Andrew login ID:______________________________________

Full Name:______________________________________

Section:______________________________________

15-213/18-243, Spring 2011

Exam 2

Thursday, April 21, 2011 v2

Instructions:

- Make sure that your exam is not missing any sheets, then write your Andrew login ID, full name, and section on the front.
- This exam is closed book and closed notes. A notes sheet is attached to the back.
- Write your answers in the space provided below the problem. If you make a mess, clearly indicate your final answer.
- The exam has a maximum score of 100 points.
- The problems are of varying difficulty. The point value of each problem is indicated. Good luck!

1 (12):
2 (15):
3 (20):
4 (18):
5 (10):
6 (15):
7 (10):
TOTAL (100):

Problem 1. (12 points):

Multiple choice.

Write the correct answer for each question in the following table:

1	2	3	4	5	6	7	8	9	10
11	12	13 X	14 X	15 X	16 X	17 X	18 X	19 X	20 X

1. What kind of process can be reaped?

   (a) Exited
   (b) Running
   (c) Stopped
   (d) Both (a) and (c)

2. Which of the following functions will always return exactly once?

   (a) `exec`
   (b) `exit`
   (c) `fork`
   (d) None of the above

3. Given an arbitrary malloc implementation that does not coalesce and a second implementation that is identical except it does coalesce, which of the following is true about their utilization scores on an arbitrary trace? (You may assume the first implementation stores enough information to make coalescing possible, so the only difference is that the second implementation actually performs the coalescing.)

   (a) The coalescing malloc will definitely get a better utilization score.
   (b) The coalescing malloc might get a better utilization score and might get the same utilization score, but it cannot get a worse utilization score.
   (c) The coalescing malloc might get a better utilization score, might get the same utilization score, and might get a worse utilization score.
   (d) The coalescing malloc will definitely get a worse utilization score.

4. Which of the following is a reason next-fit might perform better than first-fit?

   (a) If a large number of small blocks are stored at the beginning of the free list, next-fit avoids walking through those small blocks upon every allocation.
   (b) First-fit requires a traversal of the entire free list, but next-fit does not.
   (c) First-first requires that both allocated and unallocated blocks be examined, and next-fit examines only free blocks.
   (d) Next-fit is an approximation of best-fit, so it reduces internal fragmentation compared to first-fit.

5. How much virtual memory can be addressed by a 32-bit system?

   (a) 2GB
   (b) 4GB
   (c) 8GB
   (d) 240TB

6. Which of the following is a reason why a virtual memory translation would fault?

   (a) Page is not present
   (b) Page is read only
   (c) Page is empty
   (d) All of the above

7. How many bits are needed for the Virtual Page Offset if page size is 5000 bytes?

   (a) 10
   (b) 11
   (c) 12
   (d) 13

8. Which of the following is preserved across exec?

   (a) Signal handlers
   (b) Blocked signals
   (c) a and b
   (d) Neither

9. In what section of an ELF binary are initialized variables located?

   (a) .symtab
   (b) .data
   (c) .bss
   (d) .text

10. What does the call `dup2(oldfd, newfd);` do?

    (a) `newfd` and `oldfd` now both refer to `oldfd`'s entry in the open file table.
    (b) A copy of `oldfd`'s open file table entry is made, and `newfd` points to the copy.
    (c) A copy of the file `oldfd` is pointing to is made on the filesystem. The file is then opened, and `newfd` points to that open file entry.
    (d) The numerical value in oldfd is copied into newfd. No changes are made in the system.

11. Which of the following are shared between a parent and child process immediately following a fork?

    (a) Writeable physical memory
    (b) File descriptor tables
    (c) Instruction pointer
    (d) Open file structs

12. Which signals cannot be handled by the process?

    (a) SIGTSTP
    (b) SIGKILL
    (c) SIGTERM
    (d) All of the Above

Problem 2. (15 points):

Process control.

Consider the following code sample. You may assume that no call to `fork`, `exec`, `wait`, or `printf` will ever fail, and that stdout is immediately flushed following every call to `printf`.

```
int global_x = 0;

int main(int argc, char *argv[])
{
    global_x = 17;

    /* Assume fork never fails */
    if(!fork()) {
        global_x++;
        printf("Child: %d\n", global_x);
    }
    else {
        wait(NULL);
        global_x--;
        printf("Parent: %d\n", global_x);
    }

    return 0;
}
```

A. What is printed by this program?

B. How might the output change if we removed the call to `wait`? *Note: Keep your answer short, you will be penalized for excessively long answers.*

Now, consider the following two code samples. Again, you may assume that none of the previously mentioned function calls can fail. You should also note that the source for `my_child` follows the source for the invoked program.

```
int global_x = 0;

int main(int argc, char *argv[])
{
    global_x = 17;

    /* Assume fork never fails */
    if(!fork()) {
        global_x++;
        /* Assume exec never fails */
        execl("./my_child", "./my_child", NULL);
        printf("Child finished\n");
    }
    else {
        wait(NULL);
        global_x--;
        printf("Parent: %d\n", global_x);
    }

    return 0;
}
```

Code Listing for my_child:

```
int global_x;

int main(int argc, char *argv[])
{
    printf("Child: %d\n", global_x);
    return 0;
}
```

C. Is the output for this program the same as for the previous? Why or why not? *Note: Again, your answer to this question should be short.*

Problem 3. (20 points):

Dynamic memory allocation.

In this question, we will consider the utilization score of various malloc implementations on the following code:

```
#define N 32

void *pointers[N];
int i;

for (i = 0; i < N; i++) {
    pointers[i] = malloc(4);
}
for (i = 0; i < N; i++) {
    free(pointers[i]);
}
for (i = 0; i < N; i++) {
    pointers[i] = malloc(56);
}
```

A. Consider a malloc implementation that uses an implicit list with headers of size 8 bytes and no footers. In order to keep payloads aligned to 8 bytes, every block is always constrained to have size a multiple of 8. The header of each block stores the size of the block, and since the 3 lowest order bits are guaranteed to be 0, the lowest order bit is used to store whether the block is allocated or free. A first-fit allocation policy is used. If no unallocated block of a large enough size to service the request is found, `sbrk` is called for the smallest multiple of 8 that can service the request. No coalescing or block splitting is done. NOTE: You do NOT need to simplify any mathematical expressions. Your final answer may include multilpliations, additions, and divisions.

   1. After the given code sample is run, how many total bytes have been requested from `sbrk`?

   2. How many of those bytes are used for currently allocated blocks, including internal fragmentation and header information?

   3. How many of those bytes are used to store free blocks, including header information?

   4. Give the fraction of the total number of bytes requested by the user by the end of the trace (not including calls to `malloc` that have subsequently been freed) over total number of bytes allocated by `sbrk`. You do not need to simplify the fraction.

B. Consider another malloc implementation that never calls `sbrk` for a size less than 64 bytes. In every other way the implementation is identical to the implementation in question A. Note that since no block splitting is done, this means the size of each block, including the header, will always be at least 64 bytes. Again, there is no need to simplify mathematical expressions.

1. After the given code sample is run, how many total bytes have been requested from `sbrk`?

2. How many of those bytes are used for currently allocated blocks, including internal fragmentation and header information?

3. How many of those bytes are used to store free blocks, including header information?

4. Give the fraction of the total number of bytes requested by the user by the end of the trace (not including calls to `malloc` that have subsequently been freed) over total number of bytes allocated by `sbrk`. You do not need to simplify the fraction.

Problem 4. (18 points):

Virtual Memory.

Consider a 32-bit system with a page size of 4KB. A certain kernel designer wishes to analyze the merits of using 2-level page tables.

A. Consider the above system utilizing a 2-level page table (the page directory and page tables are 1 page in size).

   (a) How many entries are there in the page directory?

   (b) How many entries are there in the page table?

   (c) If a process were to use all 4GB of available virtual memory, how many page tables would be in use?

   (d) How much memory would the page directory and page tables occupy?

B. Consider the same system but now utilizing a 1-level page table.

   (a) How many entries are there in the page table?

   (b) How much memory would the page table occupy?

C. Why would the kernel designer opt for a 2-level page table when a full 2-level page table takes up more memory than a full 1-level page table?

Problem 5. (10 points):

Assume a System that has

1. A two way set associative TLB
2. A TLB with 8 total entries
3. 2^8 byte page size
4. 2^{16} bytes of virtual memory
5. one (or more) boats

TLB			
Index	Tag	Frame Number	Valid
0	0x13	0x30	1
	0x34	0x58	0
1	0x1F	0x80	0
	0x2A	0x72	1
2	0x1F	0x95	1
	0x20	0xAA	0
3	0x3F	0x20	1
	0x3E	0xFF	0

A. Use the TLB to fill in the table. Strike out anything that you don't have enough information to fill in.

Virtual Address	Physical Address
0x7E85	
0xD301	
	0x3020
0xD040	
	0x5830

Problem 6. (12 points):

Linking.

For each of the following code snippets, write down all symbols in the resulting object files from compilation. Write whether it is a weak global, strong global, or local variable, and what section of the final compiled ELF binary the variable will go into. Fill in the value if you have enough information to determine the value.

A.

main.c

```
int x = 5;
int y;
static int z = 3;

int main() {
  printf("%x\n", z());
  x = 0xdeadbeef;
  printf("%x\n", z());
}
```

foo.c

```
short x;
int y = 0x12345678;

int z() {
  return y;
}
```

File	Symbol	Strength and scope	Value	ELF section
main.o				
foo.o				

Problem 7. (10 points):

I/O

Consider the following code. Assume all system calls succeed, and that calls to read() and write() are atomic with respect to each other.

The contents of `foo.txt` are “ABCDEFG”.

A.
```
void read_and_print_one(int fd)
{
    char c;
    read(fd, &c, 1);
    printf("%c", c); fflush(stdout);
}

int main(int argc, char *argv[])
{
    int fd1 = open("foo.txt", O_RDONLY);
    int fd2 = open("foo.txt", O_RDONLY);
    read_and_print_one(fd1);
    read_and_print_one(fd2);

    if(!fork()) {
        read_and_print_one(fd2);
        read_and_print_one(fd2);
        close(fd2);
        fd2 = dup(fd1);
        read_and_print_one(fd2);
    }
    else {
        wait(NULL);
        read_and_print_one(fd1);
        read_and_print_one(fd2);
        printf("\n");
    }

    close(fd1);
    close(fd2);
    return 0;
}
```

Write out the output of this code, and the final contents of the file `foo.txt`.