55613 C Programming

Assignment 7 (Full)

Li Yihai - 23345919

Q5: Written Questions (15%)

a) IPC

1. **Why choose to use named pipe?**:
    - *Why Choose Named Pipes*: Named pipes are preferable when you need to establish communication between unrelated processes, especially when these processes do not share a common parent process. Named pipes are persistent and visible in the file system, making them accessible to any process with the appropriate permissions.
    - *Advantages*: They provide a form of IPC that can be used for unrelated processes and support bidirectional communication if opened in the right mode.
2. **For same issue, using `mmap`**:
    - *Solving IPC with `mmap`*: `mmap` can be used for IPC by creating a shared memory segment. Processes can map this segment into their address space and communicate by reading from and writing to this shared memory. This method is particularly useful for sharing complex data structures or large buffers.
    - *Advantages*: It allows sharing large amounts of data and complex structures between processes without the overhead of serialization.

b) Heavyweight vs Lightweight Processes

1. **Heavyweight Processes (Regular Processes)**:
    - *Advantage*: Each process has its own separate memory space, enhancing security and stability. If one process crashes, it does not affect the memory of another process.
    - *Disadvantage*: Greater resource consumption due to separate memory allocation, leading to higher overhead in context switching.
2. **Lightweight Processes (Threads)**:
    - *Advantage*: Threads share the same memory space within a process, leading to efficient communication and lower overhead for context switching.
    - *Disadvantage*: Increased complexity in synchronizing access to shared resources, potentially leading to issues like race conditions.

c) Synchronising

1. **Why it is important to do Synchronization and Coordination?**:

    - Synchronization is vital in multi-threaded applications to ensure that only one thread accesses a particular piece of data at a time. This prevents data corruption and ensures the consistency of shared resources.

2. **How to implement?**:

    - Mutexes, semaphores, and condition variables are common synchronization primitives. For instance, a mutex can be used to lock a resource, ensuring that only one thread accesses it at a time.