# Soal 1

Consider the following code:

```
for (i = 0; i < 20; i++)
    for (j = 0; j < 10; j++)
        a[i] = a[i] * j
```

- Give one example of the spatial locality in the code

```
The array indices a[0] and a[1] address are located near one another or in the
same block because they are most likely will be accessed after one another
```

- Give one example of the temporal locality in the code

```
the array indice a[0] is accessed 20 times, so it is likely that the value of a[0]
will be stored in the cache memory
```

# Soal 2

An I/O-bound program is one that, if run alone, would spend more time waiting for I/O than using the processor. A processor-bound program is the opposite. Suppose a short-term scheduling algorithm favors those programs that have used little processor time in the recent past. Explain why this algorithm favors I/O-bound programs and yet does not permanently deny processor time to processor-bound programs.

```
because the I/O-bound program is the program that spends more time waiting for I/O
that using the processor, the program will be first in priority to the scheduling
algorithm. But once the I/O-bound program has been executed, the processor-bound
program will be executed next. This is because the processor-bound program has not
been executed for a long time compared to the I/O-bound program, so the processor-
bound program will be executed next.
```

# Soal 3

It was stated that UNIX is unsuitable for real-time applications because a process executing in kernel mode may not be preempted. Elaborate

```

```

```
UNIX is unsuitable for real-time applications because a real-time process has a
strict scheduling and timing requirement. In UNIX, a process executing in kernel
mode may not be preempted or paused, so the real-time process may not be executed
in time. This is because the kernel mode process has a higher priority than the
real-time process, so the real-time process will be executed after the kernel mode
process has been executed, and possibly missing their timing schedule.
```

# Soal 4

Consider an enviroment in which there is a one-to-one mapping between user-level threads and kernel-level threads that allows one or more threads withing a process to issue blocking system calls while other threads continue to run. Explain why this model can make multithreaded programs run faster than their single-threaded counterparts on a uniprocessor computer.

```
This model can make multithreaded programs run faster than their single-threaded
counterparts on a uniprocessor computer because the kernel-level threads can be
executed in parallel. On a multithreaded program, when one of the user-level
threads issues a blocking system call, the kernel-level thread will be executed,
and the other user-level threads can continue to run. This will make the
multithreaded program run faster than the single-threaded program because the
kernel-level threads can be executed in parallel.
```