System Programming CSE25101 Fall 2017 Final Exam

Professor Sam H. Noh Dec. 12, 2017

1. (10 points) Define the following terms
   a. TLB

   b. working set

   c. thrashing

   d. context switch

   e. zombie process

2. (10 points) Describe what the following system call do.
   a. execve()

   b. kill()

   c. longjmp()

   d. exit()

   e. waitpid()

3. (10 points) Given
   int x = foo( ); int y = bar( ); unsigned ux = x; unsigned uy = y;
   State whether the following is TRUE or FALSE. If your answer is FALSE, describe the reason it can be FALSE.
   a. ux >> 3 = = x/8

   b. x & (x-1) != 0

   c. ux >= 0

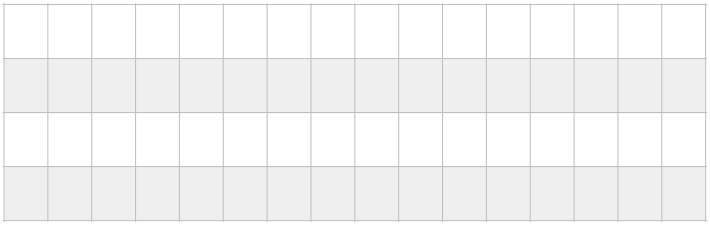
   d. x > y ==> -x < -y

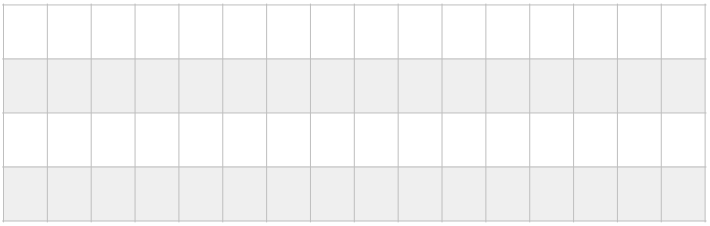
   e. x > 0 && y > 0 ==> x + y > 0

4. (8 points; 4 points each) Consider the following `struct` declaration:

```
typedef struct {
        char a[3];
        short b[3];
        long c;
        int d;
        int *e;
        char f
        long g;
        short h;
} EXAM;
```

   a. Show (using the boxes on the left) how the `struct` above would appear on the machine that we have been using in class and experiments. Label the bytes that belong to the various fields with their names and clearly make the end of the struct. Use x's to indicate bytes that are allocated in the `struct` but are not used. (A box below represents a byte.)
   b. Declare a new struct GOOD by rearranging the above fields in order to conserve the most space in the memory. Show how it would appear in memory using the boxes on the right below. Label the bytes that belong to the various fields with their names and clearly make the end of the struct. Use x's to indicate bytes that are allocated in the struct but are not used.





5. (8 points) Answer the following questions. You MUST show how you reached your solution.
   a. (2 points) Calculate the average time to access a sector on HDD that has the following characteristics: rotational rate of 12,000RPM, average seek time of 5 ms, and average number of sectors/track 300.

   b. (6 points) A potential weakness of SSDs is that the underlying flash memory can wear out. Assume that an SSD can guarantee 128 petabytes of writes before the drive wears out. Estimate the lifetime (in years) of this SSD for the following workloads.
      (a) The SSD is written to continuously at a rate of 470 MB/s, the average sequential write throughput for the device.

      (b) The SSD is written to continuously at a rate of 303 MB/s, the average random write throughput for the device.

      (c) The SSD is written to at a rate of 20 GB/day, the typical write rate for a typical user.

6. (10 points) The following is disassembled code of two functions fct1 and fct2. The left most number and right most label are for references only and are not part of the assembly code. Assuming that execution starts from Line 10, fill in the table below to trace instruction execution through to the point where the program returns to Line 11.

```
Disassembly of fct2( )
1       0000000000400540 <fct2>:
2       400540:          48 89 f8        mov %rdi, %rax
3       400543:          48 0f af c6     imul %rsi, %rax
4       400547:          c3              retq

Diassembly of fct1( )
5       0000000000400548 <fct1>
6       400548:          48 8d 77 01     lea 0x1(%rdi), %rsi
7       XXXX:            48 83 ef 01     sub $0x1, %rdi
8       400550:          e8 eb ff ff ff  callq 400540 <fct2>
9       400555:          f3 c3           repz retq
.
.
10      400560:          e8 e3 ff ff ff  callq 400548 <fct1>
11      400565:          48 89 c2        mov %rax, %rdx
```

Instruction		State values at beginning				
Line #	%rip	%rdi	%rsi	%rax	%rsp	*%rsp
10	0x400560	10	15	20	0x7fffffffe820	-
11						

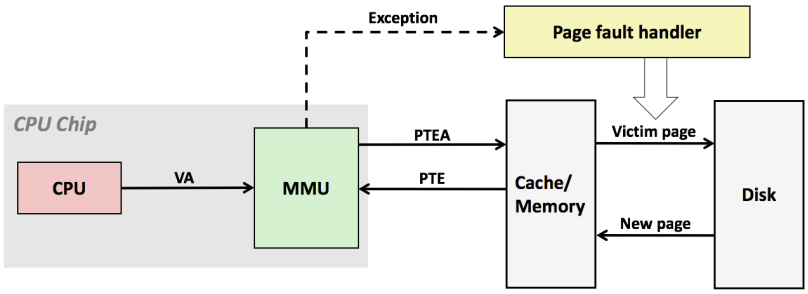
7. (10 points) In 2002, it was discovered that code supplied by Sun Microsystems to implement the XDR library, a widely used facility for sharing data structures between programs, had a security vulnerability arising from the fact that multiplication can overflow without any notice being given to the program.

   Code similar to that containing the vulnerability is shown on the right:

   a. (3 points) Explain what this code does.
   b. (3 points) Describe how the program is vulnerable.
   c. (4 points) Describe, using and example and specific code line numbers given above, under what specific conditions a malicious programmer could make use of the vulnerability of the program.

```
1  /*
2   * Illustration of code vulnerability similar to that found in
3   * Sun's XDR library.
4   */
5  void* copy_elements(void *ele_src[], int ele_cnt, size_t ele_size) {
6      /*
7       * Allocate buffer for ele_cnt objects, each of ele_size bytes
8       * and copy from locations designated by ele_src
9       */
10     void *result = malloc(ele_cnt * ele_size);
11     if (result == NULL)
12         /* malloc failed */
13         return NULL;
14     void *next = result;
15     int i;
16     for (i = 0; i < ele_cnt; i++) {
17         /* Copy object i to destination */
18         memcpy(next, ele_src[i], ele_size);
19         /* Move pointer to next memory region */
20         next += ele_size;
21     }
22     return result;
23 }
```

8. (24 points) The following diagram shows the address translation process. Use this diagram to answer the following questions.



a.(12 points) Number the arrows in the diagram in the order in which they are processed. For each of the number, describe what they are doing.

b.(4 points) In the diagram, the TLB is missing. Add the TLB in the diagram.

c.(8 points) With the TLB included, amend the answer to question a. to include the TLB. You need only describe the parts that are affected by the TLB.

9. (8 points; 4 points each) This problem tests your understanding of exceptional control flow in C programs. Answer the questions (a and b) below.

a. How many "hello" output lines does the program below print?

```
void doit() {
  fork();
  fork();
  printf("hello\n");
  return;
}

int main() {
  doit();
  printf("hello\n");
  exit(0);
}
```

b. What is the value of the counter variable that the program prints?

```
int counter = 1;

int main() {
      if (fork() == 0) {
            counter--;
            exit(0);
      }
      else {
            wait(NULL);
            counter++;
            printf("counter = %d\n", counter);
      }
      exit(0);
}
```

10. (20 points; 5 each) Each of the following programs have bugs in them. Identify the bugs. You need NOT correct the program.

a.

```
int *foo () {
   int val;
      …
   return &val;
}
```

b.

```
/* return y = Ax */
int *matvec(int **A, int *x) {
   int *y = malloc(N*sizeof(int));
   int i, j;

   for (i=0; i<N; i++)
      for (j=0; j<N; j++)
         y[i] += A[i][j]*x[j];
   return y;
}
```

c.

```
int *search(int *p, int val) {

   while (p && *p != val)
      p += sizeof(int);

   return p;
}
```

d.

```
int **p;
p = malloc(N*sizeof(int));

for (i=0; i<N; i++) {
   p[i] = malloc(M*sizeof(int));
}
```

11. (24 points) Consider an allocator that uses an implicit free list. Each memory block, either allocated or free, has a size that is a multiple of eight bytes. Thus, only the 29 higher order bits in the header and footer are needed to record the block size, which includes the header and footer and is represented in units of bytes. The usage of the remaining 3 lower order bits is as follows:
    - `bit 0` indicates the use of the current block: 1 for allocated, 0 for free.
    - `bit 1` indicates the use of the previous adjacent block: 1 for allocated, 0 for free.
    - `bit 2` is unused and is always set to be 0.

    a. (4 points) Given the description above, draw a depiction of how blocks would be allocated with this allocator. (An example diagram will suffice.)

    b. (4 points each; 20 points) Five helper routines are defined to facilitate the implementation of `free(void *p)`. The functionality of each routine is explained in the comment above the function definition. Fill in the body of the helper routines the code section label (A. or B. or C.) that implement the corresponding functionality correctly.

(a)
```
/* given a pointer p to an allocated block, i.e., p is a pointer returned by some
previous malloc()/realloc() call; returns the pointer to the header of the block*/
void * header(void* p) {
      void *ptr;
      _______;
      return ptr;
}
```
A. ptr=p-1 B. ptr=(void *)((int *)p-1) C. ptr=(void *)((int *)p-4)

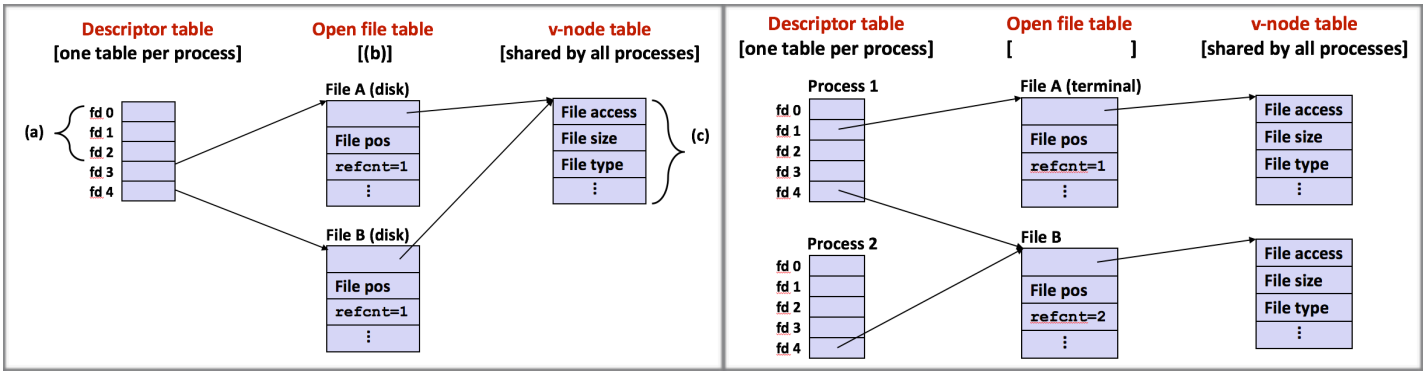
(b)
```
/* given a pointer to a valid block header or footer, returns the size of the block
*/
int size(void *hp) {
    int result;
    _______;
   return result;
}
```
A. result=(*hp)&(~7) B. result=((*(char *)hp)&(~5))<<2
C. result=(*(int *)hp)&(~7)

(c)
```
/* given a pointer p to an allocated block,i.e. p is
a pointer returned by some previous malloc()/realloc() call;
returns the pointer to the footer of the block*/
void * footer(void *p) {
      void *ptr;
      _______;
      return ptr;
}
```
A. ptr=p+size(header(p))-8 B. ptr=p+size(header(p))-4
C. ptr=(int *)p+size(header(p))-2

(d)
```
/* given a pointer to a valid block header or footer, returns the usage of the
current block,
1 for allocated, 0 for free */
int allocated(void *hp) {
      int result;
      ______;
      return result;
}
```
A. result=(*(int *)hp)&1 B. result=(*(int *hp)&0 C. result=(*(int *)hp)|1

(e)
```
/* given a pointer to a valid block header,
returns the pointer to the header of previous block in memory */
void * prev(void *hp) {
      void *ptr;
      ______;
      return ptr;
}
```
A. ptr = hp - size(hp) B. ptr = hp - size(hp-4) C. ptr = hp - size(hp-4) + 4

12. (24 points) Refer to the following figure for the questions below.



a. (2 points) fd stands for file descriptors. The first three file descriptors are assigned by default for every process that is opened. What are the names for these first three fds?

b. (2 points) Describe what should go in for (b) in the left diagram above.
c. (3 points) What is the system call that gives you back the information in (c).

d. (4 points) The diagram on the left depicts a situation where the same file is being shared. Describe how this situation can be reached.

e. (5 points) This depiction shows that a file is being shared via the v-node table. Can such sharing be done between processes? If you think no, explain why this is not possible. If you think yes, give a scenario where this can happen.

f. (6 points) Files can also be shared through the Open File Table as depicted on the right. Give a sequence of system calls whereupon you can reach this situation, that is, Process 1 and Process 2 sharing File B through the Open File Table.

g. (2 points) What is the purpose of `refcnt` in the Open File Table entries?

13. (10 points) The following byte code assigns the variable Global_value with a cookie value and executes the bang function in the IA-32 architecture. Fill in the blanks given the conditions.

```
/* Cookie : 0x45c230bd */
/* Global_value address : 0x804c1ec */
/* getbuf return address at address: 0x55683574  */
/* Local buffer starts at address:   0x55683550  */
/* Padding required:                 ①  bytes */

②          /* mov cookie, %eax */
③          /* mov %eax, global_value */
④          /* push bang */
c3         /* ret      */

⑤

/* return address */
⑥
```

Opcode	Instruction
a3	mov %eax, address
b8	mov address, %eax
c3	return from procedure
68	push address

```
0x08048bdb <bang+0>:    push   %ebp
0x08048bdc <bang+1>:    mov    %esp,%ebp
0x08048bde <bang+3>:    sub    $0x8,%esp
0x08048be1 <bang+6>:    mov    0x804c1ec,%eax
0x08048be6 <bang+11>:   cmp    0x804c1e4,%eax
                        ⋮
```

14. (24 points) Consider the code below.
    a. (4 points) Describe what a thread is in comparison with a process.
    b. (4 points) This code was discussed in class. Without going into the details of the particular functions that are called, give an outline of what the code is supposed to do.
    c. (6 points) Present a depiction of the address space of this program assuming that two Accept() function calls in the while statement are being serviced concurrently.
    d. (3 points) It was discussed in class that if

```
*connfdp = Accept(listenfd, (SA *) &clientaddr, &clientlen);
Pthread_create(&tid, NULL, thread, connfdp);
```

is replaced with

```
int connfd = Accept(listenfd, (SA *) &clientaddr, &clientlen);
Pthread_create(&tid, NULL, thread, &connfd);
```

it was possible that race could occur. Describe what a race is (as a general notion).
    e. (7 points) Given question d., describe exactly how a race can occur if the code is replaced.

```
int main(int argc, char **argv)
{
    int listenfd, *connfdp;
    socklen_t clientlen;
    struct sockaddr_storage clientaddr;
    pthread_t tid;

    listenfd = Open_listenfd(argv[1]);
    while (1) {
    clientlen=sizeof(struct sockaddr_storage);
    connfdp = Malloc(sizeof(int));
    *connfdp = Accept(listenfd, (SA *) &clientaddr, &clientlen);
    Pthread_create(&tid, NULL, thread, connfdp);
    }
    return 0;
}
```

```
/* Thread routine */
void *thread(void *vargp)
{
    int connfd = *((int *)vargp);
    Pthread_detach(pthread_self());
    Free(vargp);
    echo(connfd);
    Close(connfd);
    return NULL;
}
```