Spis treści

Rozdział 1

Metody wołania funkcji systemowych

Rozdział zawiera opis mechanizmu wołania funkcji systemowych w linuksie - jądro wersji 2.2.17. Autorem rozdziału jest Michał Sapiński.

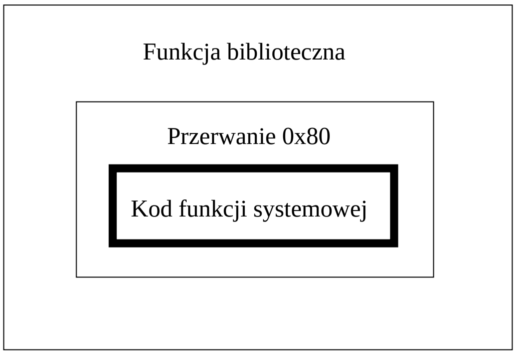
1.1 Wprowadzenie

Wywołanie funkcji systemowej nie odbywa się bezpośrednio. Musi istnieć mechanizm pośredniczący pomiędzy wywołaniem w programie użytkownika a rzeczywistym wykonaniem kodu funkcji systemowej. Jest to konieczne ze względu na potrzebę rozróżnienia pomiędzy użytkownikiem a jądrem choćby ze względu różne uprawnienia i względy bezpieczenstwa. Ogólnie można powiedziec, że jądro wykonuje funkcje systemową w imieniu programu użytkownika. W linuxie rozwiązanie polega na traktowaniu takiego wywołania jako przerwania. Chcąc wywołac funkcję zapisujemy w odpowiednich rejestrach lub na stosie argumenty a następnie wykonujemy przerwanie o numerze `0x80`. Samo przejście do trybu jądra odbywa się więc wykorzystując mechanizm przerwań. Wiadomo, że pisząc program np. w C nie wywołujemy bezposrednio przerwania lecz korzystamy z funkcji systemowej jak ze zwykłej funkcji bibliotecznej. Możemy tak robić bo większosc funkcji systemowych obudowana jest w zewnętrzną funkcją, której argumenty przekazywane są w zwykły sposób. Pokazane jest to na rysunku 1.1

1.2 Wywołania systemowe

1.2.1 Makro `_syscall`

- Użytkownik wywołując funkcję systemową korzysta z funkcji bibliotecznej. Funkcja ta tworzona jest zwykle za pomoca makr `_sycall` zdefiniowanych w `include/asm-i386/unistd.h.`
- Makra `_syscall` tworzą rodzinę - dla każdej liczby argumentów zdefiniowane jest oddzielne makro.
- Makro odkłada na stos przekazane argumenty oraz numer funkcji do wywołania i wywołuje przerwanie 0x80. Następnie w przypadku wystąpienia błedu zwraca `-1` i ustawia zmienną



Rysunek 1.1: Struktura wywołania

globalną `errno` na odpowiednia wartość. W przeciwnym wypadku zwraca wartość otrzymaną z podwywołania.

- Może zdarzyć się, że jedna funkcja systemowa obudowana jest kilkoma funkcjami bibliotecznymi. Dzieje się tak na przykład w przypadku rodziny funkji systemowych `exec`.

Poniżej znajduje się treść makra `_syscall`. Jedynka w `_syscall1` oznacza, że jest to makro dla funkcji systemowej o jednym argumencie. Analogiczne makra zdefiniowane są dla innych ilości argumentów. W tabelce 1.1 znajduje się opis argumentów makra.

nazwa	znaczenie
`type`	typ zwracanego argumentu
`name`	nazwa funkcji
`type1`	typ pierwszego argumentu
`arg1`	nazwa pierwszego argumentu

Tablica 1.1: Argumenty makra `_syscall1`

```
#define _syscall1(type,name,type1,arg1)
type name(type1 arg1) {
  long __res;
  __asm__ volatile (
   ‘"{}int \$0x80‘"{}
    : ‘"{}=a‘"{} (__res)
    : ‘"{}0‘"{} (__NR_##name),‘"{}b‘"{} ((long)(arg1)));
    __syscall_return(type,__res);
}
```

Powyżej użyte jest makro `__syscall_return`, które tworzy fragment kodu zwracający wynik wywołania funkcji systemowej. W tabelce 1.2 znajduje się opis jego argumentów.

nazwa	znaczenie
`type`	typ wyniku
`res`	wartość wyniku

Tablica 1.2: Argumenty makra `__syscall_return`

```
#define __syscall_return(type, res)
do {
  if ((unsigned long)(res) >= (unsigned long)(-125)) {
    errno = -(res);
    res = -1;
  }
  return (type) (res);
} while (0)
```

Widać tutaj mechanizm zwracania kodu błedu poprzez zmienną globalną `errno`.

1.2.2 Własciwe wywołanie

Rzeczywiste wywołanie funkcji systemowej odbywa się dzięki wywołaniu **funkcji obsługi przerwania** `(0x80)`. Kod tej funkcji znajduje sie w pliku `/arch/i386/kernel/entry.S` rozpoczynając od `_system_call()`. W pliku tym zdefiniowana jest rownież tablica `sys_call_table` zawierająca przypisania funkcji systemowych dla odpowiednich numerów. **Algorytm 1** przedstawia funkcję obsługującą przerwanie `0x80`. Należy pamiętać, że jest ona także wykonywana w przypadku powrotu z przerwań nie związanych z wywołaniem funkcji systemowej (od momentu obsługującego dolne części przerwań - linia 13). Plik `entry.S` zapisany jest w asemblerze więc siłą rzeczy **algorytm 1** nie odzwierciedla dokładnie źródeł. Żeby dokładniej zrozumnieć co się dzieje w funkcji obsługi przerwania `0x80` można zapisać ją w pseudokodzie C (na podstawie [Beck 99]).

```
system_call(sys_call_num, sys_call_args) {
/* sys_call_num oznacza numer funkcji
   (zdefiniowany w unistd.h) */
  _system_call:
    _SAVE_ALL; /* zapisz rejestry procesu */
    if (sys_call_num >= NR_syscalls)
      errno = -ENOSYS;
    else {
      if (current->flags & PF_TRACESYS) {
        /* proces jest monitorowany przez
           swój proces macierzysty */
        syscall_trace();
        /*wywolanie funkcji systemowej*/
        errno = (*sys_call_table[sys_call_num])(sys_call_args);
        syscall_trace();
      }
```

Algorytm 1 Funkcja obsługi przerwania `0x80`

Dane wejściowe: numer funkcji, argumenty funkcji
Wynik: errno

```
 1: zapisz rejestry procesu
 2: jeśli numer funkji większy od ilości funkcji to
 3:   ustaw errno na -ENOSYS
 4: w przeciwnym przypadku
 5:   jeśli proces jest monitorowany to
 6:     wywołaj funkcję monitorującą
 7:     wywołaj funkcję sytemową
 8:     wywołaj funkcję monitorujacą
 9:   w przeciwnym przypadku
10:     wywołaj funkcję sytemową
11:   koniec jeśli
12: koniec jeśli
13: jeśli nastapiło przerwanie w przerwaniu to
14:   zwróć sterowanie - skok do linii 28
15: koniec jeśli
16: jeśli trzeba obsłużyć dolną część przerania to
17:   zwiększ zmienną przechowującą zagnieżdzenie przerwań
18:   włącz przerwania
19:   obsłuż dolną część przerwania
20:   zmniejsz zmienną przechowującą zagnieżdzenie przerwań
21: koniec jeśli
22: właczą przerwania
23: jeśli trzeba uruchomić program szeregujący to
24:   uruchom program szeregujący
25:   skocz do linii 13
26: koniec jeśli
27: obsłuż niezablokowane sygnały
28: przywróc rejestry procesu
```

```
    else
      /*wywołanie funkcji systemowej*/
      errno = (*sys_call_table[sys_call_num])(sys_call_args);
  }
```

Wywołanie funkcji systemowej juz nastapiło. Nastepnie wykonywana jest funkcja `ret_from_sys_call()`, która wykonuje sie nie tylko po wywołaniach systemowych ale także po przerwaniach, zawiera więc także instrukcje nie związane z wywołaniami funkcji systemowych.

```
ret_from_sys_call:
  if (intr_count)
    goto exit_now:
    /* w przypadku gdy nastąpiło ‘‘przerwanie w przerwaniu’’
      ret_from_sys_call natychmiast zwraca sterowanie */
```

Obsługa dolnych czesci przerwań jeśli zachodzi potrzeba:

```
    if (bh_mask & bh_active) {
  handle_bottom_half;
      ++intr_count;
      sti();
      do_bottom_half();
      --intr_count;
    }
    sti();
```

Jesli zostało zamówione szeregowanie wywołany jest program szeregujący:

```
  if (need_resched)
  {
reschedule:
    schedule;
    goto ret_from_sys_call;
  }
```

Obsługiwanie odebranych sygnałów:

```
  if (current->signal & ~{}current->blocked)) {
signal_return:
    do_signal();
  }
```

Następnie wywołanie systemowe lub przerwanie zwraca sterowanie.

```
  exit_now:
    RESTORE_ALL; /*przywróć rejestry procesu */
}
```

1.3 Pliki

1. `arch/i386/kernel/head.S` - rozpoczyna pracę w trybie wirtualnym (protected mode)
2. `arch/i386/kernel/entry.S` - niskopoziomowe interfejsy do funkcji systemowych i podprogramów obsługi wyjątków
3. `include/asm-i386/unistd.h` - definicja numerów funkcji systemowych i makra do generowania interfejsów do funkcji systemowych.

Literatura

[Beck 99] M.Beck, H.Bochme, M.Dziacka, U.Kunitz, R.Magnus, D.Verworner : *Linux kernel. Jądro systemu*, MIKOM 1999

[Bach 95] M. Bach : *Budowa systemu operacyjnego Unix*, WNT, 1995.

[Silberschatz 2000] A.Silberschatz, P.Galvin : *Podstawy systemow operacyjnych*, WNT 2000