Fall term 2011
KAIST EE209 Programming Structures for EE

Final exam

Thursday Dec 15, 2011

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 (10 pt)

(a) Page fault (2pt)

It is a fault that is raised (by the hardware) when a process accesses a page that is mapped in the virtual address space but not loaded in the physical memory. Or it is raised when the accessed page's entry in the process's page table is marked as "not loaded or on disk". The actual page itself may exist in the physical memory (called soft/minor page fault, e.g., happened to be loaded by another process or not written back to disk yet) or the OS may have to load the page from the disk (called hard/major page fault).

(b) Spatial locality (2pt)

Spatial locality refers to the tendency that the memory location nearby the recently accessed memory is likely to be accessed in the near future

(c) Stack frame (2pt)

The stack area used by a function call (from passed parameters to the local variables stored in the stack)

(d) System call (2pt)

A (special) service that is provided by the OS. An interface between a user-level process and the OS.

(e) Signal (2 pt)

An asynchronous notification event delivered from the OS to the user-level process/thread (e.g., access violation, process control event, etc.)

2. Programming with fork() (20 pt)

a) What does this program print out? Write every possible output. (5pt)

```
static void f(void) {
 putchar('A');
 fflush(NULL);
 if (fork() == 0) {
      putchar('B');
      exit(0);
 }
 putchar('C');
 wait(NULL);
 putchar('D');
}

int main(void) {
 putchar('E');
 f();
 putchar('F');
 return 0;
}
```

EABCDF
EACBDF

b) What does this program print out? (5pt)

```
int main(void)
{
 int i;

 putchar('A');
 fflush(NULL);
 for (i = 0; i < 3; i++)
       fork();
 putchar('B');
 return 0;
}
```

ABBBBBBBB (note 8 'B's)

c) What's the maximum number of characters that can be printed out to stdout in the following program? Please note that we got rid of fflush(NULL); from the program above (b). (5pt)

```
int main(void)
{
 int i;

 putchar('A');
 for (i = 0; i < 3; i++)
        fork();
 putchar('B');
 return 0;
}
```

16 characters (ex. ABABABABABABABAB)

d) What's the output of the second printf() (printf("parent ..."))? (3pt). How many times is the second printf(() called? (2pt)

```
int main(void)
{
 int k = 1;

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

k = 3, only once.

3. The following assembly code was generated by gcc209 by compiling a simple C function (named f). It takes two integer parameters and returns an unsigned integer value. You may assume the passed-in parameters are in the range of 1 to 10. (15pt)

```
.file   "f.c"
.text
.globl f
.type   f, @function

f:
        pushl   %ebp
        movl    %esp, %ebp
        subl    $24, %esp
        cmpl    $1, 12(%ebp)
        jne     .L2
        movl    8(%ebp), %eax
        jmp     .L3
.L2:
        movl    12(%ebp), %eax
        movl    %eax, %edx
        shrl    %edx
        movl    8(%ebp), %eax
        imull   8(%ebp), %eax
        subl    $8, %esp
        pushl   %edx
        pushl   %eax
        call    f
        addl    $16, %esp
        movl    %eax, -12(%ebp)
        movl    12(%ebp), %eax
        andl    $1, %eax
        cmpl    $0, %eax
        jne     .L4
        movl    -12(%ebp), %eax
        jmp     .L3
.L4:
        movl    8(%ebp), %eax
        imull   -12(%ebp), %eax
.L3:
        leave
        ret
```

Some hints to understand this code:

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

(a) Where are the two passed-in parameters in terms of %ebp? (4pt)

8(%ebp), 12(%ebp)

(b) What's the value of f(3, 3)? (4pt)

27

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

```
unsigned int f(unsigned int a, unsigned int b)
{
        unsigned int c;

        if   (b == 1)
          return a;

        c = f (a * a, b >> 1);

        if ((b & 1) == 0)
          return c;

        return a * c;
}
```

(d) Describe what the function does in one sentence. (2pt)

Function f(a, b) calculates and returns a^b.

4. Memory management (10 pt)
   a) Given a virtual address, 0x34233230 on a lab machine, what is the virtual page number? What is the page offset? (4pt)

      virtual page number = 0x34233 (the first 20 bits)
      page offset = 0x230 (the last 12 bits, page size = 4KB)

   b) Why could be the "best fit" memory allocation strategy a good method? (2pt)

      The best fit would reduce internal fragmentation and memory chunk splitting

   c) Why could be the "best fit" memory allocation strategy a bad method? (2pt)

      The allocation speed would be slow - in general, it should traverse the free list longer than the first fit case

   d) What does execvp() do? (1pt) Does it create a new process? (1pt)

      It changes the current execution binary of the process to the specified binary. It does not create a new process (simply changes the content of current process and starts executing the new binary)

5. Exceptions and Process control (10pt)

(a) 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)*

(b) 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*

c) I'd like to kill a process whose pid is 1234. What's the Linux command to kill the process? (Assume you have the privilege to kill the process) (2pt)

kill -9 1234 (kill –KILL 1234 or kill –SIGKILL 1234)

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])
{
        int i;

        for (i = 0; i <16; i++)
            c[i] = 0;                    /* initialize the results to 0 */
        /* or    memset(c, 0, 17 * sizeof(u_int));    */

        for (i = 0; i < 16; i++) {
            u_int temp = a[i] + b[i];
            if (temp < a[i] || temp < b[i])
               c[i+1] = 1;                /* carry from the lower elements */
            c[i] += temp;
    }
}
```

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)

(the code below is for an 32-bit OS)

```
void multiply_large(u_int a[16], u_int b[16], u_int c[32])
{
    int i, j, k;


    /* initialization */
    for (i = 0; i < 32; i++)
        c[i] = 0;

    for (i = 0; i < 16; i++) {
        for (j = 0; j < 16; j++) {
            u_int sum, low, high;
            unsigned long long temp;
            int carry = 0;

            /* 32-bit multiplication would produce a 64-bit result */
            temp = ((unsigned long long)a[i]) * b[j];
            low = ((u_int *)&temp)[0];
            high = ((u_int *)&temp)[1];

            /* low order 32- bit addition and carry propagation */
            k = i + j;
            sum = c[k] + low;
            carry = (sum < c[k] || sum < low);    /* got a carry */
            c[k++] = sum;
            if (carry) {c[k]++; while (c[k] == 0) c[++k]++;}

            /* high order 32-bit addition and carry propagation */
            k = i+j +1;
            sum = c[k] + high;
            carry = (sum < c[k] || sum < high); /* got a carry */
            c[k++] = sum;
            if (carry) {c[k]++; while (c[k] == 0) c[++k]++;}
        }
    }
}
```