Projekt - oprogramowanie systemowe

Aplikacja EFI

Maciej Winiecki 198363

1. Wstęp

Celem projektu była implementacja prostej gry typu retro w środowisku UEFI. Omawiana gra przypomina uproszczoną wersję gry “Pac-Man”, w której użytkownik porusza się po mapie i zbiera monety, unikając przy tym kontaktu z przeciwnikami. Wytworzenie aplikacji, która na pierwszy rzut oka wydaje się być niczym ciekawym, wymagała przeprowadzenia analizy architektury środowiska UEFI oraz odejścia od bibliotek standardowych.

2. Środowisko

Wstępna analiza wymagań dotyczących projektu oraz przegląd dostępnych technologii zadecydowały o wyborze **Ubuntu** (w środowisku WSL) jako platformy deweloperskiej. Na decyzję wpłynęła przede wszystkim łatwa dostępność kluczowych narzędzi, takich jak kompilator **GCC** oraz biblioteka **GNU-EFI**. Czynnikiem decydującym okazało się jednak natywne wsparcie dla emulatora QEMU w systemach Linux, co zagwarantowało stabilność środowiska testowego.

3. Emulator

Konieczne było znalezienie alternatywy dla testowania aplikacji na fizycznym sprzęcie. Wybór padł na emulator **QEMU**, który w przeciwieństwie do ciężkich maszyn wirtualnych, cechuje się niskim narzutem zasobów oraz błyskawicznym czasem rozruchu, co znacznie przyspieszyło proces deweloperski. Dodatkowo, **izolacja sprzętu** zabezpieczyła system przed ewentualnym uszkodzeniem.

QEMU emuluje sprzęt, który domyślnie jest wyposażony w **Legacy BIOS**. Uruchomienie pliku **.efi** wymaga dostarczenia do emulatora standardu **UEFI** dla maszyn wirtualnych. Zostało to zrealizowane poprzez załadowanie **OVMF (Open Virtual Machine Firmware)**. Dzięki temu, otrzymane środowisko jest wyposażone w usługi **Boot Services** oraz **Runtime Services**, które są wymagane

przez aplikację.

4. Interfejs aplikacji

Uruchomienie aplikacji powoduje wyświetlenie **menu startowego**, w którym użytkownik może wybrać jeden z dwóch trybów gry:

- **standardowy**, w którym przeciwnicy i monety pojawiają się w ustalonych miejscach,
- **losowy**, w którym początkowe położenie przeciwników i monet jest ustalane w sposób losowy

Rozgrywka polega na przemieszczaniu się używając przycisków WSAD. Gra kończy się, gdy gracz zbierze wszystkie monety. Wyświetlany jest wtedy ekran końca gry, a gracz wybiera wyjście z programu lub ponowne uruchomienie.

5. Implementacja

Kod programu został napisany w języku C z wykorzystaniem biblioteki GNU-EFI. Poszczególne funkcjonalności zostały zaimplementowane poprzez specjalne protokoły udostępniane przez środowisko UEFI. Protokoły muszą zostać zlokalizowane poprzez wykorzystanie **GUID**, czyli unikalnych identyfikatorów, do których dostęp jest zrealizowany poprzez tablicę **Boot Services**. Wywołanie funkcji **BS->LocateProtocol** z odpowiednim GUID zwraca wskaźnik na szukany protokół. Funkcje udostępniane przez EFI muszą być objęte specjalnym wrapperem – **uefi_call_wrapper**, która gwarantuje poprawne przekazywanie parametrów do funkcji (interfejs **ABI**).

a. **GOP (Graphics Output Protocol)**
   Najważniejszą funkcjonalnością w omawianej aplikacji jest wyświetlanie obrazu na ekranie. Służy do tego specjalny protokół ze środowiska UEFI – **GOP**. Program, korzystając z protokołu, pobiera adres bazowy pamięci wideo – **FrameBufferBase**. Poprzez wpisywanie wartości **ARGB** do poszczególnych komórek pamięci, zostaje zmieniany wyświetlany obraz. Zostaje również zmieniony tryb rozdzielczości oraz pobrane zostają informacje o szerokości i wysokości ekranu. W oparciu o protokół GOP zostały zdefiniowane funkcje pomocnicza **drawSquare** i **drawCircle**.

```
void drawSquare(uint32_t *buffer, uint32_t posX, uint32_t posY, uint32_t color){
    for (uint32_t y = posY; y < posY+squareSize; y++) {
        for (uint32_t x = posX; x < posX+squareSize; x++) {
            buffer[x + (y * pixels)] = color;
        }
    }
}
```

b. **RNG PROTOCOL (Random Number Generator Protocol)**
Losowy tryb gry wymaga generowania liczb losowych. W standardowych aplikacjach wysokiego poziomu jest to zazwyczaj realizowane przez funkcje systemowe, np. rand(). W środowisku UEFI należy do tego wykorzystać natywny protokół **RNG**. Pozwala to aplikacji na pobieranie wartości losowych bezpośrednio z generatora sprzętowego. Służy do tego funkcja **GetRNG**. Aby funkcja zadziałała poprawnie, musimy zamontować do emulatora wirtualne urządzenie, odpowiedzialne za generowanie wartości losowych. W innym wypadku zwracana będzie zawsze wartość 0.

```
uint32_t random(){
    EFI_RNG_PROTOCOL *rng;
    EFI_STATUS Status;
    EFI_GUID rngGuid = EFI_RNG_PROTOCOL_GUID;

    Status = uefi_call_wrapper(BS->LocateProtocol, 3, &rngGuid, NULL, (VOID**)&rng);

    uint32_t randomVal = 0;
    if(!EFI_ERROR(Status)){
        uefi_call_wrapper(rng->GetRNG, 4, rng, NULL, sizeof(uint32_t), (UINT8*)&randomVal);
    }
    return randomVal;
}
```

c. **SIMPLE_TEXT_INPUT_PROTOCOL**
Służy do pobierania informacji od użytkownika. Protokół jest dostępny w systemowej tablicy **ConIn**. Do pobrania przycisku z klawiatury wykorzystuje się funkcję **ReadKeyStroke**. W aplikacji obsługa klawiatury zrealizowana jest na dwa sposoby:

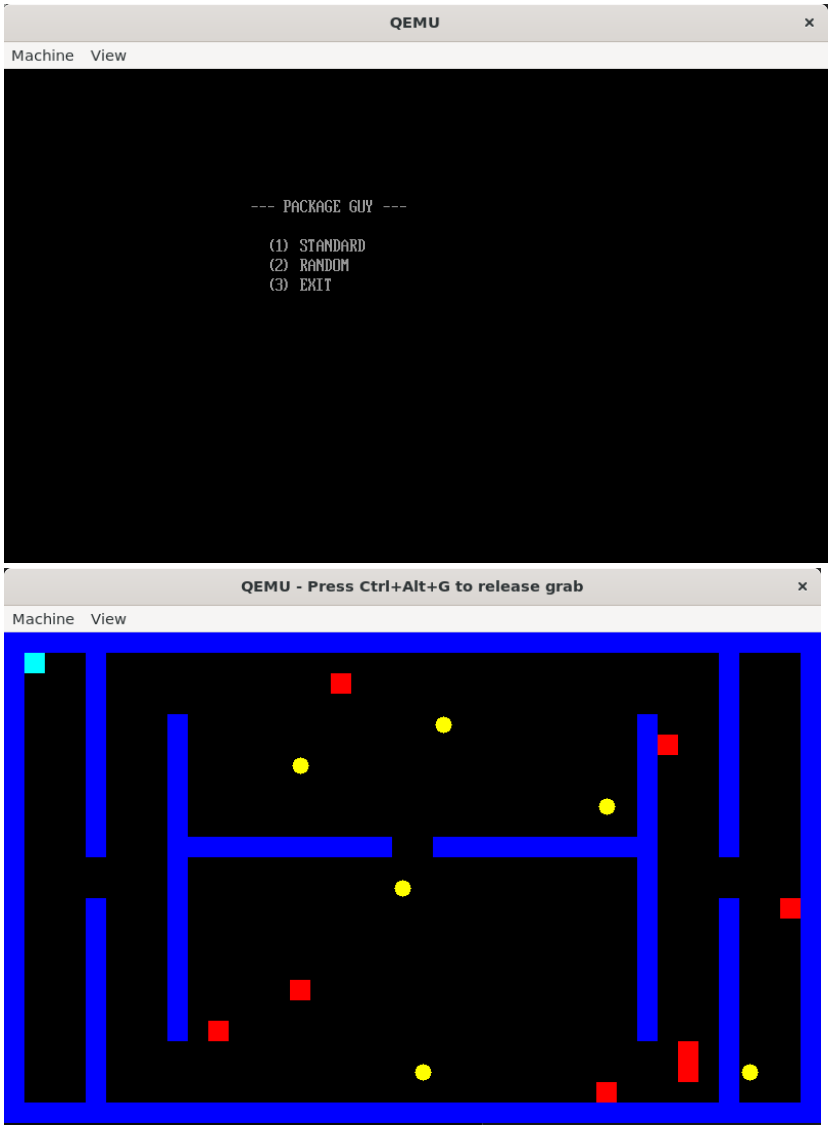
   i. **Tryb blokujący - WaitForEvent**
   W sekcji menu aplikacja jest wstrzymywana do momentu naciśnięcia przycisku.
   ii. **Tryb nieblokujący**
   ReadKeyStroke jest wywoływana w każdej klatce gry, co służy do obsługi sterowania postacią.

d. **Event System**
Mechanizm poruszania się przeciwników został zrealizowany w oparciu o

obiekt zdarzenia typu **EVT_TIMER**. Rozwiązanie to pozwala na wykonywanie logiki gry w ustalonych odstępach czasu, niezależnie od szybkości taktowania procesora. Za pomocą funkcji **CreateEvent** tworzony jest obiekt zdarzenia. Następnie, poprzez funkcję **SetTimer** ustawiany jest tryb **TimePeriodic** (automatyczny, cykliczny sygnał) oraz ustalana jest wartość opóźnienia (6000000 * 100 ns).

```
EFI_EVENT Timer;
uefi_call_wrapper(BS->CreateEvent, 5, EVT_TIMER, TPL_CALLBACK, NULL, NULL, &Timer);
uefi_call_wrapper(BS->SetTimer, 3, Timer, TimerPeriodic, 6000000);
```

6. Aplikacja



7. Uruchamianie emulatora

Emulator QEMU jest uruchamiany poniższą komendą:

qemu-system-x86_64 -bios /usr/share/ovmf/OVMF.fd -drive format=raw,file=fat:rw:. -net none -device virtio-rng-pci

a. Parametr **–bios** wskazuje na ścieżkę do pliku obrazu **OVMF**
b. **Wirtualizacja systemu plików**. QEMU mapuje bieżący katalog hosta jako partycję FAT: **-drive format=raw,file=fat:rw:**.
c. **Wyłączenie emulacji kart sieciowych**: **-net none**
d. **Dołączenie sprzętowego generatora liczb losowych**: -device virtio-rng-pci

8. Kompilacja

W celu uproszczenia etapu kompilacji został przygotowany skrypt. Najpierw plik .c jest kompilowany do pliku obiektowego. Wykorzystane są przy tym odpowiednie flagi, w tym: -freestanding (bez standardowej biblioteki systemowej), -fpic (kod niezależny od pozycji w pamięci)

Proces linkowania łączy plik obiektowy z biblioteką **GNU-EFI**.

Na koniec, używając **objcopy**, zmieniany jest format pliku z ELF, na PE (wymagane przez efi). Otrzymany plik jest umieszczany w standardowej ścieżce rozruchowej **EFI/BOOT/BOOTX64.EFI**, co powoduje, że gra jest uruchamiana automatycznie podczas startu systemu.

9. Wnioski

Realizacja projektu pozwoliła na praktyczne zrozumienie specyfiki pracy w środowisku UEFI, gdzie brak systemu operacyjnego wymusza bezpośrednią interakcję z protokołami sprzętowymi (takimi jak GOP czy RNG). Implementacja gry uwidoczniła fundamentalne różnice między programowaniem aplikacyjnym a systemowym, kładąc nacisk na ręczne zarządzanie zasobami i brak wsparcia standardowych bibliotek.