TASK 1

1.a Define the following briefly:

Process: is an program who running in the computer and use memory

Demon: is a process who running on the background on computer to performing specific tasks

system call: is a request make by a program to the operating system for a service

client & server is an architecture where the client send the request to the server and the server process it and send black the response to the client

peer to peer is an architecture where every node(peer) communicate witch every other node independently without a centralized server

1.b List and briefly explain the different types of Unix system call for IPC

- pipe()
  creates a pipe(unidirectional data channel) and is used for communication between related processes(parent-child).

- poll()/epoll() wait for events on file descriptors
- syslog() logs messages to the system logger can be used for inter-process logging and monitoring

1.c Explain for least 2 kernel architectures , how IPC is handled

- Monolithic kernel means that the whole operating system runs in kernel mode
  - centralized IPC management: All PIC operations are handle by the kernel
  - low-level control: provides fine-grained control over IPC mechanisms

- A microkernel prefers an approach where core functionality is isolated from system services and device drivers (which are basically just system services)
  - decoupled IPC: Separates IPC handling from core kernel functions
  - Modular Design: allows for easier extension and modification

TASK 2

2.a Define the various methods for InterProcess communication and provide advantages and disadvantages respectively

1.Message passing is a mechanism where processes send and receive messages to each other

Advantages:

- loose coupling: Processes are loosely coupled, allowing for greater independence
- Asynchronous Communication: processes can operate concurrently
- Flexibility and scalability

Disadvantages:

- Overhead: can be slower than shared memory for small messages.
- Buffering : requires careful buffer management to avoid blocking

2.Shared Memory

allows multiple processes top access the same block of memory

Advantage:

- High performance very fast frequent update to shared data
- low overhead minimal overhead per access
- Real-time capabilities suitable for real-time systems

Disadvantages:

- Synchronization challenges : requires careful synchronization to prevent race conditions
- limited scope

3. Message Queues

allow processes to send and receive messages stored in a queue

Advantages:

- FIFO ordering : messages are processed in the order received
- Multiple Readers/Writers: can Have multiple readers and writers

Disadvantages:

- Overhead: Can be slower than shared memory for small messages
- Complexity: more complex to implement
- Buffering: requires careful buffer management

-

2.b what IPC facilities are currently on your system ?

shared memory

```
(base) fuhetienne@fuhs-MacBook-Air homebrew % sysctl -a | grep sysv
kern.sysv.shmmax: 4194304
kern.sysv.shmmin: 1
kern.sysv.shmmni: 32
kern.sysv.shmseg: 8
kern.sysv.shmall: 1024
kern.sysv.semmni: 87381
kern.sysv.semmns: 87381
kern.sysv.semmnu: 87381
kern.sysv.semmsl: 87381
kern.sysv.semume: 10
security.mac.sysvmsg_enforce: 1
security.mac.sysvsem_enforce: 1
security.mac.sysvshm_enforce: 1
(base) fuhetienne@fuhs-MacBook-Air homebrew %
```

show the current activity in them

2.c create two separate program which implements inter process communication (between parent process and child process) using shared memory an pipes , using an programming language of your choice

Shared Memory Example

```
import multiprocessing
import time

def child_process(shared_value):
    print("Child process starting...")
    time.sleep(2)  # Simulate some work
    shared_value.value += 10  # Modify the shared value
    print(f"Child process updated shared value to:
{shared_value.value}")

if __name__ == "__main__":
    # Create a shared value
    shared_value = multiprocessing.Value('i', 5)  # 'i' is for integer

    print(f"Initial shared value: {shared_value.value}")

    # Create and start the child process
    p = multiprocessing.Process(target=child_process,
args=(shared_value,))
    p.start()
```

```
    # Wait for the child process to finish
    p.join()

    print(f"Parent process sees shared value: {shared_value.value}")
```

Using Pipes

```
import multiprocessing
import time

def child_process(pipe):
    print("Child process starting...")
    time.sleep(2)  # Simulate some work
    message = "Hello from child!"
    pipe.send(message)  # Send a message through the pipe
    print("Child process sent message.")

if __name__ == "__main__":
    # Create a pipe
    parent_conn, child_conn = multiprocessing.Pipe()

    # Create and start the child process
    p = multiprocessing.Process(target=child_process, args=(child_conn,))
    p.start()

    # Wait for the child process to finish
    p.join()

    # Receive message from the child process
    message = parent_conn.recv()
    print(f"Parent process received message: {message}")
```