Spring Semester 2016

KAIST EE209

Programming Structures for Electrical Engineering

Final Exam

Name: _____

Student ID: _____

This exam is open 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. You have 140 minutes (1:00 PM – 3:20 PM) to complete your exam. Be wise in managing your time. Good luck.

Question 1	/ 25
Question 2	/ 20
Question 3	/ 10
Question 4	/ 15
Question 5	/ 15
Question 6	/ 15

Total / 100

Name: Student ID:

1. (25 points) Simple questions

(a) Consider a computer system with virtual memory with 32-bit addresses, and 16KB page size. How many bits are needed to represent a byte offset in a page? How many virtual pages can a process have? (5 points)

⇨ 14 bits are needed to represent a byte offset
⇨ 18 bits are used to represent a page, so a process can have up to 2^18 (256K) virtual pages.

(b) For caching in a memory hierarchy, what is the motivation for a *larger* cache block size? Check the answer. (5 points)

___ Temporal locality

_V__ Spatial locality

(c) Assume you can choose either 1% of page faults or 10% of cache miss for running your program instructions. Which one would you choose for better performance and **why**? (5 points)

⇨ 10% cache miss rate is better since a page fault requires disk activity, which is about 10,000 to 100,000 times slower than a cache miss on memory access.

Name: Student ID:

(d) On Linux, you can copy fileA to fileB like "**`cp fileA fileB`**". During execution, what kind of exceptions does `cp` *possibly* experience and **why**? Assume `cp` does not have any bugs, and fileA is a large file on a disk. (5 points)

- ⇨ Faults: it would experience page faults during execution.
- ⇨ Traps (system calls): it would invoke system calls to read and write to a file.
- ⇨ Interrupts: when reading or writing to disk is done, interrupts are generated.

(e) In assignment 5 part 1, we designed a memory chunk to have a header and a footer and maintain the free chunk list as a doubly-linked list. Why do you need a footer instead of having two pointers (prev and next pointers) as part of the header? Be as specific as possible in your answer. (5 points)

- ⇨ A footer in a chunk allows us to find its previous _adjacent_ chunk in O(1) time without traversing the free list from the start. Having two pointers in the header won't do that.

Name: Student ID:

2. (20 points) Key-value hash table once again

```
enum {BUCKET_COUNT = 1024};
struct Node {
   const char *key;
   int value;
   struct Node *next;
};
struct Table {
   struct Node *array[BUCKET_COUNT];
};

struct Table *TableCreate(void)
{
   struct Table *t;
   t = (struct Table*)calloc(1, sizeof(struct Table));
   return t;
}

int TableInsert(struct Table *t, const char *key, int value)
{
   int h;
   struct Node *p = (struct Node*)malloc(sizeof(struct Node));
   if (p == NULL) return -1;
   if ((p->key = strdup(key)) == NULL) {
      free(p);
      return -1;
   }
   h = hash(key);
   p->value = value;
   p->next = t->array[h];
   t->array[h] = p;
   return 0;
}
```

Name: Student ID:

(a) Fill in the body of the following function. (10 points)

- int TableSearch(struct Table *t, const char *key, int *value) finds the node with 'key' and updates its value at 'value' of the parameter.
- It returns 0 if such a node exists. Otherwise, it returns -1.
- Assume int hash(const char *) is a hash function that returns a value between 0 and (BUCKET_COUNT-1).
- You may use any C runtime library functions. No need to comment your code.

```
int TableSearch(struct Table *t,  const char *key, int *value)
{
   int h = hash(key);
   struct Node *p;

   for (p = t->array[h]; p != NULL; p = p->next) {
      if (!strcmp(p->key, key)) {
         *value = p->value;
         return 0;
      }
   }
   return -1;
}
```

Name: Student ID:

(b) Fill in the body of the following function. (10 points)

- int TableRemove(struct Table *t, const char *key) removes the node with 'key'.
- It returns 0 if such a node exists. Otherwise, it returns -1.
- *Assume int hash(const char *)* is a hash function that returns a value between 0 and (BUCKET_COUNT-1).
- You may use any C runtime library functions. No need to comment your code.

```
int TableRemove(struct Table *t, const char *key)
{
   int h = hash(key);
   struct Node *p, *prev;

   for (p = t->array[h], prev = NULL; p != NULL; p = p->next) {
      if (!strcmp(p->key, key)) {
         if (prev == NULL)
            t->array[h] = p->next;
         else
            prev->next = p->next;
         free(p->key);
         free(p);
         return 0;
      }
      prev = p;
   }
   return -1;
}
```

Name: Student ID:

3. (10 points) Programming with fork()

(a) How many ‘A’ do you see when you call printA()? (Assume every fork() succeeds.) (5 points)

```
void printA(void)
{
    printf("A");
    fflush(stdout);
    if (fork() == 0) {
       fork();
       printf("A");
     }
     printf("A");
}
```

- ⇨ Common: A
- ⇨ first fork – parent: A (fork != 0)
- ⇨ first fork – child: second fork –parent AA
- ⇨ first fork – child: second fork –child AA

- ⇨ In total, we see 6 As

(b) How many ‘A’ do you see when you comment out the line, fflush(stdout);? (5 points)

- ⇨ first fork – parent: AA (fork != 0)
- ⇨ first fork – child: second fork –parent AAA
- ⇨ first fork – child: second fork –child AAA

- ⇨ In total, we see 8 As

Name: Student ID:

4. (15 points) Manipulating stdout and stderr

Fill in the body of main() function such that it swaps stdout and stderr in printEE209() function. With your added code, you should see “Final exam is hard\n” in *stdout* and “Final exam is easy\n” in *stderr* when you execute the program. Fill free to use any C runtime library functions and system calls without worrying about header files. (15 points)

```
void printEE209(void)
{
   fprintf(stdout, “Final exam is easy\n”);
   fprintf(stderr, “Final exam is hard\n”);
}

void main(void)
{
   /* error processing is omitted for brevity*/
   int fd = dup(1);
   dup2(2, 1);
   dup2(fd, 2);
   printEE209();
}
```

Name: Student ID:

5. (15 points) IA-32 Assembly language programming

Translate the following C code into IA-32 assembly language code. (15 points)

C code:

```
int add2int(char *a, int b[], int n)
{
   int sum = atoi(a);
   int i;

   for (i = 0; i < n; i++)
       sum += b[i];
   return sum;
}
```

Assembly language code: Please fill in the code after “push %ebp”.

⇨ There could be many solutions. The following is one of them.

```
add2int:
  pushl %ebp
  movl  %esp, %ebp
  pushl %ebx            # -4($ebp)
  pushl %esi            # -8(%ebp)
  pushl %edi            # -12(%ebp)
  subl   $4, %esp       # for  local var, sum
  movl   8(%ebp), %eax  # for parameter, a
  pushl  %eax           # parameter for atoi
  call   atoi
  addl   $4, %esp
  movl   %eax, -16(%ebp) # store the return val to sum
  movl    $0, %ebx       #  use %ebx as i, i  = 0
  jmp     LCHECK
LOOP:
  movl    12(%ebp), %eax   # $eax has parameter, b
  leal    (,%ebx,4), %edx  # $edx = 4*i
```

Name: Student ID:

```
    addl    %edx, %eax      # $eax = &b[i]
    movl  (%eax), %eax      # $eax = b[i]
    addl  %eax, -16(%ebp)   # sum += b[i]
    incl  %ebx              #  i++
LCHECK:
    cmpl  16(%ebp), %ebx    # i < n
    jl    LOOP
    movl  -16(%ebp), %eax   # store return value
    movl  -12(%ebp), %edi
    movl -8(%ebp), %esi
    movl -4(%ebp), %ebx
    movl  %ebp, %esp
    popl %ebp
    ret
```

Name: Student ID:

6. (15 points) Ordered processing with child

The following code prints out two strings into stdout. However, as we learned in class, we do not know which string will be printed out first in this program. Your task is to modify the code such that the child process ***always*** prints out its string before the parent prints out its own string. After printing out the string, the child process has to call infiloop() in the end. You can use any C runtime library functions or system calls. Do not worry about header files. You do not need to worry about performance, but don't depend on any heuristics (e.g., the child will start at least within n seconds). Hint: use a signal.

```
void infiloop()
{
   while (1) sleep(1);
}

int main(void)
{
    if (fork() == 0) {
      printf("I am a child\n");
      infiloop();
    } else {
      printf("I am a parent\n");
    }
    return 0;
}
```

Name: Student ID:

Your modified code should be here:

```
void infiloop()
{
   while (1) sleep(1);
}
static int isChildReady = 0;
void SetChildReadyFlag(int signum)
{
    isChildReady = 1;
}

int main(void)
{
    /* any reasonable signal number  is fine */
    signal(SIGINT, SetChildReadyFlag);

    if (fork() == 0) {
      printf("I am a child\n");
      /* send the signal to parent process */
      kill(getppid(), SIGINT);
      infiloop();
    } else {
     /*  any  reasonable  code  that  waits  for  the  flag
         change is fine */
      while (!isChildReady) sleep(0);
      printf("I am a parent\n");
    }
    return 0;
}
```