Konrad Gotfryd

STER

Dokumentacja projektu

Temat projektu: Sterownik do komunikacji IPC

Sterownik testowany był na dwóch komputerach o platformie x86_64, na systemach ArchLinux opartych na kompilowanych z kodów źródłowych jądrach Linux 4.1.2 i 4.2.0 oraz na systemie Knoppix V7.4.2 (jądro 3.16.3).

Razem z plikami źródłowymi dołączone są pliki Makefile – celem kompilacji należy wpisać jedynie make w obu folderach. Po załadowaniu sterownika należy zmienić uprawnienia dostępu do pilku urządzenia przez komendę „chmod a+rw /dev/events”.

Programy testowe zostały przygotowane tak, by działały przy standardowych parametrach zmiennych globalnych, z wyjątkiem dwóch programów: **test_glob_compl_cnt_max i test_glob_proc.** Programy testowe przy błędnych parametrach mogą działać nieprawidłowo. Każdy program testowy prezentuje na początku scenariusz testu.

Algorytm wykrywaniania zakleszczenia:

Zastosowano algorytm rekurencyjny. Algorytm ma za zadanie sprawdzić, czy istnieje wątek taki, który jednocześnie:

- Nie czeka jeszcze na żadne zdarzenie.
- Może wyemitować zdarzenie, na który wątek badany chce oczekiwać, lub przez który inny wątek zostanie odblokowany i ten inny wątek będzie mógł rzucić zdarzeniem, na który wątek badany chce oczekiwać.

Przykład 1: 3 wątki o nazwach 1, 2, 3. Wątek 1 rzuca zdarzeniem „a” i wątek ten już czeka na „c”. Wątek 3 rzuca zdarzeniem „c”. Wątek 2 emituje zdarzenie „b” i chce czekać na zdarzenie „a”. Zakleszczenie nie wystąpi, bo wątek 3 na nic nie czeka.
Przykład 2: 2 wątki o nazwach 1, 2. Wątek 1 rzuca zdarzeniem „a” i wątek ten już czeka na zdarzenie „b”. Wątek 2 wysyła zdarzenie „b”. Wątek ten nie może czekać na zdarzenie „a”.

Algorytm opiera się na funkcjach:

1 - int events_check_not_deadlock(struct events *cmc, struct task_struct *task, struct event *event)

2 - ssize_t events_search_task(struct task_struct **arr, ssize_t size, struct task_struct *task)

3 - struct event *events_check_if_proc_waits(struct events *cmc, struct task_struct *task)

1 funkcja sprawdza każdy proces/wątek, który może emitować zdarzenie będące argumentem funkcji – event, tzn. zdarzenie, na które badany proces/wątek chce oczekiwać.

2 funkcja sprawdza, czy obecnie badany proces rzucający dane zdarzenie już oczekuje na to samo zdarzenie (tzn. sytuacja, gdy są dwa procesy: 1, 2 i jedno zdarzenie „a”). Jeśli tak jest, pomijany jest taki proces w dalszym badaniu.

3 funkcja sprawdza, czy proces rzucający, który jest badany przez funkcję 1, już na coś czeka. Jeśli tak, do funkcji 1 kierowane jest zdarzenie, na które to proces oczekuje. Jeśli nie, oznacza to teoretyczny brak zakleszczenia.

Opis API użytkownika:

int event_set(char *name) – funkcja tworzy nowe zdarzenie o nazwie z argumentu lub

dopuszcza proces/wątek do użytkowania istniejącego już zdarzenia o tej samej nazwie co w podanym argumencie. Funkcja zwraca numeryczne oznaczenie zdarzenia (deskryptor pliku pełni tę funkcję), gdy proces został dopuszczony do zdarzenia, lub w przypadku błędu zwraca wartość -1 i ustawia odpowiednio wartość zmiennej errno.

int event_wait(int event_num) - funkcja, której celem jest oczekiwanie na nazwane zdarzenie. Funkcja zwraca 0, gdy oczekiwanie się powiodło, lub wartość -1, gdy z jakiegoś powodu funkcja nie może oczekiwać, lub oczekiwanie zostało przerwane. Stosownie do błędu ustawia zmienną errno.

int event_wait_group(char **events, int events_cnt) – oczekiwanie na jedno z grupy zdarzeń. Parametr events_cnt określa liczbę zdarzeń, która ma być pobrana ze wskaźnika char **events. Funkcja zwraca numer zdarzenia w przekazanej tablicy, które zostało odebrane, lub wartość -1 w przeciwnym wypadku. Stosownie do błędu ustawia zmienną errno.

int event_throw(int event_num) - rzucanie zdarzeniem. Funkcja zwraca 0, gdy rzucenie zdarzeniem jest możliwe, -1 w przeciwnym wypadku. Errno ustawione stosownie do błędu.

int event_unset(int event_num) - Usuwanie nazwanego zdarzenia ze sterownika. Funkcja zwraca wartość -1 i ustawia odpowiednio errno w sytuacji błędu. W przypadku gdy errno=EDEADLK, nie może dojść do usunięcia zdarzenia, ponieważ będzie to skutkowało zakleszczeniem innych procesów/wątków.

int event_check_error_exit(int rt, char *st) oraz int **event_check_error(int rt, char *st)** – prezentacja błedu. Pierwsza funkcja powoduje zakończenie wątku/procesu.

Opis sterownika:

Struktury sterownika:

```
struct events {
        const char *driver_name;
        struct cdev *cdev;
        dev_t dev;
        struct class *class;
        struct file_operations fops;
        struct device *device;
        struct list_head event_list;
        struct mutex lock;
        unsigned int event_cnt;
};
```

Powyższa struktura jest tworzona tylko raz, poprzez zmienną globalną.

Opis niektórych parametrów struktury:
struct mutex lock – mutex kontrolujący dostęp do zasobów sterownika.

Struct list_head event_list – lista zdarzeń.

Unsigned int event_cnt - licznik zdarzeń. W przypadku gdy jest liczba zdarzeń jest równa 0 i zmienna ma zostać dekrementowana, następuje generacja poważnego błędu systemowego Oops przez funkcję pomocniczą generate_oops().

Każde zdażenie reprezentowane jest przez strukturę:

```
struct event {
        char *name;
        struct completion **wait;
        char **completed_by;
        unsigned int s_comp;
        unsigned int g_comp;
        struct list_head element;
        struct task_struct **proc_throws;
        struct task_struct **proc_waits;
```

};
Zmienne s_comp i g_comp są zmiennymi pomocniczymi reprezentującymi liczbę wątków aktualnie oczekującymi na zdarzenie. s_comp oznacza liczbę wątków oczekujących na pojedyncze zdarzenie. g_comp oznacza liczbę wątków oczekujących na jedno z grupy zdarzeń. Zmienne struct task_struct **proc_throws/waits wykorzystywane przez algorytm wykrywający zakleszczenie. Tablice te inicjalizowane są przez macro current – macro to przekazuje wskaźnik na strukturę task_struct wątku wywołującego.

Zmienna wait jest wskaźnikiem na tablicę struct completion ze względu na dynamiczne tworzenie struktur completion.

Zmienna completed_by zastosowana dla otrzymywania informacji o tym, które zdarzenie zostało odebrane przy oczekiwaniu grupowym.

Zmienne globalne sterownika:

Sterownik posiada 4 parametry, które można zmienić podczas ładowania modułu:
- glob_name_size – zmienna określająca maksymalną liczbę znaków nazwy zdarzenia.
- glob_event_cnt_max – zmienna określająca maksymalną liczbę zdarzeń jaką sterownik w jednym momencie może obsługiwać.
- glob_compl_cnt_max – zmienna określająca maksymalną liczbę struktury completion.
- glob_proc – zmienna określająca maksymalną liczbę procesów mogących emitować określone zdarzenie lub czekać na określone zdarzenie.

Ponadto globalna jest struktura pomocnicza o nazwie cmc, typu struct events.

Najważniejsze funkcje sterownika:

int events_set(struct file *file, struct events *cmc, const char __user *buf)

Znaczenie argumentów:
- struct file *file – wykorzystywany do powiązania otwartego pliku ze zdarzeniem, dzięki czemu nie ma konieczności ciągłego wyszukiwania zdarzenia podczas np. emisji zdarzenia. Jest to możliwe przez wykorzystanie pola file->private_data (w tym polu umieszczany jest wskaźnik na zdarzenie).
- const char __user *buf – bufer w którym znajduje się nazwa zdarzenia do utworzenia/do którego proces/wątek ma się podłączyć.
- Struct events *cmc – wskaźnik do pomocniczej struktury events.

Zasada działania funkcji:
- Kopiowanie nazwy użytkownika do bufora w kernelspace'ie.
- Wyszukiwanie zdarzenia o takiej nazwie: jeśli nie ma takiego zdarzenia tworzone jest nowe zdarzenie, jeśli jest krok tworzenia jest pomijany.
- Proces zgłaszający zdarzenie dodawany jest do listy proc_throws zdarzenia.

int events_wait(struct file *file, struct events *cmc)

Znaczenie argumentów:
- struct file *file – z pola private_data pobierany jest wskaźnik na zdarzenie na które program na oczekiwać.
- struct events *cmc – wskaźnik do struktury events, na której sterownik się opiera.

Zasada działania funkcji:
- Inkrementacja zmiennej s_comp należącej do zdarzenia.
- Dodanie wskaźnika struktury obecnego zadania przez macro current do tablicy proc_waits danego zdarzenia.
- Sprawdzenie, czy występuje zakleszczenie. Jeśli tak – cofnięcie powyższych zmian.
- Jeśli nie – następuje przejście do funkcji wait_for_completion_interruptible
- Dekrementacja zmiennej s_comp i usunięcie procesu z tablicy proc_waits.

int events_group_wait(struct events *cmc, const char __user *user_buf)

Funkcja odpowiedzialna za oczekiwanie grupowe.
Znaczenie argumentów:

- const char __user *user_buf – bufor w którym zawarty jest wskaźnik do łańcucha tekstowego zawierającego nazwy zdarzeń do wczytania oraz liczba bajtów do przeczytania z podanego wskaźnika.
- Struct events *cmc – wskaźnik do pomocniczej struktury events.

Zasada działania funkcji:

- Pobranie zdarzeń, na które ma proces oczekiwać.
- Utworzenie nowej struktury completion.
- Umieszczenie w każdym zdarzeniu, na który proces ma oczekiwać wskaźnika do struktury completion oraz wskaźnika do struktury task_structure w tablicy proc_waits.
- Sprawdzenie, czy nie następuje deadlock. Jeśli tak powyższe zmiany są cofane.
- Przystąpienie do oczekiwania przez wywołanie funkcji wait_for_completion_interruptible.
- Usunięcie powyższych zmian po zakończeniu oczekiwania.

int events_throw(struct file *file, struct events *cmc)

Znaczenie argumentów:

- struct file *file – Zmienna zawiera w polu private_data wskaźnik do zdarzenia mającego zostać wyemitowane.
- Struct events *cmc – wskaźnik do pomocniczej struktury events.

Zasada działania funkcji:

- Sprawdzanie, czy obecny proces może emitować zdarzenie.
- Wywołanie funkcji complete_all dla oczekiwania pojedynczego o ile takie istnieje.
- Sprawdzenie, czy istnieją struktury completion dla oczekiwania grupowego. Jeśli tak następuje dla nich wywołanie funkcji complete_all.

int events_unset(struct file *file, struct events *cmc)

Znaczenie argumentów:

- struct file *file – Zmienna zawiera w polu private_data wskaźnik do zdarzenia mającego zostać usunięte.
- struct events *cmc – wskaźnik do pomocniczej struktury events.

Zasada działania funkcji:

- Sprawdzenie, czy możliwe jest usunięcie procesu ze zdarzenia: weryfikacji podlega:
  - Czy dany proces zarejestrował zdarzenie,
  - Czy usunięcie zdarzenia spowoduje zakleszczenie innych wątków/procesów.
- Jeśli istnieje możliwość wyrejestrownia procesu ze zdarzenia usuwany jest wskaźnik do struktury task_struct procesu wywołującego funkcję events_unset.
- Jeśli nie ma żadnego procesu emitującego zdarzenie, zdarzenie jest usuwane.

int events_release(struct inode *inode, struct file *file)

Znaczenie argumentów:

- struct inode *inode – niewykorzystywany
- struct file *file — Zmienna zawiera w polu private_data wskaźnik do zdarzenia mającego zostać poprawione przez funkcję.

Funkcja jest wywoływana podczas zamykania pliku urządzenia. Dzięki temu, że system automatycznie zamyka otwarte pliki przy usuwaniu procesu/wątku, możliwa jest stabilna praca sterownika przy nieprawidłowo zakończonym procesie (tzn. z pominięciem odrejestrowania

procesu od zdarzenia).
static void __exit events_exit(void)

Standardowa funkcja wywoływana podczas usuwania modułu z jądra. Funkcja zwalnia wszystkie wcześniej zaalokowane dynamicznie obszary pamięci. Dodatkowo wykrywa czy nie doszło do wycieku pamięci. Jeśli doszło, pojawia się komunikat systemowy.