A Month Off

Migrating a Robot Controller from the Proprietary INtime RTOS to Linux

CLOOS

Weld your way.
Robot arc welding at Cloos

Introduction

History
Status Quo
A tempting offer
Doing a study
How to do Linux evaluation with a Windows team?

A study with surprises on the way

Methodology
Object directories
Processes
Threads
Semaphores
Memory
What have we achieved?
Lessons learned
About me

- Born in the 70s with a fascination for electronics and computers
- German “Diplom Ingenieur” since 1997
- First job in industrial automation
- Did embedded linux systems for 17 years
- Joined Cloos robot controller team in 2019
- Married, three kids
What is this talk about?

- Cloos is a manufacturer of industrial robots
- In 2021 we decided to do a study on porting the robot controller from the proprietary INtime RTOS to Linux/Preempt-RT
- This is about our experience on the way and what we learned from it
Robot arc welding at Cloos

Introduction

History

Status Quo
A tempting offer
Doing a study

How to do Linux evaluation with a Windows team?

A study with surprises on the way

Methodology
Object directories
Processes
Threads
Semaphores
Memory

What have we achieved?
Lessons learned
Robotics at Cloos

Building robots since 1981
Inhouse designed robot controller based on Intel processors since 1986
Using the PL/M programming language and custom RTOS
1995: First robot controller based on PC hardware in industry
TenAsys INtime RTOS history

- Started as Intel RMX in the 1970s to create demand for their processors
- In 1996 RadiSys acquired the iRMX/INtime RTOS technology
- INtime 1.0 was originally introduced in 1997 in conjunction with Windows NT
- TenAsys company was founded in 2000 as a spin-off of RadiSys Corporation
Robot arc welding at Cloos

Introduction
History
Status Quo
A tempting offer
Doing a study
How to do Linux evaluation with a Windows team?

A study with surprises on the way
Methodology
Object directories
Processes
Threads
Semaphores
Memory
What have we achieved?
Lessons learned
Processes at Cloos

- Welding processes for manual and automated applications
  - Gas shielded metal arc welding
  - Tungsten inert gas welding
  - Laser welding
- Plasma and laser cutting
- Grinding
What is inside a robot controller?

▶ Industrial PC with a custom PCI card providing digital and analog I/O, plus CAN-Bus
▶ PC is EtherCAT master using a standard Ethernet controller
▶ Servo drives for all connected axes (up to 32) are EtherCAT slaves
Realtime requirements

- Axes run in cyclic synchronous position mode (CSP)
- Must provide a target position for each axis every buscycle (typical 1-4ms)
- Motion planner does all calculations in a multiple of the busclock (typical 8ms)
- Interpolation between motion planner clocks in software
Realtime requirements

- Torque Feedforward values based on dynamic model to help speed controller
- Adapt speed controller gain depending on load
- Consider values from sensors to adapt motion
The codebase

- About 3 million lines of sourcecode
- Started in the 80s in PL/M
- later converted to C
- mostly still compatible to Intel C386 compiler (ANSI C) from 1997
- split in different processes that communicate via shared mem and semaphores
- Legacy MFC-GUI
- New Qt-GUI, not feature complete, still depends on legacy GUI
The codebase

- Lots of technical debt
- Knowledge encoded from people that have left long time ago
- Will have to support legacy systems for decades (we still support building with Intel C386 from 1997)
- Carefully refactoring little by little
TenAsys INtime RTOS technology

- Separate, independent kernel outside of the Windows kernel
- Schedules all real-time processes to run first, at a higher priority than Windows
- Windows application threads communicate with their real time counterparts on the INtime kernel via NTX-API
TenAsys INtime RTOS diagram

INtime software application

- Windows process
- Real-time processes

NTX libraries

Windows executive

- Windows kernel
- Transport driver
- Real-time kernel
Robot arc welding at Cloos

Introduction
History
Status Quo
A tempting offer
Doing a study
How to do Linux evaluation with a Windows team?

A study with surprises on the way
Methodology
Object directories
Processes
Threads
Semaphores
Memory
What have we achieved?
Lessons learned
In 2021 we got a tempting offer from Keba, an automation solutions supplier.

Keba D3 controller is an automation controller running Linux.

Debian buster ia32 with Preempt-RT kernel.

Integrated safety solution suitable for robotics.

Flexcore means you can run your own realtime applications besides the Keba application.
Benefits

- Safety problem is solved
- Using off the shelf hardware
- Only one controller, no inter-device communication required
- Getting rid of license fees: Windows, INtime and EtherCAT Master
Robot arc welding at Cloos

Introduction
History
Status Quo
A tempting offer
Doing a study
How to do Linux evaluation with a Windows team?

A study with surprises on the way
Methodology
Object directories
Processes
Threads
Semaphores
Memory
What have we achieved?
Lessons learned
How far can we get in 6 weeks?

- How could we find out, if it would be possible to run our system on Keba Flexcore?
- Theoretical evaluation is hard
- I came up with the idea to do a study and see how far we can get in six weeks.
- All the realtime and GUI developers would work fulltime on this
- To my amazement, everybody agreed
Robot arc welding at Cloos

Introduction
History
Status Quo
A tempting offer
Doing a study

How to do Linux evaluation with a Windows team?

A study with surprises on the way
Methodology
Object directories
Processes
Threads
Semaphores
Memory
What have we achieved?
Lessons learned
Don’t use force, Luke

- The robot controller team members are experts with decades of experience
- Honor that
- Forcing anybody to use Linux would break motivation
- Involve everybody in the team in the decisionmaking process
- Make sure the decision is open
Consider that this is whole new world for the team, outside their comfort zone

Make sure everbody gets the time to make himself familiar

Give support wherever you can, there are no stupid questions

Don’t make fun of anyone

Encourage and join the fun when progress happens

Make the environment as comfortable as possible
Robot arc welding at Cloos

Introduction
History
Status Quo
A tempting offer
Doing a study
How to do Linux evaluation with a Windows team?

A study with surprises on the way

Methodology
Object directories
Processes
Threads
Semaphores
Memory
What have we achieved?
Lessons learned
Where to get help?

- When in doubt, consider Kerrisk
- [http://support.tenasys.com/intimehelp](http://support.tenasys.com/intimehelp)
- Axel Scholz, my knowledgeable colleague who is doing INtime and its predecessors since the 80s and has seen a lot
- We did the API implementation together
Implement the API completely in userspace as a shared library
Since the INtime application will have to be built from the same codebase, inflict as few changes as possible
Only implement the part of the API that we are actually using
grintime - generic reimplementation of the INtime API
How to get started?

- Build the application without linking the INtime libs
- See what functions are missing
- Start implementing ;)
- Use a 32-Bit Debian Buster VM for building and testing
- Evaluate Visual Studio CMake-based linux builds to support the GUI team
Grintime topology

libgrintime → grintimed

application
Robot arc welding at Cloos

Introduction
History
Status Quo
A tempting offer
Doing a study
How to do Linux evaluation with a Windows team?

A study with surprises on the way

Methodology
Object directories
Processes
Threads
Semaphores
Memory
What have we achieved?
Lessons learned
Resources like process, thread, semaphore are objects with unique 16-Bit id (RTHANDLE)

Object directory is a per process key/value store where the key is a string and the value is the 16-Bit id

Access to the object directories is global, so can be used for IPC
Object directory API

- `BOOLEAN CatalogRtHandle(RTHANDLE process, RTHANDLE object, LPCSTR name)`
- `RTHANDLE LookupRtHandle(RTHANDLE process, LPCSTR name, DWORD milliseconds)`
- `BOOLEAN UncatalogRtHandle(RTHANDLE process, LPCSTR name)`
- Object array is stored in shared memory
- libgrintime maps this memory in main()
- any object create/delete is registered there
- Each process maps this memory
- Getting the object from a 16 bit id can be done in realtime
struct system_object
{
    bool enabled;
    union {
        struct process_type process;
        struct thread_type thread;
        struct segment_type segment;
        struct heap_type heap;
        struct semaphore_type semaphore;
        struct mailbox_type mailbox;
        struct file_type file;
    };
    WORD object_type;
};
Setting up shared memory

```c
off_t size = sizeof(struct system_object) * SYSTEM_OBJECT_COUNT;
int flags = create_shm ? O_CREAT | O_EXCL : 0;

fd = shm_open(SYSOBJECT_SHM_NAME, flags | O_RDWR, S_IRUSR | S_IWUSR);
if (fd == -1)
    exit(1);
if (create_shm && (ftruncate(fd, size) == -1))
    exit(1);

system_objects = mmap(NULL, size, PROT_READ | PROT_WRITE,
                       MAP_SHARED, fd, 0);
if (system_objects == MAP_FAILED)
    exit(1);
```

13.09.2022 Dirk Eibach, Cloos Schweisstechnik GmbH
The directory itself is implemented as a std::map.
- The key is a pair of process id and object name.
- The value is the object id.
- Implemented as a gRPC service in grintimed.
- Querying the directory is not a realtime operation.
Robot arc welding at Cloos

Introduction
History
Status Quo
A tempting offer
Doing a study
How to do Linux evaluation with a Windows team?

A study with surprises on the way
Methodology
Object directories
Processes
Threads
Semaphores
Memory
What have we achieved?
Lessons learned
Process API

- **RTHANDLE** `CreateRtProcess(LPCSTR filename, LPCSTR arguments, const LPPROCESSATTRIBUTES attributes, DWORD flags)`
- **BOOLEAN** `DeleteRtProcess(RTHANDLE process, DWORD flags)`
- **BOOLEAN** `WaitForRtProcess(RTHANDLE process, DWORD milliseconds, DWORD *exit_code)`
- **VOID** `ExitRtProcess(VOID)`
main()

- main()-function in libgrintime
- Setup RT-scheduling and prio, signal handler and memory allocator
- No changes to codebase required
- To deal with the existing main(), we add -Dmain=main_grintime to the Compiler commandline
Creating a new process

```c
switch (child_pid = fork()) {
    case -1: /* fork() failed */
        ... Error handling ...
        return BAD_RTHANDLE;
    case 0: /* Child of successful fork() comes here */
        sem_wait(sem);
        printf("returns %d\n", execv(filename, argv)); /* Never returns on success; returns -1 on error */
        return BAD_RTHANDLE;
    default:
        hdl = register_process(child_pid, basename(filename));
        sem_post(sem);
        set_status_code(E_OK);
        return hdl;
}
```
Robot arc welding at Cloos

Introduction
History
Status Quo
A tempting offer
Doing a study
How to do Linux evaluation with a Windows team?

A study with surprises on the way

Methodology
Object directories
Processes
Threads
Semaphores
Memory
What have we achieved?
Lessons learned
Thread API (excerpt)

- **RTHANDLE** CreateRtThread(**BYTE** priority, **LPPROC** entry, **DWORD** stack_size, **LPVOID** parameter)
- **BOOLEAN** DeleteRtThread(**RTHANDLE** thread)
- **BYTE** GetRtThreadPriority(**RTHANDLE** thread)
- **BOOLEAN** SetRtThreadPriority(**RTHANDLE** object, **BYTE** priority)
- **BOOLEAN** SuspendRtThread(**RTHANDLE** thread)
- **BOOLEAN** ResumeRtThread(**RTHANDLE** thread)
- **BOOLEAN** RtSleep(**DWORD** milliseconds)
- **VOID** knRtSleep(**DWORD** kernel_ticks)
Found no concept for `SuspendRtThread()` and `ResumeRtThread()` in linux
So we implemented it using per thread signal handler
Using syscall `tgkill` to send the signal to a thread
Not in glibc, so this seems a little hacky
Comments welcome
Thread priorities

- INtime thread priorities range from 0(highest) to 254
- Linux realtime scheduler priorities range from 99(highest) to 1
- Fix this with lookup table, which is application specific
Robot arc welding at Cloos

Introduction
History
Status Quo
A tempting offer
Doing a study
How to do Linux evaluation with a Windows team?

A study with surprises on the way
Methodology
Object directories
Processes
Threads

Semaphores
Memory
What have we achieved?
Lessons learned
Semaphore API

- **RTHANDLE** CreateRtSemaphore(WORD init_count, WORD max_count, WORD flags)
- **BOOLEAN** DeleteRtSemaphore(RTHANDLE semaphore)
- **DWORD** WaitForRtSemaphore(RTHANDLE semaphore, WORD unit_count, DWORD milliseconds)
- **BOOLEAN** ReleaseRtSemaphore(RTHANDLE semaphore, WORD release_count)
Not much to see here

Decided to go with Posix semaphores which means we can only use count “1”

Matches our application

To support other count values, we would need to go with System V semaphores

You always have to leave some room for improvements in the future
Robot arc welding at Cloos

Introduction
History
Status Quo
A tempting offer
Doing a study
How to do Linux evaluation with a Windows team?

A study with surprises on the way

Methodology
Object directories
Processes
Threads
Semaphores

Memory

What have we achieved?
Lessons learned
Memory API (excerpt)

- PVOID AllocateRtMemory(DWORD size)
- DWORD FreeRtMemory(PVOID offset)
- RTHANDLE CreateRtMemoryHandle(PVOID offset, DWORD size)
- BOOLEAN DeleteRtMemoryHandle(RTHANDLE segment)
- PVOID MapRtSharedMemory(RTHANDLE segment)
- BOOLEAN ReleaseRtSemaphore(RTHANDLE semaphore, WORD release_count)
- DWORD GetRtPhysicalAddress(PVOID offset, DWORD size, DWORD _reserved)
- PVOID MapRtPhysicalMemory(DWORD abs_addr, DWORD size)
RTMEMORY?

THIS IS WHERE THE TEARS COME IN
Shared memory

- In INtime, all heap memory can simply be used as shared memory
- With POSIX shared memory objects this is not that easy
- Use a custom allocator
- dlmalloc can be configured to use mmap for allocations
- Allows to define a custom mmap function
- Keep track of all allocations in case they will be mapped later
ftruncate(shm_fd, shm_size + page_aligned_size);

mmap(NULL, page_aligned_size, PROT_READ | PROT_WRITE, flags, shm_fd, shm_size);

allocations[allocation_idx].ptr = ptr;
allocations[allocation_idx].size = size;
allocations[allocation_idx].offset = shm_size;
allocations[allocation_idx].mapped = false;
allocation_idx++;

shm_size += page_aligned_size;
ONE DOES NOT SIMPLY

MAP A PHYSICAL ADDRESS FROM USERSPACE
In INtime you can map physical memory addresses using `MapRtPhysicalMemory()`

To my horror this was used in our application IPC

The physical address of a data structure was sent to another process and used there

We started implementing this using the `pagemap` of the sending process and `/dev/mem`

Decided this was such a bad design decision originally that we fixed it in the application using shared mem
Robot arc welding at Cloos

Introduction
History
Status Quo
A tempting offer
Doing a study
How to do Linux evaluation with a Windows team?

A study with surprises on the way
Methodology
Object directories
Processes
Threads
Semaphores
Memory
What have we achieved?
Lessons learned
Achievements

- INtime helloworld example is working
- Cloos main realtime applications are starting up
- GUI project files have been ported to CMake
- Lots of windows dependencies removed
Robot arc welding at Cloos

Introduction

History

Status Quo

A tempting offer

Doing a study

How to do Linux evaluation with a Windows team?

A study with surprises on the way

Methodology

Object directories

Processes

Threads

Semaphores

Memory

What have we achieved?

Lessons learned
Lessons learned

- Doing a study was a good approach to the problem
- Porting a RTOS API can get quite addictive, I have spent every free minute there
- Pay respect to people and you can achieve amazing things
- Microsoft has done an amazing job in supporting Linux development from Visual Studio
- There’s still some rough edges though
Consequences

- All developers involved voted for continuing the Linux/Flexcore approach
- Component crisis hit us hard, had to suspend porting and spent the last year adding new hardware support
- grintime will be opensourced so everybody can join the fun
- Follow https://github.com/grintime to keep posted