ICT1012

OPERATING SYSTEMS

LABORATORY INSTRUCTIONS

Quiz 1

Contents

Part A: MCQs (5 marks)

Refer to "LMS/xSiTe – Examena Quiz" for MCQ Quiz questions.

Part B: Lab extension tasks (8 marks)

Prerequisite:

Download the "lab.zip" file from "LMS/xSiTe" folder:

"Lab Quizzes (Prerequisites) / Quiz 1".

Question 1: "sixfive" (4 marks)

- Your solution should be in the file "**user/sixfive.c**".
- For this task, use the system calls "*open*" and "*read*", "*C strings*", and processing text files in C.
- For each input file, "sixfive" must print all the numbers in the file that are a sequence of the characters 5 and 6 with no more than 3 digits.
- Numbers are a sequence of decimal digits separated by characters in the string " -\r\t\n./,".
- For the six in "*xv6*", "*sixfive*" shouldn't print 6 but, for example, *"/6,"* it should.
- Hints:
  - Read the input file a character at the time
  - Test if a character match any of the separators using "*strchr*" (see "*user/ulib.c*").
  - Start and end of file are implicit separators.
  - Add the program to "*UPROGS*" in "*Makefile*".
  - Changes to the file system persist across runs of "*qemu*"; to get a clean file system run "**make clean**" and then "**make qemu**".
- The following example illustrates "*sixfive's*" behavior:

```
$ sixfive sixfive.txt
555
56
666
565
$
```

Question 2: "memdump" (4 marks)

- Refer to the task "*memdump*" from the lab assignment "*xv6labs-w1*".
- Your solution should be in the file "**user/memdump.c**".
- The task is to extend and add a new format option "**q**" that would take a 16-bytes of Hex string argument and to print the corresponding ASCII characters; ignore non-printable ASCII characters.
- Keep all other previous formats (i, p, h, c, s, S).

Code snippets:

```
// Convert a single hex character ('0'-'9', 'a'-'f', 'A'-'F') to its
numeric value
int hexval(char c)
{
     if(c >= '0' && c <= '9') return c - '0';
     if(c >= 'a' && c <= 'f') return c - 'a' + 10;
     if(c >= 'A' && c <= 'F') return c - 'A' + 10;
     return -1; // invalid character
}
```

```
// add the following to the switch/if statement in memdump()

case 'q': {
    // Loop over 16 hex characters (8 bytes)
    for(int j = 0; j < 16; j += 2){
          int hi = hexval(p[j]); // high nibble of the byte
          int lo =   ; // fill the missing code

      if(hi < 0 || lo < 0)
          continue; // skip invalid hex chars

      // combine high and low nibbles into a byte
      char v = ; // fill the missing code

      // printable only
      if(v >=  && v <= ) // fill the missing values. Refer to the ASCII Table.
         printf("%c", v); // print printable ASCII
    }
    printf("\n");
    p += 16; // move pointer past the 16 hex chars
    break;
}
```

Sample output:

```
$
$ echo 5465737420313233 | memdump q
Test 123
```

Reference: ASCII Table

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	NUL	25	19	EM	51	33	3	77	4D	M	103	67	g
1	01	SOH	26	1A	SUB	52	34	4	78	4E	N	104	68	h
2	02	STX	27	1B	ESC	53	35	5	79	4F	O	105	69	i
3	03	ETX	28	1C	FS	54	36	6	80	50	P	106	6A	j
4	04	EOT	29	1D	GS	55	37	7	81	51	Q	107	6B	k
5	05	ENQ	30	1E	RS	56	38	8	82	52	R	108	6C	l
6	06	ACK	31	1F	US	57	39	9	83	53	S	109	6D	m
7	07	BEL	32	20	space	58	3A	:	84	54	T	110	6E	n
8	08	BS	33	21	!	59	3B	;	85	55	U	111	6F	o
9	09	HT	34	22	"	60	3C	<	86	56	V	112	70	p
10	0A	LF	35	23	#	61	3D	=	87	57	W	113	71	q
11	0B	VT	36	24	$	62	3E	>	88	58	X	114	72	r
12	0C	FF	37	25	%	63	3F	?	89	59	Y	115	73	s
13	0D	CR	38	26	&	64	40	@	90	5A	Z	116	74	t
14	0E	SO	39	27	'	65	41	A	91	5B	[	117	75	u
15	0F	SI	40	28	(	66	42	B	92	5C	\	118	76	v
16	10	DLE	41	29	)	67	43	C	93	5D	]	119	77	w
17	11	DC1	42	2A	*	68	44	D	94	5E	^	120	78	x
18	12	DC2	43	2B	+	69	45	E	95	5F	_	121	79	y
19	13	DC3	44	2C	,	70	46	F	96	60	`	122	7A	z
20	14	DC4	45	2D	-	71	47	G	97	61	a	123	7B	{
21	15	NAK	46	2E	.	72	48	H	98	62	b	124	7C	|
22	16	SYN	47	2F	/	73	49	I	99	63	c	125	7D	}
23	17	ETB	48	30	0	74	4A	J	100	64	d	126	7E	~
24	18	CAN	49	31	1	75	4B	K	101	65	e	127	7F	DEL
			50	32	2	76	4C	L	102	66	f			

Part C: Challenge – "Swap32 – 32-bit Endian Swap" (7 marks)

This challenge is designed to test your understanding of user-space programming, bitwise operations, and kernel system calls in xv6.

With reference to the given example for 16-bit Endian Swap

```
#include "kernel/types.h"
#include "kernel/stat.h"
#include "user/user.h"

int main(int argc, char *argv[])
{
  If (argc != 2) {
    fprintf(2, "usage: swap16 <16-bit hex value>\n");
    exit(1);
  }
```

```
  // Parse hexadecimal (base 16)
  uint val = strtol(argv[1], 0, 16) & 0xFFFF;

  // 8-bit (byte) swap of 16-bit value
  uint high = (val & 0xFF00) >> 8;
  uint low  = (val & 0x00FF);
  uint swapped = (low << 8) | high;

  printf("Input:  0x%04x\n", val);
  printf("Output: 0x%04x\n", swapped);

  exit(0);
}
```

And example execution:

```
$ swap16 0x1234
Input:  0x1234
Output: 0x3412
```

Part 1: User-Space 32-bit Endian Swap

Implement a user-space program "**swap32_sys.c**" in the "**user**" directory.

The program should:

- Accept one argument, a 32-bit hexadecimal value (e.g., 0x1234ABCD).
- Perform a byte-level endian swap.
- Print both the input and swapped output in hexadecimal

Hints:

- Use bit masking and shifts (<<, >>) to extract and reposition each byte.
- Combine the bytes using the bitwise OR (|) operator.

Part 2: Kernel System Call – 32-bit Endian Swap

Implement a system call "*endianswap(uint x)*" and a user program "swap32_sys.c" to call your syscall.

Step 1: Implement the syscall

- Add a new "*syscall endianswap(uint x)*" in xv6 kernel.
- Implement the logic in "*kernel/sysproc.c*".
- Use "*argint()*" to retrieve the argument.
- Perform byte-level swap of a 32-bit integer in kernel space.
- Register the syscall.

Step 2: User Program to test the system call

- Create "**swap32_sys.c"** in the "*user*" directory.
- Accept one 32-bit hex value.
- Call the "*endianswap*" system call.
- Print the input and then print the swapped output in hexadecimal in the next line.

```
Input:  0x12345678
Output: 0x78563412
```

- If the argument is invalid, print the input and then print "**Invalid argument**" in the next line.

```
Input:  0xZYXWVUTS
Invalid argument
```

Quiz submission

1. After "**make grade**" and "**make zipball**", submit the "**lab.zip**" file to Gradescope Assignment "**xv6labs-quiz1**" for autograding.
2. Autograder will execute the same "**make grade**" when you submit your "**lab.zip**" file. This is for submission verification only. The score will not be included in the total score.
3. Test case script may not cover all the possibilities and it is your responsibility to carry out further testing,
4. After the quiz due date and time, **all submissions will be regraded again** with different set of test cases for Part B and C.
5. To ensure your submission goes through without technical issues, please submit at least 5 minutes before the deadline.