Spring Semester 2019

KAIST EE415/PD511

Operating Systems and System Programming for EE

Mid-term Exam

Name: _____

Student ID: _____

This exam is closed book and notes. Read the questions carefully and focus your answers on what has been asked. You are allowed to ask the instructor/TAs for help only in understanding the questions, in case you find them not completely clear. Be concise and precise in your answers and state clearly any assumption you may have made. You have 120 minutes (1:00 PM – 3:00 PM) to complete your exam. Be wise in managing your time. Good luck.

Question 1	/ 20
Question 2	/ 30
Question 3	/ 15
Question 4	/ 15
Question 5	/ 20

Total	/ 100

1. Process/thread address space (20 pts)

Please refer to the following C code for problems 1-(a) to 1-(c). Assume a 32-bit address space for this process (e.g., sizeof(void *) = 4).

```
char **ptr;

void *thread(void *vargp)
{
  int myid = (int) vargp;
  static int cnt = 0;

  printf("vargp=%p\n", vargp);          // Line A
  printf("[%d]: %s (svar=%d)\n",myid, ptr[myid], ++cnt);
}

int main()
{
  int i;
  pthread_t tid[2];
  char *msgs[2] = {"Hello from foo", "Hello from bar"};

  ptr = msgs;
  for (i = 0; i < 2; i++)
     pthread_create(&tid[i], NULL, thread,(void *)i);
  pthread_join(tid[0], NULL);
  pthread_join(tid[1], NULL);
  return 0;
}
```

(a) When you run this program, how many threads (e.g., max # of concurrently-running threads) of this process would you see in the system? (1 pt)

(b) Enumerate all values that “Line A” prints out. (2 pts)

(c) Which memory section do these variables (or strings) get allocated? Indicate whether the variables (or memory area) are shared by multiple threads or local to a single thread. Fill out the table below. Note that a memory section name can be one of code, stack, data or heap. The notation for a variable is *function_name::variable_name.* For example, variable i in main() is expressed as main::i, myid in thread() is thread::myid. (14 pts)

Variable notation	Memory section	Local vs. shared
main::i	stack	local
main::tid		
main:msgs[1]		
main::"hello from foo"		
::ptr		
thread::vargp		
thread::myid		
thread:cnt		

(d) Write all possible value(s) of thread::cnt during the execution of this process. (3 pts)

2. Understanding locks (30 pts)

(a) One way to implement a lock is to disable interrupts at entering a critical section and enabling them again at leaving the section. Explain two drawbacks of this implementation. (4 pts)

(b) The textbook says one can easily implement a lock using the atomic test-and-set instruction (logical operation described in C code below). Unfortunately, Bori (a poodle who does not believe in locks) licked to remove the important code piece below. Please recover the function lock() using TestAndSet(). (5 pts)

```
1  int TestAndSet(int *old_ptr, int new) {
2      int old = *old_ptr; // fetch old value at old_ptr
3      *old_ptr = new;     // store 'new' into old_ptr
4      return old;         // return the old value
5  }
```

```
1  typedef struct __lock_t {
2      int flag;
3  } lock_t;
4
5  void init(lock_t *lock) {
6      // 0: lock is available, 1: lock is held
7      lock->flag = 0;
8  }
9
10 void lock(lock_t *lock) {
11     Bori ripped this code snippet.
12
13 }
14
15 void unlock(lock_t *lock) {
16     lock->flag = 0;
17 }
```

(c) After recovering the code above, you find two problems in terms of efficiency and fairness. Explain each problem with an example scenario. (4 pts)

(d) Fortunately, the textbook suggests the following code to overcome the problem in (c). But after studying this code, you realize that it is not entirely correct. Explain the scenario where threads using this implementation would malfunction. (5 pts)

```
1  typedef struct __lock_t {
2      int flag;
3      int guard;
4      queue_t *q;
5  } lock_t;
6
7  void lock_init(lock_t *m) {
8      m->flag  = 0;
9      m->guard = 0;
10     queue_init(m->q);
11 }
12
13 void lock(lock_t *m) {
14     while (TestAndSet(&m->guard, 1) == 1)
15         ; //acquire guard lock by spinning
16     if (m->flag == 0) {
17         m->flag = 1; // lock is acquired
18         m->guard = 0;
19     } else {
20         queue_add(m->q, gettid());
21         m->guard = 0;
22         park();
23     }
24 }
25
26 void unlock(lock_t *m) {
27     while (TestAndSet(&m->guard, 1) == 1)
28         ; //acquire guard lock by spinning
29     if (queue_empty(m->q))
30         m->flag = 0; // let go of lock; no one wants it
31     else
32         unpark(queue_remove(m->q)); // hold lock
33                                     // (for next thread!)
34     m->guard = 0;
35 }
```

(e) The following code implements a classic synchronization problem called producer-and-consumer with a single shared buffer. After studying the code, you realize there is a problem with the code. Explain one scenario where one producer thread and two consumer threads would malfunction. (Note: count is initialized as 0. mutex and cond are a mutex variable (initialized as unlocked) and a condition variable, respectively. put(i) produces an item at index i and get() consumes a valid item and returns its index) (5 pts)

```
3
4   void *producer(void *arg) {
5       int i;
6       for (i = 0; i < loops; i++) {
7           Pthread_mutex_lock(&mutex);            // p1
8           while (count == 1)                     // p2
9               Pthread_cond_wait(&cond, &mutex); // p3
10          put(i);                                // p4
11          Pthread_cond_signal(&cond);            // p5
12          Pthread_mutex_unlock(&mutex);          // p6
13      }
14  }
15
16  void *consumer(void *arg) {
17      int i;
18      for (i = 0; i < loops; i++) {
19          Pthread_mutex_lock(&mutex);            // c1
20          while (count == 0)                     // c2
21              Pthread_cond_wait(&cond, &mutex); // c3
22          int tmp = get();                       // c4
23          Pthread_cond_signal(&cond);            // c5
24          Pthread_mutex_unlock(&mutex);          // c6
25          printf("%d\n", tmp);
26      }
27  }
```

(f) Suggest a fix to the code in (e). (5 pts)

(g) In class, professor emphasized that one should be careful with the usage of Pthread_cond_signal() and Pthread_cond_wait(). Please describe two de-facto rules you need to observe when you write the code with these functions to avoid subtle race conditions that might arise. (2 pts)

3. CPU Scheduling (15 pts)

(a) We learned five process states in class. Explain “ready” and “terminated” states. (3 pts)

(b) What is convoy effect? Which scheduling policy suffers from it? (3 pts)

(c) Explain the Shortest Completion-to-Time First (SCTF) scheduling policy. (3 pts)

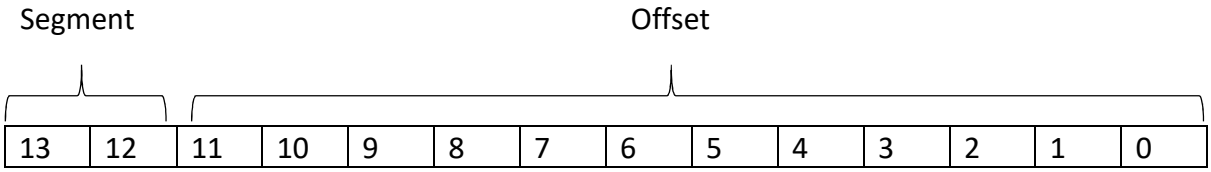
(d) Explain the strength and weakness of SCTF. (3 pts)

(e) Stride scheduling brings an advantage that eliminates the randomness of lottery scheduling. Describe two weaknesses of stride scheduling over lottery scheduling (3 pts)

4. Virtual memory (15 pts)

We're using segmentation to translate a virtual address into a physical address. The segment table is shown below. And a virtual address consists of 2 bits of a segment index and 12 bits of an offset as shown below. (a) to (c)

Segment	Base	Size	Grow Positive	Protection
Code	32K (=32768)	2048	1	Read/Execute
Heap	34K (=34816)	2048	1	Read/Write
Stack	28K (=28672)	2048	0	Read/Write



Virtual address space: program code: 0 to (2K-1), heap: 4K to (6K-1), Stack: 14K to (16K-1)

(a) What is the physical address of a virtual address 4100? (2 pts)

(b) What is the physical address of a virtual address, 11 1100 0000 0100? (3 pts)

(c) What is the main drawback of segmentation? Provide the two-word term for the problem and describe it in detail (3 pts)

(d) What is the role of TLB in paging? (2 pts)

(e) Recent x86 CPU supports 4KB, 4MB, and 1GB of page size. What is the main benefit of having a huge page size like 1GB? (3 pts)

(f) What is the main benefit of multi-level page table? (2 pts)

5. Argument Passing and System Calls in Pintos (20 pts)

While debugging my code for the Pintos project 2, I execute the following command line with the pintos OS.

echo -l foo bar a b c d e f g hhh hello

Note that echo is compiled from echo.c whose code is shown below. During the execution, I called hex_dump() at the start of syscall_hander() in userprog/syscall.c

echo.c

```
int main(int argc, char **argv)
{
 int i;
 for (i = 0; i < argc; i++)
    printf("%s ", argv[i]);
 printf("\n"); return 0;
}
```

```
bffffe50                                        09 00 00 00 |            ....|
bffffe60  XX XX XX XX YY YY YY YY-ZZ ZZ ZZ ZZ 4c 9a 04 08 |............L...|
bffffe70  01 00 00 00 a4 fe ff bf-04 00 00 00 a4 fe ff bf |................|
bffffe80  0d 00 00 00 a0 ff ff bf-a4 fe ff bf 17 9b 04 08 |................|
bffffe90  b9 a3 04 08 44 ff ff bf-57 9a 04 08 a4 fe ff bf |....D...W.......|
bffffea0  00 00 00 00 62 61 72 20-20 00 00 00 00 00 00 00 |....bar  .......|
bffffeb0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 |................|
bffffec0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 |................|
bffffed0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 |................|
bffffee0  00 00 00 00 a8 fe ff bf-04 00 00 00 01 00 00 00 |................|
bffffef0  00 00 00 00 00 00 00 00-03 00 00 00 32 9b 04 08 |............2...|
bfffff00  01 00 00 00 b9 a3 04 08-44 ff ff bf 00 00 00 00 |........D.......|
bfffff10  00 00 00 00 00 00 00 00-00 00 00 00 ef 84 04 08 |................|
bfffff20  b9 a3 04 08 44 ff ff bf-00 00 00 00 00 00 00 00 |....D...........|
bfffff30  00 00 00 00 00 00 00 00-00 00 00 00 c6 80 04 08 |................|
bfffff40  b9 a3 04 08 e4 ff ff bf-00 00 00 00 00 00 00 00 |................|
bfffff50  00 00 00 00 00 00 00 00-78 ff ff bf 00 00 00 00 |........x.......|
bfffff60  00 00 00 00 00 00 00 00-00 00 00 00 fe 80 04 08 |................|
bfffff70  00 00 00 00 fe 80 04 08-0d 00 00 00 a0 ff ff bf |................|
bfffff80  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 |................|
bfffff90  00 00 00 00 00 00 00 00-0d 00 00 00 a0 ff ff bf |................|
bfffffa0  d8 ff ff bf dd ff ff bf-e0 ff ff bf e4 ff ff bf |................|
bfffffb0  e8 ff ff bf ea ff ff bf-ec ff ff bf ee ff ff bf |................|
bfffffc0  f0 ff ff bf f2 ff ff bf-f4 ff ff bf f6 ff ff bf |................|
bfffffd0  fa ff ff bf 00 00 00 00-65 63 68 6f 00 2d 6c 00 |........echo.-l.|
bfffffe0  66 6f 6f 00 62 61 72 00-61 00 62 00 63 00 64 00 |foo.bar.a.b.c.d.|
bffffff0  65 00 66 00 67 00 68 68-68 00 68 65 6c 6c 6f 00 |e.f.g.hhh.hello.|
```

```
#define syscall3(NUMBER, ARG0, ARG1, ARG2)                  \
       ({                                                   \
          int retval;                                        \
          asm volatile                                       \
            ("pushl %[arg2]; pushl %[arg1]; pushl %[arg0]; "\
             "pushl %[number]; int $0x30; addl $16, %%esp"  \
               : "=a" (retval)                               \
               : [number] "i" (NUMBER),                      \
                 [arg0] “r" (ARG0),                          \
                 [arg1] “r" (ARG1),                          \
                 [arg2] “r" (ARG2)                           \
               : "memory");                                  \
          retval;                                            \
    })


  int
  write (int fd, void *buffer, unsigned size)
  {
    return syscall3(SYS_WRITE, fd, buffer, size);
  }
```

Note that SYS_WRITE is 9 in decimal. Also, note that 0x08049a4c is the return address to the caller of the write() system call.

(a) Fill in the 12 bytes starting at address 0xbffffe60 in the hex_dump()’ed output. Note that the leftmost XX/YY/ZZ below represents one byte in hexadecimal at address 0xbffffe60/0xbffffe64/0xbffffe68, respectively. (6 pts)

(b) What is the ASCII code for alphabet ‘h’ in the hexadecimal format (0xYY)? (2 pts)

(c) What is the value of argc in the main() function of echo.c? That is, write the result of printf("%d", argc); (2 pts)

(d) What is the memory address of argc in the main() function of echo.c? That is, write the result of printf("%p", &argc); (2 pts)

(e) What is the value of argv in the main() function of echo.c? That is, write the result of printf("%p", argv); (4 pts)

(f) What is the value of 'i' in the main() function of echo.c at the time of hex_dump()? That is, write the result of printf("%d", i); at the time of hex_dump() above (2 pts)