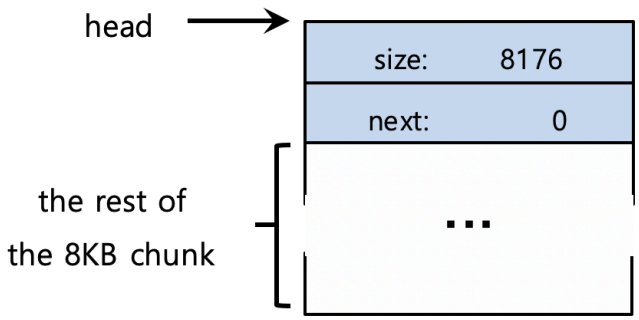
# Midterm Exam: Fall 2019
# EE415: Introduction of Operating System
School of Electrical Engineering, KAIST
Monday, Oct. 21, 2019 13:00 - 14:15

1. A process can have one of the three states: RUNNING, READY and BLOCKED. Please answer to the following questions.
   a. (1pt) What is the state of the process that is currently being executed by CPU?
   b. (1pt) A currently running process has used up its time quantum. Then, the CPU scheduler schedules the next process from the queue of processes that are waiting to be executed. What is the state of process that has been running after it has used up its time quantum?
   c. (1pt) A process that has been running is now waiting for the user input. What is the state of process when the process is waiting for the user input?
   d. (1pt) A process has requested an IO to read a disk page from the storage device. The process is waiting for the completion of an IO. The disk has transferred the requested page from the disk to the host's memory. After the IO request completes, what will be the state of process that has been waiting for the IO completion?

2. There is an 8 Kbyte free memory chunk as follows. A free memory chunk has 16 byte header: 8 byte for chunk size and 8 byte for the address of the next free memory chunk. For the allocated memory chunk, the pointer field is replaced with the magic number. Please answer to the following questions.




   a. (3pt) Draw the memory layout after executing the following three `malloc()` calls. Provide necessary assumption. In your layout, please show the pointers, `ptr1`, `ptr2` and `ptr3`.

```
ptr1 = malloc(200) ;
ptr2 = malloc(300) ;
ptr3 = malloc(200) ;
```

   b. (3pt) Continued from the above question 2-(a), we free the memory chunks pointed by `ptr2` and `ptr1`. Draw the memory layout after executing the following two `free()` calls. `head` points to the first free chunk in the free chunk list. The freed chunk is always inserted at the front of the free chunk list.

```
free(ptr2) ;
free(ptr1) ;
```

   c. (5pt) Continued from (b), we free the memory chunk pointed by `ptr2`, again. Note that we have freed the memory chunk pointed by `ptr2` in 2-(b). Draw the memory layout after

executing the following free call. We call this phenomenon as 'double free'. Explain the problem in double free.

```
free(ptr2) ;
```

3. (10pt) Consider the multi-level feedback queue CPU scheduling algorithm. In MLFQ, the process priority is computed using the following formula. The priority of the currently running process is computed as follows: the value of `recent_cpu` increases by one in every clock tick. In every four clock tick, the priority of the running process is recomputed. The priorities of the other processes are re-computed in every second. The larger the priority value is, the process has higher priority. We assume preemptive scheduling. In every clock tick, the CPU scheduler is triggered and selects the process with the highest priority.

```
priority = PRI_MAX - recent_cpu/4 - nice*2
```

Assume that clock speed is 10 Hz. The clock ticks in every 100 msec. PRIM_MAX is 64, and the value of nice is 0. The initial value of the process priority is set to 31. Initial value of recent_cpu is 0. Assume there are three processes, A, B and C. Show the priorities of the process A, B, and C for each clock tick. Also, for each clock tick, show the "running" process.

4. A process' virtual address space contains the segments for `code`, `heap`, `stack`, `data` and `BSS`. When the program is loaded onto memory for execution, OS initializes the some of the segments with the contents loaded from the disk. Consider the following code. Answer to the following questions.

```
int a_array [1000] ;     //—(1)
int b_array [1000] = 0 ; // —(2)

main(){
int first_integer ;      //    —(3)
static int second_integer ;// —(4)

//;;;
}
```

   a. (3pt) Among the five segments listed above, which segments are initialized with respect to the contents from the binary file? Please provide the detailed reasoning to get the full credit.
   b. (3pt) There are four variable declarations in the code above. They are numbered from (1) to (4). Specify the segment where each of these variables resides. Provide detailed reasoning behind your answer to get the full credit.

5. Following questions are about `exec()` system call in Unix. Please answer to the questions.

   a. (3pt) There are five segments in the process' virtual address space: `text`, `stack`, `heap`, `data` and `BSS`. Which of these segments are replaced with a new content when the `exec()` is called?

   b. (5pt) Assume the the caller of `exec()` has 4 KByte code segment. The `exec()` system call needs to load 12 KByte binary file. We need larger code segment. How does the `exec()` system call handle this situation? Does `exec()` system call ask for the additional memory space without returning the existing one? Or does `exec()` system call return all segments that need to be expanded and reallocate those segments? Provide details of your reasoning to get the full credit.

6. (5pt) The CPU scheduler needs to allocate a certain share of CPU cycles to each process. There are two approaches: lottery scheduling and stride scheduling. Explain the advantage and disadvantage of these approaches.

7. A process has 64 KByte virtual address space. Page size is 1 KByte. There are 4 MByte physical memory in the system. Please answer to the following questions.

   a. (1pt) Consider an address of 0x3CF0. Convert it to the binary number.
   b. (2pt) Provide the virtual page number and page offset for this address.
   c. (2pt) How many bits do we need to represent 4 MByte physical address space?
   d. (1pt) What is the number of page table entries in this system?
   e. (2pt) A page table entry contains dirty bit, reference bit, present bit and etc in additional to the page frame number. In this system, each page table entry has 4 Byte. What is the page table size in this system?

8. Refer the following assembly code. The initial value of `%eax` register is 0.

```
address     instruction
0x1024      movl $0x0, (%edi, %eax, 4)    // M[%edi + %eax*4] = 0 ;
0x1028      incl %eax                     // ++ %eax ;
0x102c      cmpl $0x03e8, %eax            //  if( %eax == 1000) ;
0x1030      jne 0x1024                    // goto 0x1024 if (%eax != 1000)
```

   a. (3pt) How many times is the page table accessed in the course of executing this code? Assume single level page table. Provide the detailed reasoning to get the full credit.
   b. (3pt) Assume TLB is initially empty. The capacity of TLB is 32. If there is a TLB, how many times in the page table accessed?
   c. (3pt) Assume that the time to access TLB is 1/1000 of the time to access memory. Compute the total memory access time with and without TLB.

9. When the hardware interrupt is raised, the currently running program is temporarily suspended and OS executes the interrupt handler.
   a. (3pt) When the interrupt is raised, the registers used by the user process are stored before the interrupt handler is executed. Where are the register values of the user process stored? Are they stored at user stack? or at kernel stack? Please provide detailed reasoning to get the full credit.
   b. (3pt) OS performs context switch to allocate CPU from one process to another. When the interrupt handler is executed, does the context switch happen?