Introduction to Computer Systems Homework

Vincent Lee
ihalbmond@gmail.com

2017年9月16日

目录

1 A Tour of Computer Systems

2 Representing and Manipulating Information

2.53

```c
#define POS_INFINITY 1e400
#define NEG_INFINITY (-POS_INFINITY)
#define NEG_ZERO (-1.0/POS_INFINITY)
```

2.60

suppose we number the bytes in a w-bit word from 0 (less significant) to w/8-1 (most significant). write code for the followign c function, which will return an unsigned value in which byte i of argument x has been replaced by byte b:

```c
unsigned replace_byte (unsigned x, int i, unsigned char b)
{
    unsigned char *a = (unsigned char*)(&x);
    a[i] = b;
    return x;
}
```

2.61

Write C expressions that evaluate to 1 when the following conditions are true, and to 0 when they are false. Assume x is of type int.

(A) Any bit of x equals 1.

(B) Any bit of x equals 0.

(C) Any bit in the least significant byte of x equals 1.

(D) Any bit in the most significant byte of x equals 0.

Your code should follow the bit-level integer coding rules (page 120), with the additional restriction that you may not use equality (==) or inequality (!=) tests.

```
!!x
!!(~x)
!!(x << ( (sizeof(int) - 1) << 3 ) )
!!(~(x >> ( (sizeof(int) -1) << 3 ) ) )
```

2.65

Write code to implement the following function:

```
/* Return 1 when x contains an odd number of 1s; 0 otherwise. Assume w=32. */
int odd_ones(unsigned x);
```

Your function should follow the bit-level integer coding rules (page 120), except that you may assume that data type int has w = 32 bits. Your code should contain a total of at most 12 arithmetic, bit-wise, and logical operations.

```
/*
    using divide and conquer algorithm
*/
int odd_ones(unsigned x)
{
    x ^= x >> 16;
    x ^= x >> 8;
    x ^= x >> 4;
    return 0x69966996 >> x & 1;
}
```

2.86

Intel-compatible processors also support an “extended precision” floating-point format with an 80-bit word divided into a sign bit, k = 15 exponent bits, a single integer bit, and n = 63 fraction bits. The integer bit is an explicit copy of the implied bit in the IEEE floating-point representation. That is, it equals 1 for normalized values and 0 for denormalized values. Fill in the following table giving the approximate values of some “interesting” numbers in this format:

	Extended precision	
Description	value	Decimal
Smallest positive denormalized	2^{-16445}	3.6452×10^{-4951}
Smallest positive normalized	2^{-16382}	3.3621×10^{-4932}
Largest normalized	$(2 - 2^{-63}) \times 2^{16383}$	1.1897×10^{4932}

2.95

Following the bit-level floating-point coding rules, implement the function with the following prototype:

```
/* Compute 0.5*f. If f is NaN, then return f. */
float_bits float_half(float_bits f);
```

For floating-point number f, this function computes $0.5 * f$. If f is NaN, your function should simply return f. Test your function by evaluating it for all 2^{32} values of argument f and comparing the result to what would be obtained using your machine’s floating-point operations.

```
float_bits float_half(unsigned uf) {
    unsigned char exp_bit = uf >> 23; // unsigned int to unsigned char, overflow
    if (exp_bit == 0xff) // f is NaN
        return uf;
    if (exp_bit <= 1) // exp_bit == 0 or exp_bit == 1
        return (uf >> 31 << 30) + (uf >> 1) + (uf & (uf >> 1) & 1); // round to even
    return uf - 0x800000;
}
```

2.97

Following the bit-level floating-point coding rules, implement the function with

```
the following prototype:
/* Compute (float) i */
float_bits float_i2f(int i);
```

For argument i, this function computes the bit-level representation of (float) i. Test your function by evaluating it for all 2^{32} values of argument f and comparing the result to what would be obtained using your machine’s floating-point operations.

```
float_bits float_i2f(int x) {
    int sign_bit = x & 0x80000000;
    int exp_bit = 0x4f800000; // upper bound of exp
    int carry = 0;
    if (sign_bit) x = -x;
    if (x == 0) return 0;
    while (x > 0)
    {
        exp_bit -= 0x800000; // 2^23
        x <<= 1;
    }
    carry = ((x & 0x80) && (x & 0x17f));// round to even
    return sign_bit + exp_bit + (x >> 8) + carry;
}
```

3 Machine-Level Representation of Programs

3.58

```
long decode2(long x, long y, long z)
{
    y -= z;
    x *= y;
    long rax = y;
    rax = rax << 63 >> 63;
    rax ^= x;
    return rax;
}
```

3.59

3.60

(A) x in $\%rdi$, n in $\%esi$, $result$ in $\%rax$, $mask$ in $\%rdx$.

(B) $result = 0$, $mask = 1$

(C) $mask \neq 0$

(D) $mask = mask << n$

(E) $result| = x\&mask$

(F)
```
long loop(long x, int n)
{
    long result = 0;

    long mask;
    for(mask = 1; mask != 0; mask = mask << n) {
        result |= x & mask;
    }
    return result;
}
```

3.61

```c
long cread_alt(long *xp) {
    long t = 0;
    return *(xp ? xp : &t);
}
```

3.62

3.63

3.64

$R = 7$, $S = 5$, $T = 13$

3.65

3.66

3.67

3.68

$A = 9$, $B = 5$

3.69

$CNT = 7$

```c
typedef struct {
    int first;
    a_struct a[CNT];
    int last;
} b_struct;

typedef struct {
    long idx;
    long x[4];
} a_struct;
```

4 Processor Architecture

4.43

(A) In light of analysis done in Problem 4.6, does this code sequence correctly describe the behavior of the instruction pushl %esp? Explain.

(a) This code pushes %esp - 4 into stack, while ”pushl %esp” pushes %esp into stack.

(b) This code would change conditional code.

(B) How could you rewrite the code sequence so that it correctly describes both the cases where REG is %esp as well as any other register?

```
movl   REG,    -4(%esp)
leal   -4(%esp),   %esp
```

4.44

(A) In light of analysis done in Problem 4.7, does this code sequence correctly describe the behavior of the instruction popl %esp? Explain.

(a) This code changes %esp to (%esp) + 4, while ”popl %esp” changes %esp to (%esp)

(b) This code would change conditional code.

(B) How could you rewrite the code sequence so that it correctly describes both the cases where REG is %esp as well as any other register?

```
leal   4(%esp),    %esp
movl   -4(%esp),   REG
```

4.45

(A) Write and test a C version that references the array elements with pointers, rather than using array indexing.

```
void bubble_a(int *data, int count) {
    int i, last;
    for (last = count-1; last > 0; last--) {
        int *p_last = data + last;
        for (int *p1 = data; p1 != p_last; ++p1) {
            p2 = p1 + 1;                        /* compare two adjacent pointers */
            if (*p2 < *p1) {
                int temp = *p2; *p2 = *p1; *p1 = temp;
            }
        }
    }
}

/*
    Test Code Part
*/
int a[] = {7, 6, 8, 3, 0};

int main() {
    int i;
    bubble_a(a, 5);
    for(i = 0; i < 4; i++)
        if(a[i] > a[i+1])
            return 1;
    return 0;
}
```

(B) Write and test a Y86 program consisting of the function and test code. You may find it useful to pattern your implementation after IA32 code generated by compiling your C code. Although pointer comparisons are normally done using unsigned arithmetic, you can use signed arithmetic for this exercise.

```
    .pos 0
init:
    irmovl Stack, %esp
    irmovl Stack, %ebp
    call Main
    halt

bubble_a:
    pushl   %edi
    pushl   %esi
    pushl   %ebx
    mrmovl  20(%esp), %edi
```

```
    irmovl  $1, %eax
    rrmovl  %edi, %esi
    subl    %eax, %esi
    mrmovl  16(%esp), %eax
    andl    %esi, %esi
    jle L1
    rrmovl  %eax, %edi
    rrmovl  %esi, %ebx
    addl    %ebx, %ebx
    addl    %ebx, %ebx
    addl    %eax, %ebx
    jmp L3
L7:
    mrmovl  4(%eax), %edx
    mrmovl  (%eax), %ecx
    pushl   %edx
    subl    %ecx, %edx
    popl    %edx
    jge L4
    rmmovl  %ecx, 4(%eax)
    rmmovl  %edx, (%eax)
L4:
    pushl   %ebx
    irmovl  $4, %ebx
    addl    %ebx, %eax
    popl    %ebx
    pushl   %eax
    subl    %ebx, %eax
    popl    %eax
    jne L7
L6:
    pushl   %eax
    irmovl  $4, %eax
    subl    %eax, %ebx
    irmovl  $1, %eax
    subl    %eax, %esi
    popl    %eax
    je  L1
L3:
    andl    %esi, %esi
    jle L6
    rrmovl  %edi, %eax
    jmp L7
L1:
    popl    %ebx
    popl    %esi
```

```
    popl    %edi
    ret

Main:
    irmovl  $8, %eax
    subl    %eax, %esp
    irmovl  $5, %eax
    rmmovl  %eax, 4(%esp)
    irmovl  $a, %eax
    rmmovl  %eax, (%esp)
    call    bubble_a
    irmovl  $8, %eax
    addl    %eax, %esp
    xorl    %eax, %eax
    ret

    .align 4
a:
    .long   7
    .long   6
    .long   8
    .long   3
    .long   0

    .pos 0x1000
Stack:
```

This is the output in terminal:

```
Stopped in 235 steps at PC = 0x11.  Status 'HLT', CC Z=1 S=0 O=0

Changes to registers:
%ecx:   0x00000000  0x00000001
%esp:   0x00000000  0x00001000
%ebp:   0x00000000  0x00001000

Changes to memory:
0x00dc: 0x00000007  0x00000000
0x00e0: 0x00000006  0x00000003
0x00e4: 0x00000008  0x00000006
0x00e8: 0x00000003  0x00000007
0x00ec: 0x00000000  0x00000008
0x0fe0: 0x00000000  0x000000e0
0x0ff0: 0x00000000  0x000000d1
0x0ff4: 0x00000000  0x000000dc
0x0ff8: 0x00000000  0x00000005
0x0ffc: 0x00000000  0x00000011
```

Look at the line 9 to line 13, the array was sorted.

4.46

```
    .pos 0
init:
    irmovl Stack, %esp
    irmovl Stack, %ebp
    call Main
    halt

bubble_a:
    pushl   %edi
    pushl   %esi
    pushl   %ebx
    mrmovl  20(%esp), %edi
    irmovl  $1, %eax
    rrmovl  %edi, %esi
    subl    %eax, %esi
    mrmovl  16(%esp), %eax
    andl    %esi, %esi
    jle L1
    rrmovl  %eax, %edi
    rrmovl  %esi, %ebx
    addl    %ebx, %ebx
    addl    %ebx, %ebx
    addl    %eax, %ebx
    jmp L3
L7:
    mrmovl  4(%eax), %edx
    mrmovl  (%eax), %ecx
    pushl   %ebx
    pushl   %edx
    subl    %ecx, %edx
    popl    %edx
    cmovl   %ebx, %edx
    rmmovl  %edx, 4(%eax)
    rmmovl  %ecx, (%eax)
    irmovl  $4, %ebx
    addl    %ebx, %eax
    popl    %ebx
    pushl   %eax
    subl    %ebx, %eax
    popl    %eax
    jne L7
L6:
    pushl   %eax
    irmovl  $4, %eax
    subl    %eax, %ebx
```

```
    irmovl  $1, %eax
    subl    %eax, %esi
    popl    %eax
    je  L1
L3:
    andl    %esi, %esi
    jle L6
    rrmovl  %edi, %eax
    jmp L7
L1:
    popl    %ebx
    popl    %esi
    popl    %edi
    ret


Main:
    irmovl  $8, %eax
    subl    %eax, %esp
    irmovl  $5, %eax
    rmmovl  %eax, 4(%esp)
    irmovl  $a, %eax
    rmmovl  %eax, (%esp)
    call    bubble_a
    irmovl  $8, %eax
    addl    %eax, %esp
    xorl    %eax, %eax
    ret

    .align 4
a:
    .long   7
    .long   6
    .long   8
    .long   3
    .long   0

    .pos 0x1000
Stack:
```

4.47

See the table below for detail.

4.48

See the table below for detail.

Stage	Instruction	
	iaddl V, rB	*leave*
Fetch	$icode : ifun \leftarrow M_1[PC]$ $rA : rB \leftarrow M_1[PC + 1]$ $ValC \leftarrow M_4[PC + 2]$ $ValP \leftarrow PC + 6$	$icode : ifun \leftarrow M_1[PC]$ $ValP \leftarrow PC + 1$
Decode	 $ValB \leftarrow R[rB]$	$ValA \leftarrow R[\%ebp]$ $ValB \leftarrow R[\%ebp]$
Execute	$ValE \leftarrow ValB + VaclC$ $SetCC$	$ValE \leftarrow ValB + 4$
Memory		$ValM \leftarrow M_4[ValA]$
Write-back	$R[rB] \leftarrow ValE$	$R[\%esp] \leftarrow ValE$ $R[\%ebp] \leftarrow ValM$
PC-update	$PC \leftarrow ValP$	$PC \leftarrow ValP$

4.49

See *archlab partC* for detail.

4.50

```
#/* $begin seq-all-hcl */

####################################################################

#  HCL Description of Control for Single Cycle Y86 Processor SEQ   #

#  Copyright (C) Randal E. Bryant, David R. O'Hallaron, 2010       #

```

```
####################################################################
## Your task is to implement the iaddl and leave instructions
## The file contains a declaration of the icodes
## for iaddl (IIADDL) and leave (ILEAVE).
## Your job is to add the rest of the logic to make it work
####################################################################
#    C Include's.  Don't alter these                               #
####################################################################
quote '#include <stdio.h>'
quote '#include "isa.h"'
quote '#include "sim.h"'
quote 'int sim_main(int argc, char *argv[]);'
quote 'int gen_pc(){return 0;}'
quote 'int main(int argc, char *argv[])'
quote '  {plusmode=0;return sim_main(argc,argv);}'
####################################################################
#    Declarations.  Do not change/remove/delete any of these       #
####################################################################
```

```
##### Symbolic representation of Y86 Instruction Codes #############

intsig INOP     'I_NOP'

intsig IHALT    'I_HALT'

intsig IRRMOVL  'I_RRMOVL'

intsig IIRMOVL  'I_IRMOVL'

intsig IRMMOVL  'I_RMMOVL'

intsig IMRMOVL  'I_MRMOVL'

intsig IOPL 'I_ALU'

intsig IJXX 'I_JMP'

intsig ICALL    'I_CALL'

intsig IRET 'I_RET'

intsig IPUSHL   'I_PUSHL'

intsig IPOPL    'I_POPL'

# Instruction code for iaddl instruction

intsig IIADDL   'I_IADDL'

# Instruction code for leave instruction

intsig ILEAVE   'I_LEAVE'



##### Symbolic represenations of Y86 function codes                  #####

intsig FNONE    'F_NONE'        # Default function code



##### Symbolic representation of Y86 Registers referenced explicitly #####

intsig RESP     'REG_ESP'       # Stack Pointer

```

```
intsig REBP     'REG_EBP'       # Frame Pointer
intsig RNONE    'REG_NONE'      # Special value indicating "no register"

##### ALU Functions referenced explicitly                            #####
intsig ALUADD   'A_ADD'     # ALU should add its arguments

##### Possible instruction status values                             #####
intsig SAOK 'STAT_AOK'      # Normal execution
intsig SADR 'STAT_ADR'  # Invalid memory address
intsig SINS 'STAT_INS'  # Invalid instruction
intsig SHLT 'STAT_HLT'  # Halt instruction encountered

##### Signals that can be referenced by control logic ####################

##### Fetch stage inputs        #####
intsig pc 'pc'              # Program counter
##### Fetch stage computations      #####
intsig imem_icode 'imem_icode'      # icode field from instruction memory
intsig imem_ifun  'imem_ifun'       # ifun field from instruction memory
intsig icode      'icode'       # Instruction control code
intsig ifun   'ifun'        # Instruction function
intsig rA     'ra'          # rA field from instruction
intsig rB     'rb'          # rB field from instruction
```

```
intsig valC   'valc'        # Constant from instruction

intsig valP   'valp'        # Address of following instruction

boolsig imem_error 'imem_error'     # Error signal from instruction memory

boolsig instr_valid 'instr_valid'   # Is fetched instruction valid?



##### Decode stage computations     #####

intsig valA 'vala'          # Value from register A port

intsig valB 'valb'          # Value from register B port



##### Execute stage computations    #####

intsig valE 'vale'          # Value computed by ALU

boolsig Cnd 'cond'          # Branch test



##### Memory stage computations     #####

intsig valM 'valm'          # Value read from memory

boolsig dmem_error 'dmem_error'     # Error signal from data memory





####################################################################

#    Control Signal Definitions.                                   #

####################################################################



################ Fetch Stage     ###################################

```

```

# Determine instruction code
int icode = [
    imem_error: INOP;
    1: imem_icode;      # Default: get from instruction memory
];

# Determine instruction function
int ifun = [
    imem_error: FNONE;
    1: imem_ifun;       # Default: get from instruction memory
];

bool instr_valid = icode in
    { INOP, IHALT, IRRMOVL, IIRMOVL, IRMMOVL, IMRMOVL,
           IOPL, IJXX, ICALL, IRET, IPUSHL, IPOPL, IIADDL };

# Does fetched instruction require a regid byte?
bool need_regids =
    icode in { IRRMOVL, IOPL, IPUSHL, IPOPL,
             IIRMOVL, IRMMOVL, IMRMOVL, IIADDL };

# Does fetched instruction require a constant word?
```

```
bool need_valC =
    icode in { IIRMOVL, IRMMOVL, IMRMOVL, IJXX, ICALL, IIADDL };

################ Decode Stage    ###################################

## What register should be used as the A source?
int srcA = [
    icode in { IRRMOVL, IRMMOVL, IOPL, IPUSHL  } : rA;
    icode in { IPOPL, IRET } : RESP;
    1 : RNONE; # Don't need register
];

## What register should be used as the B source?
int srcB = [
    icode in { IOPL, IRMMOVL, IMRMOVL, IIADDL  } : rB;
    icode in { IPUSHL, IPOPL, ICALL, IRET } : RESP;
    1 : RNONE;  # Don't need register
];

## What register should be used as the E destination?
int dstE = [
    icode in { IRRMOVL } && Cnd : rB;
    icode in { IIRMOVL, IOPL, IIADDL } : rB;
```

```
    icode in { IPUSHL, IPOPL, ICALL, IRET } : RESP;
    1 : RNONE;  # Don't write any register
];

## What register should be used as the M destination?
int dstM = [
    icode in { IMRMOVL, IPOPL } : rA;
    1 : RNONE;  # Don't write any register
];

################ Execute Stage   ###################################

## Select input A to ALU
int aluA = [
    icode in { IRRMOVL, IOPL } : valA;
    icode in { IIRMOVL, IRMMOVL, IMRMOVL, IIADDL } : valC;
    icode in { ICALL, IPUSHL } : -4;
    icode in { IRET, IPOPL } : 4;
    # Other instructions don't need ALU
];

## Select input B to ALU
int aluB = [
```

```
    icode in { IRMMOVL, IMRMOVL, IOPL, ICALL,
              IPUSHL, IRET, IPOPL, IIADDL } : valB;
    icode in { IRRMOVL, IIRMOVL } : 0;
    # Other instructions don't need ALU
];

## Set the ALU function
int alufun = [
    icode == IOPL : ifun;
    1 : ALUADD;
];

## Should the condition codes be updated?
bool set_cc = icode in { IOPL, IIADDL };

################ Memory Stage    ###################################

## Set read control signal
bool mem_read = icode in { IMRMOVL, IPOPL, IRET };

## Set write control signal
bool mem_write = icode in { IRMMOVL, IPUSHL, ICALL };
```

```
## Select memory address

int mem_addr = [

    icode in { IRMMOVL, IPUSHL, ICALL, IMRMOVL } : valE;

    icode in { IPOPL, IRET } : valA;

    # Other instructions don't need address

];



## Select memory input data

int mem_data = [

    # Value from register

    icode in { IRMMOVL, IPUSHL } : valA;

    # Return PC

    icode == ICALL : valP;

    # Default: Don't write anything

];



## Determine instruction status

int Stat = [

    imem_error || dmem_error : SADR;

    !instr_valid: SINS;

    icode == IHALT : SHLT;

    1 : SAOK;

];
```

```
################ Program Counter Update ############################

## What address should instruction be fetched at

int new_pc = [
    # Call.  Use instruction constant
    icode == ICALL : valC;
    # Taken branch.  Use instruction constant
    icode == IJXX && Cnd : valC;
    # Completion of RET instruction.  Use value from stack
    icode == IRET : valM;
    # Default: Use incremented PC
    1 : valP;
];
#/* $end seq-all-hcl */
```

4.51

If instruction follows too closely after one that writes register, slow it down, Hold instruction in decode, Dynamically inject nop into execute stage.

Source Registers srcA and srcB of current instruction in decode stage.

Destination Registers dstE and dstM fields. Instructions in execute, memory, and write-back stages.

Special Case

1. Don't stall for register ID 15 (0xF).
2. Don't stall for failed conditional move (use e_dstE instead of E_dstE).

4.52

See *archlab partC* for detail.

4.53

See *archlab partC* for detail.

4.54

Because jump instruction doesn't use ALU. We could transport valC signal to Memory Stage by ALU :

$ValE := 0 + ValC$

Here is the difference between the original file and the modified file :

```


diff pipe-nt.hcl pipe-nt-origin.hcl

142,144c142
<
<   M_icode == IJXX && M_ifun != UNCOND && M_Cnd : M_valE;      # changed
<
---
>   M_icode == IJXX && !M_Cnd : M_valA;
188,191c186
<
<   f_icode in { ICALL } : f_valC;
<   f_icode in { IJXX } && f_ifun == UNCOND : f_valC;   # changed
<
---
>   f_icode in { IJXX, ICALL } : f_valC;
255c250
<   E_icode in { IIRMOVL, IRMMOVL, IMRMOVL,     IJXX } : E_valC;    # added IJXX
---
>   E_icode in { IIRMOVL, IRMMOVL, IMRMOVL } : E_valC;
265c260
<   E_icode in { IRRMOVL, IIRMOVL,          IJXX } : 0;     # added IJXX
---
>   E_icode in { IRRMOVL, IIRMOVL } : 0;
351c346
<   (E_icode == IJXX && E_ifun != UNCOND && e_Cnd) ||
---
>   (E_icode == IJXX && !e_Cnd) ||
362c357
<   (E_icode == IJXX && E_ifun != UNCOND && e_Cnd) ||
---
>   (E_icode == IJXX && !e_Cnd) ||
```

4.55

similar to 4.54

4.56

4.57

(A) the condition of *load forwarding* :

```
D_icode in { IRMMOVL, IPUSHL } && E_dstM == d_srcA
```

(B) Here is the difference between the original file and the modified file :

```
diff pipe-lf.hcl pipe-lf-origin.hcl

274,277d273
<
<   # added
<   M_icode in { IMRMOVL, IPOPL } && E_icode in { IRMMOVL, IPUSHL } && M_dstM == E_srcA : m_valM;
<
336,340c332
<   (
<       E_icode in { IMRMOVL, IPOPL} && E_dstM in { d_srcA, d_srcB }    # load/use
<       && !(D_icode in { IRMMOVL, IPUSHL } && E_dstM == d_srcA )   # load/forwarding
<   )
<   ||
---
>   0 ||
349,353c341
<   (
<                 E_icode in { IMRMOVL, IPOPL} && E_dstM in { d_srcA, d_srcB }    # load/use
<                 && !(D_icode in { IRMMOVL, IPUSHL } && E_dstM == d_srcA )       # load/forwarding
<          )
< ;
---
>   0;
371,375c359
<   (
<           E_icode in { IMRMOVL, IPOPL} && E_dstM in { d_srcA, d_srcB }    # load/use
<           && !(D_icode in { IRMMOVL, IPUSHL } && E_dstM == d_srcA )       # load/forwarding
<          )
< ;
```

```
---
>    0;
```

4.58

program performance can be greatly enhanced if the compiler is able to generate code using conditional data transfers rather than conditional control transfers.

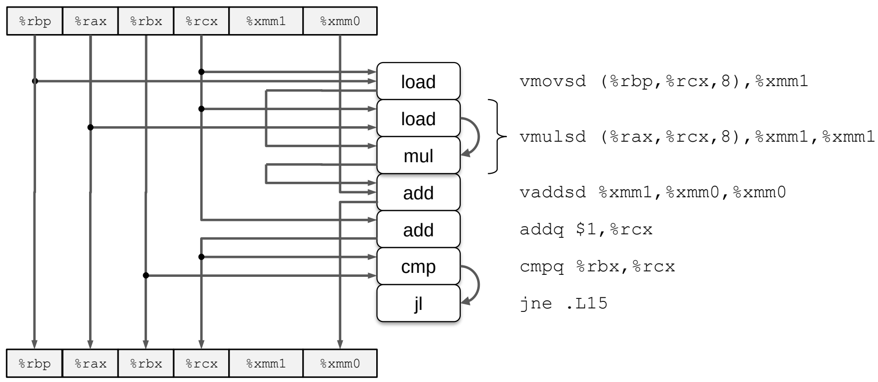
in page 530, CS:APP 2e:

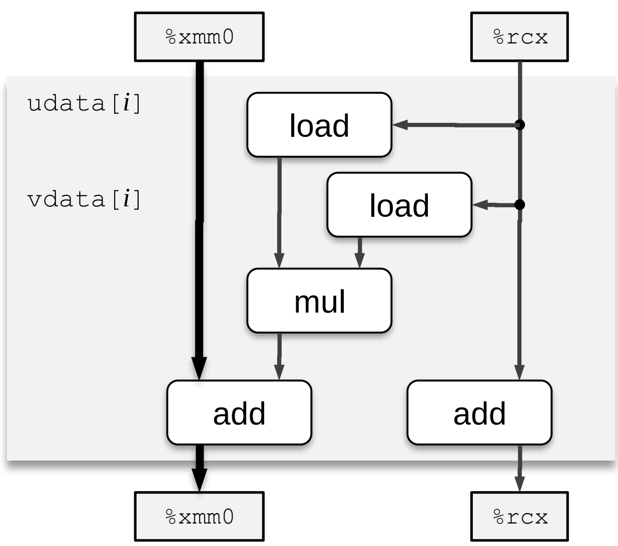
```
// Version 1.
for (i = 0; i < n; i++) {
    if (a[i] > b[i]) {
        int t = a[i];
        a[i] = b[i];
        b[i] = t;
    }
}
// Version 2.
for (i = 0; i < n; i++) {
    int min = a[i] < b[i] ? a[i] : b[i];
    int max = a[i] < b[i] ? b[i] : a[i];
    a[i] = min;
    b[i] = max;
}
```

5 Optimizing Program Performance

5.13

(A) In the figure below, *critical path* is emphasized by using bold.





(B) The addition operator of double, *latency* $= 3.0$

(C) The addition operator of int, $latency = 1.0$

(D) floating-point addition is in the *critical path* while floating-point mutiplication isn't.

With 2 functional units capable of performing floating-point mutiplication, the processor can potentially sustain a rate of 2 operations per cycle. The next mutiplication launches before the end of the previous mutiplication.

5.14

(A) With 4 functional units capable of performing integer addition, the processor can potentially sustain a rate of 4 operations per cycle. Unfortunately, the need to read elements from memory creates an additional throughput bound. The 2 load units limit the processor to reading at most 2 data values per clock cycle. So processor can't achieve a CPE less than 1.00.

(B) With 1 functional unit capable of performing floating-point addition, the processor can potentially sustain a rate of 1 operation per cycle.

the performance is restricted by the Capacity of floating-point addition.

6 The Memory Hierarchy

6.23

7 Linking

7.8

(A) (a) $REF(main.1) \rightarrow DEF(main.1)$

(b) $REF(main.2) \rightarrow DEF(main.2)$

(B) Here we have two weak definitions of x, so the symbol resolution in this case is UNKNOWN (Rule 3). But, notice that *double* is larger than *int*. It's usually behave as below:

(a) $REF(main.1) \rightarrow DEF(main.2)$

(b) $REF(main.2) \rightarrow DEF(main.2)$

(C) There are two strong definitions of x (Rule 1), so this is an *ERROR*.

7.9

When this program is compiled and executed on a Linux system, it prints the string "0x48" and terminates normally, even though p2 never initializes variable main.

Can you explain this?

Because of Rule 2, the strong symbol associated with the function main in foo6.o overrides the weak symbol associated with the variable main in bar6.o.

Thus, the reference to variable main in bar6 resolves to the value of symbol main, which in this case is the address of the first byte of function main. This byte contains the hex value 0x48, which is the binary encoding of *pushq %rbp*, the first instruction in procedure main

7.12

(A) *0xa*

(B) *0x22*

7.13

(A) *How many object files are contained in the versions of libc.a and libm.a on your system?*

libc.a has 1082 members and libm.a has 373 members.

(B) *Does* ***gcc -O2*** *produce different executable code than* ***gcc -O2 -g****?*

The code in the *.text* section is identical, whether a program is compiled using *-g* or not. The difference is that the *-O2 -g* object file contains debugging info in the *.debug* section, while the *-O2* version does not.

(C) *What shared libraries does the gcc driver on your system use?*

On our system, the gcc driver uses the standard C library (libc.so.6) and the dynamic linker (ld-linux.so.2).

8 Exceptional Control Flow

8.9

Process pair	Concurrent?
AB	N
AC	Y
AD	Y
BC	Y
BD	Y
CD	Y

8.10

In this chapter, we have introduced some functions with unusual call and return behaviors: setjmp, longjmp, execve, and fork. Match each function with one of the following behaviors:

(A) Called once, returns twice.

(B) Called once, never returns.

(C) Called once, returns one or more times.

function	behavior
setjmp	C
longjmp	B
execve	B
fork	A

8.11

4

8.12

8

8.13

any of the following sequences represents a possible output:

$x = 4$	$x = 4$	$x = 2$
$x = 3$	$x = 2$	$x = 4$
$x = 2$	$x = 3$	$x = 3$

8.14

3

8.15

5

8.16

$counter = 2$

8.17

there are only three possible outcomes (each column is an outcome):

Hello	Hello	Hello
1	1	0
Bye	0	1
0	Bye	Bye
2	2	2
Bye	Bye	Bye

8.18

$A\ C\ E$

8.19

2^n

8.20

```c
#include "csapp.h"

int main(int argc, char **argv) {
    if (!getenv("COLUMNS"))
    {
        setenv("COLUMNS", 80, 1);
        Execve("/bin/ls", argv, environ);
        unsetenv("COLUMNS");
    }
    else
    {
        Execve("/bin/ls", argv, environ);
    }
    exit(0);
}
```

8.21

1. *bac*

2. *abc*

8.22

Write your own version of the Unix system function

```c
int mysystem(char *command);
```

The mysystem function executes command by calling *"/bin/sh -c command"*, and then returns after command has completed. If command exits normally (by calling the exit function or executing a return statement), then mysystem returns the command exit status. For example, if command terminates by calling exit(8), then system returns the value 8. Otherwise, if command terminates abnormally, then mysystem returns the status returned by the shell.

```c
int mysystem(char *command) {
    pid_t pid;
    int status;

    if (!(pid = Fork())) { /* child */
        char *argv[4];
        argv[0] = "sh";
        argv[1] = "-c";
        argv[2] = command;
        argv[3] = NULL;
        execve("/bin/sh", argv, environ);
        exit(-1); /* control should never reach here */
    }

    /*
     * parent
     * In fact, we could use sigsuspend() instead of tight loop, which is better.
     */
    while (1) {
        if (waitpid(pid, &status, 0) > 0) {
            if (WIFEXITED(status))
                return WEXITSTATUS(status);
            else
                return status;
        }
        else if (errno != EINTR) /* restart waitpid if interrupted */
                return -1;
    }
}
```

8.25

```c
#include "csapp.h"
#include <unistd.h>

static sigjmp_buf env;

/* SIGALRM handler */
static void handler(int sig) {
    siglongjmp(env, 1);
    return;
}

char *tfgets(char *s, int size, FILE *stream) {
    static handler_t *old_handler;
    switch (sigsetjmp(env, 1)): {
        case 0: {
            old_handler = Signal(SIGALRM, handler);
            Alarm(5);
            Fgets(s, size, stream); /* return user input */

            /*
             * Signal(SIGALRM, SIG_IGN) is WRONG, the previous handler may not be SIG_IGN;
             */
            Signal(SIGALRM, old_handler); /* restore the old signal handler */
            return s;
        }
        case 1: {
            Signal(SIGALRM, old_handler); /* restore the old signal handler */
            return NULL; /* return NULL if fgets times out */
        }
    }
}

int main() {
    static char buf[1500012786];
    while (1) {
        if (tfgets(buf, sizeof(buf), stdin) != NULL)
            printf("read: %s\n", buf);
        else
            printf("timed out\n");
    }
    exit(0);
}
```

8.26

plz see *tshlab* for detail.

9 Virtual Memory

9.11

(A) Virtual address format

13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	0	1	1	1	1	1	0	0

(B) Address translation

Parameter	Value
VPN	0x9
TLB index	0x1
TLB tag	0x2
TLB hit?	N
Page fault?	N
PPN	0x17

(C) Physical address format

11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	1	1	1	1	1	1	0	0

(D) Physical memory reference

Parameter	Value
Byte offset	0
Cache index	0xf
Cache tag	0x17
Cache hit?	N
Cache byte returned	-

9.12

(A) Virtual address format

13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	1	1	0	1	0	1	0	0	1

(B) Address translation

Parameter	Value
VPN	0xe
TLB index	0x2
TLB tag	0x3
TLB hit?	N
Page fault?	N
PPN	0x11

(C) Physical address format

11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	1	1	0	1	0	0	1

(D) Physical memory reference

Parameter	Value
Byte offset	0
Cache index	0xa
Cache tag	0x11
Cache hit?	N
Cache byte returned	-

9.13

(A) Virtual address format

13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	1	0	0	0	0	0	0

(B) Address translation

Parameter	Value
VPN	0x1
TLB index	0x1
TLB tag	0x0
TLB hit?	N
Page fault?	Y
PPN	-

(C) N/A

(D) N/A

9.14

Given an input file hello.txt that consists of the string "Hello, world!\n", write a C program that uses *mmap* to change the contents of hello.txt to "Jello, world!\n".

```c
#include "csapp.h"

/*
 * mmapwrite - uses mmap to modify a disk file
 */
void mmapwrite(int fd, int len)
{
    char *bufp;

    /* bufp = Mmap(NULL, len, PROT_READ|PROT_WRITE, MAP_SHARED, fd, 0); */
    bufp = Mmap(NULL, len, PROT_READ|PROT_WRITE, MAP_PRIVATE, fd, 0);
    bufp[0] = 'J';
}

/* mmapwrite driver */
int main(int argc, char **argv)
{
    int fd;
    struct stat stat;

    /* check for required command line argument */
    if (argc != 2) {
        printf("usage: %s <filename>\n", argv[0]);
        exit(0);
    }

    /* open the input file and get its size */
    fd = Open(argv[1], O_RDWR, 0);
    fstat(fd, &stat);
    mmapwrite(fd, stat.st_size);
    exit(0);
}
```

9.15

Determine the block sizes and header values that would result from the following sequence of *malloc* requests. Assumptions:

(1) The allocator maintains double-word alignment, and uses an implicit free list with the block format from Figure 9.35.

(2) Block sizes are rounded up to the nearest multiple of 8 bytes.

Request	Block size	Block header
malloc(3)	8	0x9
malloc(11)	16	0x11
malloc(20)	24	0x19
malloc(21)	32	0x21

9.16

Determine the minimum block size for each of the following combinations of alignment requirements and block formats. Assumptions: Explicit free list, 4-byte *pred* and *succ* pointers in each free block, zero-sized payloads are not allowed, and headers and footers are stored in 4-byte words.

Alignment	Allocated block	Free block	Minimum block size
Single word	Header and footer	Header and footer	16
Single word	Header, but no footer	Header and footer	16
Double word	Header and footer	Header and footer	16
Double word	Header, but no footer	Header and footer	16

9.17

Develop a version of the allocator in Section 9.9.12 that performs a next-fit search instead of a first-fit search.

I defined a global roving pointer *(void *rover)* that points initially to the front of the list.

```c
/*
 * If NEXT_FIT defined use next fit search, else use first-fit search
 */
#define NEXT_FIT
```

```c
/* Global variables */
static char *heap_listp = 0;  /* Pointer to first block */
#ifdef NEXT_FIT
static char *rover;           /* Next fit rover */
#endif

/*
 * mm_init - Initialize the memory manager
 */
int mm_init(void)
{
    /* Create the initial empty heap */
    if ((heap_listp = mem_sbrk(4*WSIZE)) == (void *)-1) //line:vm:mm:begininit
        return -1;
    PUT(heap_listp, 0);                          /* Alignment padding */
    PUT(heap_listp + (1*WSIZE), PACK(DSIZE, 1)); /* Prologue header */
    PUT(heap_listp + (2*WSIZE), PACK(DSIZE, 1)); /* Prologue footer */
    PUT(heap_listp + (3*WSIZE), PACK(0, 1));     /* Epilogue header */
    heap_listp += (2*WSIZE);                     //line:vm:mm:endinit

#ifdef NEXT_FIT
    rover = heap_listp;
#endif

    /* Extend the empty heap with a free block of CHUNKSIZE bytes */
    if (extend_heap(CHUNKSIZE/WSIZE) == NULL)
        return -1;
    return 0;
}
```

```
/*
 * find_fit - Find a fit for a block with asize bytes
 */
static void *find_fit(size_t asize)
{
#ifdef NEXT_FIT
    /* Next fit search */
    char *oldrover = rover;

    /* Search from the rover to the end of list */
    for ( ; GET_SIZE(HDRP(rover)) > 0; rover = NEXT_BLKP(rover))
        if (!GET_ALLOC(HDRP(rover)) && (asize <= GET_SIZE(HDRP(rover))))
            return rover;

    /* search from start of list to old rover */
    for (rover = heap_listp; rover < oldrover; rover = NEXT_BLKP(rover))
        if (!GET_ALLOC(HDRP(rover)) && (asize <= GET_SIZE(HDRP(rover))))
            return rover;

    return NULL;  /* no fit found */
#else
    /* First-fit search */
    void *bp;

    for (bp = heap_listp; GET_SIZE(HDRP(bp)) > 0; bp = NEXT_BLKP(bp)) {
        if (!GET_ALLOC(HDRP(bp)) && (asize <= GET_SIZE(HDRP(bp)))) {
            return bp;
        }
    }
    return NULL; /* No fit */
#endif
}
```

After coalescing a block, Make sure the rover isn't pointing into the free block that we just coalesced.

```
static void *coalesce(void *bp)
{
    int prev_alloc = GET_ALLOC(FTRP(PREV_BLKP(bp)));
    int next_alloc = GET_ALLOC(HDRP(NEXT_BLKP(bp)));
    size_t size = GET_SIZE(HDRP(bp));

    if (prev_alloc && next_alloc) { /* Case 1 */
        return bp;
    }

    else if (prev_alloc && !next_alloc) { /* Case 2 */
        size += GET_SIZE(HDRP(NEXT_BLKP(bp)));
        PUT(HDRP(bp), PACK(size, 0));
        PUT(FTRP(bp), PACK(size,0));
    }

    else if (!prev_alloc && next_alloc) { /* Case 3 */
        size += GET_SIZE(HDRP(PREV_BLKP(bp)));
        PUT(FTRP(bp), PACK(size, 0));
        PUT(HDRP(PREV_BLKP(bp)), PACK(size, 0));
        bp = PREV_BLKP(bp);
    }

    else { /* Case 4 */
        size += GET_SIZE(HDRP(PREV_BLKP(bp))) +
        GET_SIZE(FTRP(NEXT_BLKP(bp)));
        PUT(HDRP(PREV_BLKP(bp)), PACK(size, 0));
        PUT(FTRP(NEXT_BLKP(bp)), PACK(size, 0));
        bp = PREV_BLKP(bp);
    }

#ifdef NEXT_FIT
    /* Make sure the rover isn't pointing into the free block */
    /* that we just coalesced */
    if ((rover > (char *)bp) && (rover < NEXT_BLKP(bp)))
        rover = bp;
#endif

    return bp;
}
```

See *code/vm/malloc/mm.c* for detail.

9.18

The allocator in Section 9.9.12 requires both a header and a footer for each block in order to perform constant-time coalescing. Modify the allocator so that free blocks require a header and footer, but allocated blocks require only a header.

```
#define ALLOC_NO_FOOTER
```

```
/* Read the size and allocated fields from address p */
#define GET_SIZE(p)  (GET(p) & ~0x7)                    //line:vm:mm:getsize
#define GET_ALLOC(p) (GET(p) & 0x1)                     //line:vm:mm:getalloc

#ifdef ALLOC_NO_FOOTER
#define GET_PREV_ALLOC(p)  (GET(p) & 0x2)
#endif
```

```
/*
 * mm_init - Initialize the memory manager
 */
int mm_init(void) {
    /* Create the initial empty heap */
    if ((heap_listp = mem_sbrk(4*WSIZE)) == (void *)-1) //line:vm:mm:begininit
        return -1;
    PUT(heap_listp, 0);                          /* Alignment padding */
    PUT(heap_listp + (1*WSIZE), PACK(DSIZE, 1)); /* Prologue header */

#ifdef ALLOC_NO_FOOTER
    PUT(heap_listp + (2*WSIZE), PACK(DSIZE, 0x3)); /* Prologue footer */
#else
    PUT(heap_listp + (2*WSIZE), PACK(DSIZE, 1)); /* Prologue footer */
#endif

    PUT(heap_listp + (3*WSIZE), PACK(0, 1));     /* Epilogue header */
    heap_listp += (2*WSIZE);                     //line:vm:mm:endinit

    /* Extend the empty heap with a free block of CHUNKSIZE bytes */
    if (extend_heap(CHUNKSIZE/WSIZE) == NULL) return -1; else return 0;
}
```

```
/*
 * place - Place block of asize bytes at start of free block bp
 *         and split if remainder would be at least minimum block size
 */
static void place(void *bp, size_t asize)
{
#ifdef ALLOC_NO_FOOTER
    size_t csize = GET_SIZE(HDRP(bp));

    if ((csize - asize) >= (2*DSIZE)) {
        PUT(HDRP(bp), PACK(asize, 1));
        bp = NEXT_BLKP(bp);
        PUT(HDRP(bp), PACK(csize-asize, 0x2));
        PUT(FTRP(bp), PACK(csize-asize, 0x2));
    }
    else {
        PUT(HDRP(bp), PACK(csize, 1));
        bp = NEXT_BLKP(bp);
        GET(HDRP(bp)) |= 0x2;
        GET(FTRP(bp)) |= 0x2;
    }
#else
    size_t csize = GET_SIZE(HDRP(bp));

    if ((csize - asize) >= (2*DSIZE)) {
        PUT(HDRP(bp), PACK(asize, 1));
        PUT(FTRP(bp), PACK(asize, 1));
        bp = NEXT_BLKP(bp);
        PUT(HDRP(bp), PACK(csize-asize, 0));
        PUT(FTRP(bp), PACK(csize-asize, 0));
    }
    else {
        PUT(HDRP(bp), PACK(csize, 1));
        PUT(FTRP(bp), PACK(csize, 1));
    }
#endif
}
```

9.19

You are given three groups of statements relating to memory management and garbage collection below. In each group, only one statement is true. Your task is to indicate which statement is true.

(1) (a) In a buddy system, up to 50% of the space can be wasted due to internal fragmentation.

(b) The first-fit memory allocation algorithm is slower than the best-fit algorithm (on average).

(c) Deallocation using boundary tags is fast only when the list of free blocks is ordered according to increasing memory addresses.

(d) The buddy system suffers from internal fragmentation, but not from external fragmentation.

(2) (a) Using the first-fit algorithm on a free list that is ordered according to decreasing block sizes results in low performance for allocations, but avoids external fragmentation.

(b) For the best-fit method, the list of free blocks should be ordered according to increasing memory addresses.

(c) The best-fit method chooses the largest free block into which the requested segment fits.

(d) Using the first-fit algorithm on a free list that is ordered according to increasing block sizes is equivalent to using the best-fit algorithm.

(3) Mark & sweep garbage collectors are called conservative if:

(a) They coalesce freed memory only when a memory request cannot be satisfied.

(b) They treat everything that looks like a pointer as a pointer.

(c) They perform garbage collection only when they run out of memory.

(d) They do not free memory blocks forming a cyclic list.

(1) a

(2) d

(3) b

9.20

Write your own version of malloc and free, and compare its running time and space utilization to the version of malloc provided in the standard C library.

See *Malloc Lab* for detail.

10 System-Level I/O

10.6

What is the output of the following program?

```
#include "csapp.h"

int main()
{
    int fd1, fd2;
    fd1 = Open("foo.txt", O_RDONLY, 0);
    fd2 = Open("bar.txt", O_RDONLY, 0);
    Close(fd2);
    fd2 = Open("baz.txt", O_RDONLY, 0);
    printf("fd2 = %d\n", fd2);
    exit(0);
}
```

On entry, descriptors 0-2 are already open. The open function always returns the lowest possible descriptor, so the first two calls to open return descriptors 3 and 4. The call to the close function frees up descriptor 4, so the final call to open returns descriptor 4, and thus the output of the program is $fd2 = 4$.

10.7

Modify the cpfile program in Figure 10.5 so that it uses the Rio functions to copy standard input to standard output, MAXBUF bytes at a time.

```
#include "csapp.h"

int main(int argc, char **argv) {
    int n;
    char buf[MAXBUF];

    while((n = Rio_readn(STDIN_FILENO, buf, MAXBUF)) != 0)
    Rio_writen(STDOUT_FILENO, buf, n);
    exit(0);
}
```

10.8

Write a version of the statcheck program in Figure 10.10, called fstatcheck, that takes a descriptor number on the command line rather than a file name.

Just invoke fstat instead of stat, Here is my solution:

```
#include "csapp.h"

int main (int argc, char **argv)
{
    struct stat stat;
    char *type, *readok;
    int size;

    if (argc != 2) {
        fprintf(stderr, "usage: %s <fd>\n", argv[0]);
        exit(0);
    }
    Fstat(atoi(argv[1]), &stat);
    if (S_ISREG(stat.st_mode)) /* Determine file type */
        type = "regular";
    else if (S_ISDIR(stat.st_mode))
        type = "directory";
    else if (S_ISCHR(stat.st_mode))
        type = "character device";
    else
        type = "other";

    if ((stat.st_mode & S_IRUSR)) /* Check read access */
        readok = "yes";
    else
        readok = "no";

    size = stat.st_size; /* check size */

    printf("type: %s, read: %s, size=%d\n", type, readok, size);

    exit(0);
}
```

10.9

Consider the following invocation of the fstatcheck program from Problem 10.8:

```
unix> fstatcheck 3 < foo.txt
```

You might expect that this invocation of fstatcheck would fetch and display metadata for file foo.txt. However, when we run it on our system, it fails with a ***"bad file descriptor."*** *Given this behavior, fill in the pseudo-code that the shell must be executing between the fork and execve calls:*

```
if (Fork() == 0) { /* Child */
    /* What code is the shell executing right here? */
    Execve("fstatcheck", argv, envp);
}
```

Before the call to execve, the child process opens foo.txt as descriptor 3, redirects stdin to foo.txt, and then (here is the kicker) closes descriptor 3:

```
if (Fork() == 0) { /* child */
    fd = Open("foo.txt", O_RDONLY, 0); /* fd == 3 */
    Dup2(fd, STDIN_FILENO);
    Close(fd);
    Execve("fstatcheck", argv, envp);
}
```

When fstatcheck begins running in the child, there are exactly three open files, corresponding to descriptors 0, 1, and 2, with descriptor 1 redirected to foo.txt.

10.10

Modify the cpfile program in Figure 10.5 so that it takes an optional command line argument infile. If infile is given, then copy infile to standard output; otherwise, copy standard input to standard output as before. The twist is that your solution must use the original copy loop (lines 9–11) for both cases. You are only allowed to insert code, and you are not allowed to change any of the existing code.

```
1  #include "csapp.h"
2
3  int main(int argc, char **argv) {
4      int n;
5      rio_t rio;
6      static char buf[MAXLINE];
7
8      if ((argc < 1) || (argc > 2) ) {
9          fprintf(stderr, "usage: %s <infile>\n", argv[0]);
10         exit(1);
11     }
12
13     if (argc == 2) {
14         int fd;
15         if ((fd = Open(argv[1], O_RDONLY, 0)) < 0) {
16             fprintf(stderr, "Couldn't read %s\n", argv[1]);
17             exit(1);
18         }
19         Dup2(fd, STDIN_FILENO);
20         Close(fd);
21     }
22
23     Rio_readinitb(&rio, STDIN_FILENO);
24     while((n = Rio_readlineb(&rio, buf, MAXLINE)) != 0)
25         Rio_writen(STDOUT_FILENO, buf, n);
26     exit(0);
27 }
```

11 Network Programming

11.6

(A) It's the same as our *echo server*.

(B)

```
$ ./tiny 8000
Accepted connection from (localhost, 40734)
GET / HTTP/1.1
Host: localhost:8000
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:50.0) Gecko/20100101 Firefox/50.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Upgrade-Insecure-Requests: 1

Response headers:
HTTP/1.0 200 OK
Server: Tiny Web Server
Connection: close
Content-length: 120
Content-type: text/html

Accepted connection from (localhost, 40736)
GET /godzilla.gif HTTP/1.1
Host: localhost:8000
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:50.0) Gecko/20100101 Firefox/50.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://localhost:8000/
Connection: keep-alive

Response headers:
HTTP/1.0 200 OK
Server: Tiny Web Server
Connection: close
Content-length: 12155
Content-type: image/gif
```

(C) HTTP/1.1

(D) Host: 请求资源所在的域名IP以及端口号。
User-Agent: 代理（浏览器）。
Accept: 指定接受的介质类型。本例中为首选网页文件和图片文件。
Referer: 来源网页信息。本例中图片文件来源为home.html。
Accept-Encoding: 指定接受的编码方法，通常为压缩方法。本例中为首选gzip压缩格式。
Accept-Language: 指定接受的语言。本例中为首选中文。
Connection: 指定完成本次连接后是否断开连接。本例中为保持连接。

11.8

Modify *Tiny* so that it reaps *CGI* children inside a *SIGCHLD* handler instead of explicitly waiting for them to terminate.

Install a *SIGCHLD* handler in the main routine and delete the call to *wait* in *serve dynamic*.

```c
void sigchld_handler(int sig)
{
    pid_t pid;

    while ((pid = waitpid(-1, NULL, 0)) > 0)
        printf("Handler reaped child %d\n", (int)pid);

    if (errno != ECHILD)
        unix_error("waitpid error");

    return;
}
```

12 Concurrent Programming

12.16

```c
#include "csapp.h"

void *thread(void *vargp);

int main(int argc, char **argv)
{
    pthread_t *tid;
    int i, n;

    if (argc != 2)
    {
        fprintf(stderr, "usage: %s <nthreads>\n", argv[0]);
        exit(0);
    }
    n = atoi(argv[1]);
    tid = Malloc(n * sizeof(pthread_t));

    for (i = 0; i < n; i++)
        Pthread_create(&tid[i], NULL, thread, NULL);

    for (i = 0; i < n; i++)
        Pthread_join(tid[i], NULL);
        exit(0);
}

/* thread routine */
void *thread(void *vargp)
{
    printf("Hello, world!\n");
    return NULL;
}
```

12.17

This is the student' s first introduction to the many synchronization problems that can arise in threaded programs.

(A) The problem is that the main thread calls exit without waiting for the peer thread to terminate. The exit call terminates the entire process, including any threads that happen to be running. So the peer thread is being killed before it has a chance to print its output string.

(B) We can fix the bug by replacing the exit function with either pthread exit, which waits for outstanding threads to terminate before it terminates the process, or pthread join, which explicitly reaps the peer thread.

12.18

(A) unsafe

(B) safe

(C) unsafe

12.22

Test your understanding of the select function by modifying the server in Figure 12.6 so that it echoes at most one text line per iteration of the main server loop.

```c
#include "csapp.h"
void echo(int connfd);
void command(void);

int main(int argc, char **argv)
{
    int listenfd, connfd, port;
    socklen_t clientlen = sizeof(struct sockaddr_in);
    struct sockaddr_in clientaddr;
    fd_set read_set, ready_set;

    if (argc != 2) {
        fprintf(stderr, "usage: %s <port>\n", argv[0]);
        exit(0);
    }
    port = atoi(argv[1]);
    listenfd = Open_listenfd(port);

    FD_ZERO(&read_set); /* Clear read set */
    FD_SET(STDIN_FILENO, &read_set); /* Add stdin to read set */
    FD_SET(listenfd, &read_set); /* Add listenfd to read set */


    /*
     * * rio init
     */
    rio_t rio;
    Rio_readinitb(&rio, connfd);

```

```c

    while (1) {
        ready_set = read_set;
        Select(listenfd+1, &ready_set, NULL, NULL, NULL);
        if (FD_ISSET(STDIN_FILENO, &ready_set))
        command(); /* Read command line from stdin */
        if (FD_ISSET(listenfd, &ready_set)) {
            connfd = Accept(listenfd, (SA *)&clientaddr, &clientlen);

            /*
             * * echoes at most one text line per iteration.
             */
            size_t n;
            static char buf[MAXLINE];

            if ((n = Rio_readlineb(&rio, buf, MAXLINE)) != 0) {
                printf("server received %d bytes\n", n);
                Rio_writen(connfd, buf, n);
            }

            Close(connfd);
        }
    }
}

void command(void) {
    char buf[MAXLINE];
    if (!Fgets(buf, MAXLINE, stdin))
    exit(0); /* EOF */
    printf("%s", buf); /* Process the input command */
}
```

12.23

12.24

The functions in the Rio I/O package (Section 10.5) are thread-safe. Are they reentrant as well?

(A) Each of the buffered Rio functions is passed a pointer to a buffer, and then operates exclusively on this buffer and local stack variables. If they are invoked properly by the calling function, such that none of the buffers are shared, then they are reentrant.

(B) We can fix the bug by replacing the exit function with either pthread exit, which waits for outstanding threads to terminate before it terminates the process, or pthread join, which explicitly reaps the peer thread.

12.25

In the prethreaded concurrent echo server in Figure 12.28, each thread calls the *echo_cnt* function (Figure 12.29). Is *echo_cnt* thread-safe? Is it reentrant? Why or why not?

The *echo_cnt* function is thread-safe because (a) It protects accesses to the shared global *byte_cnt* with a mutex, and (b) All of the functions that it calls, such as *rio_readlineb* and *rio_writen*, are thread-safe. However, because of the shared variable, *echo_cnt* is not reentrant.

12.26

```
/*
 * gethostbyname_ts - A thread-safe wrapper for gethostbyname
 */
#include "csapp.h"

static sem_t mutex; /* protects calls to gethostbyname */

static void init_gethostbyname_ts(void) {
    Sem_init(&mutex, 0, 1);
}

struct hostent *gethostbyname_ts(char *hostname) {
    struct hostent *sharedp, *unsharedp;

    unsharedp = Malloc(sizeof(struct hostent));
    P(&mutex);
    sharedp = gethostbyname(hostname);
    *unsharedp = *sharedp; /* copy shared struct to private struct */
    V(&mutex);
    return unsharedp;
}

int main(int argc, char **argv) {
    char **pp; struct in_addr addr; struct hostent *hostp;

    if (argc != 2) {
        fprintf(stderr, "usage: %s <hostname>\n", argv[0]);
        exit(0);
    }

    init_gethostbyname_ts();
    hostp = gethostbyname_ts(argv[1]);
    if (hostp) {
        printf("official hostname: %s\n", hostp->h_name);
        for (pp = hostp->h_aliases; *pp != NULL; pp++)
            printf("alias: %s\n", *pp);
        for (pp = hostp->h_addr_list; *pp != NULL; pp++) {
            addr.s_addr = *((unsigned int *)*pp);
            printf("address: %s\n", inet_ntoa(addr));
        }
    }
    else
        printf("host %s not found\n", argv[1]);
}
```