# Real-Time Systems Programming

## Summer-Semester 2002
## Lecture 6
## 26 April 2002

# *Overview*

- Application Programming Interfaces
- Who defines POSIX?
- POSIX-compliance – what does it mean?
- The structure of POSIX
- POSIX systems
- Mandatory vs. optional parts
- Compile-time and run-time checking
- POSIX-compliance – the application's part

# *Application Program Interfaces*

- ***Application program interface*** (***API***):
  - An abstraction layer between an application and the OS
- Application *A* complies with (= is conformant to) an API standard *S*:
  - *A* must be source code level portable across OSs that conform to *S*
- OS *O* complies with an API standard *S*:
  - *O* provides all the interface functions required by *S*
  - *O* may provide additional functions if these do not conflict with the required functions

# *POSIX*

- ***POSIX***, the ***Portable Operating System Interface***, is an API based on the UNIX process model
  - ***POSIX.1*** specified basic UNIX calls (1990)
  - ***POSIX.4*** is a set of RT extensions to POSIX.1 (1993)

# *Who Defines POSIX?*

- POSIX is a (still evolving) ***IEEE standard***
- Responsible for defining POSIX:
  - ***Portable Applications Standards Committee*** (***PASC***)
  - Is part of IEEE Computer Society
- There also exist POSIX standards from ***ISO***, ***ANSI***, and national standards organizations (such as ***DIN***)
- POSIX is developed by independent, industry wide committees
  - Includes representatives from all sides
  - Including the historically different UNIX fractions (Berkeley, AT&T, OSF, etc.)



- *See* www.pasc.org

# *POSIX is on the Shopping Lists!*

- SW procurement requirement documents (e.g. from governments) often also refer to POSIX
  - ***Example***: ***FIPS 151-2***, specified by the ***US National Institute of Standards Technology*** (***NIST***), specifies POSIX.1
  - A product may become certified against FIPS 151-2
  - An agency may become certified as a FIPS 151-2 certifier
- However, governments tend to get out of the business of specifying standards
  - Gradually shift to referring to “commonly accepted,” non-government standards
  - E.g., FIPS 151-2 was deactivated in 1998

# *POSIX Compliance – What Does It Mean?*

- Most OSs claim "POSIX-compliance" of some sort – however, it is not obvious what exactly this buys you
  - What components does POSIX consist of?
  - How do I find out which POSIX components my OS supports?

# *The Structure of POSIX*

- POSIX consists of many individual standards
  - Not all of these directly related to the OS API
- The POSIX components are identified by ***project numbers***
  - ***Example***: Core POSIX real-time extensions are typically referred to as "POSIX 1b," which refers to project number 1003.1b
- In 1994, POSIX projects have been ***renumbered***, such that projects that result in amendments to 1003.1 (POSIX.1) now all have the 1003.1 prefix
  - ***Example***: 1003.1b had been 1003.4 (POSIX.4) prior to renumbering

| New | Old | Title | Approval Date |
|---|---|---|---|
| *1003.0* | 1003.0 | Guide to POSIX OSE | 1995-03 |
| ***1003.1*** | **1003.1** | **System Interface** | **1988** |
| *1003.1a* | 1003.1a | System Interface Extensions | |
| ***1003.1b*** | **1003.4** | **Realtime** | **1993-09** |
| ***1003.1c*** | **1003.4a** | **Threads** | **1995-06** |
| *1003.1d* | 1003.4b | **Realtime Extensions** | 1999-09 |
| *1003.1e/.2c* | 1003.6 | Security | |
| *1003.1f* | 1003.8 | Transparent File Access | (withdrawn) |
| *1003.1g* | 1003.12 | Protocol Independent Interfaces | |
| *1003.1h* | | Fault Tolerance | |
| *1003.1i* | | **Fixes to 1003.1b** | 1995-06 |
| *1003.1j* | 1003.4d | **Advanced Realtime Extensions** | 2000-01 |
| *1003.1k* | | Removable Media API | (withdrawn) |
| *1003.1m* | | Checkpoint/Restart | (withdrawn) |
| *1003.1n* | | Fixes to 1003.1/1b/1c/1i | |
| *1003.1p* | | Resource Limits | |
| *1003.1q* | | Tracing | 2000-09 |
| *1003.1r* | | Alignment with Single Unix Spec | (withdrawn) |
| *1003.1s* | | Sync Clock | |

| New | Old | Title | Approval Date |
|---|---|---|---|
| *1003.2* | 1003.2 | **Shell and Utilities** | 1992-09 |
| *1003.2a* | 1003.2a | Shell and Utilities -<br>Tools & User Port. Ext. | 1992-09 |
| *1003.2b* | 1003.2b | Additional Utilities | |
| *1003.2d* | 1003.15 | Batch Processing | 1994-12 |
| *1003.4c* | 1003.4c | Realtime LIS | (withdrawn) |
| *1003.5* | 1003.5 | ***Ada*** binding to 1003.1 | 1992-06 |
| *1003.5a* | | Ada update | (withdrawn) |
| *1003.5b* | 1003.20 | Ada Realtime | 1996-06 |
| *1003.5c* | | Ada binding to 1003.1g | 1998-08 |
| *1003.5d* | 1003.5d | ADA PII - Sockets | (withdrawn) |
| *1003.5f* | | Ada binding to 1003.21 | |
| *1003.5g* | | Ada binding to Real-Time Interfaces | |
| *1003.5f* | | Ada binding to 1003.1s | |
| *1003.9* | 1003.9 | ***Fortran*** binding to 1003.1 | 1992-06 |
| *1003.10* | 1003.10 | Supercomputing profile | 1995-06 |
| *1003.11* | 1003.11 | Transaction Processing profile | (withdrawn) |

| New | Old | Title | Approval Date |
|---|---|---|---|
| *1003.11* | 1003.11 | Transaction Processing profile | (withdrawn) |
| *1003.13* | 1003.13 | Realtime profile | 1998-03 |
| *1003.13a* | | **Embedded Systems AEP** | |
| *1003.13b* | | Additional Real-time Profiles | |
| *1003.14* | | Multi-Processing profile | (withdrawn) |
| *1003.16* | 1003.16 | 1003.1 LIS-C Binding | (withdrawn) |
| *1003.17* | 1003.17 | Directory Services | 1993-03 |
| *1003.18* | 1003.18 | POSIX Profile | (withdrawn) |
| *1003.19* | 1003.19 | ***Fortran 90*** binding for 1003.1 | (withdrawn) |
| *1003.21* | 1003.21 | **Realtime Distributed System Communication LIS** | |
| *1003.22* | 1003.22 | Security Framework guide | |
| *1003.23* | 1003.23 | Guide for Developing User Organization OSE Profiles | 1998-12 |
| *1003.24* | 1003.5e | Ada binding: X Window Modular Toolkit | |
| *1224* | | ASN.1 Object Mgmt | 1993-03 |
| *1224.1* | | **DS (X.400) API - LIS** | 1993-03 |
| *1224.2* | | **DS (X.500) API - LIS** | 1993-03 |
| *1295* | | ***Motif*** | 1995 |

| New | Old | Title | Approval Date |
|---|---|---|---|
| *1326.2* | | DS (X.500) API - Test Methods for LIS | |
| *1327.2* | | DS (X.500) API - C Binding | |
| *1328.2* | | DS (X.500) API - Test Methods for C Binding | |
| *1351 & 1353* | 1238 | ACSE & Pres. Layer: LI (1351). C (1353) | 1994-09 |
| *1387* | 1003.7 | **System Administration** | |
| *1387.1* | 1003.7 | System Administration Overview | (withdrawn) |
| *1387.2* | 1003.7.2 | Software Admin | 1995-06 |
| *1387.3* | 1003.7.3 | User/Group Acct Admin | 1996-12 |
| *1387.4* | 1003.7.1 | Print Admin | (withdrawn) |
| *2000.1* | | **Year 2000 Terminology** | 1998-06 |
| *2000.1a* | | Year 2000 Terms: Date Range Invariance | |
| *2000.2* | | Year 2000: Test Methods | 1999-06 |
| *2003* | 1003.3 | **Test Methods** | |
| *2003.1* | 1003.3.1 | Test Methods for 1003.1 | 1992-12 |
| *2003.1b* | | Test Methods for 1003.1b | |
| *2003.2* | 1003.3.2 | Test Methods for 1003.2 | 1996-06 |
| *2003.5* | | Test Methods for Ada | |

## *... and it's even more complicated:*

- In addition to the project number, also need the *year* to refer to a standard precisely
- ***IEEE POSIX 1003.1-1996*** integrates
  - 1003.1-1990
  - 1003.1b-1993
  - 1003.1c-1995
- Furthermore, ***IEEE POSIX 1003.1-2001*** integrates
  - 1003.1d-1999
  - 1003.1j-2000
  - 1003.1q-2000
  - P1003.1a draft standard
  - 1003.2d-1994
  - P1003.2b draft standard
  - Selected parts of 1003.1g-2000



- *See* http://csa.compaq.com/CTJtext/Article29.html
- *See* http://www.unix-systems.org/version3/online.html

# *Which Parts do We Care About?*

- The most relevant parts for RT systems are
  - ***POSIX.1*** (1003.1): the basic OS interfaces (**`fork`**, **`exec`**, ...)
  - ***POSIX.1b*** (1003.1b, was 1003.4): the RT extensions
  - ***POSIX.1c*** (1003.1c, was 1003.4a): the threads extensions
- Each of these parts consists of
  - ***Mandatory parts*** and
  - ***Optional parts***
- Furthermore, there are still standard UNIX functions that we may care about and that are ***not part of POSIX*** – such as **`select`**

# *POSIX Systems*

- POSIX is a standard to allow ***source-code portability***
  - On a system conforming to a particular version of POSIX, one should be able to just compile and run those applications that use only those POSIX functions
- POSIX support consists of
  - A compilation system
  - Headers
  - Libraries
  - A run-time system

# *The POSIX Compilation System*

- The compilation system is supposed to support a standard language
- We'll assume that this is ***ANSI C*** (and thus, by extension, ***Java***)
- In addition (or instead), we may also have
  - Kernighan and Ritchie (***K&R***) C
  - ***Ada***
  - ***Fortran*** etc.
- May have to specify compile options to get POSIX support linked in
  - ***Example***: in ***LynxOS***, may use **`gcc -mposix1b`**

# *POSIX Headers*

- The system has to provide a set of headers that define the supported POSIX interface
- These are usually in **`/usr/include`**, but could also be elsewhere, esp. when cross-developing
- These headers are included in standard C fashion – e.g. **`#include <unistd.h>`**

# *POSIX Libraries*

- Libraries are ***pre-compiled***, vendor-supplied objects that implement the POSIX functionality
- The libraries are either
  - ***statically linked*** into the application at compile time, or
  - ***dynamically shared*** at run time
- Normally, we do not want to look at libraries
- However, we may care about
  - ***which libraries*** are used, and
  - the ***order*** in which libraries are linked in
- Sometimes, archive tools such as **`ar`** or **`nm`** may be helpful (specified in POSIX.2)

# The POSIX Run-Time System

- After building the program, the run-time system (the OS) allows you to run your program
- For non-RT systems, not embedded systems the run time system is typically also the development system
- However, for RT applications, we are often ***cross-developing***, and we have to deal with
  - The ***compilation environment***, a workstation or PC, which provides a fairly user-friendly environment
  - The ***run-time environment***, onto which an application is downloaded, which often has only bare-bones facilities

# *Mandatory vs. Optional parts*

- Each of the POSIX parts may consist of
  - ***Mandatory parts*** and
  - ***Optional parts***
- Much of the functionality that is relevant for us is ***not mandatory*** (semaphores, RT signals, shared memory, message queues etc.)
- Therefore "POSIX.1b compliance" is still not necessarily enough!
- We still need means to find out the details about the system that we are running on

# *Mandatory vs. Optional POSIX.1*

- POSIX.1 is fairly monolithic – that is, ***most of it is in the mandatory part***
- The only optional parts are those that were present in some UNIX systems, but not in others
- Examples for optional capabilities:
  - Suspension and resumption of process groups
  - Restricted **chown**
  - No silent truncation of overlong pathnames
  - Disabling of some terminal characters

# *Mandatory vs. Optional POSIX.1b*

- POSIX.1b is considered less basic, and therefore there is only a very small mandatory part
  - *Example*: the presence of real-time queued signals (**`SA_SIGINFO`**, **`SIGRTMIN`**, **`SIGRTMAX`**)
- ***Everything else is optional!***
  - This includes for example additional signal functions (**`sigwaitinfo`**, **`sigtimedwait`**, **`sigqueue`**)
- *Thus, "POSIX.1b" compliance by itself means very little!*

# *How to Find Out What is There*

- A solid development path has to consist of two parts:

***1. Compile-time checking***

- ➢ Which ***version*** of POSIX am I using?
- ➢ Are the ***options*** present that are needed by my application?
- ➢ What are the numerical ***limits***?

***2. Run-time checking***

- ➢ What is the run-time ***system configuration***?
- ➢ What is the ***file-dependent configuration***?
- ➢ How is the ***implementation-defined behavior*** (e.g., what I/O operations are permitted for certain file types)?

- The latter is especially relevant for cross-development

# *Which Version of POSIX?*

- The macro **_POSIX_VERSION** tells us
  - whether POSIX is present at all
  - if yes, which version of POSIX we are talking about

```
#include <unistd.h>

main() {
  /* Is this is at all a POSIX system? */
#ifndef _POSIX_VERSION
  printf("POSIX is not supported!\n");

#else
  /* Retrieve the POSIX version */
  printf("_POSIX_VERSION = %d\n", _POSIX_VERSION);
#endif
}
```

# *Examples of POSIX Versions*

- On a ***Solaris*** system (***SunOS 5.7***), the test program returns the following version:

```
daisy{rvh} cc posix-test.c; a.out
_POSIX_VERSION = 199506
daisy{rvh}
```

- On ***Linux Kernel 2.4.0*** (Suse 7.1 distribution), we get the same result:

```
rvh@sekunde> cc posix-test.c; a.out
_POSIX_VERSION = 199506
rvh@sekunde>
```

# *Typical POSIX Versions*

- The common values of **`_POSIX_VERSION`** are
  - ***198808***: August 1988  is the approval date of POSIX.1 as IEEE standard
    - This usually implies that the system passes the US FIPS 151-1 test suite
  - ***199009***: The system conforms to the 1990, ISO version of POSIX
    - This is not significantly different from 198808
    - However, the system then also passes FIPS 151-2, which is better and harder to pass
  - ***199309***: mandatory POSIX 1003.1b (real-time) is supported
  - ***199506***: mandatory POSIX 1003.1c (threads) is supported
  - ***200112L***: supports POSIX 1003.1-2001

# *Which Options are Present?*

- There is a ***feature test macro*** defined for every optional part of POSIX
- For example, we can find out whether a system supports message passing as follows:

```
#include <unistd.h>

main() {
  /* Compile-time check */
#ifdef _POSIX_MESSAGE_PASSING
  printf("Message passing is supported\n");

#else
  printf("Message passing is not supported\n");
#endif
}
```

# *Examples of Message Passing Support*

- ***SunOS 5.7*** (***Solaris 7***) supports message passing:

```
daisy{rvh} cc posix-test.c; a.out
  Message passing is supported
daisy{rvh}
```

- However, ***Linux 2.4.0*** does not support it:

```
rvh@sekunde> cc posix-test.c; a.out
  Message passing is not supported
rvh@sekunde>
```

# *What are the Numerical Limits?*

- If a feature is present, there may still be ***numerical limits*** associated with this feature
- Again there are feature test macros for this, defined in **`<limits.h>`**
- The standard specifies ***minimal limits***
  - These are also defined in <limits.h>
  - If our application requires more, we have to test the actual system limits

# *Example of a Numerical Limit*

- The results on ***SunOS 5.7:***

```
#include <unistd.h>
#include <limits.h>

main() {
#ifdef _POSIX_MESSAGE_PASSING
  printf("Message passing is supported\n");
  printf("POSIX standard: at most %d \
      message queues per process\n",
      _POSIX_MQ_OPEN_MAX);
#ifdef MQ_OPEN_MAX
  printf("This system: at most %d \
      message queues per process\n", MQ_OPEN_MAX);
#endif
#endif
}
```

```
Message passing is supported
POSIX standard: at most 8 message queues per process
```

# *Run-Time Checking*

- So far, we have discussed compile-time capabilities for examining our system capabilities
- However, there are several reasons to perform run-time checks as well:
  - The development platform may not be the target platform, and there may be uncertainties about the target platform
  - The target platform may change over the lifetime of our application
  - There may be implementation-dependent behavior for which there are no feature test macros available

# *POSIX Run-Time Checking*

- POSIX provides the following functions, declared in **`<unistd.h>`**:
  - **`sysconf`**: tests for the presence, absence, and numerical limits of an option on a per-system basis
  - **`pathconf`**: as **`sysconf`**, but testing is done on a per-file basis
  - **`fpathconf`**: as **`pathconf`**, but takes instead of a path a file descriptor to a file that we have already opened

```
#include <unistd.h>

long sysconf(int name);
long pathconf(const char *pathname, int name);
long fpathconf(int fd, int name);
```

# Per-System Run-Time Checks

```c
#include <errno.h>
#include <stdio.h>
#include <unistd.h>

#define CHECK_ERRNO if (errno != 0) \
  { perror("Error in posix-test"); errno = 0; }

main() {
  /* Have to reset errno */
  errno = 0;

  /* Run-time check for presence of message passing */
  printf("sysconf(_SC_MESSAGE_PASSING) = %d\n",
    sysconf(_SC_MESSAGE_PASSING));
  CHECK_ERRNO;

  /* sysconf() may also return a numeric value */
  printf("sysconf(_SC_OPEN_MAX) = %d\n", sysconf(_SC_OPEN_MAX));
  CHECK_ERRNO;

  /* Better check errno for validity of argument! */
  printf("sysconf(12345) = %d\n", sysconf(12345));
  CHECK_ERRNO;
}
```

# *Run-Time Results Per System*

- The results on ***SunOS 5.7***:

```
sysconf(_SC_MESSAGE_PASSING) = 1
sysconf(_SC_OPEN_MAX) = 64
sysconf(12345) = -1
Error in posix-test: Invalid argument
```

- The results on ***Linux 2.4.0***:

```
sysconf(_SC_MESSAGE_PASSING) = -1
sysconf(_SC_OPEN_MAX) = 1024
sysconf(12345) = -1
Error in posix-test: Das Argument ist ung?ltig
```

# *Per-File Run-Time Checks*

```
...

main() {
  /* Have to reset errno */
  errno = 0;

  /* Request for file-specific information */
  printf("pathconf(\"my_file\", _PC_SYNC_IO) = %d\n",
    pathconf("my_file", _PC_SYNC_IO));
  CHECK_ERRNO;

  /* Can also get information on directories */
  printf("pathconf(\".\", _PC_NAME_MAX) = %d\n",
    pathconf(".", _PC_NAME_MAX));
  CHECK_ERRNO;

  /* pathconf() complains if file does not exist! */
  printf("pathconf(\"non_existing_dir\", _PC_NAME_MAX) = %d\n",
    pathconf("non_existing_dir", _PC_NAME_MAX));
  CHECK_ERRNO;
}
```

# *Run-Time Results Per File*

- The results on ***SunOS 5.7***:

```
pathconf("my_file", _PC_SYNC_IO) = 1
pathconf(".", _PC_NAME_MAX) = 255
pathconf("non_existing_dir", _PC_NAME_MAX) = -1
Error in posix-test: No such file or directory
```

- The results on ***Linux 2.4.0***:

```
pathconf("my_file", _PC_SYNC_IO) = -1
pathconf(".", _PC_NAME_MAX) = 255
pathconf("non_existing_dir", _PC_NAME_MAX) = -1
Error in posix-test: Datei oder Verzeichnis
nicht gefunden
```

# *POSIX Compliance of the Application*

- So far, we looked at how the ***OS*** expresses its POSIX compliance
- However, the ***applications*** may also have varying degrees of POSIX compliance
- An application can express that it uses the POSIX API functions (and ***only*** those functions) by defining **`_POSIX_C_SOURCE`** with a value that expresses the POSIX version it is conforming to
- ***Note***: Earlier versions of POSIX just defined **`_POSIX_SOURCE`**, without giving it a value

```
#define _POSIX_C_SOURCE 199506
```

# *Summary on the Mechanics of POSIX*

- POSIX is a collection of ***IEEE standards***
  - The aim is ***source-code compatibility***
  - POSIX is in many areas today's ***de-facto industry standard***
- POSIX consists of many different parts, mostly stemming from ***UNIX***, and originally designed for ***C***
  - However, OSs such as Windows NT and applications written in languages such as Ada or Fortran can claim POSIX compliance as well
- The most relevant parts for us are
  - ***POSIX.1***
  - ***POSIX.1b*** (was: POSIX.4): RT extensions
  - ***POSIX.1c*** (was: POSIX.4a): threads

# *Summary on POSIX Compliance Checking*

- ***Compile-time checking***, using feature test macros:
  - Test for **`_POSIX_VERSION`**
  - Test for presence of ***options*** required by app
  - If ***numerical limits*** guaranteed by the standard are not sufficient, evaluate the actual limits
- ***Run-time checking***, esp. when cross-developing:
  - Check system configuration, using **`sysconf`**
  - Check file-dependent configuration, using **`pathconf`** or **`fpathconf`**
  - Perform ad-hoc checks on ***implementation-dependent behavior***

# *To Go Further*

- ***POSIX and RT programming in general:***
  - ☞ [Gallmeister 1995]
- ***IEEE POSIX Std 1003.1-2001:***
  - ☞ http://www.unix-systems.org/version3/online.html

# *Problem Set 3 – Due: 2 May 2002*

1) What are the pros and cons of using an API standard? Discuss from the standpoint of the *application developer*, the *OS developer*, and the *final user*. **(3 pts)**
2) Which version of POSIX does the OS you are typically using support? **(2 pts)**
3) In the IEEE POSIX Std 1003.1-2001 document
   a) What are the differences between "may", "can", and "should"? **(1 pt)**
   b) What are the differences between "unspecified" and "undefined"? **(1 pt)**
4) Find two features where Solaris and Linux currently differ in their level of POSIX support (apart from the ones mentioned in class) **(2 pts)**