Midterm Exam: Spring 2020
EE415: Introduction of Operating System

School of Electrical Engineering, KAIST
13:00 - 14:15, Monday, May 2020

Read carefully before start solving the questions.
Please submit your solution in pdf. Submit your answer file to na94jun@gmail.com. At the first line of the file, please specify your name, student id, and email address.
You can write your answer in Korean or in English. You can take a screen shot to show your answer. The subject of the email should contain "[EE415]".

The answer email should be received by 14:20, May 4, 2020. The emails that arrive after 14:20 will be rejected.

1. (10pt) What is the advantage of separating fork and exec?

2. (10pt) In pintos, list up all line number and the function name that calls schedule().

3. Process Creation
   (a) (10pt) Does pintos allocate a user stack when the process is created? What is the size of the stack when a process is first created?
   (b) (10pt) Does pintos allocate a kernel stack when the process is created? what is the size of the kernel stack when a process is first created?
   (c) (10pt) Does pintos allocate a heap when the process is created? what is the size of the heap when a process is first created?

4. The interrupt frame is a data structure that is allocated and initialized when a user program executes "`int n`" instruction. The hardware and OS store the register values and the return address at the interrupt frame. When the pintos creates a new thread, pintos sets up the interrupt frame for the newly created process. In the interrupt frame, the CPU normally pushes the return address which is the "next instruction" it needs to execute after the interrupt returns, the value of eip register.
   (a) (10pt) In Pintos, what is the size of the interrupt frame?
   (b) (10pt) In Interrupt frame structure, explain which part of the interrupt frame is stored by the hardware and which part of the interrupt frame is stored by the software.
   (c) (10pt) Sets a break point at line 64 of intr-stubs.S and run Pintos. Show the value of ESP register right after executing the "iret" for the first time. What does it point to? A user stack? or a kernel stack? Please provide detailed reasoning to get the full credit.

5. You are required to run a test program that is supplied along with the Pintos. You need to setup a break point properly to answer the following question. Pintos OS maintains a bitmap (`struct bitmap`) for physical page frame (`struct pool`). Set a break point at `palloc_get_page()` and start the Pintos.

(a) (10pt) Print the size of the bitmap for the free page. How many physical page frame are there in PintOS?

(b) (10pt) `palloc_get_page()` returns either the user page or the kernel page based upon the parameter which the caller has supplied. You can type 'continue' to go to the next call to the `palloc_get_page`. In the third call to the `palloc_get_page()`, what does it return, user page or kernel page?

6. Let us examine the details of the scheduling. First, set up a break point at the "echo" statement of the user program. The address of echo statement is `0x080480a0`. So, please set up a break point at `0x080480a0`. Continue executing to this breakpoint. Then, do the following.

(a) (10pt) Schedule() can be called either from the user process or from the timer interrupt. We like to examine the stack content when the schedule() is called from the user process. Set up a new break point at schedule(). You have set up a break point at "echo" statement already. Run gdb till the break point, schedule(). Print the stack top address of the next thread to run.

(b) (5pt) Type `backtrace` command at the gdb and shows the output. You can take the screen shot of the screen.

(c) (5pt) Print the parameter value of the intr_handler() function that appear on the output of the backtrace command above.

(d) (10pt) The thread enters the kernel when the interrupt occurs. When the interrupt occurs, the hardware sets up the interrupt frame and stores the contents of the five registers. One of the five register is eip. That is the instruction pointer. It represents the address of the instruction that has been executing when it enters the kernel. Print the content of the eip register that was stored at the caller thread's interrupt frame.

(e) (10pt) In this situation, does `schedule()` call `switch_thread()`? If it does, what is the address of the thread structure of the next thread to run? If not, why does it not? Please explain in detail.

7. (10pt) Consider the CFS scheduler. There are two processes A and B. Their nice value corresponds to -5 and 0. The weight of each nice value is as follows. Assume that the scheduling latency is 48 msec. What is the length of time slice for process A and process B?

```
static const int prio_to_weight[40] = {
 /* -20 */     88761,     71755,     56483,     46273,     36291,
 /* -15 */     29154,     23254,     18705,     14949,     11916,
 /* -10 */      9548,      7620,      6100,      4904,      3906,
 /*  -5 */      3121,      2501,      1991,      1586,      1277,
 /*   0 */      1024,       820,       655,       526,       423,
 /*   5 */       335,       272,       215,       172,       137,
 /*  10 */       110,        87,        70,        56,        45,
 /*  15 */        36,        29,        23,        18,        15,
};
```