# Wykład 12
## Zarządzanie pamięcią (część III)
## oraz urządzenia sieciowe

# Funkcja mmap na pliku specjalnym

- Funkcje systemową mmap możemy również wywołać dla pliku specjalnym reprezentującym urządzenie (nie każdy sterownik musi ją implementować operacje mmap)
- Wynikiem tej funkcji jest odwzorowanie w przestrzeni adresowej wywołującego (poniżej granicy 3GB) procesu bufora urządzenia. Obowiązuje normalna semantyka mmap – proces może się odwoływać do bufora przy pomocy wskaźników.
- Może to być zarówno bufor sprzętowy tzn. pamięć znajdująca się na urządzeniu (np. bufor ramki) karty SVGA jak i programowy tzn. realizowany w pamięci operacyjnej komputera (np. bufor karty dźwiękowej).
  - Przykład: /dev/mem - pozwala na odwzorowanie całej pamięci fizycznej RAM (oczywiście tylko dla superużytkownika), wykorzystywane przez XServer.
- Główna różnica w porównaniu z operacją vremap polega na tym, że vremap wykonuje odwzorowanie w pamięci jądra, zmieniając tablice stron wszystkich procesów - odwzorowanie jest niedostępne z poziomu użytkownika. Z kolei operacja mmap modyfikuje tablice stron jednego procesu - i pamięć ta jest dostępna z poziomu użytkownika.

# Operacja mmap w strukturze file_operations

```
int (*mmap) (struct inode *, struct file *, struct vm_area_struct *);
```

- pola vm_start oraz vm_end zawierają początek obszaru, vm_offset przesunięcie pliku, vm_page_prot prawa dostępu
- Generalnie operację mmap możemy zaimplementować na dwa sposoby:
  - Z wykorzystaniem funkcji `int remap_page_range(unsigned long from, unsigned long offset, unsigned long size, pgprot_t prot)`. Funkcja ta odwzorowuje począwszy od adresu wirtualnego from obszar pamięci fizycznej rozpoczynający się od adresu offset, o długości size i o prawach dostępu prot.
  - Stworzyć własną tablicę metod obszaru pamięci (struct vm_operations_struct) i zaimplementować metodę nopage. Operacja mmap inicjalizuje wyłącznie wskaźnik na tablicę metod obszaru, a metoda napage wywoływana jest w przypadku braku strony.
- Należy pamiętać, że odwzorowanie pozostaje aktualne, także po zamknięciu pliku, w celu wykrycia usunięcia odwzorowania (np. licznik odniesień modułu, ewentualne zwolnienie pamięci) należy implementować metodę unmap.

# Przykład - urządzenie /dev/mem

```
static int mmap_mem(struct inode * inode, struct file * file, struct vm_area_struct *
   vma)

{
   if (vma->vm_offset & ~PAGE_MASK) // Nadmiarowe (sprawdzenie czy przesunięcie
       return -ENXIO;               // jest na granicy strony

   // Odwzorowanie pamięci fizycznej spod adresu offset
   if (remap_page_range(vma->vm_start, vma->vm_offset, vma->vm_end - vma->vm_start,
         vma->vm_page_prot))
            return -EAGAIN;
   vma->vm_inode = inode;
   inode->i_count++; // Przy usunięciu odwzorowania wywoływany jest algorytm iput
   return 0;
}
```

- Zwiększenie licznika odniesień zapobiega zwolnieniu i-węzła (struktury struct inode w pamięci) po zamknięciu pliku.

# Obsługa błędu strony

- Błąd strony może zostać wygenerowany przez jądro (błąd w jądrze) lub proces użytkownika. Bład może być spowodowany brakiem praw bądź brakiem strony w pamięci.
- Po wystąpieniu błędu strony wywoływana jest funkcja do_page_fault (./arch/i386/mm/fault.c). Rejestr CR2 procesora przechowuje adres, którego dotyczy odwołanie. Wykorzystując drzewa AVL jądro najpierw sprawdza (funkcja find_vma) czy próba odwołania dotyczy jakiegoś obszaru przestrzeni adresowej procesu. Jeżeli tak to:
  - Przy próbie odczytu z obszaru nie mającego prawa odczytu bądź wykonania następuje skok do procedury obsługi błędu (bad_area).
  - Przy próbie zapisu do obszaru nie mającego prawa zapisu również następuje skok do bad_area.
  - Uwaga: Prawa obszaru to co innego niż prawa strony np. Strona może być nieobecna (not present) lub zabezpieczona przed zapisem jako COW (Copy ON Write), podczas gdy obszar ma prawa do zapisu.
  - W przeciwnym wypadku wywoływana jest funkcja do_wp_page (oznacza próbę zapisu do strony obecnej ale chronionej przed zapisem) albo do_no_page (próba odczytu bądź zapisu nieobecnej strony).
- Powyższe kroki są powtarzane, jeżeli mamy do czynienia ze wzrostem stosu (po ewentualnym zwiększeniu jego rozmiaru)

# Sytuacja błędna (bad_area)

- Jeżeli błąd spowodował przez proces wykonywujący się w trybie użytkownika, to wyślij mu sygnał SIGSEGV.
- Jeżeli błąd został spowodowany przez proces wykonywujący się w trybie jądra, po wykonaniu wywoływania systemowego, to proces ten jest przerywany (Komunikat “Oooops”). Taka sytuacja zawsze wskazuje na błąd w jądrze, ponieważ proces nie ma prawa zmusić jądra to generowania błędnych adresów.
- Jeżeli błąd wewnątrz obsługi przerwania – to nie mamy czego zabijać – panika jądra.

# Funkcja do_no_page

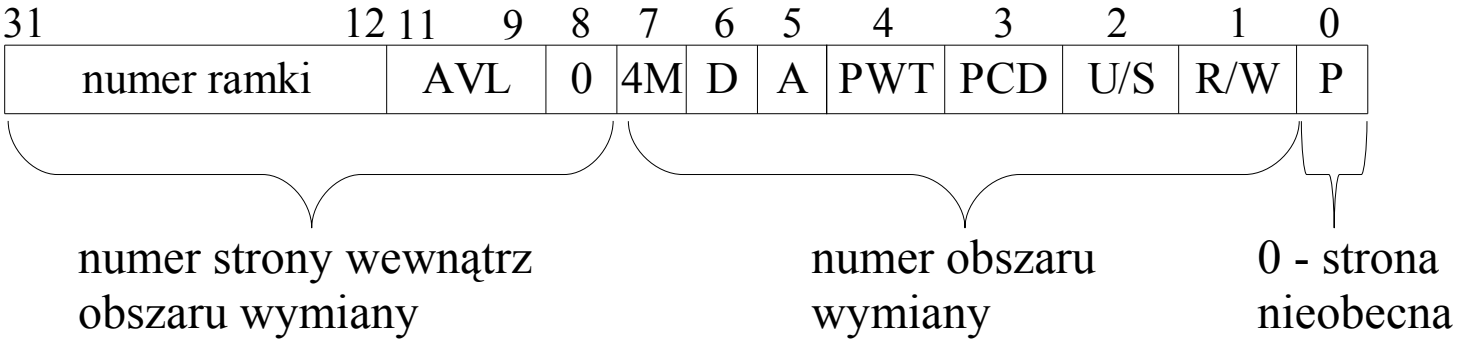
- do_no_page wywoływana jest w przypadku strony nieobecnej w pamięci (wyzerowany bit present), ale należącej do poprawnego obszaru pamięci
- Jeżeli obszar ma zdefiniowaną operację nopage, to jest ona wywoływana (i fub
- W przypadku obszaru, dla którego nie ma tablicy operacji lub też nie jest zdefiniowana operacja nopage, oraz pozycja w tablicy stron jest równa zeru, rozróżniane są dwa przypadki (etykieta anonymous page):
  - Przy odczycie mapowana jest specjalna ramka tylko do odczytu wypełniona samymi zerami (adres tej ramki to ZERO_PAGE).
  - Ciekawostka: gdyby zatem proces przydzielił np. 1GB pamięci, a następnie odczytał ten 1GB, to nie spowodowałoby to jakiejkolwiek alokacji pamięci.
  - Przy zapisie alokowana jest nowa strona (aby nie powodować kolejnego wyjątku stronicowania - ochrona przed zapisem).
- Jeżeli nie ma operacji nopage, a pozycja w tablicy stron jest różna zeru to wywoływana jest funkcja do_swap_page w celu sprowadzenia strony do pamięci.

# Funkcja do_wp_page

- do_wp_page obsługuje przypadek gdy obszar ma prawa do zapisu, jednak że strona jest zabezpieczona przed zapisem. Jest przypadek strony Copy On Write (strony współdzielonej i skopiowanej "na niby" po wykonaniu fork). Funkcja fork zabezpiecza stronę przed zapisem i zwiększa o jeden licznik referencji strony. Kopia strony jest wykonywana dopiero przy próbie zapisu przez jeden z procesów.
- Algorytm COW jest następujący (a) zaalokuj nową ramkę (b) zmniejsz o jeden licznik referencji współdzielonej strony i skopiuj ją do nowej ramki (c) zamapuj nową ramkę w pamięci adresowej procesu (oczywiście tym razem nowa ramka ma zezwolenie na zapis)
- Wykonywana jest oczywista optymalizacja w przypadku, gdy licznik referencji współdzielonej strony jest równy jeden.
  - Do takiej sytuacji dochodzi np. gdy (a) wywołane jest fork (b) jeden z procesów (potomny lub rodzicielski się kończy) (c) drugi proces próbuje wykonać zapis do strony.
  - W takiej sytuacji do_wp_page po prostu modyfikuje pozycję tablicy stron zezwalając na zapis.

# Sprowadzanie strony z obszaru wymiany

- Funkcja do_swap_page sprowadza stronę z obszaru wymiany. Jeżeli obszar pamięci ma zdefiniowaną metodę swapin to jest ona wywoływana.
- W przeciwnym wypadku strona jest sprowadzana z obszaru wymiany. Obszarem wymiany może być zwykły plik dyskowy lub partycja (oczywiście wykorzystanie z partycji prowadzi do dużo lepszej wydajności)
- Pozycja tablicy stron, dla strony zapisanej w obszarze wymiany, w architekturze x86 interpretowana jest następująco:




- Zatem maksymalnie możemy mieć 128 obszarów wymiany, a każdy 2^24 strony, co daje łącznie 256 GB w każdym obszarze.

# Pamięć podręczna stron (ang. page cache)

- Realizuje ona współdzielenie stron w przypadku plików odwzorowanych w pamięci. (Załadowanie programu lub biblioteki to prywatne odwzorowanie w pamięci).
- Wszelkie żądania odczytu stron z odwzorowanego pliku (w wyniku błędu braku strony - operacja nopage) przechodzą przez tą pamięć.
- Wyszukiwanie jest oparte na tablicach mieszających, a kluczem jest para <adres_i-węzła, przesunięcie_w_pliku>.
- W przypadku, odczytu strony będącej już w pamięci podręcznej nie jest wykonywany odczyt z dysku.
  - W przypadku gdy błęd braku strony spowodowała operacja zapisu i mamy do czynieniem z odwzorowaniem prywatnym wykonywana jest kopia strony z pamięci podręcznej => COW
  - W pozostałych przypadkach strona z pamięci jest odwzorowywana w przestrzenie procesu (zabezpieczona przed zapisem=> COW).
- Uwaga: odczyt z pliku przy pomocy funkcji generic_file_read, wykorzystuje również pamięć podręczną stron (***a nie pamięć podręczną buforową !!!***) => generic_readpage, generic_file_mmap.
  - Z tych trzech funkcji możemy skorzystać jeżeli zaimplementowana jest mmap bmap i-węzła.

# Urządzenia sieciowe

- Urządzenie sieciowe jest w pewnym stopniu podobne do urządzenia blokowego: wysyła dane w pakietach (ramkach), chociaż pakiety mają zmienną długość.
  - Na przykład ramka Ehernet
- Najważniejsza różnica: w przypadku urządzenia sieciowego odbiór ramek następuje w wyniku ich nadejścia z zewnątrz.
  - Jądro prosi urządzenie blokowe o odczyt danych.
  - Urządzenie sieciowe prosi (na ogół w wyniku przerwania) jądro o odbiór danych
- Urządzenia sieciowe nie mają plików specjalnych oraz numerów podrzędnych i nadrzędnych.
- Rejestracja (i usunięcie) urządzenia sieciowego przy pomocy funkcji:

```
int register_netdev(struct device *dev);

void register_netdev(struct device *dev);
```

# Struktura device - fragmenty

```
struct device
{
   char *name; // nazwa urządzenia;


 volatile unsigned char  start, interrupt;
 unsigned long tbusy;
```

- Powyższe trzy pola służą do synchronizacji pomiędzy urządzeniem i jądrem. start=1 oznacza, że urządzenie może wysyłać (i odbierać) dane. Interrupt powinno być ustawiane na 1 (przez sterownik) na czas obsługi przerwania. tbusy=1 oznacza, że urządzenie prosi jądro o nieprzesyłanie mu dalszych ramek.

```
  int (*init)(struct device *dev);
```

- Adres do funkcji inicjalizującej urządzenie

```
unsigned short flags;
```

- Flagi interfejsu.
- Ponadto struktura zawiera masę pól związanych z konkretnymi typami intefejsów.

# Metoda init

- Powinna wywołać funkcję ether_setup(dev) – dla urządzenia Ethernet.
- Powinna ustalić adresy pozostałych metod w strukturze device. Najważniejsze z nich to:

```
int (*open)(struct device *dev);
int (*stop)(struct device *dev);
int (*hard_start_xmit) (struct sk_buff *skb, struct device *dev);
```

- Open wywoływane jest po otwarciu urządzenia (polecenie ifup). Rezerwacja DMA, przerwań, zwiększenie licznika odniesień modułu.
  - Oraz ustawienie pola start na jeden.
- Stop jest odwrotnością open
- Niektóre opcjonalne metody, to:

```
int (*do_ioctl)(struct device *dev, struct ifreq *ifr, int cmd);
//Zmiana adrsu karty sieciowej
int (*set_mac_address)(struct device *dev, void *addr);
// Zmiana maksymalnej długości pakietu
int (*change_mtu)(struct device *dev, int new_mtu);
```

# Funkcja hard_start_xmit

- Wywoływana jest w celu inicjalizacji wysyłania pakietu.
- Na ogół karta ma sprzęt (np. wewnętrzny kontroler DMA albo pamięć RAM) pozwalający na kolejkowanie pakietów zgłoszonych przez oprogramowanie – Nie trzeba czekać ze zgłoszeniem kolejnego pakietu na zakończenie transmisji poprzedniego.
- W związku z tym jądro może kolejny raz wywołać funkcję hard_start_xmit zaraz po jej powrocie.
- Problemem powstaje, gdy sprzętowa kolejka się przepełni.
  - Ustawienie flagi tbusy na jeden sprawia, że jądro nie będzie zgłaszało nowych pakietów do wysłania.
  - Aby przywrócić możliwość wysyłania sterownik musi ustawić tbusy na zero i wywołać funkcję mark_bh (NET_BH).
  - Jeżeli jądro dojdzie do wniosku, że karta się zawiesiła (ang. trasmitter lockup), to i tak może wywołać funkcję hard_start_xmit.

# Struktura sk_buff (socket buffer)

- Generalnie przechowuje dane do wysłania, sprzętowy nagłówek (adres etc). Struktury sk_buff są alokowane przez jądro.
- Przy wysyłaniu: hard_start_xmit otrzymuje wskaźnik do struktury sk_buff. Po pomyślnym zakończeniu transmisji sterownik musi zwolnić strukturę.
  - dev_kfree_skb
- Przy odbiorze: sterownik musi zaalokować strukturę i wypełnić ją niezbędnymi danymi i przekazać jądru.
  - dev_alloc_skb.
- Struktura zawiera wiele pól, w tym sprzętowy nagłówek oraz dane do wysłania (data, len).
- Ze względu na zagnieżdżanie ramek wielu protokołów nie muszą się rozpoczynać od początku bufora.

# Obsługa przerwania karty

- Pomyślne wysłanie (odbiór) pakietu bądź grupy pakietów jest potwierdzone zgłoszeniem przerwania.
- W przypadku potwierdzenia wysłania sterownik ma obowiązek zwolnić bufory sk_buf wszystkich pakietów, których dotyczy potwierdzenie (funkcja dev_kfree_skb).
- W przypadku potwierdzenia odbioru sterownik powinien:
  - Zaalokować (dev_alloc_skb) bufor. Alokacja następuje wewnątrz przerwania i może się nie powieść – zgubienie ramki.
  - Skopiować dane do bufora i wypełnić kilka pól informacyjnych (numer protokołu, urządzenie)
  - Wywołać funkcję netif_rx aby przekazać bufor jądru.
- Widzimy, że trudno zaimplementować sterownik urządzenia sieciowego nie wykorzystując przerwań – można użyć timerów jądra.