# Number Conversion

Fill in the table below

| Binary | Octal | Decimal | Hexadecimal |
|---|---|---|---|
| 1001101111000 | | | |
| | 2017 | | |
| | | 1919 | |
| | | | 0xDEAD |
| | 567 | | |

# Binary Operations

1) Given 8-bits wide A and B with hexadecimal expression 0x5A and 0x9C respectively. Calculate the values of the following expressions

a) A & B

b) A && B

c) A | B

d) A || B

e) ~A ^ ~B

f) (A || B) ^(A &B)

2) Fill in the table below with the results of shift operation given below (Assume X is 8-bit wide) . Please answer in hexadecimal.

| X | X << 2 | X>>3(Logical) | X>>3(Arithmetic) |
|---|---|---|---|
| 0x4C | | | |
| 0xEA | | | |

3) Design a C expression, which generates a word (32-bit) consisting of the lower 24 bits of x and the remaining 8 bits of y.

For example, x = 0x89ABCDEF and y = 0x76543210, it will generate 0x76ABCDEF

# Homework 2

## I. Integer Size

You are given the task of writing a procedure `int_size_is_32()` that yields 1 when run on a machine for which an int is 32 bits, and yields 0 otherwise. You are not allowed to use the `sizeof` operator. Here is a first attempt:

```
int bad_int_size_is_32() {

    int set_msb = 1 << 31;
    int beyond_msb = 1 << 32;

    return set_msb && !beyond_msb;
}
```

When compiled and run on a 32-bit SUN SPARC, however, this procedure returns 0. The following compiler message gives us an indication of the problem:

```
warning: left shift count >= width of type
```

1. In what way does our code fail to comply with the C standard? (Hint: you can search online for the C language standard on shift operations)

2. Modify the code to run properly on any machine for which data type int is at least 32 bits.

```
int int_size_is_32() {
    // fill in C codes ...

}
```

3. Modify the code to run properly on any machine for which data type int is at least 16 bits.

```
int int_size_is_32_16bit() {
    // fill in C codes ...

}
```

## II. Byte Order

1. Consider the following function [`sizeof(int) == 4`]

```
typedef unsigned char byte;
void show_bytes(byte *start, int len) {
     int i;
     for(i = 0; i < len; i++)
        printf("%.2x", start[i]);
}
unsigned int val = 0x1234abcd;
byte *valp = (byte *) &val;
```

(1) What is the value of valp[4] (if consider it as an array of 4 elements of "`byte`")?
Little-endian: {valp[0], valp[1], valp[2], valp[3]} = {(a), (b), (c), (d)}.
Big-endian: {valp[0], valp[1], valp[2], valp[3]} = {(e), (f), (g), (h)}

(2) What is the output of the following call to show_bytes on big-endian and little-endian machines respectively? (You can have a try on your machine!)

| | Little-endian | Big-endian |
|---|---|---|
| show_bytes(valp, 1) | | |
| show_bytes(valp, 2) | | |
| show_bytes(valp, 4) | | |

2. Write a procedure `is_little_endian` that will return 1 when compiled and run on a little-endian machine, and will return 0 when compiled and run on a big-endian machine. This program should run on any machine, regardless of its word.

```
int is_little_endian(void) {
   // fill in C codes ...

}
```

3. Write a procedure `to_little_endian` that will transform an 32 bit unsigned. integer number into little endian format and return the pointer to the little endian integer. This program should run on both little endian and big endian machine. (Hint: you can use `is_little_endian` function written before)

```
byte *to_little_endian(unsigned int num) {
     // fill in C codes ...

}
```

# Homework 3

1. In C language, if an evaluation expression contains both unsigned and signed values, then signed values will be implicitly casted into unsigned ones before evaluation. Please fill the following table with "<", ">" or "=". (Assume **int value is encoded using 16 bits**)

| Constant A | Constant B | A ? B |
|---|---|---|
| -2U | -1U | |
| -1 | 1 | |
| -1 | 100U | |
| -1 | 65535U | |
| -32767 | 32768U | |

2. There is a illustration of code vulnerability similar to that found in FreeBSD's implementation of getpeername(). Find one bug in the following codes and try to fix it.

```
/* Copy n bytes from src to dest */
/* Note: size_t means unsigned int */
void *memcpy(void *dest, void *src, size_t n);

/* Kernel memory region holding user-accessible data */
#define KSIZE 1024
char kbuf[KSIZE];

/* Copy at most maxlen bytes from kernel region to user buffer */
/* Must not copy more than maxlen bytes */
 int copy_from_kernel(void *user_dest, int maxlen) {
    /* Byte count len is minimum of buffer size and maxlen */
    int len = KSIZE < maxlen ? KSIZE : maxlen;
    memcpy(user_dest, kbuf, len);
    return len;
}
```

3. Assume x and y are both 4 bit signed integers. Fill the following table. Truncate all the results to 4 bits with 2's complement and write their value in decimal.

| | x+y | x-y | x*y | -y |
|---|---|---|---|---|
| x=4, y=7 | | | | |
| x=-6, y=-8 | | | | |
| x=5, y=-1 | | | | |
| x=-3, y=6 | | | | |

# Homework 4

## JUMP INSTRUCTIONS

In the following excerpts from a disassembled binary, some of the information has been replaced by `X`s. Answer the following questions about these instructions.

A. What is the target of the `je` instruction below? (You don't need to know anything about the `callq` instruction here.)

```
4003fa: 74 02                   je    XXXXXX
4003fc: ff d0                   callq *%rax
```

B. What is the target of the `je` instruction below?

```
40042f: 74 f4                   je    XXXXXX
400431: 5d                      pop   %rbp
```

C. What is the address of the `ja` and `pop` instructions?

```
XXXXXX: 77 02                   ja    400547
XXXXXX: 5d                      pop   %rbp
```

D. In the code that follows, the jump target is encoded in PC-relative form as a 4-byte, two's-complement number. The bytes are listed from least significant to most, reflecting the little-endian byte ordering of x86-64. What is the address of the jump target?

```
4005e8: e9 73 ff ff ff          jmp XXXXXX
4005ed: 90                      nop
```

## CONDITIONAL MOVES

In the following C function, we have left the definition of operation `OP` incomplete:

```
#define OP ____ /* Unknown operator */

long arith(long x) {
    return x OP 8;
}
```

When compiled, GCC generates the following assembly code:

*long arith(long x)*

*x in %rdi*

```
arith:
  leaq      7(%rdi), %rax
  testq     %rdi, %rdi
  cmovns    %rdi, %rax
  sarq      $3, %rax
  ret
```

What operation is OP (only one operation) and explain how it works.

## LOOPS

Executing a `continue` statement in C causes the program to jump to the end of the current loop iteration. The stated rule for translating a `for` loop into a `while` loop needs some refinement when dealing with `continue` statements. For example, consider the following code:

```
/* Example of for loop using a continue statement */
/* Sum even numbers between 0 and 9 */
long sum = 0;
long i;
for (i = 0; i < 10; i++) {
    if (i & 1)
        continue;
    sum += i;
}
```

A. What would we get if we naively applied our rule for translating the `for` loop into a `while` loop? What would be wrong with this code?
B. How could you replace the `continue` statement with a `goto` statement to ensure that the `while` loop correctly duplicates the behavior of the `for` loop?

# Homework 4

## 1. Jump Table

Fill the blanks in assembly using jump table. **NOTE**: you can fill one or several instructions or symbols in one blank.

```
long x = <some value>;
long result = 0;
switch(x){
    case 5:
        result = x + 1;
        break;
    case 6: case 9:
        result = x + x;
        // fall through
    case 7:
        result = result * 5;
        break;
    case 11:
        result = x;
        break;
    default:
        result = -1;
}
return result;
```

```
    .section  .rodata
    .align 8
.L4:
    .quad ________
    .quad ________
    .quad ________
    .quad ________
    .quad ________
    .quad ________
    .quad ________
prog:
    movq  [x], -16(%rbp)
    movq  $0,  -8(%rbp)
    movq  -16(%rbp), %rax
    ________________
    jmp *_________
.L3:
    movq  -16(%rbp), %rax
    addq  $1, %rax
    ________________
.L5:
    movq  -16(%rbp), %rax
    addq  %rax, %rax
    ________________
.L6:
    movq  -8(%rbp), %rdx
    leaq  _____________, %rax
    ________________
.L7:
    movq  -16(%rbp), %rax
    ________________
.L2:
    movq  $-1, -8(%rbp)
.L8:
    movq -8(%rbp), %rax
    // function return...
```

## 2. Procedure call

There are two functions P and Q. ICSTA writes the assembly of these two functions. Read them and answer the following questions.

```
long Q(long n)
{
        long result;
        if (n <= 1)
                result = 1;
        else
        result = n * Q(n-1);
        return result;
}
long P(long x) {
        long a0 = x;
        long a1 = x + 1;
        long a2 = x + 2;
        long a3 = x + 3;
        long a4 = x + 4;
        long a5 = x + 5;
        long a6 = x + 6;
        long a7 = x + 7;
        h = proc(a0,a1,a2,a3,a4,a5,a6,&a7); // proc is another function
        return h;
}

Assembly of P:
P:
    pushq %r15
    pushq %r14
    pushq %r13
    pushq %r12
    pushq %rbp
    pushq %rbx
    subq $24, %rsp
    movq %rdi, %rbx
    leaq 1(%rdi), %r15
    leaq 2(%rdi), %r14
    leaq 3(%rdi), %r13
    leaq 4(%rdi), %r12
    leaq 5(%rdi), %rbp
    leaq 6(%rdi), %rax
    movq %rax, (%rsp)
    leaq 7(%rdi), %rdx
```

```
    movq %rdx, 8(%rsp)
    _________________  // you should pass the parameters to proc() here
    call proc
    ...  // then the function returns
Assembly of Q:
Q:
    movq %rdi, %r12
    movl $1, %eax
    cmpq $1, %rdi
    jle  .L35
    leaq -1(%rdi), %rdi
call Q
    imulq %r12, %rax
.L35:
    ret
```

1. Where are the local variables a0-a7 in function P stored in? Write the register name or memory address.
2. Fill the blank before *call proc* with proper instructions.
3. There is a problem in the assembly of Q. Find it out and fix it.

# Homework 6

## Struct and union

Please answer the following questions according to the definition of the union.

```
union ele {
    struct s1 {
        char cc;
        union ele *next;
        short ss;
        long long int li;
    } e1;
    int i;
    struct s2 {
        char c;
        struct s1 (*f) (int i, short ss, long long int li);
        char str[3];
        short s;
        int *p[2];
        char c2;
        int ii;
    } e2;
} u;
```

1. Fill in the following blocks. (please represent address with Hex)

| | |
|---|---|
| sizeof(u.e1) | |
| sizeof(u.e2) | |
| sizeof(union ele) | |
| u | 0x601060 |
| u.e1.next | |
| u.e1.li | |
| u.e2.f | |
| u.e2.p[1] | |

2. How many bytes are WASTED in struct s2 under x86-64? If you can rearrange the declarations in the struct s2, how many bytes of memory can you SAVE in struct s2 compared to the original declaration under x86-64?

# Pointers and array

Answer the following questions and explain why. Assume we use x86-64 machines.

1. Is the value of &(a[1]) equals to value of (b+1) when
   ```
   int a[2]; char *b = a;
   ```

2. Is the value of &(a[1]) equals to value of (b+1) when
   ```
   int a[2]; char **b = a;
   ```

3. Is the value of &(a[1]) equals to value of (b+1) when
   ```
   int *a[2]; char **b = a;
   ```

4. Is the value of &(a[1]) equals to value of (b+1) when
   ```
   int a[2]; char (*b)[2][2] = a;
   ```

5. Is the value of &(a[1]) equals to value of (b+1) when
   ```
   int a[2]; char (**b)[2][2] = a;
   ```

6. What is a?
   ```
   int *(*a[3])(int *, int);
   ```

# Homework 7

## 1. Buffer Overflow

One of TAs of ICS wrote a buggy program. The following C code and assembly code are executed on a **64-bit little endian** machine. He used **`gets()`** functions in section 3.10.3 on CSAPP.

```
void buggy(){
    char buf[0x10];
    gets(buf);
}
int main(){
    buggy();
    return 0;
}
```

```
00000000004004e6 <buggy>:
  4004e6:      55                    push   %rbp
  4004e7:      48 89 e5              mov    %rsp,%rbp
  4004ea:      48 83 ec 10           sub    $0x10,%rsp
  4004ee:      48 8d 45 f0           lea    -0x10(%rbp),%rax
  4004f2:      48 89 c7              mov    %rax,%rdi
  4004f5:      e8 17 00 00 00        callq  400511 <gets>
  4004fa:      c9                    leaveq
  4004fb:      c3                    retq

00000000004004fc <main>:
  4004fc:      55                    push   %rbp
  4004fd:      48 89 e5              mov    %rsp,%rbp
  400500:      b8 00 00 00 00        mov    $0x0,%eax
  400505:      e8 dc ff ff ff        callq  4004e6 <buggy>
  40050a:      b8 00 00 00 00        mov    $0x0,%eax
  40050f:      5d                    pop    %rbp
  400510:      c3                    retq
```

Now the TA uses different strings to feed the **`gets()`** in **`buggy()`**. Give the corresponding return address of function **`buggy()`** to each return address. (NOTE: the ASCII number of '0' is 48.

**a.** “”

b. "0123456789"

c. "01234567890123456789"

d. "012345678901234567890123"

e. "0123456789012345678901234567890123456789"

## 2. Floating point

Consider a 16-bit floating-point representation based on the IEEE floating-point format, with 1 sign bit, 5 exp bits, 10 frac bits, called Float16.

(1) Fill in the following table. Represent M in the form x or x/y where x is an integer and y is an integral power of 2, and represent Value in the form a or a * 2^b where a and b are integers.

| Description | Hex | M | E | Value |
|---|---|---|---|---|
| -0 | | | | -- |
| Largest negative Normalized value | | | | |
| $+\infty$ | | -- | -- | -- |
| Largest Denomalized value | | | | |
| $(11.375)_{10}$ | | | | |
| Number with hex representation 0x4BF7 | 0x4BF7 | | | |

(2) Assume we use IEEE round-to-even mode to do the approximation. Now a, b are both Float16, with a = 0x4663 and b = 0x394c represented in hex. Compute a+b and represent the answer in hex.

(3) Using Float16, what's the difference between 2^15 + 0.5 – 2^15 and 2^15 – 2^15 + 0.5? Calculate them to explain why.

# Homework 12

## Understanding Memory Performance

1. Consider the following function to copy the contents of one array to another:

```
void copy_array(long *src, long *dest, long n)
{
    long i;
    for (i = 0; i < n; i++)
        dest[i] = src[i];
}
```

Suppose `a` is an array of length 1000 initialized so that each element `a[i]` equals `i`.

(a) What would the array become if call `copy_array(a+1, a, 999)`?

(b) What would the array become if call `copy_array(a, a+1, 999)`?

(c) Our performance measurements indicate that the call of part a has a CPE of 1.2, while the call of part b has a CPE of 5.0. To what factor do you attribute this performance difference?

(d) What performance (CPE) would you expect for the call `copy_array(a, a, 999)`? Please explain your answer.

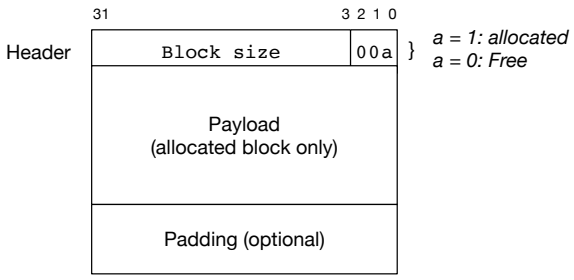
## Dynamic Memory Allocation

1. Determine the block sizes and header values that would result from the following sequence of `malloc` requests.

| Request | Block size (decimal bytes) | Block header (hex) |
|---|---|---|
| `malloc(1)` | ____ | ____ |
| `malloc(5)` | ____ | ____ |
| `malloc(12)` | ____ | ____ |
| `malloc(13)` | ____ | ____ |

Assumptions:

- *words* are 4-byte objects and *double words* are 8-byte objects.
- The memory allocated to the user is at the granularity of *word*. That is, the size requested are rounded up to the nearest multiple of 4 bytes.
- The allocator maintains double-word alignment and uses an imiplicit free list with the block format as below.




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

| Alignment | Allocated block | Free block | Minimum block size (bytes) |
|---|---|---|---|
| Single word | Header and footer | Header and footer | ____ |
| Single word | Header, no footer | Header and footer | ____ |
| Double word | Header and footer | Header and footer | ____ |
| Double word | Header, no footer | Header and footer | ____ |

# Homework 1

# Problem 1

Suppose we have a hard disk with:

512 B sector size

206 Sectors per track in average

50864 tracks per surface

2 surfaces per platter

14 platters per disk

A. What is the capacity of disk in GB? Please use the unit conversion 1KB=1024B and similar conversions on MB and GB.

B. The manufacturer claims it to be a 146.8GB hard disk. If you didn't make anything wrong in the first sub-problem, the actual capacity is smaller than the claimed one.
   a) Why could there be such a difference?

   b) How much percentage of capacity is lost between the actual one and the claim one?

# Problem 2

Suppose that a 1MB file consisting of 512-byte logical blocks is stored on a disk drive with the following characteristics:

Rotational rate: 10,000 RPM

$T_{avg\ seek}$: 5 ms

Average # sectors/track: 1000

Surfaces: 4

Sector size: 512 B

For each case below, suppose that a program reads the logical blocks of the file sequentially, one after the other, and that the time to position the head over the first block is $T_{avg\ seek} + T_{avg\ rotation}$.

A. Best case: Estimate the optimal time (in ms) required to read the file given the best possible mapping of logical blocks to disk sectors (i.e., sequential).

B. Random case: Estimate the time (in ms) required to read the file if blocks are mapped randomly to disk sectors.

# Problem 3

Permute the loops in the following function so that it scans the three-dimensional array a with a stride-1 reference pattern.

```
1.  #define N 8
2.  int sumarray3d (int a[N][N][N])
3.  {
4.      int i, j, k, sum = 0;
5.
6.      for (i = 0; i < N; i++) {
7.          for (j = 0; j < N; j++) {
8.              for (k = 0; k < N; k++) {
9.                  sum += a[k][i][j];
10.             }
11.         }
12.     }
```

# Homework 2

1. The following table gives the parameters for a number of different caches. Fill in the missing fields in the table for each cache. Recall that m is the number of memory address bits, C is the cache size (number of data bytes), B is the block size in bytes, E is the associativity, S is the number of cache sets, t is the number of tag bits, s is the number of set index bits, and b is the number of block offset bits.

| Cache | C | S | E | B | m | t | s | b |
|---|---|---|---|---|---|---|---|---|
| A | 1024 | 16 | _____ | 4 | 64 | _____ | 4 | 2 |
| B | 32768 | 128 | 8 | _____ | _____ | 20 | _____ | _____ |
| C | 2048 | 32 | _____ | _____ | 32 | _____ | 5 | 6 |
| D | _____ | _____ | 2 | 8 | 64 | 53 | 8 | 3 |

2. Bob has a machine with 4-way cache. The cache line size is 64 bytes. There are 4 sets in the cache. Alice executes the code below on this machine. Suppose `sum`, `i`, `j`, `k` are stored in registers and `sizeof(int)` will return 4.

```
 1 #define M 16
 2 #define N 16
 3
 4 int a[M][N];
 5
 6 int sum()
 7 {
 8     int i, j;
 9     int sum = 0;
10
11     for (i = 0; i < M; i++)
12         for (j = 0; j < N; j++)
13             sum += a[i][j];
14
15     return sum;
16 }
```

   (a) How many memory accesses in total are there in the loop between line 11 and line 13?

   (b) How many cache misses in the loop in total?

   (c) Suppose the latency of cache hit is 4 cycles. An access to main memory requires 200 cycles for 64 bytes. What is the average latency of accessing array elements in cycles when executing the loop between line 11 and line 13?

   (d) Bob wants to execute this program on a new machine, whose cache doubles in size, to get better performance. The larger cache comes in different styles. Which one(s) of the followings will help?

      1. Double the associativity of a set

2. Double the number of sets
3. Double the size of cache line

3. Suppose we have a 16-bit machine whose cache is two-way set associative (E=2), with a 4-byte block size (B=4) and four sets (S=4). The contents of the cache are shown as follows, with all addresses, tags, and values given in hexadecimal notation.

| Set index | Tag | Valid | Byte 3 | Byte 2 | Byte 1 | Byte 0 |
|---|---|---|---|---|---|---|
| 0 | 281 | 0 | 43 | 42 | 41 | 40 |
| 0 | 361 | 1 | D0 | CC | 97 | FE |
| 1 | 98E | 0 | 47 | 46 | 45 | 44 |
| 1 | 361 | 1 | 15 | 05 | 20 | 15 |
| 2 | 0B5 | 1 | 48 | 4A | 49 | 48 |
| 2 | 412 | 0 | 25 | E6 | 68 | 06 |
| 3 | 5FF | 1 | FF | 03 | C0 | 9A |
| 3 | 084 | 1 | 9F | 62 | 47 | 5B |

For each of the following memory accesses, indicate if it will be a cache hit or miss when carried out in sequence as listed. Also give the value of a read if it can be inferred from the information in the cache.

| Operation | Address | Hit (Y or N) | Read value (or unknown) |
|---|---|---|---|
| Read | 0x3615 | _____ | _____ |
| Read | 0x2816 | _____ | _____ |
| Read | 0x084F | _____ | _____ |
| Read | 0x412B | _____ | _____ |
| Read | 0x2817 | _____ | _____ |

## Homework 5

# Problem 1

Please specify **which kind of exception** (Faults, Aborts, Traps, Interrupts) will occur in the given scenario, point out whether it is **asynchronous** or **synchronous**, and specify **where the exception handler will return to**.

A. Access content at address 0x0.

B. The memory of your PC corrupted.

C. You run a command "kill -9 <pid>" in your shell.

D. You click your mouse.

# Problem 2

Consider three processes with the following starting and ending times:

| Process | Start time | End time |
|---|---|---|
| A | 0 | 2 |
| B | 1 | 4 |
| C | 3 | 5 |

A. For each pair of processes (AB, AC, BC), indicate whether they run concurrently or not.

B. Suppose the three processes are running on a single-core machine. Process A is running a program with an infinite loop, as is shown below:

```
1. int main(void) {
2.         while (1) {
3.                 /* waste CPU time */
4.         }
5. }
```

Would this process A block the execution of process B? Why?

# Problem 3

A. A process in user mode is not allowed to execute privileged instructions such as those intend to initiate an I/O operation. How does a C program manage to print characters on your screen when it invokes 'printf' function in user mode?

B. Operating system kernel lives in the top portion of every process's private address space, inaccessible to processes in user mode. Can you give some reasons of why not let kernel live in its separate address space for protection? (Hint: what might be different for system calls?)

# Homework 6

1. *FORK AND EXECVE.* Read the C program and answer the questions below. NOTE: `/bin/echo` is an executable file that will print its arguments on the screen.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>

char *ch;

int main()
{
        ch = malloc(1);
        *ch = 'A';

        if (fork() == 0) {
                *ch = 'B';
                printf("%c\n", *ch);

                if (fork() == 0) {
                        printf("C\n");
                }
                else {
                        exit(0);
                }
        }
        else {
                while (waitpid(-1, NULL, WUNTRACED) > 0);
                char *my_argv[] = {"/bin/echo", ch, 0};
                execve(my_argv[0], my_argv, 0);
        }

        free(ch);
        return 0;
}
```

   (a) What is the possible output of this program? Is the output deterministic? Please explain why.

   (b) Is there any memory leakage or double free issue for the variable `ch` in each process? Please explain why.

2. *BLOCKING AND UNBLOCKING SIGNALS.* Linux provides an explicit mechanism for blocking signals. Read the C program and answer the questions below.

```
1 #include <stdio.h>
2 #include <signal.h>
3 #include <sys/types.h>
4 #include <unistd.h>
```

```
 5
 6 void sig_han(int sig)
 7 {
 8       printf("signal␣handled\n");
 9 }
10
11 int main()
12 {
13       sigset_t set;
14       int i;
15
16       signal(SIGKILL, sig_han);
17       signal(SIGINT, sig_han);
18       sigemptyset(&set);
19       sigaddset(&set, SIGINT);
20       sigprocmask(SIG_BLOCK, &set, NULL);
21
22       for (i = 0; i < 3; i++) {
23               printf("send␣signal\n");
24               kill(getpid(), SIGINT);
25       }
26
27       sigprocmask(SIG_UNBLOCK, &set, NULL);
28       return 0;
29 }
```

(a) When run, the call at line 16 fails. Why? And what is the return value of this call and what value would `errno` be set to? (You can run the program or refer to the `man` page `signal(2)`).

(b) What is the output of this program? Please explain your answer.

3. (Optional) *SIGNAL HANDLER INSIDE*. For a user-defined signal handler (suppose a x86-64 machine runnnig Linux),

   (a) does it run in user mode or kernel mode (that is, in non-privileged mode or privileged mode)?

   (b) what stack does it use, does it share the same stack with your normal functions? Use `GDB` to stop inside a signal handler and check the stack address.

   (c) where does the signal handler function return to, what is the next instruction after the signal handler function `retq`? Use `GDB` to stop inside a signal handler and step instruction-by-instruction to check where is the handler function returns to (You may find `GDB`'s `si`, `ni`, `disassemble` commands useful).

# Homework 7

# Problem 1

```
volatile sig_atomic_t counter = 0;
void handler(int sig){
    int olderrno = errno;
    sigset_t hmask, hprev;
    sigfillset(&hmask);
    while (counter){
        waitpid(-1, NULL, 0);
        sigprocmask(SIG_BLOCK, &hmask, &hprev);
        sio_putl((long)(--counter));
        sigprocmask(SIG_SETMASK, &hprev, NULL);
        sio_puts("Children running\n");
    }
    errno = olderrno;
}
int main(){
    Signal(SIGCHLD, handler);
    sigset_t mask, prev;
    sigfillset(&mask);
    for(int i = 0; i < 5; i++){
        if (fork() == 0){
            printf ("Child\n");
            exit(0);
        }
        sigprocmask(SIG_BLOCK, &mask, &prev);
        counter++;
        sigprocmask(SIG_SETMASK, &prev, NULL);
    }
    while(!counter) pause();
    exit(0);
}
```

1 The given code aims to create 5 children processes and reap them. Try to **describe** what unexpected problem may happen during execution, and **give the solution**.

# Problem 2

```
int counter = 2;

void handler1(int sig) {
    counter = counter + 1;
    printf("%d\n", counter);
    exit(0);
}
int main() {
    signal(SIGINT, handler1);
    printf("%d\n", counter);
    if ((pid = fork()) == 0) {
        while(1) {};
    }
    kill(pid, SIGINT);

    counter = counter - 1;
    printf("%d\n", counter);
    waitpid(-1, NULL, 0);
    counter = counter + 1;
    printf("%d\n", counter);
    exit(0);
}
```

1. The above program validates some guidelines in section 8.5.5, please point out which ones are validated (even if unnecessary) and rewrite the **handler** according to the guidelines (**HINT**: you can use **`Sio_puts`** as thread safe **`printf`** if needed).

2 . Please write down all the possible outputs of the original programs.

# Problem 3

```
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
void handler(int sig) {
    static int beeps = 0;
    printf("BEEP\n");
    if (beeps < 5) {
        beeps += 1;
        fork();
        alarm(1); /* next SIGALRM will be delivered in 1s */
    } else{
        printf ("BOM!\n");
        exit(0);
    }
}
int main() {
    signal(SIGALRM, handler); /* install SIGALRM handler */
    alarm(1); /* next SIGALRM will be delivered in 1s */
    /* signal handler returns control here each time */
    while (1);
    exit(0);
}
```

1. How many seconds will this program remain approximately?
2. How many BEEPs and BOOMs will be printed if you run the above program?

# Problem 4

| | |
|---|---|
| 1 int main(){<br>2 int fd1, fd2;<br>3 char c;<br>4 fd1 = open("c.txt", O_RDONLY, 0);<br>5 int i = 0;<br>6 if(fork() == 0){<br>7 read(fd1, &c, 1);<br>8 }<br>9 read(fd1, &c, 1);<br>10 printf("%c\n", c);<br>11 exit(0);<br>12 } | a.txt<br><br>12345 |

Please give **all** the possible output and one execution order for each. You can use line Cx or line Px to distinguish the same line of code executed by child and parent.

# Homework 8

**Problem 1**

```
int main(){
    int fd1, fd2, fd3;
    char *buf1=(char*)malloc(10);
    char *buf2=(char*)malloc(10);
    fd1 = open("a.txt", O_RDWR, 0);
    fd2 = open("b.txt", O_RDWR|O_APPEND, 0);
    fd3 = open("a.txt", O_RDWR, 0);
    if(fork()==0){
        read(fd2, buf1, 2);
        dup2(fd1, fd2);
        read(fd2, buf1, 1);
        exit(0);
    }
    waitpid(-1, NULL, 0);
    read(fd2, buf1, 3);
    write(fd1, buf1, 3);
    read(fd1, buf1, 10);
    printf("%s\n", buf1);
    read(fd3, buf2, 10);
    dup2(fd2, 1);
    printf("%s\n", buf2);
    free(buf1);
    free(buf2);
    exit(0);
}
```

a. txt

```
abcdefg
```

b. txt

```
0123456789
```

1. What will the contents of a.txt and b.txt be after the program completes?
2. What will be printed on stdout?

**Problem 2**

1. Please give three implementations of the following function, one uses *getnameinfo*, one uses *inet_ntop, ntohs* and one uses only *ntohs*:

   ***void print_sin(const struct sockaddr_in *addr);***

   given an IPv4 address struct, print it as "x.x.x.x:port"
   (e.g. 127.0.0.1:1234)

   Note: For simplicity, you do NOT need to take care of error handling.

2. Assume we initialize addr as following:

   ```
   struct sockaddr_in addr;
   memset(&addr, 0, sizeof(addr));
   addr.sin_family = AF_INET;
   addr.sin_addr.s_addr = 0x13784293;
   addr.sin_port = 12387;
   ```

   What's the output of print_sin? Why the port number is not 12387?

# Homework 12

## I. Virtual Memory

1. Which type of address is used in each of following scenarios, virtual or physical address?
    1) The address of variables in C program
    2) The address stored in a C pointer
    3) The address of a C pointer
    4) The address used in looking up L1 cache
    5) The value in CR3
    6) The address in L2 PTE
    7) The address in L4 PTE
    8) The value in PC register
2. The lookup of L1 cache usually consists of three steps: locating the set; comparing the tag of each cache line in the set; returning bytes from cache or loading value from next level memory system. As the lookup uses physical address, the hardware can only start the cache lookup after address translation is completed. How to make cache lookup and address translation parallelized? Show the requirements to the parameters of your paging and cache system.

## II. Page Table

1. Why could multiple level page table save memory?
    1) If we only use 0x400000-0x800000, compare the memory usage of page table between single level and 4 level.

    2) If we have enough physical memory and use the whole $2^{48}\mathrm{B}$ virtual space, compare the memory usage again.

2. Given the following page table and cr3=0x1000, please translate the virtual addresses to physical addresses.
   (Assume the machine is x86_64, which uses 4 level page table)

| Physical Address of PTE | Address Part in PTE |
|---|---|
| 0x1000 | 0x4000 |
| 0x1008 | 0x4000 |
| 0x4000 | 0x5000 |
| 0x5000 | 0x7000 |
| 0x5008 | 0x8000 |
| 0x5010 | 0x9000 |
| 0x5018 | 0x3000 |
| 0x5020 | 0x5000 |
| 0x7008 | 0x8000 |
| 0x7010 | 0x10000 |

| Physical Address | Virtual Address |
|---|---|
| 0x1234 | |
| 0x2234 | |
| 0x8000802135 | |

# III. Demand paging & TLB

Assume on a x86_64 machine without cache, we have an `int a[70]` at virtual address 0x8000344f00, a program access a[0], a[1], ..., a[69] one by one. Before the first access, we have only an empty L1 page table.

1. How many PP is used after the program access a[0], a[64] and a[69]? (Remember the page table is stored in PP too)

2. How many memory accesses and page fault happened in the program if we have no TLB?

3. How many memory accesses and page fault happened in the program if we have a full-associative infinite TLB?

## Homework 13

# Problem 1

Imagine a system with the following specifications:

- The system has 4GB of virtual memory
- The system has 256KB of physical memory
- The page size is 4KB
- The page table size is 4KB
- Length of each PTE is 4B
- The TLB is 4-way set associative with 16 total entries
- The L1 d-cache has 4 sets and 4 bytes in each line
- LRU replacement policy in TLB and d-cache

1. Please fill the blanks about the characteristics of the system:

   bits of page offset:
   bits of PPN:
   bits of VPN:
   number of PTEs in one of page table:
   bits of VPN of each level:
   number of levels:
   bits of tag in TLB:
   bits of tag in cache:

2. Part of the content of TLB, page tables, and L1 d-cache are given below:

| TLB | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Ind | Tag | PPN | Valid | Tag | PPN | Valid | Tag | PPN | Valid | Tag | PPN | Valid |
| 00 | 03 | -- | 0 | 09 | 0D | 1 | 00 | -- | 0 | 07 | 02 | 1 |
| 01 | 03 | 2D | 1 | 02 | -- | 0 | 04 | -- | 0 | 0A | -- | 0 |
| 10 | 02 | -- | 0 | 08 | -- | 0 | 06 | -- | 0 | 03 | -- | 0 |
| 11 | 07 | -- | 0 | 04 | 0D | 1 | 0A | 34 | 1 | 02 | -- | 0 |

| L1 Page Table | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| VPN | PPN | Valid | VPN | PPN | Valid | VPN | PPN | Valid | VPN | PPN | Valid |
| 00 | 18 | 1 | 05 | -- | 0 | 0A | 0C | 1 | 0F | -- | 0 |
| 01 | -- | 0 | 06 | -- | 0 | 0B | -- | 0 | 10 | -- | 0 |
| 02 | 13 | 1 | 07 | -- | 0 | 0C | -- | 0 | 11 | 1A | 1 |
| 03 | 12 | 1 | 08 | -- | 0 | 0D | 1D | 1 | 12 | -- | 0 |
| 04 | -- | 0 | 09 | 22 | 1 | 0E | -- | 0 | 13 | 0D | 0 |

| L2 Page Table @ 0x18000 | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| VPN | PPN | Valid | VPN | PPN | Valid | VPN | PPN | Valid | VPN | PPN | Valid |
| 00 | 28 | 1 | 05 | -- | 0 | 0A | 1C | 1 | 0F | -- | 0 |
| 01 | -- | 0 | 06 | -- | 0 | 0B | -- | 0 | 10 | -- | 0 |
| 02 | 33 | 1 | 07 | 17 | 1 | 0C | -- | 0 | 11 | 3A | 1 |
| 03 | 02 | 1 | 08 | -- | 0 | 0D | 2D | 1 | 12 | 22 | 1 |
| 04 | -- | 0 | 09 | 02 | 1 | 0E | -- | 0 | 13 | 0D | 1 |

| L1 d-cache | | | | | | |
|---|---|---|---|---|---|---|
| Index | Tag | Valid | block0 | block1 | block2 | block3 |
| 0 | 3A27 | 1 | 0x0A | 0x0B | 0x0C | 0x0D |
| 1 | -- | 0 | -- | -- | -- | -- |
| 2 | 1C28 | 0 | 0x1A | 0x1B | 0x1C | 0x1D |
| 3 | D27 | 1 | 0x00 | 0x01 | 0x02 | 0x03 |

Step through the following address translation and cache accessing. If there is a cache miss, enter "--" for "Cache Byte Returned". If there is a page fault, enter "--" for "PPN" & "Physical address" and leave the second table empty.

1) VA: 0x1327c

| **Parameter** | **Value** |
|---|---|
| VPN | |
| TLB index | |
| TLB tag | |
| TLB hit? (Y/N) | |
| Page fault? (Y/N) | |
| PPN | |
| Physical Address | |

| **Parameter** | **Value** |
|---|---|
| Cache Byte Offset | |
| Cache index | |
| Cache tag | |
| Cache hit? (Y/N) | |
| Cache Byte Returned | |

2) VA: 0x0A289

| **Parameter** | **Value** |
|---|---|
| VPN | |
| TLB index | |
| TLB tag | |

| | |
|---|---|
| TLB hit? (Y/N) | |
| Page fault? (Y/N) | |
| PPN | |
| Physical Address | |

| Parameter | Value |
|---|---|
| Cache Byte Offset | |
| Cache index | |
| Cache tag | |
| Cache hit? (Y/N) | |
| Cache Byte Returned | |

3) VA: 0x411272

| Parameter | Value |
|---|---|
| VPN | |
| TLB index | |
| TLB tag | |
| TLB hit? (Y/N) | |
| Page fault? (Y/N) | |
| PPN | |
| Physical Address | |

| Parameter | Value |
|---|---|
| Cache Byte Offset | |
| Cache index | |
| Cache tag | |
| Cache hit? (Y/N) | |
| Cache Byte Returned | |

3. Calculate the **total page table size** (sum of all page tables of all levels) of this system. For another system with the same virtual memory space, page size, and PTE size, but only uses a **single** page table, calculate the **total page table size** of this system. Compare these two sizes, and **explain the benefits of using multi-level page table**.