University of Maryland College Park
Dept of Computer Science
CMSC216 Spring 2016
Midterm II

Last Name (PRINT): _____

First Name (PRINT): _____

University Directory ID (e.g., umcpturtle)_____

Lab TA (Circle One):

8am 0101 Andrej R	11am 0104 Swati S	3pm 0203 Saurabh K	1pm 0302 Gabriella F	1pm 0402 Andrew V
9am 0102 Andrej R	12pm 0201 Matthew E	4pm 0204 Saurabh K	3pm 0303 Andrew V	4pm 0403 Xuefang X
10am 0103 Swati S	1pm 0202 Matthew E	12pm 0301 Gabriella F	8am 0401 Aditya M	

I pledge on my honor that I have not given or received any unauthorized assistance on this examination.

Your signature: _____

Instructions

- If the answer to a question depends on the architecture involved, assume the question applies to behavior on linux.grace.umd.edu.
- This exam is a closed-book and closed-notes exam.
- Total point value is 200 points.
- The exam is a 75 minutes exam.
- Please use a pencil to complete the exam.
- WRITE NEATLY.
- Cheat sheets can be found at the end of the exam.
- There are **FOUR** problems in the exam.
- **Your code must be efficient.**
- **You don't need to use meaningful variable names; however, we expect good indentation.**
- **In any code you write, you should check for errors in library and system calls (e.g., fork).**

Grader Use Only

#1	Problem #1 (Miscellaneous)	(46)	
#2	Problem #2 (Processes)	(34)	
#3	Problem #3 (Linked Lists)	(60)	
#4	Problem #4 (Assembly)	(60)	
Total	Total	(200)	

Problem #1 (Miscellaneous)

1. (8 pts) What is the output of the following program?

```
#include <stdio.h>

int main() {
   unsigned char a = 0x3b, b = 0xc4;

   printf("%02x\n", a | b);
   printf("%02x\n", a & b);
   printf("%02x\n", a ^ b);
   printf("%02x\n", a << 2);

   return 0;
}
```

2. (8 pts) The following code compiles. Identify any errors in the code. If there are no errors write NONE.

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

void process(int *p) {
   p[0] = 10;
   printf("%d\n", *p);
   free(p);
}

int main() {
   int number, *k = &number, *m;

   scanf("%d", k);
   m = malloc(sizeof(int) * number);
   process(m);
   printf("%d", *m);
   free(m);
   free(k);

   return 0;
}
```

3. (8 pts) The file desk.c includes the file desk.h and desk.h includes the file temp.h. Write one or more makefile rules that generates the file desk.o. You do not need to use any makefile macros.

4. (4 pts) Suppose a function written in Y86 assembly language has allocated space for one local variable, call it **x**, and the function then pushes the contents of three registers onto the stack. Write a single instruction that places the contents of the local variable **x** into the %eax register. You must use the %esp register and not the %ebp register in the instruction.

5. (2 pts) The wait() system call reaps:

    a. The first process the parent created.
    b. The last process the parent created.
    c. Any process that has finished.
    d. Any process that has not finished.
    e. None of the above.

6. (2 pts) Suppose a program just executed scanf("%d", &x) where x is an integer variable. If we type CTRL-C (Ctrl key followed by C) what will take place? Circle all those that apply.

    a. A signal will be generated.
    b. The program will be terminated.
    c. The EOF character will be returned by scanf.
    d. A segmentation fault (segfault) will take place.
    e. None of the above.

7. (14 pts) Implement the function **free_array** with prototype shown below that frees the dynamically-allocated memory associated with each array element in the first parameter passed to the function. The function needs to handle the scenario where two or more array entries are pointing to the same dynamically-allocated memory (only one free can be applied to a memory location). It is OK to modify the original array as part of your implementation, but you MAY NOT define a new array in the function and you may not allocate more memory. If you do you will lose a significant number of points.

```
void free_array(void *a[], int length) {
```

Problem #2 (Processes)

Write a C program that reads a command (string) and creates a child that executes the command using execlp. For this problem:

1. There are only two processes: the parent and the child.
2. The command is a single string (e.g., pwd, bla) and it must be read using scanf.
3. The command will be executed by the child process the parent creates.
4. If the command cannot be executed (invalid command) or the child finished abnormally the parent (**NOT the child)** must display the message "Invalid command". Keep in mind that the child does not have any output statements. If the child executes the command successfully, the parent will not display any output and it will execute the **date** command by using execlp (the parent will become the **date** command).
5. You can assume a command has a maximum of 80 characters.
6. You do not need to show any #include statements.

Problem #3 (Linked Lists, Dynamic Memory Allocation)

The following structures and definitions will be used for the questions below. The Node structure represents a node in a doubly linked list where **next** points to the next node and **prev** to the previous node. A Linked_list structure keeps track of the actual head of the list and the size.

```
#define MAX_NAME_LENGTH 80

typedef struct hackathon {
   char name[MAX_NAME_LENGTH + 1];
   int participants;
} Hackathon;

typedef struct node {
   Hackathon *hackathon;
   struct node *next;
   struct node *prev;
} Node;

typedef struct list {
   Node *head;
   int size;
} Linked_list;
```

1. Define the function **create_list** that has the prototype below. It initializes the Linked_list pointer variable with the address of a Linked_list structure where **head** has been initialized to NULL and **size** to zero. The following code fragment illustrates how to use this function:

```
Linked_list *my_list;
create_list(&my_list);
```

```
void create_list(Linked_list **list);
```

2. Define the function **add** that has the prototype below. The function adds a **Hackathon** to the doubly linked list. The list must be kept sorted based on the number of participants (in increasing order).

```
void add(Linked_list *list, Hackathon new_hackathon);
```

For this problem:

1. The function must make a copy of the Hackathon parameter using dynamic memory allocation and add the copy to the list.
2. Your implementation must not be recursive.

IMPLEMENT YOUR CODE IN THE NEXT PAGE

PAGE FOR YOUR CODE

Problem #4 (Assembly)

Write the assembly function named **remove_char** on the next page. The function implements the C **remove_char** function written below.

- You only need to provide assembly code for the **remove_char** function.
- Do **not** provide assembly code for main.
- On the next page you will see assembly code representing the array.
- Your assembly code must work for any array (not just the one in the example).
- **Provide comments with your assembly code.**
- **Your code must be efficient.**
- The function must save and restore the base/frame pointer.
- Parameters must be passed on the stack.
- You may not modify the provided C function and must implement the version below.
- **You need to define and use the local variable (value).** That is, at the beginning of the **remove_char** function you need to reserve space on the stack for this local variable and you must use (read and write) the space.
- Use **%ebx** for the characters in the array **src, %edx** for the characters in the **result** array and **%ecx** for the character **to_remove**.
- Remember the ASCII value of the letter **A** is 65.
- **Your solution must be recursive otherwise you will lose a significant amount of the credit.**

```
int remove_char(char *src, char *result, char to_remove) {
   int value = 0;

   if (*src == '\0') {
      *result = '\0';
      return 0;
   }
   if (*src != to_remove) {
      *result = *src;
      result++;
   } else {
      value = 1;
   }

   return value + remove_char(++src, result, to_remove);
}

char src[81] = "ABACADEF";
char result[81];

int main() {
   printf("instances removed %d\n", remove_char(src, result, 'A'));
   printf("%s\n", result);

   return 0;
}
```

Output

```
instances removed 3
BCDEF
```

```
remove_char:

end_remove_char:

.align 4
src:
   .long 65
   .long 66
   .long 65
   .long 67
   .long 65
   .long 68
   .long 69
   .long 70
   .long 0
result:
```

EXTRA PAGE

CHEAT SHEETS

Standard IO Cheat Sheet

- FILE *fopen(const char *path, const char *mode);
- int fscanf(FILE *stream, const char *format, ...);
- char *fgets(char *s, int size, FILE *stream);
- int sscanf(const char *str, const char *format, ...);
- int fputs(const char *s, FILE *stream);
- int fprintf(FILE *stream, const char *format, ...);

Assembly Cheat Sheet

- Registers: %eax, %ecx, %edx, %ebx, %esi, %edi, %esp, %ebp
- Assembler Directives: .align, .long, .pos
- Data movement: irmovl, rrmovl, rmmovl, mrmovl
- Integer instructions: addl, subl, multl, divl, modl
- Branch instructions: jmp, jle, jl, je, jne, jge, jg
- Reading/Writing instructions: rdch, rdint, wrch, wrint
- Other: pushl, popl, call, ret, halt
- Ascii code for newline character: 0x0a
- Ascii code for space: 0x20

Process Cheat Sheet

- pid_t fork(void);
- pid_t wait(int *status);
- pid_t waitpid(pid_t pid, int *status, int options);
- int execvp(const char *file, char *const argv[]);
- int execlp(const char *file, const char *arg, ...);