Exercise 1

1. Programs may have virtual address spaces larger than the available memory capacity. Paging splits the virtual address space into chunks and loads them in the available slots called frames in RAM. Thus a program actually to large for memory can still execute due to paging.

2. Paging splits the virtual address space into smaller chunks and loads them as needed. swapping stores currently inactive processes e.g. minimized GUIs to disk to make space for other processes. Paging deals with loading parts of processes address space while swapping deals with loading and storing processes.

3. Overlays split programs into little pieces and loading them as needed using an overlay manager. Using overlays, the programmer had to define the split and take care of loading the right overlay at the right point in time. With paging all the splitting and loading work is outsourced from the program to the operating system including loading mechanisms.

4. A page is a piece of a program, thus virtual memory. A page frame is a chunk of RAM and thus corresponds to a sequence of addresses in physical memory.

5. They can have different sizes, but should have the same size to avoid fragmentation. The size of a page defines the size of one piece of the program, while the same is for the chunk of RAM and the page frame.

6. In order to match up with addresses (addressing is done in binary format) and address translation (which is done by the MMU) and to allow for intelligent splits (e.g. the buddy algorithm). In particular, bit masks are used by the MMU to split between page number and offset which would not be so fast and easy when using a non-binary system (or at least some system that is easily decomposable into binary e.g. octal or heaxdecimal).

7. A page table stores the mapping between virtual pages and physical frames. It is integral to address translation: Page frame address + offset = physical address

8. A page fault is what happens when a page is looked up in a page table and the absent bit is set (no page frame is stored for this page). On encountering a page fault, the operating system loads the page into a frame and updates the page table entry.

9. When data structures of dynamic size grow and shrink, it may be that some run out of allocated space while others have plenty of unused allocated memory. To avoid this, segmentation can be used to introcuce independent address spaces so called segements that are automatically allocated in pages to fit and may be redistributed to avoid external fragmentation

10. Yes, e.g. Intel x86 and MULTICS proposed this

Exercise 2

Resident: Page 0 (code). Array uses $\frac{64^2}{128}$ Pages $= \frac{(32 \cdot 2) \cdot 64}{128} = \frac{128 \cdot 32}{128} = 32$ Pages.

1. Fragment A:
   As the inner loop iterates over the rows, each second access leads to a page fault as only two elements of a page are accessed before the next page is needed to answer the queried element. This leads overall to $\frac{64^2}{2} = \frac{2 \cdot 32 \cdot 64}{2} = 64 \cdot 32 = 2^6 \cdot 2^5 = 2^{11} = 2048$ page faults.

2. Fragment B:
   As the outer loop iterates over i here and the inner over j, the accesses are sequential in memory. This means that two rows $X[i][]$ and $X[i+1][]$ are accessed without page fault. This means that overall 32 page faults occur.

Exercise 3

Expression	Description
`v`	value of variable `v`
`&v`	address of variable `v`
`*p`	value at address stored by `p`
`a[2]`	value of third element of the array. Also value of the address `a + 2 * sizeof(*a)`
`**p`	value at address stored by value at address stored by `p`

Exercise 4

1. $5555_10 = 1010110110011_2$.
   Page ID $= (1110000000000_2 \& 1010110110011_2) >> 10 = 101_2 = 5$
   Offset $= 0001111111111_2 \& 1010110110011_2 = 0110110011_2$
   Page ID 5 is in Page frame 0.
   $\rightarrow$ Physical address $= 0000110110011_2$

2. $4100_10 = 1000000000100_2$.
   Page ID $= (1110000000000_2 \& 1000000000100_2) >> 10 = 100_2 = 4$
   Offset $= 0001111111111_2 \& 1000000000100_2 = 0000000100_2$
   Page ID 4 is not present $\Rightarrow$ Page fault.

3. $2048_10 = 0100000000000_2$.
   Page ID $= (1110000000000_2 \& 0100000000000_2) >> 10 = 010_2 = 2$
   Offset $= 0001111111111_2 \& 0100000000000_2 = 0000000000_2$
   Page ID 2 is in Page frame 3.
   $\rightarrow$ Physical address $= 0110000000000_2$

4. $2047_10 = 0011111111111_2$.
   Page ID $= (1110000000000_2 \& 0011111111111_2) >> 10 = 001_2 = 1$
   Offset $= 0001111111111_2 \& 0011111111111_2 = 1111111111_2$
   Page ID 1 is in Page frame 7.
   $\rightarrow$ Physical address $= 1111111111111_2$

Exercise 5

1. siehe address_translation.c