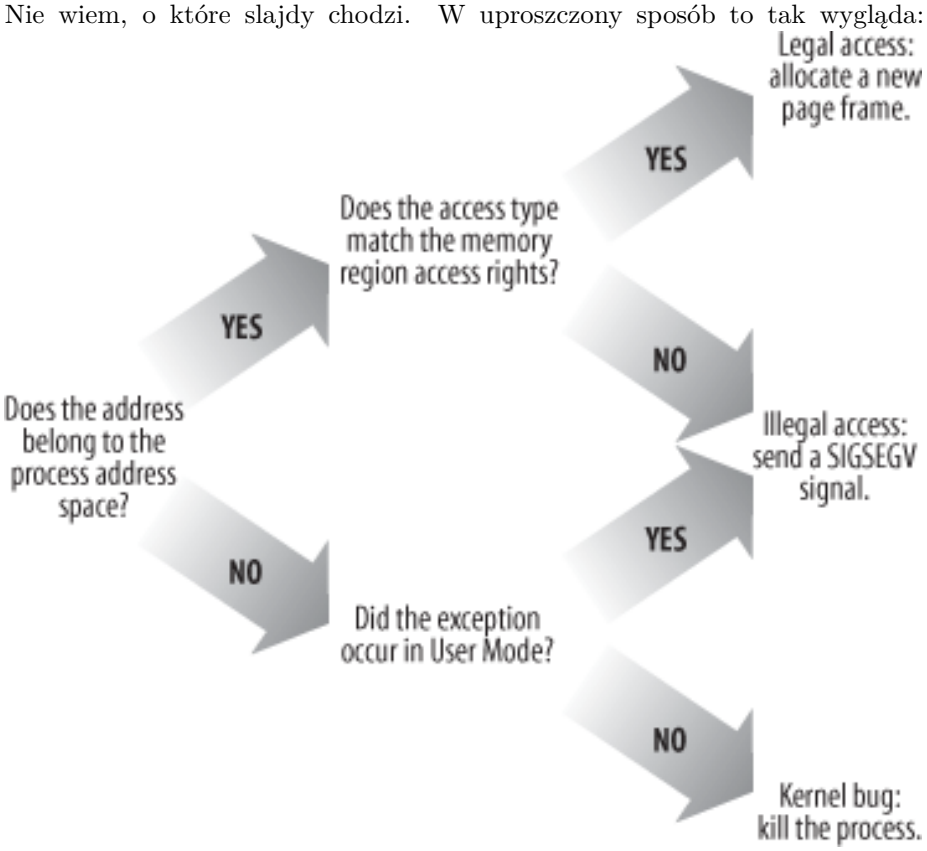
Zadanie 3.

Nie wiem, o które slajdy chodzi. W uproszczony sposób to tak wygląda:



There are two common kinds of **SEGV**, which is an error that results from an invalid memory access:

- A page was accessed which had the wrong permissions. E.g., it was read-only but your code tried to write to it. This will be reported as **SEGV_ACCERR**.
- A page was accessed that is not even mapped into the address space of the application at all. This will often result from dereferencing a null pointer or a pointer that was corrupted with a small integer value. This is reported as **SEGV_MAPERR**.

Minor - the page is loaded in memory at the time the fault is generated. The page fault handler in the operating system merely needs to make the entry for that page in the memory management unit point to the page in memory and indicate that the page is loaded in memory. This could happen if the memory is shared by different programs and the page is already brought into memory for other programs.

Major - the page is not loaded in memory at the time of the fault. The page fault handler in the OS needs to find a free location: either a free page in memory, or a non-free page in memory. Once the space has been made available, the OS can read the data for the new page into memory, add an entry to its location in the memory management unit, and indicate that the page is loaded.

Under Linux, the **Page Cache** accelerates many accesses to files on non volatile storage. This happens because, when it first reads from or writes to data media like hard drives, Linux also stores data in unused areas of memory, which acts as a cache. If this data is read again later, it can be quickly read from this cache in memory.

Backing object - obiekt, który odpowiada za prawdziwy dostęp do danych, np. może to być jakiś plik, urządzenie lub pamięć anonimowa.

Nie potrafię odpowiedzieć, kiedy wystąpi błąd strony przy zapisie mimo, że pole *vm_prot* pozwala na zapisdo obiektu wspierającego.

SIGBUS is generated for file-based mappings if we access a part of the mapping for which no corresponding region exists in the file (i.e., the mapping is larger than the underlying file).

Zadanie 4.

- **Real UID** - This is the UID of the user/process that created process. It can be changed only if the running process has EUID=0.
- **Effective UID** - This UID is used to evaluate privileges of the process to perform a particular action. EUID can be changed either to RUID, or SUID if EUID!=0. If EUID=0, it can be changed to anything. When you run a setuid program, set to the program's owner UID.
- **Saved UID** - If the executable, that was launched has a Set-UID bit on, SUID will be the UID of the owner of the file. Otherwise, SUID will be the RUID. The saved user ID (suid) is used when a program running with elevated privileges needs to do some unprivileged work temporarily; changing euid from a privileged value (typically 0) to some unprivileged value (anything other than the privileged value) causes the privileged value to be stored in suid. Later, a program's euid can be set back to the value stored in suid, so that elevated privileges can be restored; an unprivileged process may set its euid to one of only three values: the value of ruid, the value of suid, or the value of euid.

Proces	A	B
RUID	2000	2000
EUID	0	1000
SUID	0	1000

Instrukcja:

```
setreuid(2000, 2000);
```

setreuid, setregid - set real and/or effective user or group ID

```
int setreuid(uid_t ruid, uid_t euid);
```

Unprivileged processes may only set the effective user ID to the real user ID, the effective user ID, or the saved set-user-ID.

Unprivileged users may only set the real user ID to the real user ID or the effective user ID.

If the real user ID is set (i.e., ruid is not -1) or the effective user ID is set to a value not equal to the previous real user ID, the saved set-user-ID will be set to the new effective user ID.