Systemy operacyjne II

Wojciech Kwedlo
Wydział Informatyki PB, p. 205
wkwedlo@ii.pb.bialystok.pl
aragorn.pb.bialystok.pl/~wkwedlo

Pracownia specjalistyczna: Wojciech Kwedlo
Krzysztof Bandurski

Treść zajęć

- W semestrze pierwszym nastąpiło przedstawienie teoretycznych podstaw działania systemów operacyjnych.
- Semestr drugi poświęcony jest przedstawieniu jądra *rzeczywistego* systemu operacyjnego. Systemem tym jest Linux 2.0.36.
- Dlaczego taka stara wersja? 2.0.36 to oczywiście jądro przestarzałe z punktu widzenia bieżącego stanu Linuksa. Jednak:
  - Prosta (w porównaniu z najnowszymi wersjami) postać jądra ułatwi jego zrozumienie przez studentów.
  - Niewielkie wymagania sprzętowe pozwalają na w miare bezpiecznie uruchomienie na emulatorze (w naszym przypadku qemu).
  - Jednocześnie nawet te przestarzałe jądro spełnia wszystkie wymagania stawiane dorosłemu systemowi operacyjnemu.
  - Nie wykluczamy możliwości przejścia na nowsze jądra w przyszłości.
- Egzamin: Kod prostego sterownika urządzenia + pytania testowe
- Pracownia specjalistyczna: jedno zadania wprowadzające i dwaprojekty.

Literatura

Bardzo dużo pozycji na temat jądra Linux-a, poniższe dotyczą jądra 2.0.x

- M. Beck, H. Bohm, M. Dziadzka, U. Kunitz, R. Magnus, D. Verworner, Linux Kernel - Jądro systemu, wydanie II, Wydawnictwo MIKOM, Warszawa, 2000.
- A. Rubini. Linux sterowniki urządzeń.
  - Pierwsze wydanie dotyczy starszego jądra, drugie dostępne w sieci, niestety po angielsku.
- D. A. Rusling. The Linux kernel.
- Strona www dr Janiny Mincer-Daszkiwicz z Instytutu Informatyki UW.

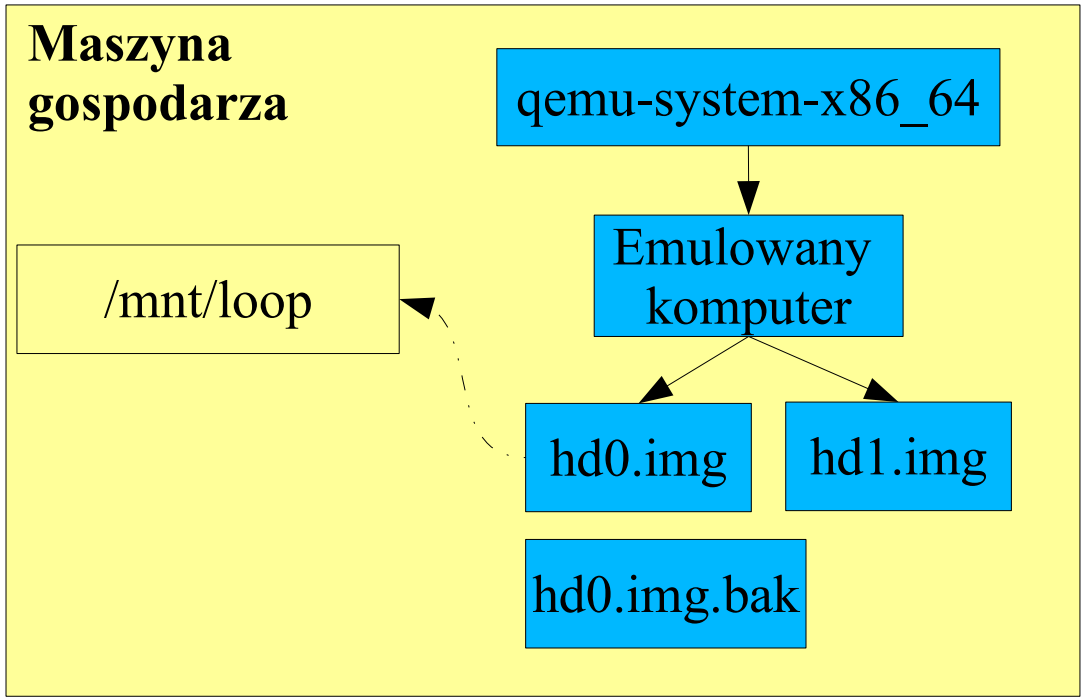
Plan wykładu

- Organizacja kodu jądra.
- Procesy: structura *task_struct*, stany procesów, kolejki procesów, sygnały.
- Sterowniki urządzeń, urządzenia blokowe i znakowe, moduły jądra.
- Algorytm szeregowania procesów, funkcja *schedule.*
- Implementacja wywołań systemowych.
- Zarządzenie pamięcią: alokator pamięci jądra – funkcja *kmalloc*, stronicowanie, pamięć wirtualna, pamięć podręczna stron.
- Pamięć podręczna buforów.
- Systemy plików: mechanizm VFS, system plików ext2.
- Linux 2.0 na architekturach SMP.
- Dalsze wersje jądra.

Pracownia specjalistyczna

- Realizacja 2 projektów polegających ma wprowadzeniu znacznych modyfikacji w jądrze. Przykładowe tematy:
  - Implementacja nowego systemu plików.
  - Implementacja sterownika urządzenia.
  - Implementacja nowego mechanizmu zastępowania stron.
- Realizowane na emulowanej maszynie (emulator qemu wraz z maszyną maszyną wirtualną z zainstalowanym Linuxem pobrać od prowadzących).
  - Zajęcia w systemie Linux, sale 222 oraz 223m, architektura x86_64, emulator pracuje także pod Windows [powodzenia :)]
- Realizacja projektu może być związana z przygotowaniem raportu opisującego podsystemy jądra, których będzie dotyczył dany projekt

Konfiguracja maszyny wirtualnej w laboratoriach 222,223



- hd0.img – główny system plików i partycja swap. Można zamontować na /mnt/root na maszynie gospodarze.
- hd1.img – dysk z kopią źródeł Linuksa tylko do odczytu.

- Nie wolno jednocześnie mieć zamontowanego podkatalogu w /mnt/loop i uruchamiać emulator.
- hd0.img.bak – przegrać na hd0.img jak emulowany dysk „nie nadaje się do użytku” :)
- hd1.img – tu jest kod źródłowy jądra, tylko do odczytu. Wszelkie patche robimy względem tego katalogu.

Linuks to system

- Wieloprogramowy – w pamięci może być wiele programów (procesów).
- Z ochroną – procesy są przed sobą chronione.
- Z wielodostępem (wielu użytkowników może pracować jednocześnie).
- Z pamięcią wirtualną realizowaną przez stronicowanie na żądanie.

Większość systemów Uniksowych udostępnia te funkcje.

Pozycja jądra w systemie

- Każdy proces może wykonywać się w trybie jądra lub użytkownika.
  - Tryb użytkownika: proces wykonuje instrukcje swojego programu.
  - Tryb jądra: Proces zażądał przez jądro wykonania usługi np. wywołał funkcję open (otwarcie pliku).
- Tryb jądra jest trybem uprzywilejowanym. Pewne instrukcje np. blokada przerwań są dopuszczalne tylko w jądrze.
- Procesy w trybie użytkownika są chronione przed sobą. Awaria kodu jądra -> awaria systemu jako całości.
- (Tylko) jądro obsługuje przerwania (wywołanie funkcji systemowej to też przerwanie, tylko programowe).
- Idealnie procesy powinny korzystać z urządzeń wejścia-wyjścia (komunikacja z portami) za tylko pomocą jądra, ale Linux pozwala na obejście (dla użytkownika uprzywilejowanego).
- Możemy powiedzieć, że jądro
  - Izoluje procesy od sprzętu
  - Dostarcza wygodnych abstrakcji i usług (np. plik, sam proces to też abstrakcja).

Synchronizacja jądra

- W jądrze może przebywać wiele procesów. Na przykład Proces A wywołuje funkcje read aby odczytać dane z pliku, wykonuje się w trybie jądra, trzeba zaczekać aż dane nadejdą (transmisja DMA), więc ***jest usypiany***, planista przełącza procesor do procesu B, a ten również wchodzi do jądra. Musimy więc zadbać o synchronizację.
- Jądro Linuksa 2.0.x (ale późniejsze już nie), oparte jest na klasycznej zasadzie: Kod jądra jest niewywłaszczalny. Procesowi, który wykonuje się w trybie jądra ***nie zostanie odebrany procesor w celu przekazania innemu procesowi***.
  - Proces tylko może dobrowolnie zrzec się procesora, np. uśpić się jak w powyższym scenariuszu.
  - Ponadto proces może być przerwany przerwaniem (ale cli() przed tym zabezpiecza).
- Upraszcza to znacznie synchronizację jądra. Wystarczy dbać:
  - o blokowanie przerwań gdy handler przerwania odwołuje się do tej samej struktury danych co aktualna ścieżka kodu jądra.
  - O pozostawienie danych w stanie spójnym, jeżeli proces w trybie jądra będzie usypiany.
- Jest to główna przyczyna stosowanie tej wersji jądra na zajęciach. W wersjach (2.2.x i późniejszych jest o wiele trudniej).
- Uwaga: czasami uśpić proces mogą pozornie niewinne funkcje jądra.

Zanim zaczniemy coś zmieniać w jądrze należy pamiętać, że

- Nie mamy dostępu do funkcji biblioteki standardowej języka C – jednakże część z tych funkcji została zaimplementowana np. funkcje z rodziny *memcpy*.
- Błędy w jądrze mogą spowodować:
  - awaryjne przerwanie procesu (Oops) jeżeli proces ten wykonywał się w trybie jądra. Na przykład gdy proces wywołał funkcje systemową.
  - Panikę jądra (kontrolowane zawieszenie), po której musimy zrestartować maszynę.

Organizacja kodu jądra

- ./kernel zasadnicza część jądra, szeregowanie procesów: funkcja *schedule*.
- ./mm – zarządzanie pamięcią, stronicowanie, alokator pamięci jądra kmalloc/kfree.
- ./init funkcje wykonywane przy starcie jądra np. *start_kernel*
- ./fs implementacja VFS (wirtualny system plików). Podkatalogi w ./fs zawierają implementacje poszczególnych systemów plików
- ./devices sterowniki urządzeń.
- ./net – obsługa sieci.
- ./arch/i386 – część systemu zależna od architektury.
  - ./arch/i386/kernel – funkcje jądra (np. obsługa przerwań)
  - ./arch/i386/mm – zarządzanie pamięcią.
  - /arch/i386/boot – tu trafia jądro (bzImage) po kompilacji

Pożyteczne funkcje - printk

- Odpowiednik *printf*, jednakże nie obsługuje formatów zmiennoprzecinkowych – których i tak nie wolna używać w jądrze.
- include <*linux/kernel.h*>
- Przykład:

  printk("X jest rowne %d\n",x);

- Można porzedzić symbolem określającym poziom komunikatu, komunikaty o poziomie niższym od poziomu progowego zawsze trafiają na konsolę.

  Na przykład

  printk(KERN_EMERG "X jest rowne %d\n",x);

  trafi zawsze na konsolę

Pożyteczne funkcje: kmalloc/kfree

- Odpowiadają za alokację/dealokację pamięci.

*void * kmalloc(size_t size, int priority)*

- rozmiar nie może być większy od 128KB.
- Priorytet
  - GFP_KERNEL zakłada, że jądro wykonuje się w kontekście jakiegoś procesu, proces może zostać uśpiony, jeżeli brak pamięci.
  - GFP_ATOMIC – wywoływane z procedury obsługi przerwania, może zakończyć się niepowodzeniem

*void kfree (void *ptr)*