Fall term 2010
KAIST EE209 Programming Structures for EE

Final exam

Thursday Dec 16, 2010

Student's name: _____

Student ID: _____

The exam is closed book and notes. Read the questions carefully and focus your answers on what has been asked. You are allowed to ask the instructor/TAs for help only in understanding the questions, in case you find them not completely clear. Be concise and precise in your answers and state clearly any assumption you may have made. All your answers must be included in the attached sheets. You have 90 minutes to complete your exam. Be wise in managing your time.
Please do not fill in the "Score" fields below. Self-grading is not allowed. Good luck!

Scores

Question 1	/12
Question 2	/13
Question 3	/10
Question 4	/10
Question 5	/10
Question 6	/10
Question 7	/25
Total	/90

1. Dancing with pointers (12 pt)

(a) – (c)

```
int a[] = {1,2,3,4,5,6,7,8,9};
int *p = &a[1], *q = &a[5];
```

(a) What is the value of *(p+2)? (2pt)

4

(b) What is the value of *(q-3)? (2pt)

3

(c) What is the value of q –p ? (2pt)

4

(d) Define a function pointer type (use `typedef`) named “handler” which takes one integer pointer as an argument and returns a void pointer. (2pt)

*typedef void * (*handler) (int *);*

(e) What does this program print out? (4pt)

```
char a[] = “abcdefg”;
int *p = (int *)a;

p++;
((char *)p)[2] = ‘X’;

printf(“p = %s\n”,    (char *)a);
```

p = efX

2. Simple C programming (15 pt)

<qsort() programming> Qsort() is a C runtime library function that sorts an arbitrary array of any type using the quick sort algorithm. Here is the manual page for qsort().

NAME

qsort - sorts an array

SYNOPSIS

```
#include <stdlib.h>

void qsort(void *base, size_t nmemb, size_t size,
           int(*compar)(const void *, const void *));
```

DESCRIPTION

The qsort() function sorts an array with nmemb elements of size size. The base argument points to the start of the array. The contents of the array are sorted in ascending order according to a comparison function pointed to by compar, which is called with two arguments that point to the objects being compared.

The comparison function must return an integer less than, equal to, or greater than zero if the first argument is considered to be respectively less than, equal to, or greater than the second. If two members compare as equal, their order in the sorted array is undefined.

RETURN VALUE

The qsort() function returns no value.

(a) We have code that sorts and prints out the argument strings to the standard output as follows. Please fill out cmpstringp() below (8 pt).

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

static int
cmpstringp(const void *p1, const void *p2)
{
    /* fill in the code below, you can use any C runtime library functions */



    return strcmp( (const char *)p1, (const char *)p2);
        (or any code that does the same thing as strcmp)




}
```

```
int
main(int argc, char *argv[])
{
        int j;

        if (argc < 2) {
          fprintf(stderr, "Usage: %s <string>...\n", argv[0]);
          exit(EXIT_FAILURE);
        }

        qsort(&argv[1], argc - 1, sizeof(argv[1]), cmpstringp);

        for (j = 1; j < argc; j++)
                puts(argv[j]);
        exit(EXIT_SUCCESS);
}
```

(b) This function checks whether the machine is a big-endian machine or not. If it is a big endian machine, it returns 1, otherwise it returns 0. Please fill out the function. (5pt)

```
int IsBigEndian(void)
{
/* fill in the code here */

/* something similar to the following code */

int x = 0x01020304;
char *p = (char *)&x;

return (p[0] == 0x01);









}
```

3. The following assembly code was generated by gcc209 by compiling a simple C function (named g). It takes a single unsigned integer parameter and returns an unsigned integer value. You may assume the parameter passed in is in the range of 0 to 10. (10pt)

```
    .file "wow.c"
        .text
.globl g
        .type   g, @function
g:
        pushl   %ebp
        movl    %esp, %ebp
        subl    $8, %esp
        cmpl    $1, 8(%ebp)
        ja      .L2
        movl    $1, %eax
        jmp     .L3
.L2:
        movl    8(%ebp), %eax
        decl    %eax
        subl    $12, %esp
        pushl   %eax
        call    g
        addl    $16, %esp
        imull   8(%ebp), %eax
.L3:
        leave
        ret
```

Some hints to understand this code:

- "leave" releases the stack frame, copying EBP into ESP
- imull multiplies two operands and stores the result to the second operand.
- decl decrements the operand by 1

(a) What is stored in 8(%ebp)? (2pt)

The passed parameter.

(b) How many local variables does this function, g, have? (2pt)

Zero. (0).

(c) What is the value of g(3)? (2 pt)

g(3) returns 6.

(d) Describe the summary of what this function does in one or two sentences. (4pt)

Given n as a parameter, g calculates and returns n!(factorial(n)).

4. Memory management (10pt)
   a) Consider a computer system that has virtual memory with 32-bit addresses and a 8KB page size. How many bits are used to identify the byte offset in a page? How any virtual pages a process have? (2pt)

      13 bits for a byte offset in a page. (1pt)
      $2^{32-13} = 2^{19}$ virtual pages per process is possible. (1pt)

   b) Briefly explain temporal locality and spatial locality. (2pt)

      Temporal locality means that the recently accessed memory location is likely to be accessed again in the near future (1pt) while spatial locality means that the memory location nearby the recently accessed memory is likely to be used in the near future. (1pt)

   c) Why does implementing malloc() and free() with a single free list (with free blocks of different sizes) lead to a lot of virtual-memory page faults? (2pt)

      Free list traversal looking for a good chunk will visit many memory addresses in the different virtual pages, forcing swapped-out pages to be brought back to physical memory.

   d) A call to fork() creates a child process that inherits a copy of the parent's virtual address space. The virtual address space is quite large, so copying every byte would be quite time consuming. How is this overhead avoided (be specific)? (4pt)

      By copying only the page table entries of the parent process (not the bytes in each page) (2pt) and sharing the physical memory pages until there is any update on any of the page (1pt). If there is an update on a specific page, that page will be copied and allocated separately for each process – copy-on-write (COW) mechanism.(1pt)

5. Exceptions and Process control (10pt)

(a) What is exception in process execution (not java exception)? (2pt)

Deviation of execution from the normal control flow in a process.

(b) Describe four types of exceptions, and give at least one example to each type. (4pt)

- *Interrupts – keyboard/mouse click, hard disk I/O done, network packet arrival at network cards, etc. (1pt)*
- *Traps – invoking system calls (1pt)*
- *Faults – page faults. memory access violation (segmentation faults). divide-by-zero fault, etc.(1pt)*
- *Aborts – physical memory checksum error (1pt)*

(c) What is context switch? List at least three different causes for context switch. (4pt)

Context switch refers to the activity that the OS assigns the CPU to a different process. (1pt)

Causes (for any reasonable cause 1pt)

1. *Timer interrupt (time quantum)*
2. *I/O requests or faults (e.g., disk I/O that takes up long time, page fault handling)*
3. *Explicitly calls sleep() (or similar functions or locks) in the process*

6. Programming with fork(). (10 pt)

```
int main()
{
    int i;
    int x = 3;

    for (i = 0; i < 2; i++) {
        if   (fork() == 0) {
            printf("x = %d\n", --x);
        } else {
            printf("x = %d\n", ++x);
        }
    }
    printf("hello\n");
    return 0;
}
```

a) How many times "hello\n" is printed out? (3pt)

4

b) List all values of x that are printed. (5pt)

1, 2, 3, 3, 4, 5 (-1 : any wrong or missing number)

c) If (i < 2) in the for loop is replaced with (i < n), how many times "hello\n" would be printed out (in terms of n)? (2pt)

2^n

7. Code reading (25pt)

a) The following function checks something with x. What does it do? Describe what it does not how it does. (3pt)

```
int foo(unsigned int x)
{
    return (x & (x-1) == 0);
}
```

foo checks whether x is power of 2 or not.

b) Signal handling (7 pt)

```
#include <signal.h>
#include <stdio.h>

void sighandler(int x)
{
   raise(SIGINT);
   printf("sighandler called\n");
}

int main()
{
   signal(SIGINT, sighandler);
   getchar();
   return 0;
}
```

The behavior of the above program is different depending on the keyboard input. Describe what happens on a different key input. Be specific. (5pt)

Ctrl-C => infinite printing of "sighandler called\n" (4pt)
Other keys (+ enter) => finishes execution without any printouts (1pt)

Give a sequence of shell commands to stop this process when it prints out "signal called\n". Assume the process is running in the foreground. (2pt)

Ctrl-Z => figure out pid of this process by ps command => kill -9 pid

c)
```
char *x = “abcde”;
char *p = x;

p[sizeof(x)] = ‘Y’;
printf(“p=%s\n”, x);
```

What happens when you run the above code? Specify the reason of your answer. (3pt)

It is likely to crash with memory access violation (2pt) since the code tries to write to the read-only section (string literal) (1pt).

d)
```
int foo(const char* st, const char* pat)
{
    int i, j;
    int n = 0;

    for (i = 0; i < strlen(st); i++)
        for (j = 0; j < strlen(pat); j++)
            if (st[i] == pat[j])
                n++;
    return n;
}
```

What does this function do? (Assume characters in pat are unique in the string)(3pt)

Returns the number of characters in the “st” string that appear in the “pat” string.

Fix three lines that need improvement for better performance. (3pt)

strlen(st) is executed each time in the for loop.

⇨ *int len_st = strlen(st);*
for (i = 0; i < len_st; i++)

strlen(pat) is executed each time in the for loop.

⇨ *int len_pat = strlen(st);*
for (j = 0; j < len_pat; j++)

After n++, it’s better to break out of the nested for loop.

⇨ *If (st[i] == pat[j]) { n++; break;}*

Assuming st is typically really large (e.g., 10 Gigabytes), rewrite the code for better performance. You should slightly change the function prototype to be correct. (6pt)

One possible solution is below:

1. *Instead of iterating strlen(pat) times for each character of st, create a bit map that remembers which character in pat is set.*
2. *Using each character in st as an index to the map, see if the corresponding map element is set (meaning that the character is in the pat string)*
3. *Avoid using strlen(st) since it has to scan a few Gigabytes to find the '\0' character. Instead, use a pointer(p) and see if *p is '\0' to check the end of the string.*

2 pt: avoid strlen(st)
3 pt: avoid scanning pat for each character of the st string.
1 pt: use 64 bit integer (long is fine on an 64-bit OS)

```
long long int foo(const char* st, const char* pat)
{
    int i;
    char map[256] = {0}:
    int len_pat = strlen(pat);
    const char *p;
    long long int n = 0;                    /* 64 bit integer */

    for (i = 0; i < len_pat; i++)
        map[pat[i]] = 1;

    for (p = st; *p != \'0'; p++)
        if (map[*p])
            n++;

    return (n);
}
```