This handout is meant to serve as a representative for the prerequisite knowledge you might need in this module. Not all of it will be needed. You are ***not*** expected to recollect all these background details immediately and you will **not** be tested on these topics assessments directly. Use these questions to fill any gaps in your systems knowledge. A good "systems security" student should know these kinds of things.

If you're curious about the answers, use the LumiNUS class forum to discuss it with others.

1. A process on a computer has allocated only the virtual memory region 0x6000000 - 0x8000000. What happens when it accesses 0x5000000 at runtime?

a. The processor will raise a General Protection Fault (GPF) and jump to the kernel (ring-0)
b. The processor will raise a General Protection Fault (GPF) and jump to process A (ring-3)
c. The processor will halt.
d. The processor will kill the process A.

2. The OS wishes to switch context between process A (currently running) to process B (next to run). An efficient OS should perform which of the following steps :

a. Copy all memory pages of A from RAM to disk, and load all memory pages of B from disk.
b. Copy the page tables of B from disk to RAM, and write all page tables of A to disk
c. Only switch the page-table-base-register (PDBR) to point to B's page table instead of A's
d. No changes to page-table related registers / data is necessary.

3. How does an OS share a memory page between two processes? IPC memset

4. Here is the output of the `/proc/pid/maps` of a process. Can you identify the code segment, data segment of the `/bin/cat` binary executable and the stack segment?

Code -
Data -
Stack -

```
00400000-0040b000 r-xp 00000000 08:01 655384                             /bin/cat
0060a000-0060b000 r--p 0000a000 08:01 655384                             /bin/cat
0060b000-0060c000 rw-p 0000b000 08:01 655384                             /bin/cat
009a4000-009c5000 rw-p 00000000 00:00 0                                  [heap]
7fdcb3b3b000-7fdcb421d000 r--p 00000000 08:01 663756                     /usr/lib/locale/locale-archive
7fdcb421d000-7fdcb43d9000 r-xp 00000000 08:01 1446080                    /lib/x86_64-linux-gnu/libc-2.19.so
7fdcb43d9000-7fdcb45d8000 ---p 001bc000 08:01 1446080                    /lib/x86_64-linux-gnu/libc-2.19.so
7fdcb45d8000-7fdcb45dc000 r--p 001bb000 08:01 1446080                    /lib/x86_64-linux-gnu/libc-2.19.so
7fdcb45dc000-7fdcb45de000 rw-p 001bf000 08:01 1446080                    /lib/x86_64-linux-gnu/libc-2.19.so
7fdcb45de000-7fdcb45e3000 rw-p 00000000 00:00 0
7fdcb45e3000-7fdcb4606000 r-xp 00000000 08:01 1446056                    /lib/x86_64-linux-gnu/ld-2.19.so
7fdcb47ec000-7fdcb47ef000 rw-p 00000000 00:00 0
7fdcb4803000-7fdcb4805000 rw-p 00000000 00:00 0
7fdcb4805000-7fdcb4806000 r--p 00022000 08:01 1446056                    /lib/x86_64-linux-gnu/ld-2.19.so
7fdcb4806000-7fdcb4807000 rw-p 00023000 08:01 1446056                    /lib/x86_64-linux-gnu/ld-2.19.so
7fdcb4807000-7fdcb4808000 rw-p 00000000 00:00 0
7fff3b99d000-7fff3b9be000 rw-p 00000000 00:00 0                          [stack]
7fff3b9fe000-7fff3ba00000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0                  [vsyscall]
```

5. What order are the following tools invoked in to prepare an executable? What is the input-output of each component? Which of these, if any, may be present in the process when the executable runs?

A. Compiler, Linker, Assembler, Loader
B. Compiler, Linker, Loader, Assembler
C. Compiler, Assembler, Linker, Loader
D. Compiler, Loader, Linker, Assembler

6. Consider the following six C program snippets. Which of these will fault on line 3 in function **`foo`**?

```
void foo (x) {
      int a, *b, c[1] ;
      b = (int*) malloc (sizeof (int));
      scanf (“%d”, a);
}
```

```
void foo (x) {
      int a, *b, c[1] ;
      b = (int*) malloc (sizeof (int));
      scanf (“%d”, b);
}
```

```
void foo (x) {
      int a, *b, c[1] ;
      b = (int*) malloc (sizeof (int));
      scanf (“%d”, c);
}
```

6.

```
void foo (x) {
      int a, *b, c[1] ;
      b = (int*) malloc (sizeof (int));
      scanf (“%d”, &a);
}
```

```
void foo (x) {
      int a, *b, c[1] ;
      b = (int*) malloc (sizeof (int));
      scanf (“%d”, &b);
}
```

```
void foo (x) {
      int a, *b, c[1] ;
      b = (int*) malloc (sizeof (int));
      scanf (“%d”, &c);
}
```

7. The '`call 0x5000`' instruction on an x86 CPU has the following semantics :

```
RetVal := EIP+4; // EIP is the instruction pointer register
 ESP := ESP - 4;  // ESP is the stack pointer register
 [ESP] := RetVal; // [REG] denotes access to value pointed by REG
 EIP := 0x5000;
```

What do you think is the right semantics of the `ret` instruction?

```
RetVal := [ESP];
ESP := ESP + 4;
EIP:= RetVal;
```

```
ESP := ESP + 4;
RetVal := [ESP];
EIP:= RetVal;
```

```
RetVal := [ESP];
EIP:= RetVal;
```

```
RetVal := [ESP];
ESP := ESP - 4;
EIP:= RetVal;
```

Hint: Intel x86: http://www.cs.virginia.edu/~evans/cs216/guides/x86.html

8. We want to dynamically allocate a 2D integer array (matrix) in C. Let the base pointer be `int** x;`
Which of the following is the safe way of doing this --- A, B, or C?

```
A. x = malloc (100 * 50 * sizeof(int) );
        ... x[i][j] ...
```

```
B. x = malloc(100 * sizeof(int));
for (int i = 0; i < 100; i++) {
x[i] = malloc(50 * sizeof(int));
}
        ... x[i][j] ...
```

```
C. x = malloc(100 * sizeof(int*));
for (int i = 0; i < 100; i++) {
x[i] = malloc(50 * sizeof(int));
}
        ... x[i][j] ...
```

(Hint: Code taken from here)

9. Given a secret key k and a message m, which of the following is a secure one-time encryption technique? Explain why (ideally, write down a proof using the conditional probabilities Pr[m| E(m)]).

A. E(m) := m `bitwise-and` k
B. E(m) := m `bitwise-or` k
C. E(m) := m `bitwise-xor` k
D. None of the above.

10. You are asked to debug the following program.

```
void main()
{
  int x = 10, y = 20;
   f(x, y);
}

void f(int x, int y)
{
  g(x, y);
}

void g(int x, int y)
{
  qsort(x, y);
}

void qsort(int x, int y)
{
  if (x > y)
    printf("%d, %d", y,x);
  else
    printf("%d, %d", x,y);
}
```

Write down the GDB commands to perform the following steps:

a. Put a breakpoint on the entry to function g.
b. Print the address of function g
c. Print the call-stack when you hit the breakpoint set at step a.
d. Print the address of variable x
e. Print the contents of memory 0x5000
f. Print the return address at breakpoint set at step a.
g. Print the assembly instructions of function g.