Group 11

POSIX-compliance and the quest for grand unified operating system standard

Poopa Kaewbuapan
64130500226

Micko Kok
64130500203

Ratchanon Burong
64130500229

Surapong Keawwongvan
64130500251

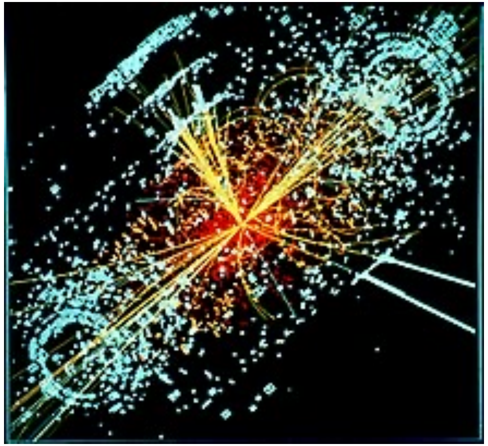
Punn Kanploy
64130500262

CSC217 Operating Systems
May 9, 2023

What is a Grand Unified Theory (GUT)?

"..a model in which, at high energies, the electromagnetic, weak, and strong forces are ***merged into a single force.****"*

Wikipedia, Grand Unified Theory (2023)



"Just borrow the name!"

Have you seen this before?

No worries, no particle physics crash course today.

GUT of OS standard

iOS

mac OS

Monolithic Kernel

android

Microkernel,
Baremetal

Current OS standard implemetation

Portable Operating System Interface (POSIX)

==

Single UNIX Specification (SUS)* UNIX® An Open Group Standard

Compliance to an OS standard is that the development of an OS meets some requirements of the standard, while being certified is meeting all requirements and has been tested by the standard comittee

* SUS is equivalent to POSIX in the requirement sense but focused only on UNIX and not OS agnostic

POSIX in a nutshell

- was created because the preceding C language standards weren't enough to make programs written for UNIX and Linux compatible
- Maintained by IEEE and The Open Group
- Ensures source code level portability
- POSIX.1 is the standard for an application programming interface in the C language
- POSIX.2 is the standard for shell and utility interface for the OS

Latest revision: POSIX-2017
(5 years old!)

The Open Group Base Specifications Issue 7, 2018 edition
IEEE Std 1003.1™-2017 (Revision of IEEE Std 1003.1-2008)
Copyright © 2001-2018 IEEE and The Open Group

POSIX.1-2017 is simultaneously IEEE Std 1003.1™-2017 and The Open Group Technical Standard Base Specifications, Issue 7.

POSIX.1-2017 defines a standard operating system interface and environment, including a command interpreter (or "shell"), and common utility programs to support applications portability at the source code level. POSIX.1-2017 is intended to be used by both application developers and system implementors and comprises four major components (each in an associated volume):

- General terms, concepts, and interfaces common to all volumes of this standard, including utility conventions and C-language header definitions, are included in the Base Definitions volume.
- Definitions for system service functions and subroutines, language-specific system services for the C programming language, function issues, including portability, error handling, and error recovery, are included in the System Interfaces volume.
- Definitions for a standard source code-level interface to command interpretation services (a "shell") and common utility programs for application programs are included in the Shell and Utilities volume.
- Extended rationale that did not fit well into the rest of the document structure, which contains historical information concerning the contents of POSIX.1-2017 and why features were included or discarded by the standard developers, is included in the Rationale (Informative) volume.

The following areas are outside the scope of POSIX.1-2017:

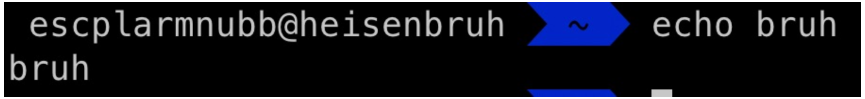
- Graphics interfaces
- Database management system interfaces
- Record I/O considerations
- Object or binary code portability
- System configuration and resource availability

POSIX.1-2017 describes the external characteristics and facilities that are of importance to application developers, rather than the internal construction techniques employed to achieve these capabilities. Special emphasis is placed on those functions and facilities that are needed in a wide variety of commercial applications.

Equivalent to: IEEE Std. 1003TM-2017, SUSv4

Still, why POSIX matters?

macOS X (UNIX-certified)*



Windows 11 (POSIX-compliant – WSL2 subsystem)

Linux (POSIX-compliant)

Being adhered to POSIX allows reproducibility and stable behavior of application across platforms!

* Equivalent to being POSIX-certified

Still, why POSIX matters? (extra info figure)

KORNSHELL	Korn Shell version 1988 (ksh88)	POSIX-compliant (incorporated in Solaris 2.x UNIX)	Proprietary (AT&T Research)
BASH THE BOURNE-AGAIN SHELL	Bourne Again Shell (bash)	POSIX-compliant	Open source (GNU)
C	malloc()	POSIX-certified (part of a specification – C stdlib memory allocation)	

Still, half a decade is quite a while

Key Hypothesis

POSIX-2017 is obsolete in a
modern OS requirement setting

What should be the move towards a novel grand unified OS standard?

- *2 examples of POSIX limitation*

+ 2 propositions of POSIX augmentation

#1

Associative Array

Example 1

```
$ages = array("Peter"=>32, "Quagmire"=>30, "Joe"=>34);
```

Example 2

```
$ages['Peter'] = "32";
$ages['Quagmire'] = "30";
$ages['Joe'] = "34";
```

Displaying Associative Array

```
<?php
$ages['Peter'] = "32";
$ages['Quagmire'] = "30";
$ages['Joe'] = "34";
echo "Peter is " . $ages['Peter'] . " years old.";
?>
```

Output

Peter is 32 years old.

https://www.scmgalaxy.com/tutorials/complete-tutorial-of-associative-arrays/

First POSIX showdown: Shell Programming – associative array

ksh88 vs bash

Finding:
Strong POSIX-compliant command shell has limited functionalities compared to weaker POSIX-compliant with utility extensions

POSIX Showdown I – associative array

- Associative array is essentially a <Key, Value> pair (refer to array indices by string)
- Associative array is not in the specification of POSIX-2017
- bash natively supports associative array meanwhile to replicate the associative array behavior in ksh88 or strong POSIX-compliant shells required the use of eval() which can introduces side-effects [1]
- Bash implements associative array as an extension aside from POSIX-compliance for ease of use (a proof that achieving POSIX pureness tends to be undesirable for user's end)

```
## ainit STEM
## Declare an empty associative array named STEM.
ainit () {
  eval "__aa__${1}=' '"
}
## akeys STEM
## List the keys in the associatve array named STEM.
akeys () {
  eval "echo \"\$__aa__${1}\""
}
## aget STEM KEY VAR
## Set VAR to the value of KEY in the associative array named STEM.
## If KEY is not present, unset VAR.
aget () {
  eval "unset $3
        case \$__aa__${1} in
          *\" $2 \"*) $3=\$__aa__${1}__$2;;
        esac"
}
## aset STEM KEY VALUE
## Set KEY to VALUE in the associative array named STEM.
aset () {
  eval "__aa__${1}__${2}=\$3
        case \$__aa__${1} in
          *\" $2 \"*) :;;
          *) __aa__${1}=\"\${__aa__${1}}$2 \";;
        esac"
}
```

https://unix.stackexchange.com/questions/111397/associative-arrays-in-shell-scripts

[1] https://stackoverflow.com/questions/2571401/why-exactly-is-eval-evil

#2

Second POSIX showdown:
GUI Programming

ncurses vs …

Finding:

…

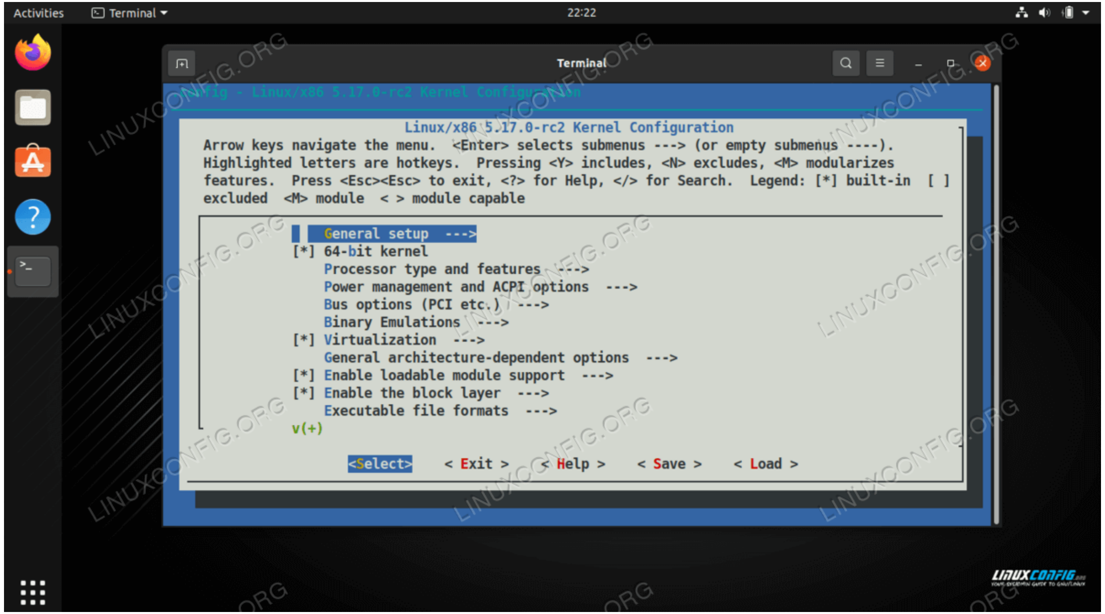
None.

None.

The following areas are outside the scope of POSIX.1-2017:

- Graphics interfaces

POSIX Showdown II – ncurses and TUI



+ Open source

- Arrow keys/ Virtual cursor navigation constraint

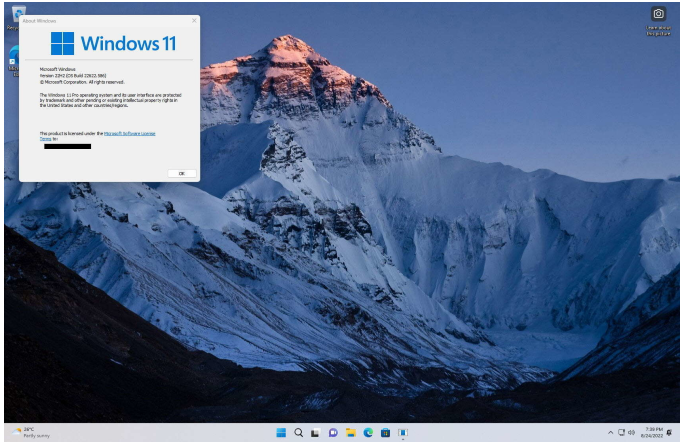
- framebuffer, Limited ANSI color palette

A Linux kernel compilation setting dialog using ncurses [1] POSIX-compliant library for text-based user interface(TUI)

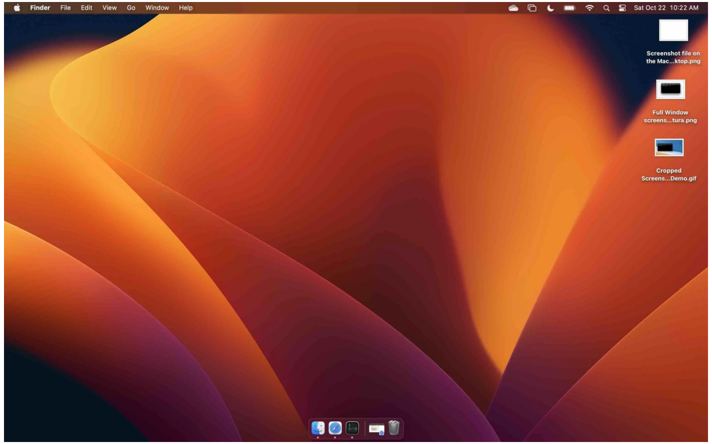
Best of POSIX interface rendering resides only in terminal because the purpose till today is just to mainly serve mainframes and 24/7 computer systems, but increasing of modern Linux desktop demand can be concerning…

[1] https://invisible-island.net/ncurses/

POSIX Showdown II – To GUI is to give money



Windows 11 –
Aero (DE*)+ DWM/Explorer (WM**)



macOS X 13 Ventura –
Aqua (DE)+ Quartz Compositor (WM)

Proprietary Model => Revenue => Hire more UX/UI Designers

= Superb user-friendliness
but *vender-locked* UI

* Desktop Environment is WM + External Graphics Libraries
** Window Manager/Compositor manages layout and interleaving of application windows

POSIX Showdown II – freedesktop.org (FOSS imple.)

X.Org Server 11 (FOSS/Proprietary – X.Org Found.)

Wayland (FOSS)

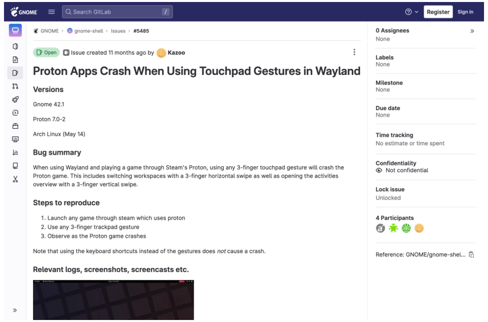
Display Server Protocol

Desktop Env.

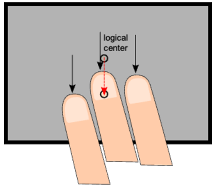
GNOME™

Window Mng.

Almost all implementations that is Linux-oriented and FOSS is POSIX-compliant via the conformance of C API usage, but the GUI environment is still considered immature and finickily



Example of Wayland bug filing



https://gitlab.gnome.org/GNOME/gnome-shell/-/issues/5485

Finding:

POSIX doesn't define graphics interface in the specification, which the implementation of Wayland and the others (FOSS) can be a starting point for GUI programming unification

#1

First POSIX Proposition: Memory-safe programming language backbone

malloc() (C) vs Rust

Key Proposition:

+ Transitioning from C to viable low-overhead functional PL
+ Formal-verifiable memory allocation

POSIX Propo. I – *malloc()* behavior

The Open Group Base Specifications Issue 7, 2018 edition
IEEE Std 1003.1-2017 (Revision of IEEE Std 1003.1-2008)
Copyright © 2001-2018 IEEE and The Open Group

NAME

malloc - a memory allocator

SYNOPSIS

```
#include <stdlib.h>

void *malloc(size_t size);
```

DESCRIPTION

[CX] The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *malloc*() function shall allocate unused space for an object whose size in bytes is specified by *size* and whose value is unspecified.

The order and contiguity of storage allocated by successive calls to *malloc*() is unspecified. The pointer returned if the allocation succeeds shall be suitably aligned so that it may be assigned to a pointer to any type of object and then used to access such an object in the space allocated (until the space is explicitly freed or reallocated). Each such allocation shall yield a pointer to an object disjoint from any other object. The pointer returned points to the start (lowest byte address) of the allocated space. If the space cannot be allocated, a null pointer shall be returned. If the size of the space requested is 0, the behavior is implementation-defined: either a null pointer shall be returned, or the behavior shall be as if the size were some non-zero value, except that the behavior is undefined if the returned pointer is used to access an object.

RETURN VALUE

Upon successful completion with *size* not equal to 0, *malloc*() shall return a pointer to the allocated space. If *size* is 0, either:

- A null pointer shall be returned [CX] and *errno* may be set to an implementation-defined value, or
- A pointer to the allocated space shall be returned. The application shall ensure that the pointer is not used to access an object.

Otherwise, it shall return a null pointer [CX] and set *errno* to indicate the error.

ERRORS

The *malloc*() function shall fail if:

POSIX-2017 says when application use malloc() and the memory is not enough, *it should return NULL* and let application handles the error…

POSIX Propo. I – *malloc()* behavior

But Out-of-Memory (OOM) killer refuses!

When malloc() Never Returns NULL—Reliability as an Illusion

Gunnar Kudrjavets
University of Groningen
9712 CP Groningen, Netherlands
g.kudrjavets@rug.nl

Jeff Thomas
Meta Platforms, Inc.
Menlo Park, CA 94025, USA
jeffdthomas@fb.com

Aditya Kumar
Snap, Inc.
Santa Monica, CA 90405, USA
adityak@snap.com

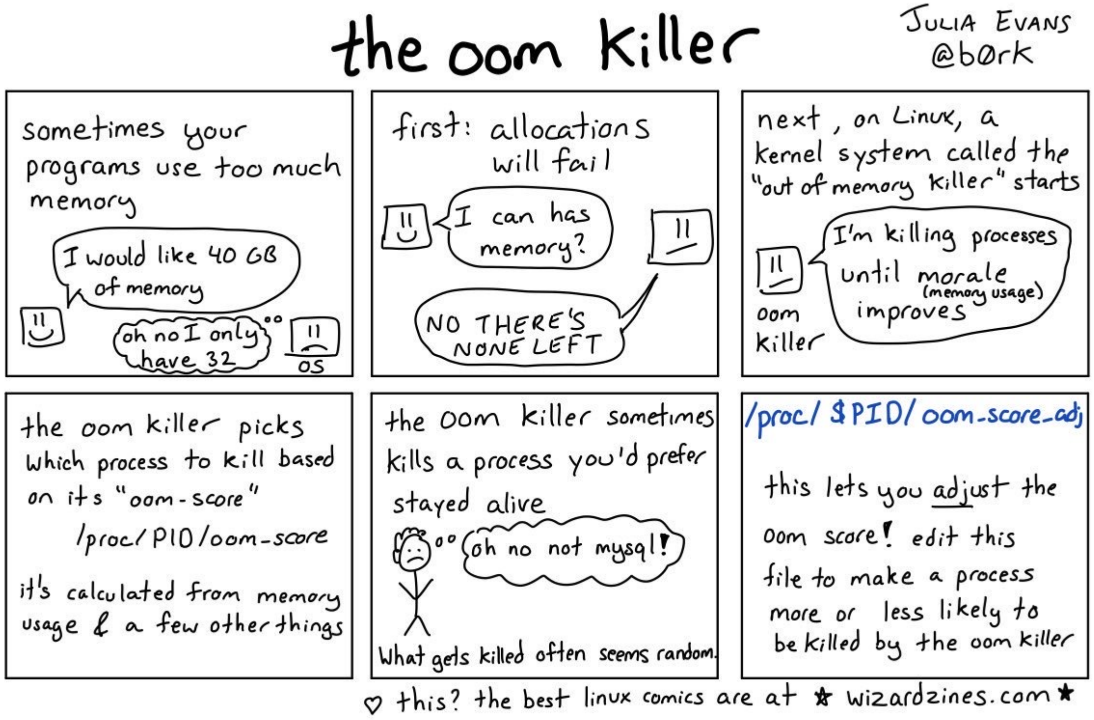
Nachiappan Nagappan
Meta Platforms, Inc.
Menlo Park, CA 94025, USA
nnachi@fb.com

Ayushi Rastogi
University of Groningen
9712 CP Groningen, Netherlands
a.rastogi@rug.nl

This paper [1] presents a behavior of many POSIX-compliant OSes as letting kernel OOM killer handles the allocation error instead of malloc() invoked by the application, thus violate the definition and introduce reliability issues

[1] Kudrjavets et al., 2023, arXiv preprint https://arxiv.org/abs/2208.08484

POSIX Propo. I – *malloc()* behavior



OOM killer illustration (courtesy of Evans, 2019 https://twitter.com/b0rk/status/1133216877839360001)

POSIX Propo. I – Getting Rust-y (intended)

Now that malloc() has been caught of spurious behavior,
let us shipped the memory-safe! (intended x2)

Theseus: an Experiment in Operating System Structure and State Management

Kevin Boos
Rice University

Namitha Liyanage
Yale University

Ramla Ijaz
Rice University

Lin Zhong
Yale University

A research endeavor [1] in radical OS changes, replacing C with Rust, single-level kernel ring, <u>make memory allocation safe via Rust type system</u>, and much more.
Theseus can be a viable choice of OS design to be merged in POSIX revision, given enough time to mature.

[1] Boos et al., 2020, OSDI '14 https://www.usenix.org/conference/osdi20/presentation/boos

POSIX Propo. I – Getting Rust-y

- Take advantage of Rust's powerful abilities
  - Rust compiler checks many built-in safety invariants
    - e.g., memory safety for objects on stack & heap
  - Extend compiler-checked invariants to *all* resources

- *Intralingual* design requires:
  1. Matching compiler's expected execution model
  2. Implementing OS semantics fully within strong, static type system

Theseus on memory allocation
(Boos et al., OSDI '14, slides p.10)

POSIX Propo. I – Getting Rust-y

Rust properties

- low-level esque with acceptable overhead compared to C in real-time operations
- A functional PL with strong type system
- Memory safety (via borrow checker)
- Formal-verificable [1] ensures correctness

Rust makes memory allocation so that only statically complie time exception can occurs.

Which mean that theoretically a complier can detect and prevent every exceptions that can occur before runtime phase, thus constituting to almost error-free runtime process.

[1] Jung et al., 2018, RustBelt: Securing the Foundations of the Rust Programming Language https://doi.org/10.1145/3158154

#2

Second POSIX Proposition:
Let POSIX certification
free of cost

POSIX Fee Schedule

Key Proposition:

+ Eliminate POSIX Fee Schedule Scheme
+ Encourage FOSS contributions

POSIX Propo. II – Go rich, getting certified

Wonder why your favorite FOSS Linux distro aren't getting some cool UNIX-certified badge and only macOS can?


THE Open GROUP
Making standards work®

Home · About · A-Z Index · Search · Contacts · Press · Register · Login

The Open Brand Fee Schedule

Certified Products

» Certification Register

Certification Guides

» Open Brand Guide
» UNIX V7 Guide
» UNIX 03 Guide

Documents

» Trademark License Agreement (TMLA)
» TMLA Signature page
» Trademark Usage Guide (TMUG)
» Checklists
» Fee Schedule
» Test Requirements
» Test Suites
» Product Standards
» Registration Forms
» FAQ

Conformance Statements

» Blank Templates
» Search

Links

» Problem Reports and Interpretations

BACK You are here: The Open Group > The Open Brand Certification Program > Fee Schedule

Reformatted Nov 6 2003

1. License Fees

The brand fees and royalties in this table are controlled by and due under the terms and conditions of the Open Brand Trademark License Agreement (Jan 1998). This document is an integral part of, and should be read in conjunction, with Schedule 2 of the above agreement.

When signing the Agreement the Licensee must :

1. Provide to The Open Group (*X/Open Company Limited trading as The Open Group*) a statement of the Company Revenue from which the Core Program Royalty buy-out is calculated.
2. Notify The Open Group which of the Additional Programs the Licensee is applying to join.
3. Notify The Open Group of the applicable fee band for each Additional Program (where appropriate).

Fees	**US$**
Trademark License Agreement (upon execution of the agreement)	2,500
Annual License Fee (second and subsequent years)	1,000
Territory Special Registration Fee (per territory)	2,200

Discounts:
The fee upon execution of the Trademark License Agreement and the Annual License Fee are waived for members of The Open Group who **only** have Identity Management Forum entitlement and VSLDAP test suite licenses and current VSLDAP support contracts.

2. Product Registration Fees (*Component and Profile*)

Fees	**US$**
Registration fee per Product Standard	800


Open Brand Certificate

This is to certify that

Apple Inc.

has entered into a Trademark License Agreement with X/Open Company Limited in accordance with which the following are registered under the X/Open Brand Program

UNIX 03

Registration No: P1217

macOS version 13.0 Ventura
on Apple silicon-based Mac computers

Date of first issue: 9 August 2022
Next renewal date: 9 August 2023

License No: L3064

THE Open GROUP

President and Chief Executive Officer
X/Open Company Limited
A wholly owned subsidiary of The Open Group

The Open Brand is the internationally accepted guarantee of conformance to recognized standards. Consult the vendor's Conformance Statement(s) for important details about the Registered Product(s). Conformance Statements and a Register of Certified Products can be found on The Open Group web site at http://www.opengroup.org/csqs and http://www.opengroup.org/openbrand/register

Motif, OSF/1, UNIX, and the "X" Device are registered trademarks, and The Open Group is a trademark of The Open Group in the US and other countries.
©Copyright 2022 The Open Group. All rights reserved.

Even considering an annual renewal fee is still 400$, topping all off with a whopping 110000$ *(assume more than 30000 copies sale)* for applying to extra UNIX program!

[1] https://www.opengroup.org/openbrand/Brandfees.htm

POSIX Propo. II – Speaking, Sparking, Sparkling

Even POSIX fee schedule [1] doesn't help, although slightly being graceful.

1003.1™-2016 Base Product Standard **Excluding subsidiary product standards.** Payment due upon application to certify a product for the 1003.1-2016 Base product standard separately to the subsidiary product standards. The subsidiary product standards must already be registered.	**$1,250**	**$2,500**

From our study so far, we were convinced that latest POSIX revision unable to address modern OS requirements, supporting that

Obsolete Standard + High Entry Fee => No Adoption => No contributions

We suggest that POSIX or a novel grand unified OS standard should restructure the fee schedule to a more resonable and accessible price or abolish the fee system to increase the influx of ecosystem contributors in a FOSS sense, with a solid code of conduct and regulations.

[1] POSIX®: *Certified by IEEE and The Open Group* Fee Schedule V1.4, 2023 http://get.posixcertified.ieee.org/docs/posix-fee-schedule.pdf

Epilogue – Future Studies Direction

1. Microkernels Standard (Redefining IPC)
- Current IPC implementation on POSIX favors monolithic kernel, how can we revamp the IPC so it can handle more communications from microkernel architecture?

2. Using of Formal Verification in checking OS model specification correctness
- To accommodate the ease of using Formal Verification, OS model should be able to incorporate high-level functional language i.e. Haskell into the specification or optimize Automated Theorem Prover to approximate answer within real-time bound.

seL4: Formal Verification of an OS Kernel

Gerwin Klein[1,2], Kevin Elphinstone[1,2], Gernot Heiser[1,2,3]
June Andronick[1,2], David Cock[1], Philip Derrin[1*], Dhammika Elkaduwe[1,2‡], Kai Engelhardt[1,2]
Rafal Kolanski[1,2], Michael Norrish[1,4], Thomas Sewell[1], Harvey Tuch[1,2†], Simon Winwood[1,2]
[1] NICTA, [2] UNSW, [3] Open Kernel Labs, [4] ANU
ertos@nicta.com.au

Thanks! Read our manuscript for extra references 🤓

<u>Group 11 Term Paper Manuscript – CSC217 Operating Systems</u>

Mr. Poopa Kaewbuapan 64130500226

Mr. Micko Kok 64130500203

Mr. Ratchanon Burong 64130500229

Mr. Surapong Keawwongvan 64130500251

Mr. Punn Kanploy 64130500262