Karlsruhe Institute of Technology

Operating Systems 2016/17
Solutions for Assignment 6

Prof. Dr. Frank Bellosa
Dipl.-Inform. Marc Rittinghaus

T-Question 6.1: Interprocess Communication

a. What are the two fundamental models of interprocess communication? Give a short explanation for each. **2 T-pt**

Solution:

Shared Memory *With shared memory the communication peers establish a mapping to a physical memory region that both have access to. Each process can then write or read data to/from the shared memory region.*

Message Passing *Message passing uses system calls to explicitly send or receive data.*

b. Why is IPC via shared memory often more difficult to use for an application developer? **1 T-pt**

Solution:
Accessing memory with multiple threads in parallel always requires some form of synchronization between the threads to maintain consistency and avoid race conditions. The application developer has to implement this synchronization, using OS or hardware supplied synchronization mechanisms.

c. Explain the concept of mailboxes for IPC. How are mailboxes uniquely identified in Linux? **2 T-pt**

Solution:
A mailbox is an indirect messaging approach, where the sender sends the message to a mailbox instead of transmitting it directly to the receiver. The message is stored in the mailbox until the receiver reads it. In contrast to direct messaging, more than one sender and receive may share a mailbox.

In Linux, mailboxes are identified via a file name in the virtual file system (VFS).

d. You have been asked to write a server application using message-based IPC. You can choose between two request processing models: **3 T-pt**

Forking For each incoming request the server forks, creating a new worker that is responsible for processing the request. The worker exits afterwards.

Worker Pool At program start a fixed number of worker threads are created. An incoming request is directed to an idle worker thread of the pool or queued if all threads are busy.

Briefly compare the two models regarding implementation cost, resources usage (CPU, memory, etc.), and complexity of data sharing between workers.

Solution:

Complexity *The forking model is generally easier to implement for simple server applications. It does not require a synchronized message queue, instead the server can implicitly pass the message to a new worker on the stack or heap without extra synchronization.*

Resource Usage *Performing a fork for every message is expensive (CPU time) and the model is prone to DoS attacks by depleting kernel resources (PCBs, kernel stacks, etc.). Furthermore, creating more (busy) workers than CPUs available in the system wastes CPU cycles for extensive switching between the many ready worker threads. The limited number of worker threads in the pool model avoids that; however, at the cost of a potentially higher response time (i.e., the time a requests sits in the queue until a worker thread starts to process it).*

Data Sharing *Data sharing with the worker pool is very easy because all workers run in the same address space and thus implicitly share all data. When using forking, data needs to be shared explicitly, for example, through a dedicated shared memory region, because each worker runs in its own address space. However, since the workers' address spaces are cloned from the parent server process, explicit shared memory is only needed if the workers need to modify the shared data.*

Total: 8 T-pt