National University of Singapore
School of Continuing and Lifelong Education

TIC1001: Introduction to Computing and Programming I
Semester I, 2019/2020

Tutorial 9
Brief Look at OS and Database

1. **[Process Management]** In the lecture, we discussed that OS is responsible for selecting a process to run on CPU. This is formally known as **process scheduling**. One way to make a good process scheduling decision is based on the **process behavior**. A process perform either:

   - Input/Output (IO)-Activity: e.g. waiting for user input (`scanf()`), reading from a file (`fgets()`), printing information on screen (`printf()`), etc. OR
   - CPU-Activity: e.g. performing heavy calculation.

   (a) Discuss the behavior of the following processes:

      i. The "desktop" interface commonly seen in most modern OSes.

      ii. The "Sieve of Erathoses" on a HUGE array (say 500 million entries).

   (b) On a system with an active user (e.g. you are coding actively on your laptop), discuss how should the OS prioritizes processes based on their behavior. As a guide, you should be thinking about, "if a process P is doing a lot I/O activity, then . . . ". You can also use (a) as a guide.

2. **[File Management]** Let us write a simple C program simulate how the OS uses File Allocation Table (FAT) to find out the disk blocks used by a file.

   You are given an input file which contains the entries of a FAT table. The first number in the file indicate the size S of the FAT, the subsequent S integers are the content of the FAT. For example, the "`sample_fat.txt`" contains:

```
32       // The size of FAT is 32 entries
-1       // Entry [0], i.e. FAT[0] contains -1, -1 indicates the end of a "file"
13       // FAT[1] contain 13
6        // FAT[2] contains 6
24       // etc.
28
...
```

   Write a program that:

   (a) Ask user for the input file name `FN` .

   (b) Reads from the file `FN` and initializes the FAT. You can assume that the largest FAT has 128 entries.

   (c) Repeatedly reads a single integer `X` from user, then:

      i. Print the disk block number `D` at FAT[`X`]

ii. Set `X` = `D`, then repeat step i, until `D` is -1

Below is a sample execution session (user input in bold):

```
File name: fat32.txt    //Sample input file is "fat32.txt"
FAT size is 32          //Show the size of FAT read from fat32.txt
   0[-1]                //Print the FAT entries after storing them.
   1[13]
   2[ 6]
 ...
  31[16]                //This is the start of step (c)

1 13                    //File at location 1 continues with block 13
5
5 15 30 20 21 23     8 0
<ctrl-d>                //to stop the program
```

You are free to design the program in any way. (If you are unsure where to start, take a look at the last page of this tutorial for some suggestions).

3. **[Database]** Given the following person table:

name	height	weight	age	IQ
Abigail	152.5	55	17	180
Trigun	175.4	85	25	135
Shelcard	195.0	120	55	120
Maurice	145.4	48	46	108
Liam	170.5	60	18	100

(a) Give the SQL statement to create the table above. You can make reasonable assumptions on the datatype of each column.

(b) Give the SQL statements to insert the 5 rows above.

(c) Give the SQL statements to find out the following information. Note that you are free to make reasonable assumptions.

i. Young Genius

ii. Giant among us

iii. The Joes (not tall, not bulky, not genius, i.e. average person)

As demonstrated in the lecture, you can use any online SQL platform to try out the SQL statements, e.g. http://sqlfiddle.com/

Hints / Suggestions for Question 2 (Don't look if you want to challenge yourself)

1. The FAT is simply an integer array.
2. Write the following functions as a start:
   (a) `int readFAT( FILE* input, int fat[])`
   - Read the FAT entries from input

- Output: the array fat[] contains the FAT entries read
- Return: the size of the fat[]

(b) `void printFAT( int fat[], int size)`

- Print the FAT entries in the form of "idx [content]"
- The size of the FAT array is given as size.

(c) `void printFileAt( int fat[], int start)`

- Print the disk blocks used by the "file" starting at block start

3. The main loop for step (c) can be achieved via:

```
while ( scanf("%d", &start) == 1) { ... }
```

This loop will read until the user press <ctrl-D> which sends an "EOF" signal to the program.