시스템 프로그래밍 언어의 크기와 수행 성능 사이의 상관관계 분석을 위한 사례 연구

*김민성, 우균
부산대학교 정보융합공학과
e-mail : {msungkim, woogyun}@pusan.ac.kr

A Case Study to Analyze The Correlation Between The Size And Performance of a System Programming Language

*Minsung Kim and Gyun Woo
Dept of Information Convergence Engineering,
Pusan National University

Abstract

A system programming language refers to a language used for various system design, and is generally a programming language that provides direct access methods to hardware. In this paper, the size of the system programming language is compared based on the size comparison criteria proposed by the author. It measures execution time for typical languages using Fibonacci sequence and factorial. The correlation between the measured time and the sizes of the system programming languages is analyzed. The case study used in this paper can be used as a basis for determining which system programming language to choose.

I. 서론

시스템 프로그래밍 언어는 각종 시스템 설계용으로 사용되는 프로그래밍 언어를 뜻하며, 일반적으로 하드웨어에 직접적인 접근 방법을 제공해주는 프로그래밍 언어이다. 시스템 프로그래밍 언어[1]는 높은 복잡성으로 인해 스크립트 언어[2]보다 어렵지만 높은 성능에 힘입어 여전히 널리 사용되고 있다. 프로그래밍 언어 성능 비교와 점유율이 높은 언어에 관한 발표한 자료[3,4]에 따르면 여전히 시스템 프로그래밍 언어는 스크립트 언어에 비해 높은 성능과 점유율을 보이고 있다. 하지만 스크립트 언어에만 익숙한 프로그래머가 시스템 프로그래밍 언어를 선택할 때에는 어떤 언어를 선택하는 것이 좋은지 판단하기 어렵다.

이에 본 논문은 시스템 프로그래밍 언어의 크기를 키워드, 연산자 의 수 등 수치적으로 계산하여, 시스템 프로그래밍 언어 간 객관적인 수치를 통해 해당 언어의 크기를 측정하고, 성능을 측정할 수 있는 함수를 통해서 언어의 크기와 언어의 성능 간에 상관관계를 밝혀 어떤 시스템 프로그래밍 언어를 선택하는 것이 좋은지 판단의 근거를 제시하기 위해 간단한 사례 연구를 진행해보았다.

II. 본론

실험에 앞서, 시스템 프로그래밍 언어 중 5가지 언어를 선정하였다. 본 연구에 선정된 언어는 현재

꾸준한 개발이 이루어지고 있으며, 많은 개발자들이 사용하고 있는 프로그래밍 언어이다. 프로그래밍 언어를 비교함에 있어, 본 연구자의 주관이 들어갔다는 것을 부인할 순 없으나, 객관적으로 개발자들이 많이 사용하는 언어 중 선별하여 선정하였다.

2.1 C

C 언어는 1972년 Kenneth Lane Thompson과 Dennis Ritchie가 Bell Lab에서 재직할 당시에 새로 개발된 유닉스 운영 체제에서 사용하기 위해 개발한 시스템 프로그래밍 언어이다[5,6]. 실질적으로는 모든 컴퓨터 시스템에서 사용할 수 있도록 설계된 프로그래밍 언어이며, 높은 생산성과 성능이 뒷받침 되어 아직까지 C언어는 많이 사용되는 언어 중 하나이다.

2.2 C++

C++언어는 1979년 Bjarne Stroustrup이 개발한 언어로서, 초기의 C++언어는 C언어에 객체지향 프로그래밍을 지원하기 위해 사용되었다. C++프로그램을 C 프로그램으로 변환한 다음 다시 C컴파일러로 컴파일 하는 방식이었기 때문에, C 언어에 대한 상위 호환성 만을 가졌었다[7]. 하지만 시간이 지남에 따라 C++만의 특징이 뚜렷해졌고, 기능과 성능이 높아져, 사실상 C 언어와 C++ 언어는 분리되었다. 현재까지 계속해서 개발되고 있으며, 시스템 프로그래밍 언어로 많이 사용되는 언어 중 하나이다.

2.3 Julia

Julia 언어는, 2012년 Jeff Bezanson, Stefan Karpinski, Viral B. Shah 외 기타 기여자들이 개발한 동적 프로그래밍 언어[8]로서, 개발의 본 목적은 수치 및 과학 컴퓨팅을 위해서 설계되었으나, C 언어와 같은 정적 언어 수준의 성능을 가지며, 저수준의 시스템 프로그래밍에서 많이 사용되는 언어이다.

2.4 Swift

Swift는 Apple의 iOS와 MacOS를 위한 프로그래밍 언어로, 2014년 6월에 처음으로 소개 된 언어이다. 기존에 사용되던 애플 운영체제용 언어인 Objective-C와 함께 공존할 목적으로 만들어졌으나[9], Objective-C에는 없는 현대 프로그래밍 언어가 갖고 있는 기능을 많이 포함 시켰으며, 스크립트 언어처럼 실시간으로 상호작용을 하며 프로그래밍이 가능한 강력한 시스템 프로그래밍 언어 중 하나이다.

2.5 Rust

Rust는 Mozilla Research에서 개발한 범용 프로그래밍 언어이다[10]. 함수형 프로그래밍, 액터 기반 병렬 프로그래밍, 객체지향 스타일 등 다양한 패러다임을 제시하는 언어이다. 컴파일 기반의 언어임과 동시에 시스템 프로그래밍 언어에 속하며, C 언어나 C++언어와 비슷한 수준의 성능을 보여주며, 개발자들에게 꾸준히 사랑받는 시스템 프로그래밍 언어 중 하나이다.

Ⅲ. 실험 설계

본 연구에서 사용된 프로그래밍 언어 간 비교는 개발자들이 많이 사용되는 언어 중에서 시스템 프로그래밍에 이용되는 언어를 선정하였으며, 객관적인 수치를 통해서 비교를 진행하였다. 언어 크기 비교 기준을 설계하는데 있어, 비교하려는 언어 중 동적 프로그래밍 언어가 포함 되어 있기 때문에 변수의 타입 관련 부분은 제외하였다. 그리고 언어의 성능을 측정하는데 있어서, 2 가지의 알고리즘을 사용해서 측정했다.

3.1 예약어의 수

예약어란, 컴파일 동작 전 이미 정의되어 있는 키워드를 의미한다. 각 언어별 표준 예약어를 기준으로 삼았다.

3.2 연산자의 수

연산자는 컴파일러에 특정한 수치 연산 혹은 논리 연산을 수행하도록 지시하는 기호이다. 본 연구에서 사용된 연산자의 유형은 산술 연산자, 관계형 연산자, 논리 연산자, 할당연산자를 기준으로 삼았다.

3.3 성능 측정

대표적인 알고리즘인 Fibonacci와 Quicksort 알고리즘을 사용해서 각 언어 간의 실행 시간을 출력하여 성능을 측정했다. 최대한 동일한 조건에서 실험하기 위하여, 라이브러리 함수가 아닌 4절 그림 2, 그림 3의 의사코드를 토대로 사용자 정의함수를 사용했다. 그리고 프로그램 종료 시에 실행 시간을 출력하게 함으로써 측정을 종료했다.

Ⅳ. 실험

2절에서 제시한 각 언어에 대한 3절의 비교기준을 적용하여 시각적으로 그림 1에 표현했다.

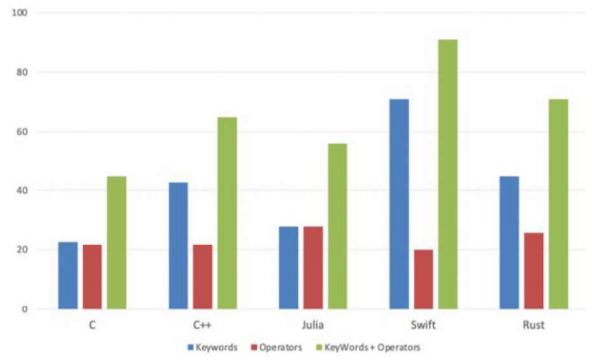

그림 1. 비교기준에 따른 언어 크기비교

그래프의 막대는 각각 예약어와 연산자, 그리고 예약어와 연산자를 합한 수치를 시각적으로 표현하였다. 그림 1을 보면 C < Juila < C++ < Rust < Swift 순으로 언어의 크기가 측정되었음을 알 수 있다. 프로그래밍 언어의 성능 측정에는 피보나치 수열과 빠른 정렬 함수를 사용했다.

Algorithm 01

```
Function functionName(Argument array::Array, start::Int, end::Int){
     If start >= end Then Return;
     pivot::Int = start i::Int = pivot + 1
     j::Int = end temp::Int
     While i <= j Then
          While i <= end and array[i] <= array[pivot] Then
               i++
          EndWhile
          While j > start and array[j] >= array[pivot] Then
               j--
          EndWhile
          If i > j Then
               temp = array[j]
               array[j] = array[pivot]
               array[pivot] = temp
          Else
               temp = array[i] array[i] = array[j] = array[j] = temp
     EndWhile
     functionName(array, start, j-1) functionName(array, j+1, end)
EndFunction
```

그림 2. Quicksort 의사 코드

Algorithm 02

```
Function functionName(Argument number::Int)
          If number is 0 or number is 1 Then
                    Return number
          Else
                    Return functionName(number - 1) + functionName(number - 2)
EndFunction
```

그림 3. Fibonacci 의사 코드

위와 같은 의사 코드로 언어 별 함수를 구현했다.

그림 4는 위와 같은 의사코드를 토대로 구현된 함수들의 언어 별 실행시간 측정과 비교기준에 따른 언어의 크기를 시각적으로 나타냈다.

	C	C++	Julia	Swift	Rust
Keyword + Operator	45.0000	65.0000	56.0000	91.0000	71.00
QuickSort Execute Time	0.0014	0.0013	0.0120	0.2310	0.11
Fibonacci Execute Time	0.5755	0.6329	0.4014	1.6747	0.80

그림 4. 언어 별 함수 시간 측정

본 연구에 사용된 함수들의 시간 측정 기준은 모두 동일한 환경, 인자를 사용해서 측정되었다. 특히 함수의 인자 부분은 재귀 함수를 활용해서 구현되었기 때문에 스택 오버 플로우가 발생할 수 있다. 따라서 Fibonacci 함수에는 정수 값 40을 대입했으며, Quicksort 함수에는 10000개의 난수가 담긴 배열을 대입하여 측정했다. 또한 상관관계분석은 Pearson 상관관계를 이용하여 실시하였으며, 두 변수는 함수의 실행 시간과 예약어, 연산자의 합으로 실시되었다.

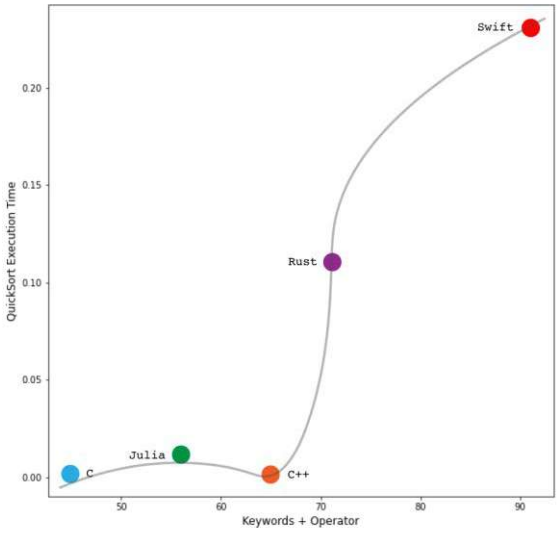

그림 5. Quicksort 함수와 비교기준 간 그래프

	Comparison	QuickSort
Comparison	1.00000	0.91122
QuickSort	0.91122	1.00000

표 1. Quicksort 함수와 비교기준 간 상관계수

그림 6을 보면, Quicksort 함수와 본 연구자가 세운 비교기준에 따른 상관계수의 절댓값은 약 0.91로 강한 양의 상관관계를 이루고 있다는 것을 알 수 있다.

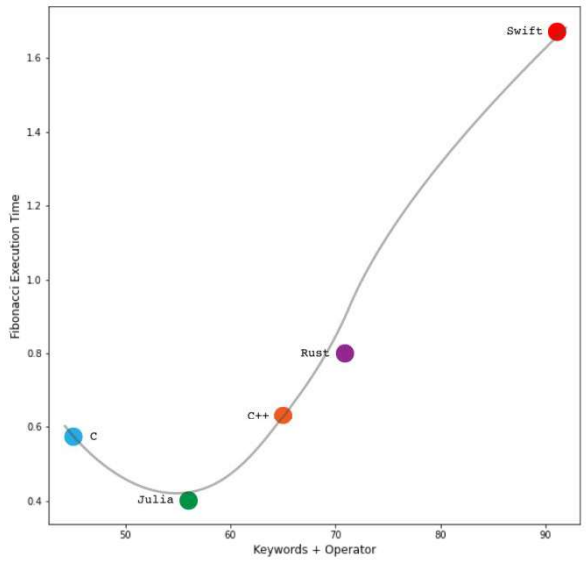

그림 7. Fibonacci 함수와 비교기준 간 그래프

	Comparison	Fibonacci
Comparison	1.00000	0.89110
Fibonacci	0.89110	1.00000

표 2. Fibonacci 함수와 비교기준 간 상관계수

그림 8을 보면, Fibonacci함수와 비교기준 간 상관계수 또한 절댓값이 약 0.89로 강한 양의 상관관계를 보인다.

V. 결론 및 향후 연구 방향

언어의 크기를 측정한다는 것은 매우 어려운 일이며, 방대한 작업이다. 그럼에도 불구하고 본 연구에서 달성하고자 하는 목표는, 언어의 크기를 새로운 방법으로 시도하여 측정하는 것에 그 목표를 두었다. 본 연구자가 세운 기준에 따라, 실험 결과를 보게 되면, 예약어와 연산자의 수가 적을 수록 함수의 실행 속도가 빠르다는 것을 알 수 있다. 상관 계수 또한 실험에 사용된 모든 함수들이 비교 기준에 강한 양의 상관관계를 지니고 있다.

이는 시스템 프로그래밍 언어를 구성을 할 때, 많은 예약어와 연산자를 가지고 있지 않은 언어라 하더라도, 성능과는 무관할 수 있으며, 오히려 성능이 더 좋을 수 있다. 따라서 키워드나 연산자의 수가 다른 언어들 보다 많은 시스템 프로그래밍 언어를 선택하기 보다, 적은 시스템 프로그래밍 언어를 선택해도 성능 측면에서 뒤떨어지지 않다는 것을 알 수 있으며, 시스템 프로그래밍 언어를 선택하는 데 있어서 좋은 판단의 근거가 될 수 있을 것이라 보여진다.

ACKNOWLEDGEMENT

이 논문은 2020 년도 정부 (과학기술정보통신부)의 재원으로 정보통신기획평가원의지원을 받아 수행된 연구임 (No.2014-3-00035, 매니코어 기반 초고성능 스케일러블 OS 기초연구 (차세대 OS 기초연구센터) .*교신 저자: 우균(부산대학교, woogyun@pusan.ac.kr).

참고문헌

[1] System programming Language, Wikipedia https://en.wikipedia.org/wiki/System_programming_language,(2020.10.28).

[2] 장한일 외, "스크립트 언어 비교: 프로그래밍 편의성 관점에서", 한국정보과학회, 32(2), pp. 973-975, 2005(2020.10.28).

[3] Benchmarks-Game, Debian Team https://benchmarksgame-team.pages.debian.net/,(2020.10.28).

[4] Tiobe, https://www.tiobe.com/tiobe-index/, (2020.10.28)

[5] Dennis M. Ritchie, "The Development of the C Language", Bell Labs/Lucent Technologies,1993,(2020.10.29)

[6] C Programming Language, Wikipedia, https://en.wikipedia.org/wiki/C_(programming_language),(2020.10.29)

[7] C++, Wikipedia https://en.wikipedia.org/wiki/C%2B%2B, (2020.10.30)

[8] Julia. https://julialang.org/,(2020.10.28).

[9] Swift, Wikipedia. https://en.wikipedia.org/wiki/Swift_(programming_language),(2020.10.30)

[10] Rust, Wikipedia. https://en.wikipedia.org/wiki/Rust_(programming_language) (2020.10.28)