Core War

Marcin Puc, Piotr Komorowski

1. Opis projektu

Core War to gra w której dwa współbieżnie działające programy rywalizują ze sobą we współdzielonej przestrzenii pamięciowej („Core”) wirtualnego komputera. W celu pokonania przeciwnika należy zmusić jego program do wykonania specjalnej instrukcji lub niedozwolonej operacji.

Programy realizowane są w specjalnym pseudoasemblerowym języku, który udostępnia zbiór instrukcji realizujących operacje na pamięci. Gracz chcący wziąć udział w pojedynku musi napisać program („wojownika”) w udostępnionym języku, skompilować i uruchomić w maszynie wirtualnej. W ramach realizacji projektu zostanie stworzony język, kompilator i maszyna wirtualna.

2. Założenia wstępne

Wirtualny komputer obsługujący grę składa się z bloku pamięci, w którego komórkach zapisane są instrukcje. Blok ten jest cykliczny, tj. po ostatniej komórce następuje przejście do pierwszej komórki bloku. Programy ładowane są do bloku w losowo wybranych pozycjach. W trakcie wykonywania programu dostępne jest wyłącznie adresowanie względne – nie można się odwołać do adresu bezwzględnego danej komórki.

Każdy wykonywany program składa się z jednego lub więcej procesów, gdzie proces definiujemy jako wskaźnik na komórkę bloku. Procesy dla każdego programu są przechowywane w kolejkach i wywoływane naprzemiennie. „Wojownik” przegrywa gdy wszystkie jego procesy zostaną usunięte z kolejki. W przypadku, gdy liczba wykonanych cykli przekroczy ustaloną wartość zanim jeden z wojowników przegra, pojedynek kończy się remisem.

Instrukcje programu składają się z trzech głównych części: *kodu operacji*, *pola A* i *pola B*. Kod operacji identyfikuje wykonywaną przez maszynę instrukcję, natomiast pola A i B służą do przechowania danych liczbowych, np. adresu instrukcji przy skoku.

3. Język programowania

W oryginalnej wersji gry programy są pisane w stworzonym przez D. G. Jonesa i A. K. Dewdneya języku *„Redcode”*. Jego zbiór instrukcji i składnia posłużą jako wzorzec w tym projekcie.

Przykładowe instrukcje:

- `MOV` – przenoszenie danych w pamięci
- `ADD` – dodawanie liczb
- `MUL` – mnożenie liczb
- `JMP` – skok do innej instrukcji
- `FRK` – stworzenie nowego procesu
- `KIL` – zakończenie działania obecnego procesu i usunięcie go z kolejki

4. Kompilator

Kompilator analizuje kod programu napisany w formacie języka bardziej czytelnym dla użytkownika i zamienia go na plik odczytywalny w prosty sposób przez maszynę wirtualną.

Funkcjonalności kompilatora obejmują przykładowo:

- zamianę etykiet na adresy (względne) instrukcji
- wstępne przeliczanie adresów do postaci bardziej czytelnej dla maszyny
- znajdowanie błędów

5. Maszyna wirtualna

Maszyna wirtualna przyjmuje skompilowane pliki i wykonuje programy poprzez załadowanie ich kodu do pamięci. Wewnętrznie blok pamięci zaimplementowany jest jako tablica instrukcji. Maszyna inicjalizuje wojowników poprzez dodanie do ich kolejek po jednym procesie w miejscu wskazanym w pliku ze skompilowanym kodem programu. Następnie gra jest przeprowadzana zgodnie z zasadami podanymi w sekcji *Założenia wstępne*. Na końcu działania maszyna wyświetla wynik pojedynku.