2025 Fall Introduction to Operating Systems

Midterm Exam

Total Points: 100

Name:

Student ID:

Attentions:

- The exam is closed-book, closed-slides, closed-notes
- No electronic devices are allowed
- Remember to write your name and student ID on the answer sheet

1. **[10 points]** Explain how the operating system kernel can regain control of the CPU from a running user-level program. Provide two distinct mechanisms.

   1) System call
   2) Interrupt

2. **[10 points]** Given a program executable file, how can we identify the system calls that will be carried out by the program?

   Look for syscall or software interrupt (e.g. int 0x80) instructions in the program code

3. **[10 points]** We can either create a child process or a new thread for concurrent program execution. Describe a key advantage of the multi-process approach over the multi-threaded approach.

   Memory isolation that prevent the memory corruption by peer programs

4. **[10 points]** (Continued from the previous question) Describe a key advantage of the multi-threaded approach over the multi-process approach.

   More lightweight

5. **[10 points]** The following C code calculates the factorial of `n`. Assume that the C compiler allocates a new stack frame for each function call and does not perform any code optimization. Illustrate what the stack memory looks like right before the first function return occurs after executing `factorial_v1(5)`.

```
int factorial_v1(int n) {
    if (n == 0)
        return 1;
    else
        return n * factorial(n - 1);
}
```

(Your drawing should illustrate the variables and their current values on the stack based on the code above.)

(Draw an arrow on the right side to indicate: upward means towards low address / downward means towards high address) 1 point for address order

n=0
n=1
n=2
n=3

n=4
n=5

6. **[10 points]** A student comes up with a different version of the factorial program as follows. Illustrate what the stack memory looks like right before the first function return occurs after executing `factorial_v2(5)`.

```
int factorial_v2(int n) {
    int result = 1;

    for (int i = 1; i <= n; ++i) {
        result *= i;
    }

    return result;
}
```

(Your drawing should illustrate the variables and their current values on the stack based on the code above.)

(Draw an arrow on the right side, indicating that moving upward is towards low address / moving downward is towards high address) 1 point for address order, 3 points for each value

n=5	result=120	i=6

7. **[10 points]** In a multi-threaded process, which of the following items are managed separately for each thread?

(a) Heap memory
(b) Stack memory
(c) CPU registers
(d) File descriptors
(e) Network socket descriptors

B, C

8. We would like to implement a critical section with the atomic `test_and_set`.

```
bool test_and_set(bool *target)
{
    bool rv = *target;
    *target = true;
    return rv;
}
```

Given the Boolean variable `lock`, which is initialized to FALSE, please show

(a) **[5 points]** How can a thread safely enter a critical section protected by `lock`? Please provide the code example.

while (test_and_set(&lock) == TRUE); (Deduct 1 point if the & symbol is written incorrectly)

(b) **[5 points]** How can a thread safely leave a critical section protected by `lock`? Please provide the code example.

lock = FALSE;

9. Following is a snippet of the [`Code for example of poll() usage`] from Homework #2.

```
……
22  while (running) {
23      printf("%d seconds\n\r", seconds++);
24      if ( poll(poll_fds, 1, 1000) < 0 ) {
25          perror("Error \n");
26          continue;
27      }
28      if (poll_fds[0].revents & POLLIN) {
29          ch = getch();
30          switch(ch) {
31          case 'q':
32          case 'Q':
33              running = false;
34              break;
35          default: ;
36          }
37      }
38  }
……
```

(a) **[3 points]** If the user does not press any key, how often will Ln 23 get executed?

1 sec

(b) **[3 points]** How many threads are involved in the execution of the while loop

1

(c) **[4 points]** Would the CPU utilization be closer to 100% or 0% when running the while loop?

0%

10. **[10 points]** In discussing the Dining Philosophers Problem, it was noted that philosophers may encounter deadlock situations. To mitigate this risk, one effective strategy is to require each philosopher to acquire both chopsticks—the left and the right—only when both are available simultaneously. This approach necessitates that Lines 2 and 3 of the following code execute atomically. Please demonstrate how this can be achieved using the synchronization primitives introduced in the class.

```
// The code of Philosopher i:
1    do {
2        wait (chopstick[i] );
3        wait (chopStick[ (i + 1) % 5] );
4
5        //  eat
6
7        signal (chopstick[i] );
8        signal (chopstick[ (i + 1) % 5] );
9
10       //  think
11
12   } while (TRUE);
```

(Each part is worth 5 points)

Create a semaphore X and set it to 1.

Use a Boolean array for chopstick[]. Initialize all elements to TRUE.

```
Replace  Line 2~3 with
     wait (X);
     if (chopstick[i] && chopstick[ (i + 1) % 5] ) {
        chopstick[i] = FALSE;
        chopstick[ (i + 1) % 5] = FALSE;
        signal(X);
     }
     Else {
         signal(X);
         continue;
     }

Replace  Line 7~8 with
     chopstick[i] = TRUE;
     chopstick[ (i + 1) % 5] = TRUE;
```