# User space

A modern computer operating system usually segregates virtual memory into **kernel space** and **user space**.[a] Primarily, this separation serves to provide memory protection and hardware protection from malicious or errant software behaviour.

Kernel space is strictly reserved for running a privileged operating system kernel, kernel extensions, and most device drivers. In contrast, user space is the memory area where application software and some drivers execute.

## Contents



## Overview

The term **userland** (or user space) refers to all code that runs outside the operating system's kernel.[1] Userland usually refers to the various programs and libraries that the operating system uses to interact with the kernel: software that performs input/output, manipulates file system objects, application software, etc.

Each user space process normally runs in its own virtual memory space, and, unless explicitly allowed, cannot access the memory of other processes. This is the basis for memory protection in today's mainstream operating systems, and a building block for privilege separation. A separate user mode can also be used to build efficient virtual machines – see Popek and Goldberg virtualization requirements. Depending on the privileges, processes can request the kernel to map part of another process's memory space to its own, as is the case for debuggers. Programs can also request shared memory regions with other processes, although other techniques are also available to allow inter-process communication

Various layers within Linux, also showing separation between the userland and kernel space

<table>
<tr><td rowspan="5">User mode</td><td>User applications</td><td colspan="5">For example, bash, LibreOffice, GIMP, Blender, 0 A.D., Mozilla Firefox, etc.</td></tr>
<tr><td>Low-level system components:</td><td>System daemons: systemd, runit, logind, networkd, PulseAudio, ...</td><td>Windowing system: X11, Wayland, Mir, SurfaceFlinger (Android)</td><td colspan="2">Other libraries: GTK+, Qt, EFL, SDL, SFML, FLTK, GNUstep, etc.</td><td>Graphics: Mesa, AMD Catalyst, ...</td></tr>
<tr><td>C standard library</td><td colspan="5"><code>open()</code>, <code>exec()</code>, <code>sbrk()</code>, <code>socket()</code>, <code>fopen()</code>, <code>calloc()</code>, ... (up to 2000 subroutines)<br>glibc aims to be POSIX/SUS-compatible, uClibc targets embedded systems, bionic written for Android, etc.</td></tr>
<tr><td rowspan="4">Kernel mode</td><td rowspan="4">Linux kernel</td><td colspan="5"><code>stat</code>, <code>splice</code>, <code>dup</code>, <code>read</code>, <code>open</code>, <code>ioctl</code>, <code>write</code>, <code>mmap</code>, <code>close</code>, <code>exit</code>, etc. (about 380 system calls)<br>The Linux kernel System Call Interface (SCI, aims to be POSIX/SUS-compatible)</td></tr>
<tr><td>Process scheduling subsystem</td><td>IPC subsystem</td><td>Memory management subsystem</td><td>Virtual files subsystem</td><td>Network subsystem</td></tr>
<tr><td colspan="5">Other components: ALSA, DRI, evdev, LVM, device mapper, Linux Network Scheduler, Netfilter<br>Linux Security Modules SELinux, TOMOYO, AppArmor, Smack</td></tr>
<tr><td colspan="7">Hardware (CPU, main memory, data storage devices, etc.)</td></tr>
</table>

# Implementation

The most common way of implementing a **user mode** separate from kernel mode involves operating system protection rings.

Another approach taken in experimental operating systems is to have a single address space for all software, and rely on the programming language's virtual machine to make sure that arbitrary memory cannot be accessed – applications simply cannot acquire any references to the objects that they are not allowed to access.[2][3] This approach has been implemented in JXOS, Unununium as well as Microsoft's Singularity research project.

# See also

- BIOS
- CPU modes
- Memory protection

# Notes

a. Older operating systems, such as DOS and Windows 3.1x, do not use this architecture.

# References


1. "userland, n." (http://www.catb.org/jargon/html/U/userland.html). *The Jargon File*. Eric S. Raymond. Retrieved 2016-08-14.
2. "Unununium System Introduction" (https://web.archive.org/web/20011215052223/http://uuu.sourceforge.net/si.php#SEC6). Archived from the original (http://uuu.sourceforge.net/si.php#SEC6) on 2001-12-15. Retrieved 2016-08-14.
3. "uuu/docs/system_introduction/uuu_intro.tex" (http://uuu.cvs.sourceforge.net/viewvc/uuu/uuu/docs/system_introduction/uuu_intro.tex?view=markup) *UUU System Introduction Guide* 2001-06-01. Retrieved 2016-08-14.

# External links

- Linux Kernel Space Definition
- Entering User Mode at the Wayback Machine (archived March 26, 2016)