CS 351 Spring 2018

Midterm Exam

Instructions:

- This exam is closed-book, closed-notes. Calculators are not permitted.
- For numbered, multiple-choice questions, fill your answer in the corresponding row on the "bubble" sheet.
- For problems that require a written solution (labeled with the prefix "WP"), write your answer in the space provided on the written solution sheet. Please write legibly and clearly indicate your final answer.
- Turn in the exam question packet, bubble sheet, and written solution sheet separately.
- Good luck!

Multiple Choice (24 points):

1. Which best describes the type of `p`, declared below?

```
char (*p[10])(int *);
```

   (a) a function which takes an int pointer and returns an array of ten pointers to chars
   (b) a function which takes an int pointer and returns a pointer to an array of ten chars
   (c) an array of 10 pointers to functions, each taking an int pointer and returning a char
   (d) a pointer to an array of 10 functions, each taking an int pointer and returning a char

2. Consider the following macro definition and variable declaration:

```
#define FOO(x) (x * x)
```

   What is the value of the expression `FOO(4 + 5)`?

   (a) 18
   (b) 26
   (c) 29
   (d) 81

3. Which of the following keywords can be used to create a "private" API in a C source file?

   (a) `static`
   (b) `extern`
   (c) `const`
   (d) `void`

4. What is wrong with the following structure declaration?

```
struct foo {
  void *val;
  struct foo *p, *q;
  struct foo x, y;
};
```

   (a) you can't refer to `struct foo` in its own definition
   (b) `x` and `y` cannot be properly allocated
   (c) `void *` isn't a valid type for an attribute
   (d) a `typedef` must be used in place of all `struct foo` references

5. Given the declaration `int i` within some C function, which of the following actions would be the biggest cause for concern?

   (a) returning the value of `&i`

   (b) calling another function with `&i` as a parameter

   (c) assigning the value of a global `int` variable to `i`

   (d) the subsequent declaration and initialization `void *p = &i`

6. Which action will *never* be taken following an abort (a form of synchronous exception)?

   (a) a different process is scheduled to execute

   (b) the process which generated the exception is terminated

   (c) the operating system shuts down (aka a "kernel panic")

   (d) the instruction generating the exception will be restarted

7. Which of the following statements about reentrant functions is false?

   (a) they can be safely interrupted and re-executed again from the start

   (b) preempted execution can be resumed after interruption without error

   (c) they are always inherently recursive

   (d) they are not permitted to access any global variables

8. Which of the following is *not* the responsibility of a typical shell program?

   (a) reaping terminated child processes

   (b) ensuring that each child process is a process group leader

   (c) forwarding `SIGINT` and `SIGTSTP` signals to foreground jobs

   (d) adopting descendant processes whose parents have terminated

9. Which of the following is retained across a successful call to `exec`?

   (a) the current stack frame

   (b) the value of the PC register

   (c) the pending signals vector

   (d) registered signal handlers

10. What is your favorite way of terminating a C program?

    (a) `exit(0)`

    (b) `return 0`

    (c) `*(int *)0 = 0`

    (d) `kill(SIGKILL, getpid())`

WP1. Memory Management (8 points):

Consider the following code, which contains a type definition and a function that uses it to dynamically allocate an *adjacency list* data structure, which consists of a gridlike arrangement of linked nodes.

```
typedef struct node node_t;

struct node {
  int data;
  node_t *right;
  node_t *down;
};

void alloc_adj_list(node_t **n, int height, int width) {
  int i, j;
  node_t *p, *q;
  *n = NULL;
  for (i=0; i<height; i++) {
    p = malloc(sizeof(node_t));
    p->right = NULL;
    for (j=0; j<width; j++) {
      q = malloc(sizeof(node_t));
      q->right = p->right;
      p->right = q;
    }
    p->down = *n;
    *n = p;
  }
}
```

The following call

```
node_t *p;
alloc_adj_list(&p, 5, 7);
```

will dynamically allocate a structure consisting of 5 downwards nodes, each with a chain of 7 nodes hanging off to the right.

Implement the function `void free_adj_list(node_t *n);`, which, when called with a pointer to an adjacency list structure (of arbitrary dimensions), will free *all* the nodes it contains. E.g., `free_adj_list(p)` will free the structure allocated above.

WP2. Process Trees (8 points):

For each of the following programs, (1) sketch the corresponding process tree — being sure to indicate any outputs and circle synchronization points, if they exist — and (2) write out *all* the distinct outputs that could be produced when it is executed.

A)
```
main() {
  for (int i=0; i<2; i++) {
    if (fork() == 0) {
      printf("%d", i);
    } else {
      wait(NULL);
      printf("%d", 3-i);
    }
  }
}
```

B)
```
main() {
  if (fork() == 0) {
    printf("0");
    for (int i=1; i<3; i++) {
      if (fork() == 0) {
        printf("%d", i);
        exit(0);
      }
    }
    printf("3");
  } else {
    wait(NULL);
    printf("4");
  }
}
```

WP3. Signal Handlers (8 points):

Consider the following program:

```
int counter = 0;

void handler (int sig) {
  counter++;
}

int main() {
  signal(SIGUSR1, handler);
  signal(SIGUSR2, handler);
  if (fork() == 0) {
    /* insert snippet here */
    exit(0);
  }
  wait(NULL);
  printf("%d\n", counter);
  return 0;
}
```

Replacing the comment in the above code with each of the snippets below, indicate *all* possible outputs of the program (i.e., the printed value of `counter`) and briefly explain why they may occur. Assume that no external signals are sent to the process. Note that `SIGUSR1` and `SIGUSR2` correspond to signal numbers 30 and 31, respectively.

A)
```
kill(getppid(), SIGUSR1);
kill(getppid(), SIGUSR1);
```

B)
```
kill(getppid(), SIGUSR1);
kill(getppid(), SIGUSR1);
kill(getppid(), SIGUSR1);
```

C)
```
kill(getppid(), SIGUSR2);
kill(getppid(), SIGUSR1);
```

D)
```
kill(getppid(), SIGUSR2);
kill(getppid(), SIGUSR1);
kill(getppid(), SIGUSR1);
kill(getppid(), SIGUSR1);
```

WP4. `runcommand` (8 points):

Implement the function `runcommand`, which takes an array of strings that represent a command suitable for use as the `argv` parameter to `execv`, and runs the command in its own child process.

The return value of `runcommand` will be one of:

- the exit status of the program, if it exits normally
- the value 255, if the command is invalid (e.g., the given `argv` specifies an invalid program)
- the value -1, if the command terminates abnormally (e.g., due to a segfault)