Ochrona funkcji systemowych – funkcje bliźniacze

1. Wstęp

Współczesne systemy operacyjne, w tym Linux, wymagają skutecznego zarządzania pamięcią, aby zapewnić stabilność, efektywność i bezpieczeństwo działania aplikacji. Dynamiczne przydzielanie pamięci jest kluczowym elementem w programowaniu systemowym i aplikacjach operujących na niskim poziomie. Funkcje takie jak `kmalloc`, `kmalloc_array` i `kcalloc` w jądrze Linux są fundamentalne do tego procesu, umożliwiając programistom elastyczne zarządzanie pamięcią w czasie rzeczywistym.

Rola funkcji funkcji bliźniaczych

Kmalloc Jest to podstawowa funkcja do dynamicznego przydzielania pamięci w jądrze Linux. Pozwala na alokację jednego bloku pamięci o określonym rozmiarze. Jest szeroko stosowana w obszarach takich jak przechowywanie danych, zarządzanie sterownikami i obsługa plików, gdzie konieczne jest dynamiczne przydzielanie i zwalnianie zasobów pamięciowych.

Kmalloc_array rozszerza możliwości kmalloc poprzez umożliwienie alokacji tablicy o określonym rozmiarze. Jest używana do przechowywania danych w postaci tablicy, co pozwala na efektywne zarządzanie zmienną liczbą elementów w czasie działania programu.

Kcalloc Jest odmianą kmalloc_array, która dodatkowo inicjuje zaalokowaną pamięć zerami. Jest szczególnie przydatna do tworzenia tablic, które wymagają początkowej inicjalizacji zerami, co zapewnia bezpieczeństwo danych i eliminuje ryzyko uzyskania przypadkowych wartości i lub tzw. Wycieku pamięci.

2. Przegląd funkcji bliźniaczych

Zestawienie funkcji bliźniaczych

Funkcja	Sygnatura	Parametry
Kmalloc	void *kmalloc(size_t size, gfp_t flags);	size, flaga
Kmalloc_array	void *kmalloc_array(size_t n, size_t size, gfp_t flags);	n, size, flaga
Kcalloc	void *kcalloc(size_t n, size_t size, gfp_t flags);	n, size, flaga

Kmalloc:

Jest to podstawowa funkcja do alokacji pamięci w Linuxie.

Praktyczne zasotowania:

- **Przechowywanie danych**: Alokacja pamięci na struktury danych, listy, bufory.
- **Obsługa sterowników**: Przechowywanie kontekstów sterowników, buforowanie danych wejściowych/wyjściowych.
- **Obsługa plików**: Bufory na dane do odczytu/zapisu.

Kmalloc_array:

Jest rozszerzeniem kmalloc i służy do alokacji dynamicznej tablicy o określonym rozmiarze.

Praktyczne astosowania:

- **Tablice dynamiczne**: Przechowywanie listy elementów o zmiennej liczbie.
- **Zarządzanie pamięcią**: Alokacja pamięci dla dynamicznych struktur danych, np. list wiązanych.

Kcalloc:

kcalloc to specjalna odmiana kmalloc_array, która dodatkowo zeruje zaalokowaną pamięć. Mówiąc krótko stosując ten typ alokacji tablicowej, dostajemy zaalokowany obszar pamięci przeznaczony na typ tablicowy, którego wszystkie elementy są inicjalizowane zerami już podczas kompilacji.

Praktyczne zastosowania:

- **Inicjalizacja zerami**: Tworzenie tablic, które wymagają początkowej inicjalizacji zerami.

- **Bezpieczeństwo danych**: Zapewnienie, że pamięć jest czysta i nie zawiera danych z poprzednich alokacji.

Charakterystyki i sygnatury funkcji bliźniaczych:

- Kmalloc:

  sygnaturaL void * kmalloc(size_t size, gfp_t flags)

  sieze_t: oznacza rozmiar alokowanej pamięci w bajtach

  flags: flagi sterujące, określają m.in. rodzaj pamięci, którą chcemy zaalokować

  Funkcja zwraca wskaźnik na zaalokowaną pamięć lub NULL w przypadku niepowodzenia

- Kmalloc_array

  sygnaturaL void * kmalloc_array(size_t n, size_t size gfp_t flags)

  sieze_t: oznacza rozmiar alokowanej pamięci w bajtach

  n: liczba elementów w tablicy

  sieze: rozmiar pojedynczego elementu

  flags: flagi sterujące, określają m.in. rodzaj pamięci, którą chcemy zaalokować

  Funkcja zwraca wskaźnik na zaalokowaną pamięć lub NULL w przypadku niepowodzenia

- Kcalloc

  sygnaturaL void * kcalloc (size_t n, size_t size gfp_t flags)

  sieze_t: oznacza rozmiar alokowanej pamięci w bajtach

  n: liczba elementów w tablicy

  sieze: rozmiar pojedynczego elementu

  flags: flagi sterujące, określają m.in. rodzaj pamięci, którą chcemy zaalokować

  Funkcja zwraca wskaźnik na zaalokowaną **wypełnioną zerami** tablicę pamięci lub NULL w przypadku niepowodzenia

Flagi i ich znaczenie:

W funkcjach `kmalloc`, `kmalloc_array` i `kcalloc` w jądrze Linux, istotną rolę odgrywają flagi, które określają sposób alokacji pamięci. Flagi te są przekazywane jako parametry do funkcji alokujących i wpływają na zachowanie procesu przydzielania pamięci. Zrozumienie tych flag jest kluczowe dla prawidłowego i efektywnego zarządzania pamięcią w jądrze Linux. W naszym projekcie przyjrzymy się dwom flagom, mianowicie: GFP_kernel oraz GFP_atomic.

1. **GFP_KERNEL**

   Flaga ta jest używana, gdy alokacja pamięci jest wykonywana w kontekście jądra, gdzie można bezpiecznie wywołać funkcje, które mogą być w trybie sleep. Ponadto ta flaga nie może być wykorzystywana, gdy używamy przerwań systemowych podczas alokacji.

   Zastosowanie:
   Jest to najczęściej używana flaga dla alokacji pamięci w większości części jądra.

   Zachowanie:
   Funkcja może blokować wykonanie i czekać na dostępność pamięci, co zapewnia wysoką szansę na powodzenie alokacji (wywołanie alokacji funkcji nie zwróci nam wartości NULL).

2. **GFP_ATOMIC**

   Flaga ta jest używana w kontekstach, gdzie alokacja pamięci nie może być w trybie sleep, czyli w sekcjach kodu, które muszą być wykonywane szybko i nieprzerwanie. Pozwala na zastosowanie przerwań systemowych.

   Zastosowanie:
   Używana w kontekstach przerwań i innych miejscach, gdzie alokacja musi być natychmiastowa.

   Zachowanie:
   Funkcja nie może blokować wykonania. Jeśli pamięć nie jest dostępna, alokacja może się nie powieść, co oznacza mniejsze szanse na powodzenie w porównaniu z GFP_KERNEL – możliwe zwrócenie wartości NULL przy próbie alokacji.

3. Niewłaściwe wykorzystanie funkcji bliźniaczych i konsekwencje

Niewłaściwe korzystanie z funkcji kmalloc_array i kcalloc

Używanie kmalloc zamiast kmalloc_array lub kcalloc do alokacji tablicy może prowadzić do błędnych obliczeń rozmiaru, co zostało omówione w punkcie poniżej.

Drugą potencjalną przyczyną może być ignorowanie potrzeby inicjalizacji pamięci zerami, co jest zapewniane przez kcalloc.

Jeśli chodzi o skutki udało się znaleźć dwa:

Nadpisywanie pamięci: Błędne obliczenia rozmiaru mogą prowadzić do nadpisywania sąsiednich obszarów pamięci.

Niezainicjalizowane wartości: Używanie nieprzydzielonej pamięci, która może zawierać dane z poprzednich alokacji, prowadzi do nieprzewidywalnych błędów.

Niewłaściwe zarządzanie rozmiarem alokowanej pamięci

Jedną z przyczyn niewłaściwego zarządzania rozmiarem alokowanej pamięci jest błędne obliczanie liczby bajtów do alokacji.

Natomiast w przypadku alokacji tablicowych (kmalloc_array, kcalloc) może być nieprawidłowe uwzględnienie rozmiaru elementów w tablicy.

Jedną z konsekwencji jest przepełnienie bufora. Alokacja mniejszej ilości pamięci niż wymagana prowadzi do nadpisywania sąsiadujących obszarów pamięci, co może spowodować awarie systemu lub błędy logiczne. Natomiast alokowanie większej ilości pamięci niż potrzebna prowadzi do marnotrawienia zasobów i może skutkować niedoborem pamięci dla innych procesów.

Nieprawidłowe zarządzanie wskaźnikami

Wynika bezpośrednio z użycia niezainicjalizowanych wskaźników lub błędnego obliczenia wskaźników co skutkuje do dostępu poza przydzielony obszar – wyciek pamięci.

Jedną z konsekwencji jest – Segmentation fault, mianowicie próbując uzyskać dostęp do nieprzydzielonej pamięci, program może spowodować błąd ochrony pamięci, co prowadzi do natychmiastowej awarii programu.

Kolejną konsekwencją jest tzw. Korupcja danych. Oznacza dostęp do niewłaściwych obszarów pamięci, który może prowadzić do nadpisywania i uszkadzania danych innych procesów.