

Python - Go 연동 실험결과

1. 구성

Golang : CGo로 C 형식의 DLL을 만듦.

Python : ctypes로 DLL을 가져와 사용.

환경 : os = windows 11, cpu = i5-13600k, ram = 32 GiB

실험 B는 980 MiB 크기의 데이터를 대상으로 7번 반복했으며, s와 MiB/s 단위로 표시된다.

<2024.04.23. 기준> stdlib5.kobj 모듈을 사용했다. kobj는 현재 데이터 입출력은 최대 1GiB, 입출력 타입은 정수(C int32), 실수(C float32), 바이트 배열(C *char)을 지원한다.

2. 실험 A (메모리 사용량 평가)

대용량 데이터는 모두 바이트 배열 형식을 통해 입출력된다. python과 go 모두 자체 GC가 있고, 관리되는 언어이다. 따라서 각 언어 측에서 할당한 메모리는 각 언어의 GC가 관리해야 하며, 다른 언어 측에서 해제할 수 없다.

PVM -> DLL : 최대 점유 메모리는 전송 데이터량의 2배이다. python 측과 go 측이 소유한 데이터는 서로 독립적이므로 메모리 사용량을 더 낮추기는 어렵다.

DLL -> PVM : 최대 점유 메모리는 전송 데이터량의 3배이다. python과 go가 각각 데이터를 소유하며, 동시에 CGo malloc으로 메모리가 할당되어야 하므로 메모리 사용량을 더 낮추기는 어렵다.

3. 실험 B (메모리 전달속도 평가)

Python은 C로 짜인 런타임 (PVM)에서 돌아가므로 기본 자료형과 C 자료형의 호환이 쉽게 가능하다. 반면 Go는 독자 런타임을 가지므로 자신의 데이터를 바로 C 형식의 자료형으로 넘겨줄 수 없다.

<데이터 왕복>	평균	최고	최저
소요시간 (s)	0.56545	0.57102	0.55832
전송속도 (MiB/s)	1733.1	1755.3	1716.2

4. 실험 C (전역 메모리 사용 평가)

DLL 파일은 하나지만, 메모리 공간은 그 공유 라이브러리를 사용하는 프로그램마다 하나씩 독립적으로 갖게 된다. DLL 파일을 사용할 때, 데이터를 계속 저장되는 전역 변수 입출력을 할 수 있었다. 하지만, DLL은 실행파일이 아니기에 진입점(main 함수)에 코드가 있어도 실행하지 않는다. 즉, 전역변수를 초기화하는 함수를 별도로 만들어 실행해 주어야 한다.

5. 실험 D (비동기 처리 평가)

goroutine을 사용하는 비동기 처리 방식 운용에 성공했다. 특정 함수를 호출하면, 그 함수는 시간이 걸리는 프로세스를 고루틴으로 호출하고, 자신은 반환된다. 고루틴은 파이썬 스레드와 무관하게 계속 돌아갈 수 있었다. 전역 변수 접근과 같이 사용하면, 처리 명령을 내리고 진행 상황을 다른 함수를 통해 가져올 수 있다.

6. 메모리 구조 해설

실험 A에서 일어난 메모리 사용을 정리했다. 앞의 사진 4개는 PVM -> DLL 입력, 뒤의 사진 4개는 DLL -> PVM 출력을 설명한다.

전체 프로그램 메모리 공간 중 힙 공간만 나타냈으며, PVM, Go DLL 공간은 각 언어의 런타임이 위치하며 각자의 GC로 관리되는 메모리가 있는 곳이다. 외부의 메모리 공간은 malloc/free 등을 통해 수동으로 관리되는 공간이다. 실험 A에서는 CGo.malloc을 통해 힙에 직접 메모리를 할당했고, 이후 다른 DLL 함수를 통해 CGo.free로 메모리 공간을 해제했다.



