Linux Application in Safety-Critical Environment

Some Real-Life Examples
SAFETY

- General
  - IEC 61508
- For Railway Applications
  - EN 50126, EN 50128, EN 50129
- Failures Classification
Electric Interlocking System for Underground

Collect inputs: information from track circuits

Calculate outputs

Pass calculated data to the outdoor equipment
Electric Interlocking System for Underground

Operator Workstations

PLCs

Physical interfaces

CAN bus

...
Concepts

• Fully synchronous design
• Safety-critical code isolation
• Keeping safety-critical code as simple as possible
• Software “safe states”
• Monitoring system health
• Redundancy
  • hardware
  • software
Concepts Synchronous Design

processing thread

ISR

{inputs} {outputs} {inputs} {outputs}

t₀ t₁ t₂

hardware
Concepts Synchronous design Real Time

- Guaranteed maximal response time
- Using interrupt threads to prioritize interrupts
- Carefully define priorities for
  - IRQ handler threads
  - realtime kernel threads
  - realtime userspace threads
Concepts: Safety-critical code isolation

- Define which code is SC
- Specify inputs and outputs
- Prove code correctness
- Set required priority
- Monitor health
Concepts Safe States

Inputs vector
\{0,1,1,\ldots,0,1\}

Processing...

Outputs vector

\text{timestamp: } \{0,1,1,\ldots,1\} \ |
\text{CRC}

\{0,0,0,\ldots,0\}

Sending...

$t_0$

$t_1$
Concepts Monitoring system health

- Monitor state of processes
  - alive status
  - memory consumption
  - scheduler statistics
- Generate alarms if something is going wrong
- Interchange health state between modules participating in hardware redundancy scheme
Concepts  Hardware Redundancy

• Equipment is duplicated
• Outputs are cross checked
• Different boards with different CPU architectures are used
• Clock are synchronized
Concepts Software Redundancy

- Two different implementations for safety-critical process
  - written on different languages by different developers
  - the solution is only taken when they give equal results
  - alarm is raised otherwise and safe states are guaranteed on outputs
- Two different OS versions on modules
Building System

• Kernel
• Filesystems
• Startup
• Shutdown
OSADL stable rt-linux recommendation:
- 2.6.31.12-rt21 at the moment

Standard tested configuration

RT-Preempt Patch:
- CONFIG_PREEMPT_RT=on
  - disable power management
  - disable high memory support
  - disable group CPU scheduler
Building System File System

• Read-Only root filesystem

• Separate filesystem for configuration data and logs
Building System Taking Off

- Configure system
- remove the limit of CPU usage of RT processes
- Startup the child process
- Configure priorities
- Wait for its completion
- monitor health
- feed watchdog
- Simplified

```c
#include <sys/wait.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

int main(int argc, char** argv) {
    pid_t safety_critical_process, appstatus;
    int status;

    /* System setup is going here*/
    /* ... */

    while(1) {
        safety_critical_process = fork();
        if ( safety_critical_process == -1 )
            exit(EXIT_FAILURE);

        if ( safety_critical_process == 0 ) {
            while (1) {
                /* Health monitoring actions here */
                sleep(INIT_QUANTUM);
            }
        } else {
            appstatus = waitpid(safety_critical_process, &status, 0);
            printf("exited: status %d\n restarting", status);
        }
    }
}
```
Building System Landing

- Graceful shutdown
  - explicitly tell neighbors that we are not operational anymore
  - make log record
What’s now

• We must be ready to use Linux when it will be possible to certify it for SIL3/4 systems

• Until then we are using it in proof of concept designs taking special care of its possible misbehavior:
  • avoid hazardous failures falling back to safe states aggressively
  • use additional modules to provide hardware redundancy and special system design to minimize non-hazardous failures