CS5600, Cheng Tan

SAMPLE MIDTERM EXAM (09/26/2021)

Note:
- Questions include material that we did not cover, and there is material that we did cover that is not represented by these questions.
- The exam will be closed book.
- The exams will cover material from lectures, labs, homework, readings, and any other taught material.
- All material is fair game for exams.

I. C and system calls (20 points)

1. [6 points] Consider the following program; read it carefully:

```
#include <stdio.h>
void func(int* p) {
    int q = 2;
    p = &q;
    *p = 5;
}
int main() {
    int a = 9;
    func(&a);
    printf("%d\n", a);
    return 0;
}
```

What does this program print?

2. [4 points] Answer questions:

Which system call that we have studied creates processes?
State the system call.

You want to learn about a system call, mmap(). What command do you issue at the shell prompt on our devbox (the Linux virtual machine) to read the system manual pages for mmap()?
State the command.

3. [10 points] Consider the program below, which makes use of fork() and exec(). As a reminder, the Unix command echo outputs its arguments.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/wait.h>
#include <errno.h>
int main()
{
    int rc = fork();
    if (rc < 0) {
        fprintf(stderr, "fork: %s\n", strerror(errno));
        exit(1);
    } else if (rc == 0) {
        char* argv[3];
        argv[0] = "echo";
        argv[1] = "xyz";
        argv[2] = NULL; // tells execvp() that there are no more arguments
        // below, execvp() is a variant of exec()
        if (execvp(argv[0], argv) < 0)
            fprintf(stderr, "exec: %s\n", strerror(errno));
        printf("abc");
    } else {
        wait(NULL);
        printf("def");
    }
    return 0;
}
```

What does this program print?

II. Processes and Shell (20 points)

4. [10 points] Consider the program below,

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

uint64_t f(uint64_t* ptr);
uint64_t g(uint64_t a);
uint64_t* q;

int main(void)
{
    uint64_t x = 0;
    uint64_t arg = 8;

    x = f(&arg);

    printf("x: %lu\n", x);
    printf("dereference q: %lu\n", *q);

    return 0;
}

uint64_t f(uint64_t* ptr)
{
    uint64_t x = 0;
    x = g(*ptr);
    return x + 1;
}

uint64_t g(uint64_t a)
{
    uint64_t x = 2*a;
    q = &x; // <-- THIS IS AN ERROR (AKA BUG)
    return x;
}
```

Why does the comment say "q = &x" is a bug?
Explain in 2-3 sentences.

What are the outputs of this program?
(hint: there might be multiple possibilities, you only need to give one answer.)

5. [10 points] What do the following shell commands do?

```
$ cat students.txt | shuf -n 1
```

Explain the shell command in 1-2 sentences.

```
$ ls > files; shuf -n 1 < files > tmp1
```

Explain the shell command in 1-2 sentences.

```
$ :(){ : | : & }; :
```

Explain the shell command in 3-5 sentences.

III. Concurrency (20 points)

6. [6 points] Let `cv` be a condition variable, and let `mutex` be a mutex. Assume that there are only two threads, and a single CPU. Consider this pattern:

```
acquire(&mutex);
if (not_ready_to_proceed()) {
    wait(&mutex, &cv);
}
release(&mutex);
```

Under the above assumptions, when is this pattern correct? Follow the concurrency commandments. Your answer should not be longer than one sentence.

7. [14 points] Below are pseudocode.

a) [4 points] Read the code below:

```
int x;

int main(int argc, char** argv) {

    tid tid1 = thread_create(f, NULL);
    tid tid2 = thread_create(g, NULL);

    thread_join(tid1);
    thread_join(tid2);

    printf("%d\n", x);
}

void f() {
    x = 1;
    thread_exit();
}

void g() {
    x = 2;
    thread_exit();
}
```

What are possible values of x after A has executed f() and B has executed g()?

b) [4 points] If f() and g() are now defined as follows:

```
int y = 12;

f() { x = y + 1; }
g() { y = y * 2; }
```

What are the possible values of x?

c. [6 points] If f() and g() are now defined as follows:

```
int x = 0;

f() { x = x + 1; }
g() { x = x + 2; }
```

What are the possible values of x?

IV. Virtual Memory (20 points)

8. [10 points] Consider a byte-addressed processor architecture with 30-bit virtual addresses. In this architecture, the memory management unit (MMU) expects a three-level page table structure: the upper 7 bits of an address determine the index into the first-level page table, the next 7 bits determine the index in the second-level page table, the next 7 bits determine the index in the third-level page table, and the bottom 9 bits determine the offset.

How many entries are in a first-level page table? Justify, in a single sentence.

How many entries are in a third-level page table? Justify, in a single sentence.

What is the page size on this machine? Justify, in a single sentence.

What is the maximum number of virtual pages per process? Justify, in a single sentence.

9. [10 points] In this problem, you will describe how the implementation of malloc() can exploit paging so that the system (as a whole) can detect certain kinds of out of bound accesses; an out of bound access is when a process references memory that is outside an allocated range. In this problem we focus on overruns. Consider this code:

```
int *a = malloc(sizeof(int) * 100); /* allocates space for 100 ints */
a[0] = 5; /* This is a legal memory reference */
a[99] = 5; /* This is also a legal memory reference */
a[100] = 6; /* This is an overrun, and is an illegal memory reference. */
```

When the above executes, the process would ideally page fault as a result of an illegal memory reference, at which point the kernel would end the process.

Assume that malloc() is a system call, so its implementation is inside the operating system, and thus can manipulate the virtual address space of the process.

Describe how the implementation of malloc() can arrange for page faults when there are overruns like the one above. Do not write more than three sentences.

V. Labs (20 points)

10. [10 points] In Lab1, file "queue.h" defines the node as follows:

```
typedef struct node_t {
    struct node_t *next;
    char data[128];
} node_t;
```

[2 points] For this implementation, what's the maximum string length in this node?

[8 points] How can we support strings with no length limits?
Describe your node struct (in C code), and describe what you have to do during enqueue() and dequeue() (in English).
Note: increasing 128 to 256, or any other large numbers, doesn’t work (which is also a waste memory!)

11. [10 points] In Lab2, ... [skip; since Lab2 hasn’t been released]

END OF MIDTERM