Assignment 2 – Proposed solution:

Problem 1 (5 pts)

Discuss the difference between Kernel mode and User mode.

Answer:

Kernel mode:

- Complete and unrestricted access to all hardware.
- Execute any instruction the machine is capable of.
- Crashes in kernel mode may halt the computer.

User mode:

- Limited access to the hardware in order to protect the OS and hardware.
- Execute only a subset of instructions and access to a subset of features.

Having kernel mode and user mode modes allows, users to run user programs in user mode and thus deny them access to critical instructions. This is also enforced by the CPU hardware.

Problem 2 (16 pts)

Evaluate if each of the following statements is TRUE or FALSE.

1. Bash shell is a part of an operating system (OS) - **FALSE**.
   Bash shell is user interface program
2. A program which copies a file belongs to an OS - **FALSE.**
   It is a software running on user mode
3. Users can operate in Kernel mode thanks to procedure calls- **FALSE**
   Nothing is true in this statement
4. Interrupt is used to switch from User mode to Kernel mode- **TRUE**
5. In Kernel mode, some instructions are forbidden – **FALSE**
   Kernel mode can execute any instructions.
6. Disks and I/O devices are abstracted as files by OS - **TRUE**
7. A process table is used to store information of a suspended process – **FALSE**
   Process tables store all information about each process, with one entry per process currently in existence. This means that a process table not only stores information of suspended processes, but every process.
8. Two processes are connected by using a file – **FALSE**
   In the context of Chapter 1, two processes are connected by using a pipe, which is considered as a pseudo file. A pseudo file is a special file, which is categorized under the "special file" section.
   However, two processes can be connected in other ways, for example message passing which has not yet been delivered up to the date of assignment. Therefore, the answer for statement: 'Two processes are connected by using a file' is **FALSE,** but the statement itself is a mistake at the time of delivering. Therefore, this statement was not considered during grading. **We are very sorry for this inconvenient.**

Problem 3 (4 pts)

Suppose that you have an OS running on your smart TV or your laptop. Can that OS be used for wearable or sensor devices? Explain your answer.

Answer: NO

Two main functions of an OS:

1) control hardware
2) provide abstraction of the hardware to user interface program.

Therefore, to some extents, the OS depends on the hardware in sense that it is built with certain hardware and hardware infrastructure as the foundation.

Smart TV and PC are too powerful, compared to wearables and other sensor devices. Therefore, OS that could be used in Smart TV or PC, are not compatible with wearable or sensor devices. In fact, wearables or sensor devices may not have enough memory for OSs used in Smart TV/PC, to be installed. In addition, they may quickly run out of battery power.

Problem 4 (5 pts)

What happens when a program calls a system call?

Steps 1-3, the calling program pushes the parameters onto the stack.

Step 4, A call to the actual library procedure is made. This instruction is the normal procedure call instruction, used to call all other procedures.

Step 5, The library procedure, places the system call number, in an expected place. Eg: a register.

Step 6, Library procedure executes a TRAP instruction. Which is used to switch from user mode, to kernel mode, and start execution at a fixed address, within the kernel.

Step 7, The kernel examines the system call number and then dispatches it to the correct system call handler. This handler is found by using a table containing a reference to the specific handler given the number.

Step 8, The system call handler runs.

Step 9, Operation has now completed, and therefore the user is given back control, once the TRAP instruction is set.

Step 10, This procedure returns to the user program.

Step 11, The OS will now clear the stack. Which is done by incrementing it till it is empty.

Source(https://technobyte.org/system-calls-in-operating-systems-simple-explanation/)

Problem 5 (20 pts). Questions in Chapter 2 of the main textbook:

- (5 pts) Question 1 In Fig. 2-2, three process states are shown. In theory, with three states, there could be six transitions, two out of each state. However, only four transitions are shown. Are

there any circumstances in which either or both of the missing transitions might occur?
Answer:
The missing transitions are: Blocked → Running, Ready → Blocked.
Blocked → Running **is** possible if the processor is idling.
Ready → Blocked **is not** possible as the process has yet to execute and therefore not possible to be blocked.

- (5 pts) Question 8 Consider a multiprogrammed system with degree of 6 (i.e., six programs in memory at the same time). Assume that each process spends 40% of its time waiting for I/O. What will be the CPU utilization?
  Answer:
  Using the CPU utilization formula we get: $1 - p^n$ => $1 - 0.4^6$ = 0.995904

- (5 pts) Question 6 A computer has 4 GB of RAM of which the operating system occupies 512 MB. The processes are all 256 MB (for simplicity) and have the same characteristics. If the goal is 99% CPU utilization, what is the maximum I/O wait that can be tolerated?
  Answer:
  goal is: 0.99 => $1 - p^n = 0.99$
  process size: 256MB
  4GB = 4096MB
  4096 – 512 = 3584
  n = 3584 / 256 = 14

  Therefore we get: $1 - p^{14} = 0.99$ => p = 0.71968

- (5 pts) Question 7
  Multiple jobs can run in parallel and finish faster than if they had run sequentially. Suppose that two jobs, each needing 20 minutes of CPU time, start simultaneously. How long will the last one take to complete if they run sequentially? How long if they run in parallel? Assume 50% I/O wait.
  Answer:
  Sequentially:
  Each job requires 20 / (1-0.5^1) => 40 minutes. Therefore, the last job will complete after 40+40 = 80 minutes.

  Parallel:
  CPU utilization: 1 – 0.5^2 = 0.75 which gives 0.75/2 = 0.375 pr-processor
  20 / 0.375 = 53.33….3 minutes for completion

Oppgave 6) (5 pts)

Discuss shortly the differences between local, global, static and static local variables. (Hint: consider the location in the address space, scope (within function, files, etc.) or initialization).
Answer:

Global:

- Exists throughout the program.
- Is defined outside all functions and is therefore accessible for all functions and variables that comes after the declaration.

- Could be accessed outside the current file by using the keyword extern and declaring the variable in both files.

Static:

- When defined in the global scope it becomes a global static variable that has file scope. Though it cannot be accessed from other files like a true global variable.
- When defined inside a block scope, it can only be accessed within that same scope.
- The value is persistent and exists throughout the program.

Local:

- Is stored, initialized and accessed only within the current block
- Only exists on the temporary stack related to the current block.

All initialized global and static variables are stored in the data section of memory. Whilst all uninitialized global and static variables are stored in the Bss section, where it will be initialized to zero on program start.

Local variables are stored on the current block or function stack.

Problem 7 (10 pts) Consider the following code fragment:

```
1  int a, b;
2
3  void foo()
4  {
5          int a, b;
6          int c;
7          a = 1;
8          b = 1000;
9          c = 10;
10
11         printf("a = %d, b = %d, c = %d \n", a, b, c);
12 }
13
14 void main()
15 {
16         int c;
17         a = 2;
18         b = 2000;
19         c = 20;
20
21         foo();
22         printf("a = %d, b = %d, c = %d\n", a, b, c);
23 }
```

- (2.5 pts) What are local variables, global variables in this program?
  Answer:
  - Local: All variables marked as blue or cyan. Which in this case are called: a, b and c.
  - Global: All variables marked red. Which in this case are called a and b.
- (5 pts) Discuss shortly the output of the program?
  Answer:
  - Output:
    a = 1, b = 1000, c = 10
    a = 2, b = 2000, c = 20
    The coloring in the picture showcases which variables are local or global in that

current program segment. Whenever a variable is shadowing another from an outer scope, it will prefer the one that is local to that function.

- (2.5 pts) Discuss shortly the output of the program if we remove line number 5?
  Answer:
  - It will output:
    a = 1, b = 1000, c = 10
    a = 1, b = 1000, c = 20
    The values for global variables a and b will now have their values changed by foo(). As those are the closest scope. As illustrated:

```
1  int a, b;
2
3  void foo()
4  {
5
6          int c;
7          a = 1;
8          b = 1000;
9          c = 10;
10
11         printf("a = %d, b = %d, c = %d \n", a, b, c);
12 }
13
14 void main()
15 {
16         int c;
17         a = 2;
18         b = 2000;
19         c = 20;
20
21         foo();
22         printf("a = %d, b = %d, c = %d\n", a, b, c);
23 }
```

Problem 8 (10 pts)

Consider the following code fragment:

```
int pow_2_10;
for(int i = 0; i < 10; i++)
{
    static int b = 2;
    b = b* 2;
    pow_2_10 = b;
}
```

- (5 pts) Predict the value of pow 2 10 after the loop finishes. Explain
  Answer:
  pow_2_10 = 2^11 = 2048. The variable pow_2_10 starts at 4 and is incremented by * 2 for each iteration. The value of variable b, is persistent inside that for block as it is a local static variable.

- (2.5 pts) Discuss the value of pow 2 10 if we remove static keyword.
  Answer:
  b is initialized with the value 2 in every iteration, which is then incremented by * 2. Thereby making pow_2_10 = 2 * 2 = 4

- (2.5 pts) Explain why we could not assign value of b to pow 2 10 outside the for loop, like this:

```
int pow_2_10;
for(int i = 0; i < 10; i++)
{
    static int b = 2;
    b = b* 2;
}
pow_2_10 = b;
```

Answer:
Static variables can only be accessed within the same block scope. Here that block scope is within the for loop. Therefore, one cannot access this variable outside of the for loop.

Problem 9 (10 pts) Consider the following code fragment:

```
while(N > 0)
{
    N--;
    fork();
}
```

Assume that N is a positive integer which is small enough to avoid fork bomb.

- (5 pts) How many child processes are created?
  Answer:
  The number of children is $2^N - 1$. Where 1 represents the root node.

- (5 pts) How many parent processes are there?
  Answer:
  A child that creates another child is itself a parent. Therefore, the number of parents is: 2^{N-1}

- (Bonus - 2.5 pts) What is a fork bomb?
  It is a DOS attack that aims to consume CPU time and take up the operating system's process table by cloning a process repeatedly. Which would also take up space in memory. Thereby making the OS unresponsive. Though in modern unix systems the cloning process uses a technique **called copy on write**. Which won't, in most cases, exhaust the memory immediately.

Problem 10 (5 pts)
Consider the following code fragment:

```
if (fork() == 0)
    a = a + 5;
else
    a = a - 5;
```

- (5 pts) Let x and y be the values of variable a in the parent and child process respectively. What can we infer about the difference of x and y after executing this code? Explain.
  Answer:
  fork() returns negative integer on fail, zero to new process and positive integer to the caller/parent. Therefore, y, the child, will be: a = a + 5. Making y = a + 5. Whilst x, the parents a value, will be: a = a – 5, making x = a - 5, and the total difference between the parent and the

child = 10

- (Bonus - 5 pts) Let u and v be the addresses of variable a in the parent and child process respectively. What can we infer about the difference of u and v after executing this code? Explain.
  Answer:
  The addresses of u and v will be equal as they share the same virtual address. The values inside the addresses of u and v will still have a difference of 10.

Problem 11 (10 pts)
The program in comparator.c reads two integers from the user input, compares them and prints out messages accordingly. The expected output of this program is shown below. However, there are some bugs inside this program which cause different errors. Fix the bugs and explain in short why there are those bugs. Remember to update the comparator.c accordingly.

Answer:
1: scanf("%d %d", num1, num2); to scanf("%d %d", &num1, &num2); as scanf requires the address to the variables in which it will store the values.

2: if(num1 = num2) to if(num1 == num2), as the goal here is to check for equality and not assign num1 to num2.

3(Optional): Altered the return of main from void to int, as is best practice. Which allows one to know whether the program exited successfully or not.