Week 5

Shared Memory: It allows multiple processes to share the same physical memory. It is one of the fastest forms of Inter-Process Communication.

shared memory is a synchronous, non-blocking form of IPC
– shared memory is used when two or more processes need fast access to common data

We will look at several methods of IPC which can be categorized
according to two criteria:
(i) blocking vs. non-blocking, and
(ii) synchronous vs. asynchronous

blocking vs. non-blocking
λ blocking methods cause the transmitting process to block until the receiving process handles the transmitted data
λ non-blocking methods do not cause the transmitting process to block
– synchronous vs. asynchronous
λ in synchronous methods, the receiving process can only handle the transmitted data at a certain point in its code; Can be blocking/nonblocking
λ in asynchronous methods, the receiving process can handle the transmitted data at any point in its code

message passing has at it’s core 3 functions: MsgSend(), MsgReceive() and MsgReply()
λ two supporting functions required in Neutrino are ChannelCreate() and ConnectAttach()

server process must create a channel ChannelCreate()

client process must connect to the server’s channel ConnectAttach()

client is done communicating it calls ConnectDetach().

server is ready to close the channel it calls ChannelDestroy().

message passing is a blocking form of IPC

If the client process calls MsgSend(), before the server calls its
MsgReceive(), it becomes “Send-blocked”.

The client process remains Send-blocked until the server process calls
MsgReceive(), at which point the client becomes “Reply-blocked”.

when the server process calls MsgReply(), the client process unblocks.

when the server calls MsgReceive() it becomes "Receive-blocked”

int ChannelCreate (unsigned flags) – Create a communication channel.

int MsgReceive (int chid, void * msg, int bytes, struct _msg_info * info) – Receive a message from a client on the communication channel.

int ChannelDestroy (int chid)– Destroy the communication channel being used.

int MsgError (int rcvid, int error) – Will unblock the thread which called MsgSend with an error.

int ConnectAttach (uint32_t nd, pid_t pid, int chid, unsigned index, int flags)- Create a connection to an already established communication channel.

int MsgSend(int coid, const void* smsg, int sbytes, void* rmsg, int rbytes) – Send a message using a communication channel.

int ConnectDetach (int coid) – Detach a communication channel from a channel.

shm_open() to create a shared memory object and set access attributes associated with this object

ftruncate() to instruct the OS to set aside a fix amount of memory for the shared memory object

mmap() to map a section of it’s own memory addresses to the shared memory space

munmap() to unmap from shared memory

shm_unlink() to free up shared memory addresses

question: How to passing message? – first we open the channerl, and then need to communicate, you need to close the channel.

Week 6

Pulses – Asynchronous, non-blocking form of IPC

pulses have a maximum size of 40 bits

code - a positive integer between 1 and 127

When should the timer go off?

A timer is either relative or absolute:
– a relative timer expires "x" seconds from now (Current Time)
– an absolute timer goes off at a particular time and date

Should the timer repeat?
λ Periodic timers repeat at regular intervals

One-shot timers are just as it says time out occurs ONCE

How should the timer notify us when it goes off?
λ Send a pulse
λ Send a signal
λ Start a thread
λ Unblock the current thread (as done with sleep functions)

nanospin() that still uses this approach in QNX to achieve high precision timing

Creating a timer is done using timer_create()

Setting and starting a timer is done using timer_settime()

disable a timer using timer_settime()
– query a running timer using timer_gettime()
– extend a running timer using timer_settime()
– delete a timer using timer_delete()

Ticksize – on most systems, the default ticksize is 1ms

The time slice: – is 4 times the ticksize so it defaults to 4ms

Interrupts are first sent to the Kernel.

Hardware will send data to a Programmable Interrupt Controller (PIC).
§ The Kernel receives the Interrupt Request (IRQ) from the PIC.
§ If your program registers to have a function called to handle an IRQ then the Kernel will call your handler (Driver A)
§ Otherwise you register for a notification which sends an event to your program and
you handle the IRQ on your time.

To handle Interrupts your program must have I/O privileges.

ThreadCtl allows you to request changes to a thread. Only threads owned by ‘root’ will be granted permission

struct sigevent - This structure defines what should happen when an
event occurs. This structure is used for timers and interrupts.

Once the sigevent struct is setup you can call InterruptAttach or
InterruptAttachEvent.