Inter-process communication (high and low level)
- Process coordination
- Memory management (high and low level)
- Drivers for hardware devices (Ethernet and tty devices)
- Network protocols
- Shell and shell applications
- Local file system
- Remote disk system
- Remote file system
- Configuration program (specifies which device drivers to use)

Before looking at the code, make a guess on how the modules will rank from largest to smallest. Then, count the actual lines of code for each module, and list them from largest to smallest.

For the purpose of the homework, we added some extra notes:

- Source code
  Use Lab 7 tarball. You can download by running the following command:

  ```
  tar xzvf /u/u3/cs354/Lab7/Xinu-code-BeagleBoneBlack.tar.gz
  ```

- Definition of 'code'
  All .h and .c files. Assembly code files (.s) are optional. An exception would be the files in config, where you may include all.

- Counting the lines
  Include every line in the code, which includes the lines with comments, empty lines, declarations, externs, macros, etc.

- List of files per module

  For each module, list the name of the files/folders that you have included in the count. For example, ‘a.h in include folder, and b.c, c.c in system folder’. If you want to enumerate all the files in a folder, you may write as ‘folder_name/*’, or ‘a/b/*’ for the nested case.

- Files to ignore

  The following lists the files/folders that you can safely ignore.

  compile/*, cross_compiler/*, lib/*, DESCRIPTION, include: prototypes.h, xinu.h, kernel.h, system: initialize.c.

- System

  Some of the functions in ‘system’ folder have nested calls (e.g., create() calls getstk() which calls disable()), or implicit calls (e.g., when a process returns from top-level function, userret() is called). You don’t have to worry about counting them; only count the functions that are directly related to each module. For example, create() is a part of process management since it’s responsible for creating a process. Count the number of lines in create.c, but not those in getstk.c.

- Header

  Please count the code in the header as well. Only count the ones that are directly related to each module. For example, process.h is relevant for process management and coordination (process table entry is used).

- Duplicates

  If a header or a function happens to be belonging to multiple modules, you may choose to add the count for each module. Just make sure to mention the file name per module.

- Folder

  Some of the modules have all the source code stored in a *dedicated* folder (i.e., the folder does not contain any code irrelevant to that module). In this case, you can simply count all the files in that folder.

2. (5 pts) Plot a pie chart of the counts found above.

3. (5 pts) List one module that agrees the most with your guess, and an another one furthest from your guess.

4. (5 pts) In terms of the size of the source code, which module surprised you the most by being smaller or larger than you expected?

Submission instruction:
Submit your PDF file on Blackboard before the due date.

Q1:

My guess on the rank from largest to smallest is:

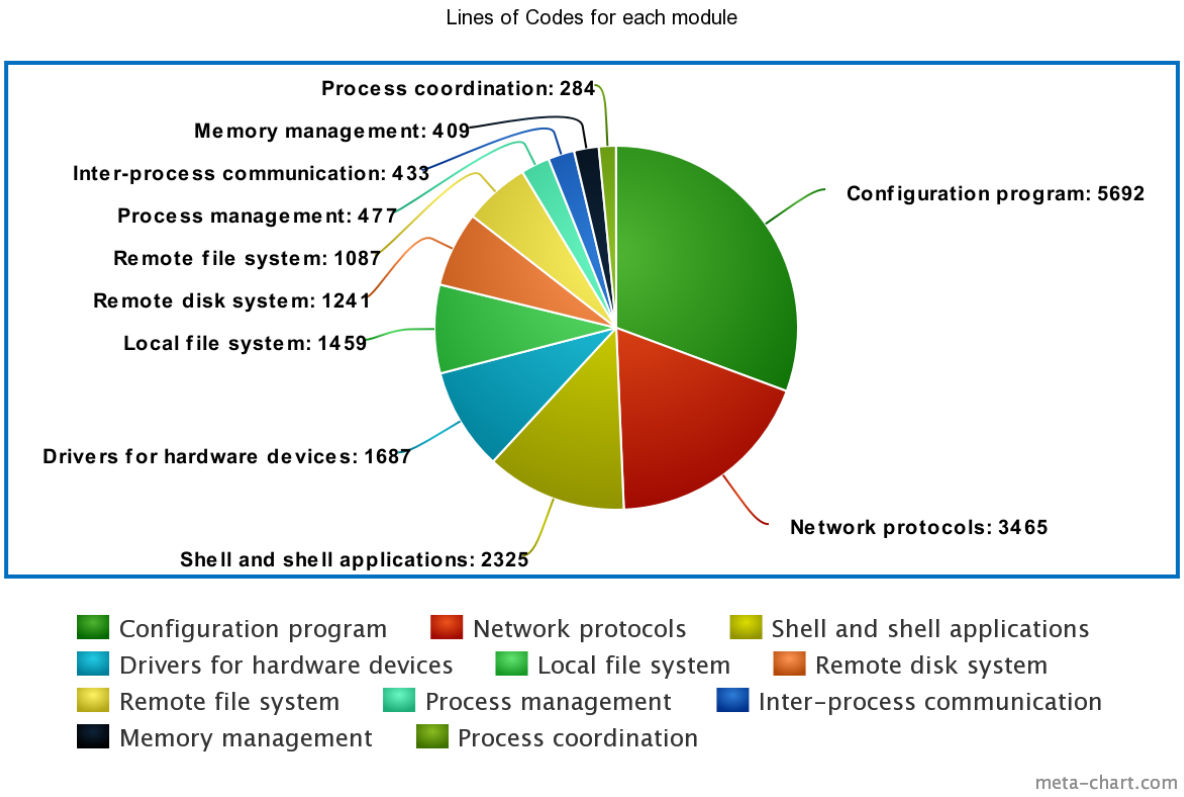
1. Process management
2. Memory management (high and low level)
3. Shell and shell applications
4. Drivers for hardware devices (Ethernet and tty devices)
5. Inter-process communication
6. Process coordination
7. Local file system
8. Remote disk system
9. Remote file system
10. Network protocols
11. Configuration program (specifies which device drivers to use)

The actual counts of lines of code ranked from largest to smallest is (with list of files):

	A	B	C
1	Modules in Xinu	Files in each module	Lines of code in each file
2	Configuration program (specifies which device drivers to use)		Total: 5692
3		config/*	5692
4	Network protocols		Total: 3465
5		include/net.h	70
6		device/net/*	3395
7	Shell and shell applications		Total: 2325
8		include/shell.h	76
9		shell/*	2249
10	Drivers for hardware devices (Ethernet and tty devices)		Total: 1687
11		device/tty/*	766
12		device/eth/*	719
13		include/ether.h	117
14		include/tty.h	85
15	Local file system		Total: 1459
16		include/lfilesys.h	177
17		device/lfs/*	1282
18	Remote disk system		Total: 1241
19		include/rdisksys.h	236
20		device/rds/*	1005
21	Remote file system		Total: 1087
22		include/rfilesys.h	293
23		device/rfs/*	794
24	Process management		Total: 477
25		include/process.h	63
26		system/resched.c	81
27		include/resched.h	17
28		system/ready.c	30
29		system/suspend.c	42
30		system/resume.c	63
31		system/kill.c	60
32		system/userret.c	13
33		system/create.c	108
34	Inter-process communication		Total: 433
35		system/receive.c	26
36		system/send.c	42
37		system/recvclr.c	26
38		include/ports.h	30
39		system/ptinit.c	46
40		system/ptcreate.c	47
41		system/ptsend.c	60
42		system/ptrecv.c	49
43		system/ptdelete.c	28
44		system/ptreset.c	28
45		system/ptclear.c	51
46	Memory management (high and low level)		Total: 409
47		include/memory.h	39
48		system/getmem.c	51
49		system/freemem.c	71
50		system/getstk.c	53
51		system/mkbufpool.c	56
52		system/getbuf.c	44
53		system/freebuf.c	40
54		system/meminit.c	30
55		include/bufpool.h	25
56	Process coordination		Total: 284
57		include/semaphore.h	23
58		system/wait.c	40
59		system/signal.c	32
60		system/signaln.c	38
61		system/semcreate.c	51
62		system/semdelete.c	37
63		system/semcount.c	27
64		system/semreset.c	36

Q2:

Pie chart of the counts



Q3:

1. The module that agrees the most with your guess: Shell and shell applications
2. The module furthest from your guess: Configuration program

Q4:

The module “Configuration program” surprised me the most. I didn’t realized configuration can take so many lines of code because I thought it would be just a small addition to the whole operating system which I thought would be fundamentally larger.