Lecture 11: Memory tools in embedded system communication

Shared memory: Two or more tasks that have access to the same areas in the physical memory of the computer system. Example: message passing or synchronization, peripheral device memory-mapped IO-registers. Shared memory has the properties of a so-called block device: programs can access arbitrary blocks on the device, in any sequence.
Global flags: easy, flexible, accessible. Cause **livelock** (problem where two tasks have rare-interleaving causing a break in mutual exclusion), or **spinlock** (problem when flag remains set so that the task continues to loop or wait).

Remote procedure calls: The sender (the client) asks the “receiver” (the server) to execute a procedure on its behalf, and send the results back. The server is on a different computer system or in a separate address region on the same computer.
Synchronization: Sender and receiver tasks both block until the communication is finished.
Remote invocation: Sender task waits until receiver task has sent back an acknowledgement that sender message was received.

FIFOs: Character device (First-in, First-Out) that is:
-loss-free
-non-blocking (except for the sender when the FIFO is completely full)
-exclusive (the sender and receiver are shielded from each other's accesses to data at the same time)
-1 to 1 (i.e., only one sender and one receiver)
-buffered (FIFOs put data in queue, the sender adds data on one end, the reader reads at the other end)
-no data is lost
-blocking is available for synchronization

Ring (Circular) buffer: Like a FIFO where data is read out from the “head” pointer location and data is written in through the “tail” pointer location.
-If writer pointer overruns the read pointer, data is potentially lost because when one of these pointers reaches the end of the buffer, it swaps back to the start of the buffer and continues from there.
-Data is read multiple times if the reader pointer overtakes the writer pointer.
-Acess is 1 to 1.
-Has properties of shared memory except that it can contain more than one data structure used in communcation.

Swing Buffer: An advanced circular buffer. Instead of using one single shared memory array, a swinging buffer uses two or more. The sender task fills up one of the buffers, while the receiver task empties another one. Every time one of the tasks reaches the end of its buffer, it starts operating on a buffer that the other task is not using. Both tasks operate on different data structures, hence no locks are used to access this data. Hence, a swinging buffer is loss-prone and non-blocking (because a task can fill or empty the same buffer multiple times), exclusive (with a “granularity” of one buffer size!), and 1-to-1.

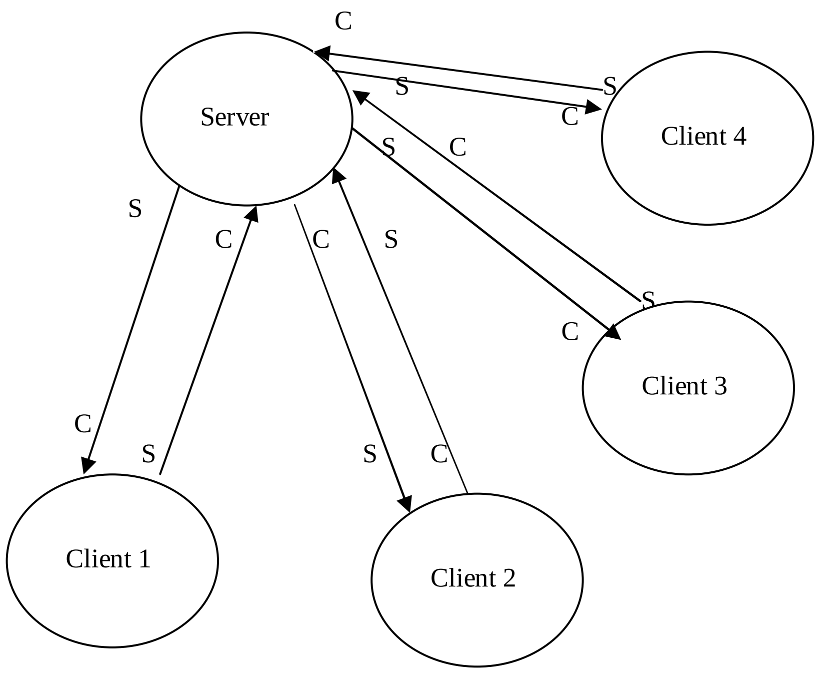
Note: Buffers in real-time applications should be locked in memory; ie they should never be “swapped” out of physical RAM due to time performance because of the large system overheads associated with paging through virtual memory.

Messages and Queues: The sender puts its message in a memory space it gets from the OS, and then sends the address of that memory to the OS, together with an identification of the receiver. The receiver asks the OS whether there are messages for it, and decides to read them or not. Reading a message is done in the same place as where it was written. Limited only by the size of physical RAM.

Lecture 12: Task communication and synchronization (cont)

Mailbox: The sender notifies the OS that it has a message, and gives the identification of the receiver. The OS then copies the message to the mailbox of the receiver. This task must read the messages in the order of arrival.   Again, only limited by the size of physical RAM.

Sockets: Software connection to a port (network node point) for the means of communicating messages between a server and multiple clients. Asynchronous parallel tasks can communicate efficiently in this manner in a 1 (server) to many (client) task (nonblocking) relationship.
Socket client: creates socket, creates memory space, connects to user supplied server address, sends message request to server, and receives data from server.
Socker server: creates socket, creates memory space, binds to socket, creates listening queue for possible client connections, accepts connection requests from clients, reads client requests and potentially spawns task to handle client requests, and writes requested data to socket.



Asynchronous socket programming diagram

Lecture 12: Task communication and synchronization (cont)

Pipes: Method of communicating (sending messages) from one process to another on the same processor. It is one way only and uniquely a 1 to 1 relationship between the parent process that writes to the child process who reads. Both sides are blocked until the "pipe" between the two is freed of a deposited message. Cannot do on multiple processors or with multiple clients. Pipes cannot prioritize messages, or handle variable length messages. And only synchronous communication is possible.

Semaphores: This name has its origin in the railroad world, where a semaphore was the (hardware) signal used to (dis)allow trains to access sections of the track: when the semaphore was lowered, a train could proceed and enter the track; when entering, the semaphore was raised, preventing other trains from entering; when the train in the critical section left that section, the semaphore was lowered again. In a computer context, a semaphore s is an integer number, with two possible function calls: up(s) and down(s). The up(s) represents a signal to increment (V(s) is sometimes used for the Dutch word Verhogen meaning to increment), and down(s) represents a wait instruction (P(s) is sometimes used for the Dutch word Proberen meaning to test). The up(s) imcrements the semaphore when an event or action is initiating. The down(s) will poll on semaphore s until it is available and reset it to 0 (or decrement it). Together up(s) and down(s) are said to implement the semaphore s.

In VxWorks, 3 types of semaphores can be distinguished:

-counting semaphore: when a task executes up(s), the semaphore counter is incremented and the task is blocked if the counter was already higher than zero; the counter is decremented when a task executes a down(s). It can count up to any integer. Multiple tasks can increment the semaphore and hence it keeps track on how many tasks are blocking on it.

-binary semaphore: the semaphore is set, or it is not set. Most efficient and fast for real-time purposes.

-mutual exclusion: a special kind of binary semaphore which is used to make sure that two tasks are excluded from each other entering the same critical section at the same time. Tasks entering a critical section raise the semaphore, and lower it when exiting the critical section.

Summary:

Communication	1 to 1	loss prone	buffer blocking	mutual exclusion
FIFO	yes	no	no	yes
Ring buffer	yes	yes	no	no
Swing buffer	yes	yes	no	yes
Shared memory	no	no	no	no
Sockets	no	yes	no	no
Pipes	yes	no	yes	yes
Semaphores	yes	no	yes	yes