ZADANIE

Zadanie 2

Otwarto: poniedziałek, 3 kwietnia 2023, 00:00
Wymagane do: sobota, 22 kwietnia 2023, 23:59

Rozproszona maszyna stosowa

Zaimplementuj w asemblerze x86_64 symulator rozproszonej maszyny stosowej. Maszyna składa się z `N` rdzeni, które są numerowane od `0` do `N − 1`, gdzie `N` jest pewną stałą ustalaną podczas kompilowania symulatora. Symulator będzie używany z języka C w ten sposób, że będzie uruchamianych `N` wątków i w każdym wątku będzie wywoływana funkcja:

```
uint64_t core(uint64_t n, char const *p);
```

Parametr `n` zawiera numer rdzenia. Parametr `p` jest wskaźnikiem na napis ASCIIZ i definiuje obliczenie, jakie ma wykonać rdzeń. Obliczenie składa się z operacji wykonywanych na stosie, który na początku jest pusty. Znaki napisu interpretujemy następująco:

- `+` – zdejmij dwie wartości ze stosu, oblicz ich sumę i wstaw wynik na stos;
- `*` – zdejmij dwie wartości ze stosu, oblicz ich iloczyn i wstaw wynik na stos;
- `-` – zaneguj arytmetycznie wartość na wierzchołku stosu;
- `0` do `9` – wstaw na stos odpowiednio wartość 0 do 9;
- `n` – wstaw na stos numer rdzenia;
- `B` – zdejmij wartość ze stosu, jeśli teraz na wierzchołku stosu jest wartość różna od zera, potraktuj zdjętą wartość jako liczbę w kodzie uzupełnieniowym do dwójki i przesuń się o tyle operacji;
- `C` – zdejmij wartość ze stosu i porzuć ją;
- `D` – wstaw na stos wartość z wierzchołka stosu, czyli zduplikuj wartość na wierzchu stosu;
- `E` – zamień miejscami dwie wartości na wierzchu stosu;
- `G` – wstaw na stos wartość uzyskaną z wywołania (zaimplementowanej gdzieś indziej w języku C) funkcji `uint64_t get_value(uint64_t n)`;
- `P` – zdejmij wartość ze stosu (oznaczmy ją przez `w`) i wywołaj (zaimplementowaną gdzieś indziej w języku C) funkcję `void put_value(uint64_t n, uint64_t w)`;
- `S` – synchronizuj rdzenie, zdejmij wartość ze stosu, potraktuj ją jako numer rdzenia `m`, czekaj na operację `S` rdzenia `m` ze zdjętym ze stosu numerem rdzenia `n` i zamień wartości na wierzchołkach stosów rdzeni `m` i `n`.

Po zakończeniu przez rdzeń wykonywania obliczenia jego wynikiem, czyli wynikiem funkcji `core`, jest wartość z wierzchołka stosu. Wszystkie operacje wykonywane są na liczbach 64-bitowych modulo 2 do potęgi 64.

Wolno założyć, że podany numer rdzenia jest poprawny. Wolno założyć, że obliczenie jest poprawne, tzn. zawiera tylko opisane wyżej znaki, kończy się zerowym bajtem, nie próbuje sięgać po wartość z pustego stosu i nie doprowadza do zakleszczenia. Zachowanie rdzenia dla niepoprawnego obliczenia jest niezdefiniowane.

Jako stosu, którego rdzeń używa do opisanych wyżej obliczeń, należy użyć sprzętowego stosu procesora. Nie wolno korzystać z żadnych bibliotek. Synchronizację rdzeni, czyli operację `S`, należy zaimplementować za pomocą jakiegoś wariantu wirującej blokady.

Wynik obliczenia `"1nS"` jest niezdefiniowany i zależy od implementacji, choć rozsądne wydaje się potraktowanie sekwencji operacji `n` i `S` jako operacji `C`.

Nie należy zakładać żadnego górnego ograniczenia na wartość `N` innego niż wynikające z architektury procesora i dostępnej pamięci.

W specyfikacji operacji wykonywanych przez rdzeń określenia „zdejmij ze stosu” i „wstaw na stos” należy traktować jako definicję operacji, która ma być wykonana, a nie jako wymaganie użycia rozkazu `pop` lub `push`.

Oddawanie rozwiązania

Jako rozwiązanie należy wstawić w Moodle plik o nazwie `core.asm`.

Kompilowanie

Rozwiązanie będzie kompilowane poleceniem:

```
nasm -DN=XXX -f elf64 -w+all -w+error -o core.o core.asm
```

gdzie `XXX` określa wartość stałej `N`. Rozwiązanie musi się kompilować w laboratorium komputerowym.

Przykład użycia

Przykład użycia znajduje się w załączonym pliku `example.c`. Kompiluje się go poleceniami:

```
nasm -DN=2 -f elf64 -w+all -w+error -o core.o core.asm
gcc -c -Wall -Wextra -std=c17 -O2 -o example.o example.c
gcc -z noexecstack -lpthread -o example core.o example.o
```

Ocenianie

Zgodność rozwiązania ze specyfikacją będzie oceniania za pomocą testów automatycznych. Sprawdzane będzie przestrzeganie reguł ABI, czyli prawidłowość użycia rejestrów i stosu procesora. Sprawdzana będzie poprawność odwołań do pamięci i zajętość pamięci. Należy dążyć do minimalizowania rozmiaru pamięci wykorzystywanej przez rozwiązanie. Od wyniku testów automatycznych zostanie odjęta wartość zależna do rozmiaru wykorzystywanej pamięci (sekcje `.text`, `.bss`, `.data`, `.rodata`, stos, sterta) – zostaną ustalone progi, których przekroczenie będzie skutkowało odjęciem punktów. Wystawienie oceny może też być uzależnione od osobistego wyjaśnienia szczegółów działania programu prowadzącemu zajęcia.

Oceniana będzie jakość kodu źródłowego, w tym komentarzy. Kod powinien być dobrze skomentowany, co oznacza między innymi, że każda procedura powinna być opatrzona informacją, co robi, jak przekazywane są do niej parametry, jak przekazywany jest jej wynik, jakie rejestry modyfikuje. To samo dotyczy makr. Dobrze jest w kodzie funkcji rozdzielić pustymi liniami poszczególne bloki funkcjonalne i każdy z nich skomentować, co robi. Komentarza wymagają także wszystkie kluczowe lub nietrywialne linie wewnątrz procedur lub makr. W przypadku asemblera nie jest przesadą komentowanie prawie każdej linii kodu, ale należy unikać komentarzy opisujących to, co widać.

Oceniane będzie spełnienie formalnych wymagań podanych w treści zadania, na przykład poprawność nazwy pliku. Kod niekompilujący się otrzyma 0 punktów.

Pytania

Pytania dotyczące zadania można zadawać w przeznaczonym do tego wątku na Moodle.

Załącznik

example.c 30 marca 2023, 23:10 PM

Dodaj pracę

Status przesłanego zadania

Status przesłanego zadania	Nie przesłano jeszcze zadania
Stan oceniania	Nieocenione
Pozostały czas	Pozostało 19 dni
Ostatnio modyfikowane	-
Komentarz do przesłanego zadania	Komentarze (0)