# Operating Systems Problem session 5

Dibran Dokter 1047390 & Marnix Lukasse 1047400

October 11, 2019

## 8.6

It does not allow a process to access memory it does not own because that would introduce security risks. A malicious user could inject a program and access the data of the rest of the processes including the operating system itself.

A process can let multiple processes share data by linking the pages of the processes to the same frame in physical memory.

It should do this when multiple users run the same program or use the same library. It should make sure however that the code does not get changed. It should not do this when the code changes something in the rest of the system (is re-entrant).

## 8.12

a) page=3,offset=13, since 3085 = 3 * 1024 + 13.
b) page=42,offset=111, since 42095 = 42 * 1024 + 111.
c) page=210, offset=161, since 215201 = 210 * 1024 + 161.
d) page=634, offset=784, since 650000 = 634 * 1024 + 784.
e) page=1953, offset=129, since 2000001 = 1953 * 1024 + 129.

## 8.13

a)
When we limit it to the 21bit logical address there can be 1024 entries since $2^{21}/2048 = 1024$.
b)
When limited to the 16bit physical address there can only be 32 entries since $2^{16}/2048 = 32$.

## 8.15

a) 8 bits, since $2^8 = 256$.
b) 18 bits, since $2^{18} = 262144$ and $2^{18} = 2^6 \cdot 2^{12}$.

## 9.1

We can encounter all the scenarios when we have the following scenario:

Since we have just referenced the address it will still be loaded in the TLB since it was just accessed. So when we access it again it is still in the TLB and we get a TLB hit and no page fault.
When we try to access the same address later in the program our address could have been removed from the TLB causing a miss and no page fault.
Maybe later in the program we try to access an address that has not yet been loaded in to memory and thus has not been cached. This would cause a miss and a page fault.
A while later we may want to access that piece of data again. This time it is still cached in the TLB and we get a hit. It has however been removed from the main memory since it has not been used for a while so we get a page fault.(this case seems unlikely since the TLB should remove the unused reference.)

## 9.18

Using this formula to calculate the time:
effective access time $= (1 - p) \cdot ma + p \cdot pagefaulttime$. where $p$ = page fault probability, $ma$ = memory-access time.

In this case : $(1 - 0.10) \cdot 1micro + 0.10 \cdot 20millis$