Wrocław 6 maja 2015

Kamil Cała  
209954  
Środa 7:15 TN

Sprawozdanie z laboratorium nr 1

*Data laboratorium: 22.04.2015r  
Rok akademicki 2014/2015, Informatyka  
Prowadzący: Mgr. Aleksandra Postawka*

# Opis i cel ćwiczenia

Celem tego ćwiczenia było zapoznanie się z metodami integracji tworzonego oprogramowania, a konkretnie używaniem języków Assembly i C w ramach jednego projektu.

Ćwiczenie składało się z 3 zadań:

1. Użycie wstawki assembly w ramach programu napisanego w C
2. Napisanie programu w C który będzie wykorzystywał funkcję napisaną w assembly i zawartą w oddzielnym pliku
3. Napisanie programu w assembly, który wykorzystywać będzie funkcje z języka C - zarówno biblioteczne jak i własne

# Zadanie 1 - program w C używający in-line assembly

Pierwsze okazało się być najprostszym. Stworzony przeze mnie program nie wykonuje żadnej praktycznej operacji, a jedynie prezentuje w jaki sposób można używać wstawek assemblerowych w programach napisanych w C. Aby to zaprezentować, stworzyłem 3 zmienne. Do dwóch pierwszych za pomocą funkcji bibliotecznej scanf wczytywane są dwie liczby całkowite.

int first, second, sum;

printf("Enter two integers: ");

scanf("%d%d", &first, &second);

Następnie obie liczby zostają dodane a wynik tej operacji zapisany zostaje w zmiennej sum. Operacja ta zostaje przeprowadzona w assembly:

//Add two numbers in asm

\_\_asm\_\_("addl %%ebx, %%eax;" : "=a" (sum) : "a"(first), "b"(second));

Jak widać, wstawka assemblerowa w C ma następującą formę:  
\_\_asm\_\_(KOD : ZMIENNE);

Wewnątrz nawiasu znajduje się kod assembly, dodający wartości rejestrów ebx i eax, zapisując wynik do eax. Warto zauważyć że nazwy rejestrów są tutaj poprzedzone podwójnym znakiem %. Po dwukropku zdefiniowane są relacje zmiennych C z rejestrami dostępnymi z poziomu assembly. Po pierwszym dwukropku znajduje się zapis:  
  
"=a" (sum)

Oznacza on, że po zakończeniu operacji wartość rejestru e**a**x powinna zostać przeniesiona do zmiennej sum. Znak równości oznacza dostęp do rejestru w trybie read-only. Z kolei po drugim dwukropku znajdziemy nastepujący zapis:

"a"(first), "b"(second)

Dzięki tej części, przed wykonaniem operacji w assembly do rejestrów e**a**x oraz e**b**x zapisywane są wartośći z podanych zmiennych. Brak znaku równości oznacza że rejestry mogą być zarówno odczytywane i zapisywane.

Na koniec, wynik operacji zostaje wypisany przy pomocy funkcji printf ze standardowej biblioteki języka C, dając spodziewany wynik.

printf( "%d + %d = %d\n", first, second, sum );

# Zadanie 2 - program w C używający funkcji napisanej w ASM zawartej w innym pliku

Aby zaprezentować ten rodzaj integracji stworzyłem wpierw funkcję mnożącą dwie liczby w assembly. Funkcja bierze dwa argumenty w sposób zgodny z ABI. Argumenty powinny więc w momencie wywołania funkcji znajdować się w konkretnie wyspecyfikowanych rejestrach. Pierwszy argument w rejestrze rdi, a drugi w rejestrze rsi (dla wersji 64bit). Zgodnie ze specyfikacją mogłyby one jednak również zostać przekazane przez stos, dlatego dla ujednolicenia zachowania programu w tym miejscu wartości z rejestrów kopiowane są na stos, “na lewo” od miejsca wskazywanego przez stack pointer %rsp. Następnie wartości są przez siebie mnożone a wynik zwracany jest poprzez rejestr eax.

.text

.globl multiply

multiply:

pushq %rbp #save whatever is in rbp

movq %rsp, %rbp #set new stack frame

movl %edi, -4(%rbp) #move first argument to stack

movl %esi, -8(%rbp) #move second argument to stack

movl -4(%rbp), %eax #move first number to eax

imull -8(%rbp), %eax #multiply numbers

popq %rbp #restore rbp value

ret #return function

W aplikacji C wywołanie funkcji niczym nie różni się od wywołania każdej innej funkcji.

int result = multiply(5,6);

Zauważmy jednak że użycie tej funkcji nie jest takie proste. Wcześniej należy poinformować kompilator że taka funkcja istnieje i zostanie dopiero w fazie linkowania dołączona do programu. W większośći projektów w C robi się to poprzez użycie plików nagłówkowych z rozszerzeniem .h w których zawarte są same definicje funkcji bez ich implementacji. W wypadku takim, jak w tym zadaniu, funkcja pochodzi z innego języka programowania. Do programu zostaje dodana dopiero na ostatnim etapie budowania projektu, czyli linkowaniu. Aby poinformować więc kompilator że taka funkcja istnieje i zostanie później dołączona, należy użyć następującej komendy:

extern int multiply(int a, int b);

Należy też zauważyć, że w pliku .s zawierającym funkcję znalazła się dyrektywa:

.globl multiply

Dzięki niej funkcji została przydzielona nazwa, która może zostac użyta z poziomu języka C.

# Zadanie 3 - program w ASM używający funkcji z języka C

Analogicznie do poprzedniego zadania, aby zaprezentować ten sposób integracji stworzyłem wpierw w oddzielnym pliku funkcję mnożącą dwie liczby. Tym razem jednak w języku C:

int multiply(int a, int b){

return a\*b;

}

Główna część programu została z kolei napisana w assembly. Program ten:

1. Wyświetla na ekranie prośbę o podanie dwóch liczb całkowitych przy użyciu funkcji języka C - printf
2. Pobiera z wejścia standardowego dwie liczby i zapisuje je na stosie. Ponownie dzieje się to przy użyci funkcji pochodzącej z biblioteki języka C. Tym razem jest to jednak funkcja scanf
3. Pobrane liczby mnożone są przez siebie przy użyciu opisanej wcześniej, stworzonej przeze mnie funkcji
4. Wynik wypisywany jest na ekran - ponownie przy użyciu funkcji printf.

Kluczowym do zrozumienia działania programu elementem jest zrozumienie sposobu przekazywania argumentów do funkcji. W 64bitowej architekturze Intela kolejne argumenty (maksimum 6) przekazywane są poprzez wczytanie ich do następujących rejestrów:

1. rdi
2. rsi
3. rdx
4. rcx
5. r8
6. r9

W momencie wywołania funkcji poprzez **call**, kolejne wartości przekazywane są do funkcji w C od lewej do prawej.

Jeżeli zachoidz potrzeba przekazania więcej niż 6 argumentów, należy umieścić je na stosie. W przypadku 64bitowej architektury będą to liczby 64bitowe, a więc 8 bajtowe. Adresowanie kolejnych argumentów wygląda więc następująco:

1. (%rsp)
2. 8(%rsp)
3. 16(%rsp)

Itd.

W wypadku tego ćwiczenia zostawały przekazane do funkcji maksymalnie 2 argumenty. Przykład tego można zobaczyć przy wywoływaniu funkcji scanf. Pierwszym argumentem jest adres ciągu znaków będących formatką wykorzystywaną przez tą funkcję.

Drugim i trzecim argumentem są adresy miejsc na stosie do których mają zostać wczytane dane. W języku C argumenty podaje się poprzez dodanie do nazwy zmiennej ampersandu (&), co powoduje zwrócenie jej adresu w pamięci komputera. Aby uzyskać ten sam efekt w assembly, należy użyć instrukcji LEA (load effective address), oczywiście z charakterystycznym dla GNU assembler’a sufixem q, oznaczającym operację na liczbach 64bitowych.

leaq -8(%rbp), %rsi #pass first 4 allocated bits address as second argument

leaq -4(%rbp), %rdx #pass next 4 allocated bits address as third argument

Przekazanie stringa zawierającego formatkę wygląda następująco

movq $InputFormat, %rdi #pass input format as first argument

# Wnioski

W aplikacjach nastawionych na ekstremalną wydajność, jak na przykład bankowych systemach HFT (High Frequency Trading), gdzie milisekundy w czasie wykonywania aplikacji mogą decydować o milionowych zyskach i stratach, często wykorzystuje się wstawki assemblerowe w ramach kodu C/C++. Pozwala to na najlepszą możliwą optymalizację krytycznych części programu, oraz na wykorzystanie najnowszych funkcji procesorów, które nie są jeszcze wspierane przez popularne kompilatory. Wstawki assemblerowe używa się także przy tworzeniu sterowników.

Operacje w drugą stronę, czyli wywoływanie funkcji języków wyższego poziomu z poziomu assemblera są rzadko wykorzystywane, jednakże zrozumienie ich działania pozwala lepiej pojąć cały proces wywoływania funkcji w ramach ABI (Application Binary Interface) procesora.

Wykonanie tego ćwiczenia pozwoliło mi zrozumieć jak komunikują się ze sobą programy na najniższych poziomach abstrakcji, oraz nauczyło mnie jak sprawić aby ze sobą współpracowały. Umiejętności te będą przydatne również w pracy z językami wyższego poziomu i pozwolą zapobiec ewentualnym błędom przy integracji oprogramowania.