Fall term 2012
KAIST EE209 Programming Structures for EE

Final exam

Thursday Dec 20, 2012

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 120 minutes to complete your exam. Be wise in managing your time.

Scores

Question 1	________/10___
Question 2	________/20___
Question 3	________/15___
Question 4	________/10___
Question 5	________/10___
Question 6	________/15___
Total	________/80___

1. Briefly explain following terms or answer the question (10 pt)

   (a) Virtual memory (2pt)

   (b) Page table (2pt)

   (c) External fragmentation (2pt)

   (d) Trap (2pt)

   (e) What is the difference between read(int fd, void* buf, size_t count) (system call) and fread(void* ptr, size_t size, size_t nmemb, FILE *stream) (C library function)? (2 pt)

2. Fun programming (20 pt)

a) Assume a[99] has 99 distinct natural numbers from 1 to 100. Fill out the following function that finds the missing number out of 1 to 100 that does not exist in a[99]. Be concise and be cautious about memory use (including the stack usage) (5pt)

```
int find_missing(int a[99])
{









}
```

b) The function stack grows from high to low address in the Intel architecture. Assume the stack could grow low to high or high to low address in general. Write a function that returns 1 if the stack grows from high to low address, 0 if it grows from low to high address. You can add another function if you want. (5pt)

```
int does_stack_grow_to_low_address(void)
{




}
```

c) What does this function do? (5pt)

```
typedef struct node {
 void *data;
 struct node *next;
} Node;


int f(Node *p)
{
   Node *p1, *p2;

   if (p == NULL) return 0;

   p1 = p2 = p;

   do {
      p1 = p1->next;
      p2 = p2->next;
      if (p2 == NULL)
         return 0;
      p2 = p2->next;
    } while (p1 != NULL && p2 != NULL && p1 != p2);

 return 1;
}
```

d) What's the output of the second printf() (printf("parent ..."))? If it's called multiple times, show every possible output (5pt).

```
int g = 1;

int main(void)
{
 int k = 1;

 if (fork() == 0) {
        k++; g++;
        printf("child k+g=%d\n", k+g);
 }
 g--;
 k += 2;
 wait(NULL);
 printf("parent k+g=%d\n", k+g);
 return 0;
}
```

3. The following assembly code was generated by gcc209 by compiling a simple C function (named f). It takes one signed integer parameter and returns a signed integer value. You may assume the passed-in parameter is in the range of 1 to 40. (15pt)

```
file    "f.c"
        .text
        .globl  f
        .type   f, @function
f:
        pushl   %ebp
        movl    %esp, %ebp
        pushl   %ebx
        subl    $20, %esp
        cmpl    $1, 8(%ebp)
        jg      .L2
        movl    $1, %eax
        jmp     .L3
.L2:
        movl    8(%ebp), %eax
        decl    %eax
        movl    %eax, (%esp)
        call    f
        movl    %eax, %ebx
        movl    8(%ebp), %eax
        subl    $2, %eax
        movl    %eax, (%esp)
        call    f
        addl    %ebx, %eax
.L3:
        addl    $20, %esp
        popl    %ebx
        popl    %ebp
        ret
```

(a) At label .L2, what do 4(%ebp) and 8(%ebp) have? (4pt, 2pt each)

(b) What’s the value of f(5)? (4pt)

(c) Write the equivalent code in C. (5pt)

(d) It looks that the assembly code is inefficient in stack memory usage. How would you change the code to reduce the unnecessary stack memory usage? (2pt)

4. Memory management (10 pt)
   a) List three kinds of cache misses and explain each of them. (3pt)

   b) What is the "first fit" memory allocation strategy? Discuss the advantages and disadvantages of "first fit"? (3pt)

c) Write a program that shows the minimum memory allocable chunk unit including the header/footer overhead. (4pt)

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

int main(void)
{










 return 0;

}
```

5. Exceptions and Process control (10pt)

   (a) What is the difference between function calls and exceptions. List at least three. (3pt)

   (b) Process context refers to the state that each process maintains. What does the context consist of? List at least three. (3pt)

   d) Explain each of the following system call functions in one sentence (4pt)

      A. fork()

      B. exec()

      C. wait()

      D. kill()

6. Big integer operations (15pt)

We are writing a library that can add and multiply two unsigned large integer values that cannot be represented by the C's built-in integer type (unsigned int, unsigned long, unsigned long long). Assume that the largest integer in the library can be represented by 32 * sizeof(unsigned int) bytes. u_int is typedef'ed to be unsigned int.

For example, 0x11111111222222223333333344444444 can be represented by an integer array of size 4. That is

```
u_int a[16] = {0};

a[0] = 0x44444444
a[1] = 0x33333333
a[2] = 0x22222222
a[3] = 0x11111111
```

represents 0x11111111222222223333333344444444. a[3] represents the most significant four bytes whereas a[0] represents the least significant four bytes.

a) add_large() takes two unsigned big integers (a[16], b[16]) and write the sum to c[17]. Please fill out the function. (5pt)

```
void add_large(u_int a[16], u_int b[16], u_int c[17])
{


}
```

b) multiply_large() takes two unsigned big integers (a[16], b[16]) and writes the result of the multiplication of the values into c[32]. Please fill out the function. (10pt)

```
void multiply_large(u_int a[16], u_int b[16], u_int c[32])
{

}
```