Comunicación con sistemas externos

Inter Process Communication

La comunicación entre procesos o IPC, es la capacidad de comunicar diferentes procesos concurrentes.

Python cuenta con una librería multiprocessing que soporta la creación de processo mediante el uso de una API similar a threading. Ofrece concurrencia local y remota y opera fuera del Global Interpreter Lock al usar subporocesos en lugar de hilos.

El mecanismo de comunicación mas común en este tipo de aplicaciones locales es el del uso de regiones de memoria compartidas y mecanismos de control de acceso a esos recuros mediante semaforos o en casos mas simples, mecanismos de exlusión mutua o mutex.

Memoria compartida distribuida

Similar al mecanismo de comunicación local por uso de regiones de memoria compartida, este método busca hacer uso de sistemas externos para el control de registro de datos.

Sistemas de archivos distribudios

- NFS
- GFS
- PanFs

Bases de datos

MariaDB

```
# Module Imports
import mariadb
import sys

# Connect to MariaDB Platform
try:
    conn = mariadb.connect(
        user="db_user",
        password="db_user_passwd",
        host="192.0.2.1",
        port=3306,
        database="employees"

    )
except mariadb.Error as e:
```

```
    print(f"Error connecting to MariaDB Platform: {e}")
    sys.exit(1)

# Get Cursor
cur = conn.cursor()
```

postgresql

```
# Note: the module name is psycopg, not psycopg3
import psycopg

# Connect to an existing database
with psycopg.connect("dbname=test user=postgres") as conn:

    # Open a cursor to perform database operations
    with conn.cursor() as cur:

        # Execute a command: this creates a new table
        cur.execute("""
            CREATE TABLE test (
                id serial PRIMARY KEY,
                num integer,
                data text)
            """)

        # Pass data to fill a query placeholders and let Psycopg perform
        # the correct conversion (no SQL injections!)
        cur.execute(
            "INSERT INTO test (num, data) VALUES (%s, %s)",
            (100, "abc'def"))

        # Query the database and obtain data as Python objects.
        cur.execute("SELECT * FROM test")
        cur.fetchone()
        # will return (1, 100, "abc'def")

        # You can use `cur.fetchmany()`, `cur.fetchall()` to return a list
        # of several records, or even iterate on the cursor
        for record in cur:
            print(record)

        # Make the changes to the database persistent
        conn.commit()
```

MongoDB

```python
from pymongo import MongoClient
def get_database():

   # Provide the mongodb atlas url to connect python to mongodb using pymongo
   CONNECTION_STRING = "mongodb+srv://user:pass@cluster.mongodb.net/myFirstDatabase"

   # Create a connection using MongoClient. You can import MongoClient or use pymongo.MongoC
   client = MongoClient(CONNECTION_STRING)

   # Create the database for our example (we will use the same database throughout the tuto
   return client['user_shopping_list']

# This is added so that many files can reuse the function get_database()
if __name__ == "__main__":

   # Get the database
   dbname = get_database()
```

Middleware basado en mensajes

- Soap
- REST
- WebSocket

Middleware basado en objetos compartidos

- Cobra
- DCOM
- JavaSpaces
- Distributed Ruby

Middleware basado en coordinación

Ejemplos de esto son los RPC (Remote Procedure Call).

- gRPC
- D-Bus
- JSON-RPC