CS360 Final Exam - April 30, 2015 - James S. Plank

Question 1 - 16 points

In this question, please assume that **malloc()** is implemented as described in class, and that pointers are 32 bits. Also, assume that the **malloc()** calls are successful. Take a look at the following snippet of code to the right:

```
void a()
{
  int *p, *q, *r;

  p = (int *) malloc(7);
  q = (int *) malloc(84);

  r = p;
  q -= 2;
  p -= 2; /* XXX */
  *p += *q;
  free(r);
}
```

Please answer the following questions:

- A: What is the most likely value of *p after statement XXX is executed?
- B: What is the most likely value of *q after statement XXX is executed?
- C: I say "most likely" value of *p. Tell me another possible value of *p, and why it would have that value instead of your answer to A?
- D: Now, assume that the first **malloc()** call returned 0x5008, and that the second **malloc()** call returned 0x8008. Explain to me exactly why subsequent **malloc()** and **free()** calls are going to be problematic.
- E: Given your answers to A and B, suppose that the first **malloc()** call returned 0x5008. Tell me a return value of the second **malloc()** call that causes no problem to subsequent **malloc()** and **free()** calls. You don't need to tell me why -- just give me the return value.

Question 2 - 16 points

Your boss suspects that your co-worker, Del Nileppez is encrypting secrets in the output of his program, **del**. **Del** is a simple program that takes no command line arguments. It reads lines of text on standard input and prints lines of text on standard output. Your boss is going to do the following. Unbeknownst to anyone else in the company, the program **del** is going to be moved to **/home/boss/del**, and then **del** is going to be replaced by a program that you are going to write. When a user launches your program, you will create a second process that executes **/home/boss/del**. You will have **/home/boss/del** read standard input from the user, but every time that **/home/boss/del** prints a line on standard output, you are going to send that line to a server that will be running on **boss.evilcorp.com**, port 6666. Of course, to be surreptitious, you will also send the line of text to the user as well.

Write the code. You don't need threads for this, and you only need to call **fork()** and **pipe()** once. You may assume that lines of text are 1000 characters or less. I advocate using **fdopen()** to make I/O easier.

Question 3 - 16 points

On the next page is a program called **q3.c.** On Page 4, there are 7 different runs of the program. You are to answer *all* possible outputs of each run of the program. Choose your answers from the multiple choice answers. They are in alphabetical order so that you can find the correct answer(s) easily. If the program is deadlocked, you may assume that it will exit immediately.

```
struct ts {                             /* This is q3.c (skipping includes) */
  pthread_mutex_t *lock;
  pthread_cond_t *cv1, *cv2;
  char *command;
  int id;
};

void *thread(void *arg)
{
  struct ts *T;
  int i;

  T = (struct ts *) arg;

  pthread_mutex_lock(T->lock);
  for (i = 0; T->command[i] != '\0'; i++) {
    if (T->command[i] == 'S') {
      pthread_mutex_unlock(T->lock);
      sleep(1);
      pthread_mutex_lock(T->lock);
    }
    if (T->command[i] == 'X') pthread_cond_wait(T->cv1, T->lock);
    if (T->command[i] == 'Y') pthread_cond_wait(T->cv2, T->lock);
    if (T->command[i] == 'A') pthread_cond_signal(T->cv1);
    if (T->command[i] == 'B') pthread_cond_signal(T->cv2);
    if (T->command[i] == 'P') { printf("%c ", T->id); fflush(stdout); }
    if (T->command[i] == 'E') { printf("\n"); exit(1); }
  }
  pthread_mutex_unlock(T->lock);
  return NULL;
}

main(int argc, char **argv)
{
  pthread_mutex_t lock;
  pthread_cond_t cv1, cv2;
  int i;
  pthread_t *tids;
  struct ts *T;
  void *dummy;

  pthread_mutex_init(&lock, NULL);
  pthread_cond_init(&cv1, NULL);
  pthread_cond_init(&cv2, NULL);

  T = (struct ts *) malloc(sizeof(struct ts)*(argc-1));
  tids = (pthread_t *) malloc(sizeof(pthread_t)*(argc-1));
  for (i = 0; i < argc-1; i++) {
    T[i].id = 'A' + i;
    T[i].lock = &lock;
    T[i].cv1 = &cv1;
    T[i].cv2 = &cv2;
    T[i].command = argv[i+1];
    pthread_create(tids+i, NULL, thread, (void *) (T+i));
  }
  for (i = 0; i < argc-1; i++) pthread_join(tids[i], &dummy);
  printf("\n");
}
```

Question 4 - 16 points

Below is a program called **q4.c.** On the next page, there are 8 different runs of the program, followed by 16 multiple choice statements. For each run of the program, circle all statements that are true. There will be four true statements per run.

You should assume that PIPE_BUF is 16K (16,384) bytes.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/uio.h>

#define BSIZE 16000

main(int argc, char **argv)
{
  int p[2];
  int id, i, j;
  char s[BSIZE];
  FILE *f;

  pipe(p);
  id = fork();

  if (id == 0) {
    id = 1;
    f = fopen("OC.txt", "w");
  } else {
    id = 2;
    f = fopen("OP.txt", "w");
  }

  for (i = 0; argv[id][i] != '\0'; i++) {
    if (argv[id][i] == 'A') close(p[0]);
    if (argv[id][i] == 'B') close(p[1]);
    if (argv[id][i] == 'C') {
      for (j = 0; j < BSIZE-2; j++) s[j] = 'A';
      s[BSIZE-1] = '\n';
      write(p[1], s, BSIZE);
    }
    if (argv[id][i] == 'D') {
      j = read(p[0], s, BSIZE);
      if (j != BSIZE) exit(1);
      fwrite(s, 1, BSIZE, f);
      fflush(f);
    }
    if (argv[id][i] == 'S') sleep(1);
  }
  exit(0);
}
```

Useful Prototypes

```
FILE *fdopen(int fd, char *mode);
FILE *fopen(char *filename, char *mode);
char *fgets(char *buf, int sz, FILE *f);
char *fputs(char *buf, FILE *f);
int accept_connection(int fd);
int dup2(int fd1, int fd2);
int execl(char *path, char *arg0, ... NULL);
int execv(char *path, char **argv);
int fflush(FILE *f);
int fork();
int fread(void *buf, int sz, int n, FILE *f);
int fwrite(void *buf, int sz, int n, FILE *f);
int pipe(int filedes[2]);
int read(int fd, void *buf, int size);
int request_connection(char *hn, int port);
int serve_socket(int port);
int wait(int *stat_loc);
int write(int fd, void *buf, int size);
```

Question 3 - The runs of q3

- Run 1: `q3 PP PP`
- Run 2: `q3 PX PX PE`
- Run 3: `q3 XYP SAP SSBP`
- Run 4: `q3 XYP SASPBP`
- Run 5: `q3 XP PASE`
- Run 6: `q3 PX SPE`
- Run 7: `q3 XP SPAB YP`

The answers:

a: `A`
b: `A A`
c: `A A B B`
d: `A B`
e: `A B A B`
f: `A B B`
g: `A B B A`
h: `A B C`
i: `A C`
j: `A C B`
k: `B`
l: `B A`
m: `B A A B`
n: `B A B`
o: `B A B A`
p: `B A C`
q: `B B`
r: `B B A`
s: `B B A A`
t: `B C`
u: `B C A`
v: `C`
w: `C A`
x: `C A B`
y: `C B`
z: `C B A`

Question 4 - The runs of q4

- Run 1: `q4 C DD`
- Run 2: `q4 ACCC BD`
- Run 3: `q4 SD CCBDD`
- Run 4: `q4 CD SCD`
- Run 5: `q4 BDD C`
- Run 6: `q4 CCCC DD`
- Run 7: `q4 CC D`
- Run 8: `q4 BD ACSC`

The statements:

a: The child process exits with a return code of 0.
b: The child process exits with a return code of 1.
c: The child process never exits.
d: The child process exits because of SIGPIPE.

e: The parent process exits with a return code of 0.
f: The parent process exits with a return code of 1.
g: The parent process never exits.
h: The parent process exits because of SIGPIPE.

i: When the program is done, **OC.txt** has 0 characters.
j: When the program is done, **OC.txt** has 16,000 characters.
k: When the program is done, **OC.txt** has 32,000 characters.
l: When the program is done, **OC.txt** has something else.

m: When the program is done, **OP.txt** has 0 characters.
n: When the program is done, **OP.txt** has 16,000 characters.
o: When the program is done, **OP.txt** has 32,000 characters.
p: When the program is done, **OP.txt** has something else.

Name: _____ **Email:** _____ **Major:** _____

QUESTION 1:

A: _____

B: _____

E: _____

C:

D:

QUESTION 3: Circle all that apply:

Run 1:	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
Run 2:	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
Run 3:	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
Run 4:	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
Run 5:	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
Run 6:	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
Run 7:	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z

QUESTION 4: Circle all that apply:

Run 1:	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p
Run 2:	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p
Run 3:	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p
Run 4:	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p
Run 5:	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p
Run 6:	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p
Run 7:	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p
Run 8:	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p

Name: _____ Email: _____ Major: _____

QUESTION 2:

Don't bother with include files.