AZ UBUNTU ON WINDOWS ÉS EGY KIEGÉSZÍTÉSÉNEK BEMUTATÁSA

Dr. Pócza Krisztián

Kivonat

A natív Linux binárisok futtatásának lehetősége a Windows Subsystem for Linux (WSL) révén került be a Windows 10-be *Bash on Ubuntu on Windows* néven. Elsődleges célja, hogy alapvetően Linuxra készült konzolos felhasználói és fejlesztői alkalmazások portolás nélkül, az eredeti binárist telepítve futtathatók legyenek Windowson. Ezt a célt többé-kevésbé meg is valósították, azonban a grafikus alkalmazások hiánya szembeötlő. Cikkem első felében ismertetem a WSL működését, technológiai megoldásait, majd a második részben egy saját kiegészítést mutatok be (https://github.com/kpocza/LoWe), amely segítségével akár mplayer vagy X alapú GUI alkalmazások is futtathatók.

Tartalomjegyzék

1. Microsoft ♥ Linux

A kétezres évek elején tapasztalható Linux-ellenes hangulat után (Steve Balmer (akkori CEO): „Linux is a cancer!” – „A Linux olyan mint a rák”) a Microsoft gyökeresen változáson ment keresztül. A Linux használójává, támogatójává lépett elő, amely köszönhető a szemléletváltásnak, nyílt gondolkodásmódnak, valamint a cég élén történt változásoknak is (Satya Nadella, mint CEO).

Az *Azure* már az indulása után röviddel támogatta a Linux-alapú virtuális gépek futtatását az alatta található Hyper-V technológia segítségével. A későbbiekben számos szerver terméket (Redis, Hadoop, No-SQL adatbázisok, deployment rendszerek) is elérhetővé tett ugyanitt (1).

Kiadásra kerül a *PowerShell (2),* valamint az *SQL Server 2016* (3) is a nyílt forrású operációs rendszerre, valamint a *Linux Foundation platina* (4) szintű támogatójává avanzsálódott a Microsoft.

A *Bash on Ubuntu on Windows* (5) teszi fel a koronát a történtekre, amelyet jelen cikk is részletesen tárgyal.

Nem egy nyilvános fórumon került már kijelentésre: *„Microsoft loves Linux”*. Ez a kijelentés nem meglepő, hiszen mindenki szereti azt, amiből rengeteg pénzt keres…

2. Unix/Linux futtatása Windowson

Több 3rd party megoldás is lehetővé teszi, hogy Linux shell és egyéb alkalmazásokat futtassunk Windows környezetben: *Cygwin (6)*, *MinGW (7)*.

Amerikai kormányzati elvárásoknak megfelelően a Windows NT rendszereket még a kilencvenes években felvértezték a *POSIX v1* támogatásával a Microsoft POSIX subsystem (8) formájában. Ezt a megoldást a Windows XP/Windows Server 2003-ban felváltotta az *SFU (Windows Services for Unix) (9)*. Míg a POSIX subsystem csak a POSIX v1 standard funkcióit implementálta az SFU már hálózati alkalmazásokat és szervereket (telnet, NFS, NIS, Perl, cron, GNU utilities, stb.) is biztosít.

Csak történeti szempontból lényeges, hogy a Microsoft Research berkein belül elindult a *Project Drawbridge* (10), azaz a *Library OS*, amely több későbbi megoldás alap építőkövét biztosította. Ezt a jelen cikk későbbi fejezetében részletesen tárgyaljuk még.

Az Android mélyén Linux dübörög. A *Project Astoria*, azaz *Windows Bridge for Android* célja natív Android alkalmazások Windowson való futtatása volt. A projektet később leállították (11).

Ez azonban nem teljesen igaz, ugyanis a Project Astoria újjászületett *Bash on Ubuntu on Windows* azaz *LXSS* azaz *WSL (Windows Subsystem for Linux)* néven, amely konzolos, fejlesztői alkalmazások natív futtatását valósítja meg.

3. Bash on Ubuntu on Windows

A *Windows 10 Anniversary Update* (2016 augusztusa) tette először elérhetővé a Bash on Ubuntu on Windows-t (továbbiakban: *Ubuntu on Windows*, *UoW*) (5). A cikk hátralevő részében az Ubuntu on Windows, UoW és LWS kifejezéseket felváltva használjuk.

Az UoW konzolos, fejlesztői Linux alkalmazások és szerverek natív futtatását valósítja meg Windows rendszeren. Számos *fejlesztőeszköz*, *nyílt forráskódú szerver alkalmazás* nem vagy csak késéssel jelenik meg Windowsra. Jó megoldás lehetne ezen szoftverek portolása, azonban egy Windows kernelen futó Linux alaprendszer, disztribúció sokkal előremutatóbb, időtállóbb megoldás. Így nincs szükség az említett alkalmazások újabb verzióinak folyamatos portolására sem.

Ezzel a megoldással a Microsoft ugyan a Windowshoz láncolja a fejlesztőket,

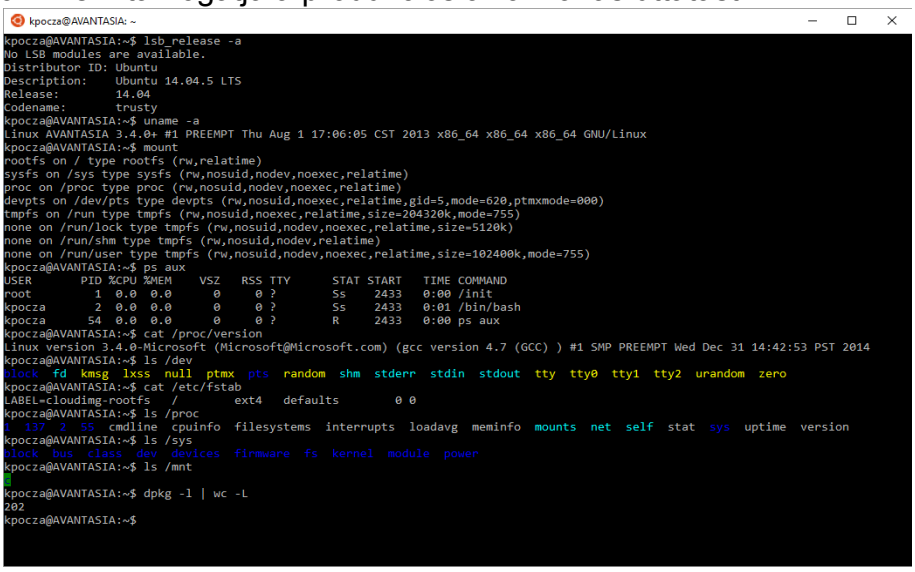
azonban az elkészült alkalmazás Linux szerveren futtatható, valamint terjeszti a *GNU* és egyéb *szabad/nyílt* alkalmazásokat is. Ezen felül lehetővé válik a fejlesztői eszközöket futtató gépen, környezetben egyéb jól megszokott alkalmazások (böngésző, levelező, stb.) futtatása is (váltogatás nélkül, lásd VM).

A beta állapotú komponens csak fejlesztői módban (*Developer mode*) látszik *„Windows Subsystem for Linux (beta)”* néven opcionálisan telepíthető komponensként (12). Telepítés után a bash.exe indításával válik az UoW használhatóvá. A *Canonical*al megvalósult együttműködés eredményeképpen az *Ubuntu 14.04 LTS* verzió *natív* (ELF), *64 bites* binárisai töltődnek le.

A bash.exe indítása nem rendszer szinten, hanem *felhasználó szinten* telepíti az UoW-t a *C:\Users\{UserName}\AppData\Local\lxss* útvonalra NTFS fájlrendszerre.

Más disztribúció is felhekkelhető az LWS környezetre nem hivatalosan támogatott módon. A Windows 10 Creators Update Ubuntu 16 LTS-re frissíti a rendszert.

Az UoW nem támogatja a produkciós alkalmazásfuttatást.



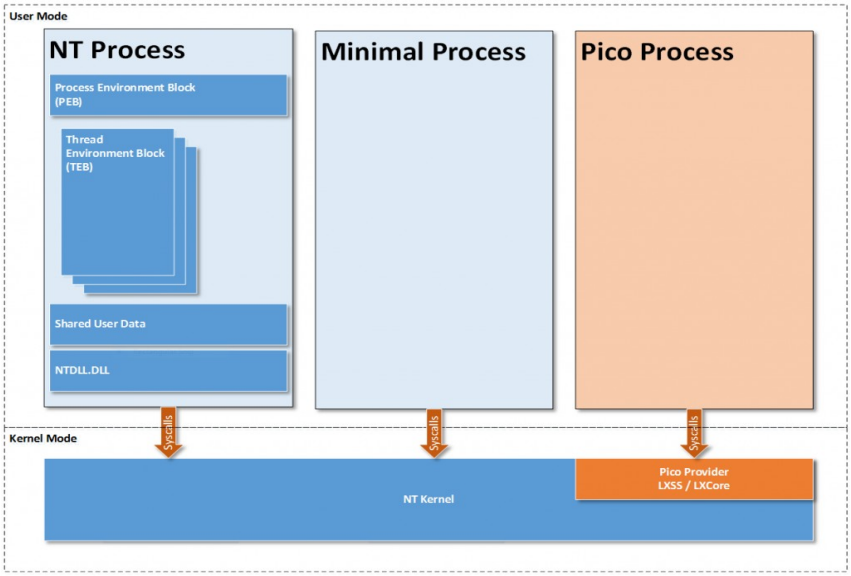
```
kpocza@AVANTASIA: /mnt/c/Prog/LoWe/src/LoWeAgent
 ProgRuntimeDispatcher.cpp  ProgRuntimeHandler.cpp
bool ProgRuntimeHandler::SpySyscallEnter()
{
        struct user_regs_struct regs;

        _currentDeviceHandler = NULL;

        ptrace(PTRACE_GETREGS, _pid, NULL, &regs);
        _syscall = regs.orig_rax;

        _log.Debug("SYSCALL:", _syscall);

        if(_syscall == SYS_open)
        {
                ReadRemoteText(regs.rdi, _openpath, sizeof(_openpath));
                _currentDeviceHandler = _deviceHandlerFactory.Create(_openpath, _pid);
        }
        else
        {
                if(_syscall == SYS_ioctl || _syscall == SYS_read || _syscall == SYS_write)
                {
                        long fd = (long)regs.rdi;
                        _currentDeviceHandler = _deviceHandlerRegistry.Lookup(fd);
                }
                if(_syscall == SYS_mmap)
                {
                        long fd = (long)regs.r8;
                        _currentDeviceHandler = _deviceHandlerRegistry.Lookup(fd);
                }
        }

        if(_currentDeviceHandler!= NULL)
        {
                _currentDeviceHandler->ExecuteBefore(_syscall, regs);
        }

        _exiting=!_exiting;

        return true;
                                                                    21,1          23%
```

4. Technológiai háttér

Ebben a fejezetben áttekintjük a WSL, mint Linux alrendszer alapvető működési ismérveit, technológiai megoldásait.

4.1. Pico process

A *Project Drawbridge* vezette be a *Pico Process* (13) fogalmát, amelyet a következő ábra magyaráz el részletesebben:



A klasszikus *NT process*ek, folyamatok Win32-es DLL-eket, szálakat, felhasználói és infrastruktúra adatokat és kódot, valamint felhasználói programkódot tartalmaznak. Egy ilyen process nem alkalmas natív Linux binárisok futtatására,

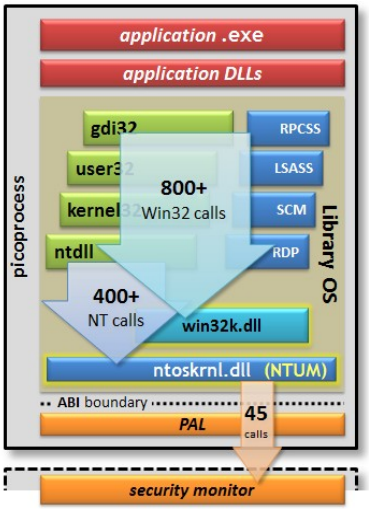
ugyanis a Win32-es alapkód már befészkelte magát. Az ábrán másodikként szereplő un. *Minimal Process* megfelelő lehetne erre a célra, ugyanis nem tartalmaz semmi extra kódot csak a felhasználói programét, amit beletöltünk; azonban a rendszerhívások NT rendszerhívásokként manifesztálódnának.

A teljes problémakört az un. *Pico Process* oldja meg, ahol a Minimal processhez hasonlóan az alapértelmezett process nem tartalmaz felhasználói és rendszerkódot, azonban a kernel hívásokat a Pico Provider kezeli, amely képessé tehető Linux-rendszerhívások kezelésére. A Pico process indulásakor meghatározásra kerül, hogy melyik *Pico Provider* felelős a tőle érkező rendszerhívások kezelésére, ezáltal tetszőleges környezet, alrendszer, operációs rendszer emulálható általa.

4.2. Library OS kitekintés

A *Project Drawbidge* keretein belül az MS Research a 2010-es évek elején alkotta meg a *Library OS*-t (10) és ezzel együtt az imént ismertetett Pico processt. Ahogy a Library OS elnevezés sejteti, az operációs rendszer kernelének nagy részét user módba költöztették, a valódi rendszerhívások nagy részét így inprocess megoldhatták. Az összesen 400 NT rendszerhívás helyett mindössze 45-el elérték azt, hogy viszonylag komoly felhasználói programok is futhassanak ebben a környezetben. Ezt a container-jellegű környezetet távoli asztal szolgáltatással érték el.

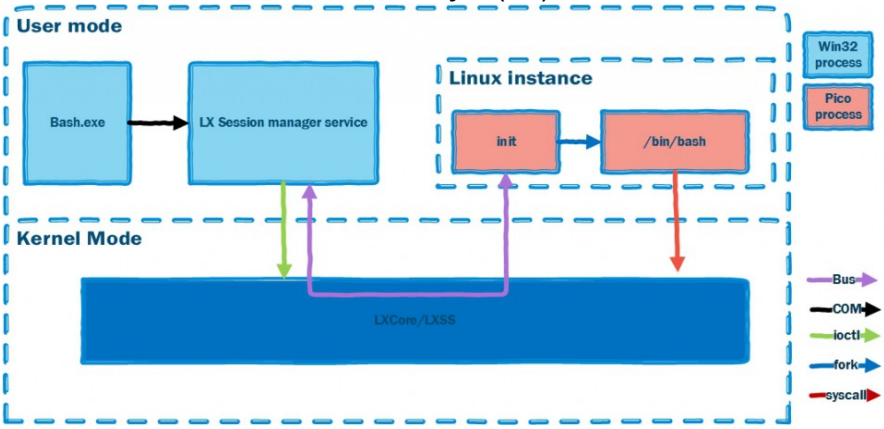
A következő ábra ad mélyebb betekintést a Library OS felépítésébe:



Ez a megoldás jelenti az alapját az *MS SQL Server 2016* Linuxon futó változatának is. Az SQL Server Win32 hívások tömkelegét végzi, mivel szorosan összeszőtt a Win32 és NT környezettel. Az a döntés született, hogy a Linux-ra „portolás” során nem ezek a hívások kerülnek lecserélésre, hanem a Win32 alaprendszert a Windows kernel egy bizonyos részével egyetemben mozgatják Linux alá pont ugyanúgy, ahogy a Library OS esetében is történik. A lényegi különbség annyi, hogy a valódi rendszerhívások (amelyek nem kezelhetők user módban) a Linux kernelhez érkeznek be. Linux kernel felett fut gyakorlatilag a Windows kernel egy csökkentett verziója a szokásos dll-ekkel valamint .NET frameworkkel együtt, amely egy hihetetlen mérnöki teljesítményként értékelhető.

4.3. LWS Architektúrája

A következő ábra az LWS architektúráját (14) szemlélteti:



Folyamat oldalról megközelítve a kérdést a bash.exe indítása után a *LX Session Manager Service* egy kernel módú buson keresztül elindítja az init processt. Az *init* process feladata jelen esetben a */bin/bash* ELF bináris futtatására koncentrálódik. Az init és a /bin/bash Pico processek egyazon Linux instanceba töltődnek be. Az ezután futtatott bash.exe már a korábban indított Linux instanceban indítja el a /bin/bash-t. Ez azt jelenti, hogy ezek a processek látják egymást, képesek egymással kommunikálni, a soron következő */dev/ttyX* kerül allokálásra részükre.

Mivel jelenleg Windows felhasználónként egy, független Linux instance jön létre, ezért ezek konkurensen belépett Windows felhasználónként függetlenek. A Linux instance megléte előre vetíti, hogy akár Windows felhasználónként több ilyen instance is futhasson.

A /bin/bash és további felhasználói programok az *LXCore/LXSS* Pico Provideren keresztül érik el a kernel szolgáltatásait.

4.4. Linux Rendszerhívások

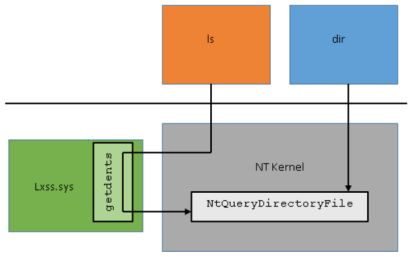
A egyik, ha nem legfontosabb szolgáltatás a kernel részéről a rendszerhívások biztosítása (15). Az korábban említett Pico Provideren keresztül a következő Linux rendszerhívások érhetők el:

accept, accept4, access, alarm, arch_prctl, bind, brk, capget, capset, chdir, chmod, chown, clock_getres, clock_gettime, clock_nanosleep, clone, close, connect, creat, dup, dup2, dup3, epoll_create, epoll_create1, epoll_ctl, epoll_wait, eventfd, eventfd2, execve, exit, exit_group, faccessat, fadvise64, fchdir, fchmod, fchmodat, fchown, fchownat, fcntl64, fdatasync, flock, fork, fsetxattr, fstat64, fstatat64, fstatfs64, fsync, ftruncate, ftruncate64, futex, getcpu, getcwd, getdents, getdents64, getegid, getegid16, geteuid, geteuid16, getgid, getgid16, getgroups, getpeername, getpgid, getpgrp, getpid, getppid, getpriority, getresgid, getresgid16, getresuid, getresuid16, getrlimit, getrusage, getsid, getsockname,getsockopt, gettid, gettimeofday, getuid, getuid16, getxattr, get_robust_list, get_thread_area, inotify_add_watch, inotify_init, inotify_rm_watch, ioctl, ioprio_get, ioprio_set, keyctl, kill, lchown, link, linkat, listen, llseek, lseek, lstat64, madvise, mkdir, mkdirat, mknod, mlock, mmap, mmap2, mount, mprotect, mremap, msync, munlock, munmap, nanosleep, newuname, open, openat, pause, perf_event_open, personality, pipe, pipe2, poll, ppoll, prctl, pread64, process_vm_readv, process_vm_writev, pselect6, ptrace, pwrite64, read, readlink, readv, reboot, recv, recvfrom, recvmsg, rename, rmdir, rt_sigaction, rt_sigpending, rt_sigprocmask, rt_sigreturn, rt_sigsuspend, rt_sigtimedwait, sched_getaffinity, sched_getparam, sched_getscheduler, sched_get_priority_max, sched_get_priority_min, sched_setaffinity, sched_setparam, sched_setscheduler, sched_yield, select, send, sendmmsg, sendmsg, sendto, setdomainname, setgid, setgroups, sethostname, setitimer, setpgid, setpriority, setregid, setresgid, setresuid, setreuid, setrlimit, setsid, setsockopt, settimeofday, setuid, setxattr, set_robust_list, set_thread_area, set_tid_address, shutdown, sigaction, sigaltstack, sigpending, sigprocmask, sigreturn, sigsuspend, socket, socketcall, socketpair, splice, stat64, statfs64, symlink, symlinkat, sync, sysinfo, tee, tgkill, time, timerfd_create, timerfd_gettime, timerfd_settime, times, tkill, truncate, truncate64, umask, umount, umount2,unlink, unlinkat, unshare, utime, utimensat, utimes, vfork, wait4, waitpid, write, writev

A fenti Linux rendszerhívásokat (16) három csoportba osztjuk aszerint, hogy hogyan kerültek megvalósításra WSL-en:

1. Egy az egyben lefedhető hívások azok, amelyekre van megfelelő Windows NT hívás
2. Saját implementációt igénylő hívások, ahol még hasonló NT hívás sem áll rendelkezésre
3. Előkészítő műveletek után egy NT hívás futtatható

Például egy directory entry lekérdezésre úgy történik, hogy az UoW-s process behív a *getdents64* rendszerhívásba a Pico Provideren keresztül, amely utána áthív az *NtQueryDirectoryFile* NT hívásba.



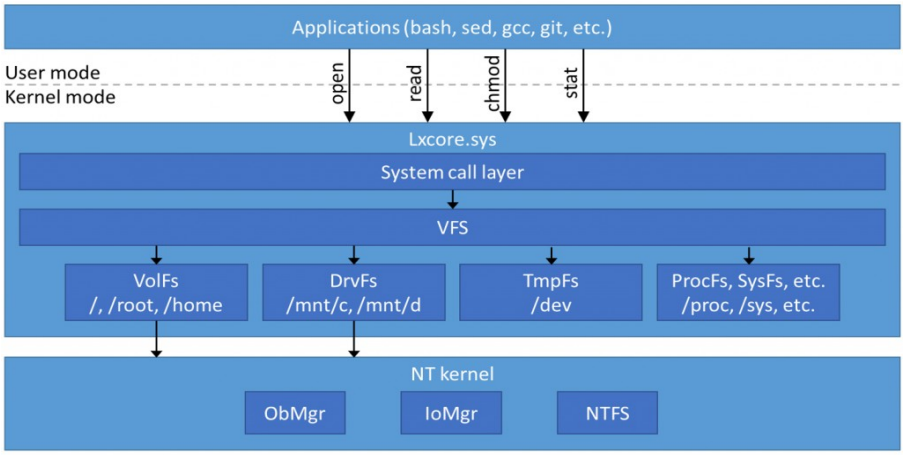
A Linuxos és Windowsos hívási konvenciók eltérnek ugyan, de abban megegyeznek, hogy az *rax* regiszter tartalmazza a rendszerhívás sorszámát, a *syscall* utasítás végzi a rendszerhívást, valamint az *rax* adja vissza a státusz kódot. A további regiszterek eltérnek, azonban ez nem okoz problémát, ugyanis a Pico Provider tetszőleges módon értelmezheti a regiszterek jelentését.

Lépés	getdents	NTQueryDirectoryFile
1	mov rax, __NR_getdents64	mov rax,#NtQueryDirectoryFile
2	mov rdi, Fd	mov rcx, Foo
3	mov rsi, Buffer	mov rdx, Bar
4	mov rdx, sizeof(Buffer)	mov r8, Baz
5	Syscall	Syscall
6	cmp rax,0xFFFFFFFFFFFFF001	test eax, eax

4.5. Fájlrendszer

A UNIX-ban így a Linuxban is *minden file* (“Everything is a file”). Ezt azt jelenti, hogy a fizikai fájlok mellett a rendszer állapotát, eszközeit is fájlok reprezentálják. Ez utóbbiak a kernel belső “lelki világának” egy megjelenési formáját manifesztálják.

A következő ábra részletesen bemutatja az LWS megvalósítását (17):



A *fizikai Linux fájlrendszerek* (/, /root, /home) mellett elérhető a */mnt/{driverletter}* útvonalon a szintén fizikailag létező Windows NTFS meghajtók adatai. *Virtuális fájlrendszerként* jelenik meg a */dev*, a */proc* és a */sys*.

Az UoW a *C:\Users\{UserName}\AppData\Local\lxss* útvonalon található (azaz a *felhasználó directoryjában*) *NTFS* fájlrendszeren. Az NTFS jogosultsági rendszere nagyban eltér a Linux fájlrendszer jogosultsági rendszerétől. A háromszintű RWX valamint a setuid, setgid és sticky biteket az *NTFS extended attribútumok* segítségével kezeli a rendszer a színfalak mögött.

4.6. Továbbiak

A hálózati szolgáltatások (18) kezelése szempontjából alapvető fontossággal bíró *socketek* a *Windows hálózati stack*jén kereszült valósulnak meg. A localhost *közös port-tartománnyal bír*, amely azt jelenti, hogy az NT és az UoW alatt futó hálózati funkciók ugyanazokat a portokat látják. A *DNS* az */etc/resolv.conf* mindig naprakészen tartásával valósul meg a Windowson megadott névszerver beállítások replikációján keresztül.

A Windows NT és az UoW alatt futó *processzek egymást korlátozott módon látják* (NT látja az UoW alattiakat, amit fordítva már nem igaz). Egymással együttműködni korlátozottan tudnak, amelyet a Windows 10 Creators Update segít némiképp (19).

Fontos megjegyezni, hogy az UoW egy némiképp zárt környezet jogosultsági szempontból. Az UoW processei a Windows felhasználó jogosultságaival futnak, legyen az normál felhasználó vagy rendszergazda. Ez a *sudo* szempontjából azt jelenti, hogy a normál felhasználó csak a UoW-ben lesz root, egyéb szolgáltatások, pl. az */mnt/c* fájljai továbbra is a normál felhasználó jogosultságaival lesznek kezelve és nem a rendszergazdáéval.

5. LoWe

Mint ahogy már említésre került korábban, a *UoW nem támogatja* a *hangeszközök* kezelését, a *grafikus* valamint *X* programok futtatását sem. Az ok, hogy a szükséges device-ok (/dev/fb0, /dev/snd/*, /dev/input/mice, stb.) nem kerültek implementálásra.

Megjegyezzük, hogy az X alkalmazások futtathatók 3rd party X server segítségével (20).

A ***Linux on Windows extender*** azaz *LoWe* (21) a következő funkciókkal bővíti ki az UoW-t:

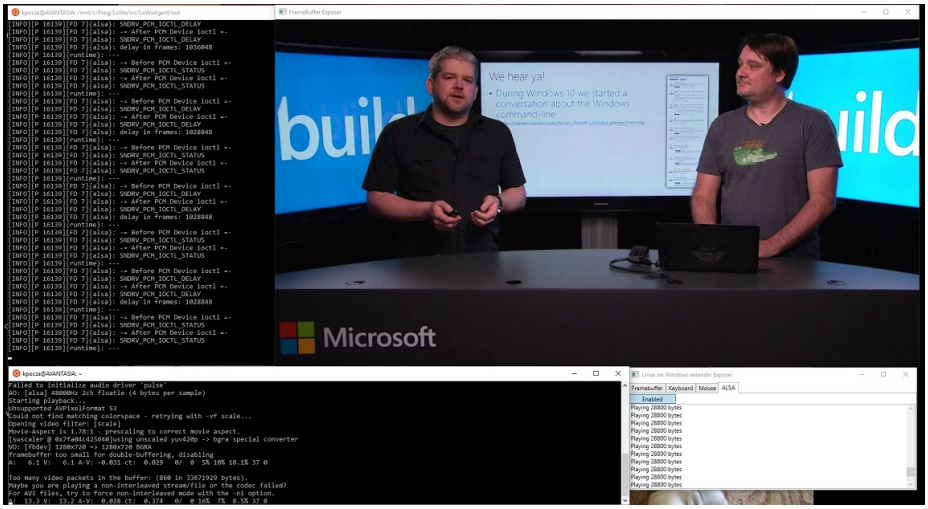
- Grafikus alkalmazások
- X alkalmazások
- ALSA - PCM alapú hang
- Egér (és billentyűzet) támogatás X alkalmazásokhoz
- Mindehhez *nincs szükség 3rd party X* serverre

A LoWe kódja a https://github.com/kpocza/LoWe címen érhető el *MIT licensz* alatt.

5.1. Programok futás közben

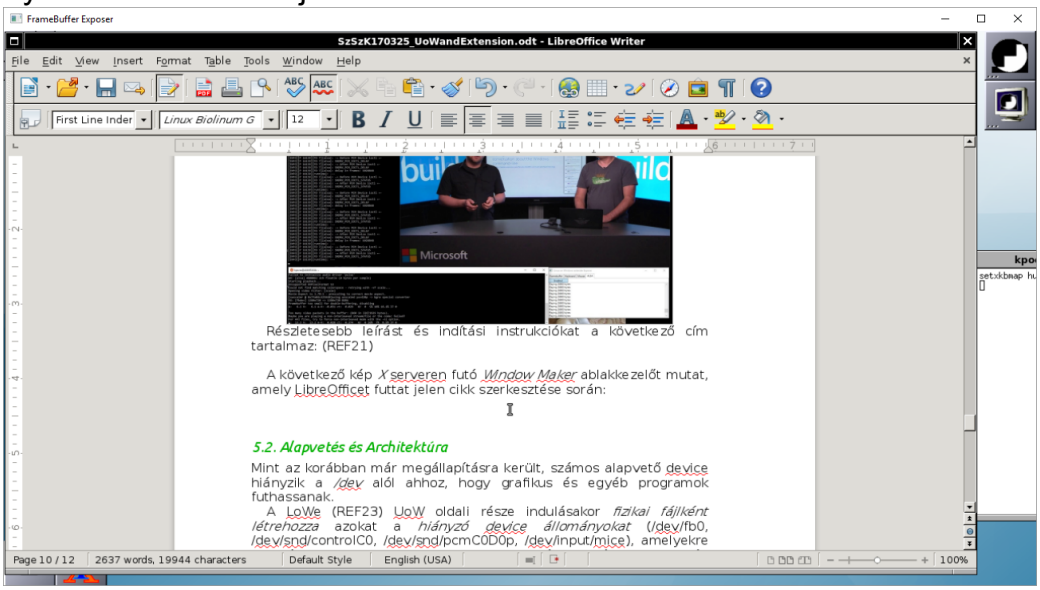
A LoWe jelenlegi verziója az *mplayer*rel valamint *X*-el került tesztelésre.

Az LoWe framebuffer és ALSA – PCM „emulációs” szolgáltatásai révén az mplayer képes videók lejátszására hanggel együtt:



Részletesebb leírást és indítási instrukciókat a következő cím tartalmaz: (22)

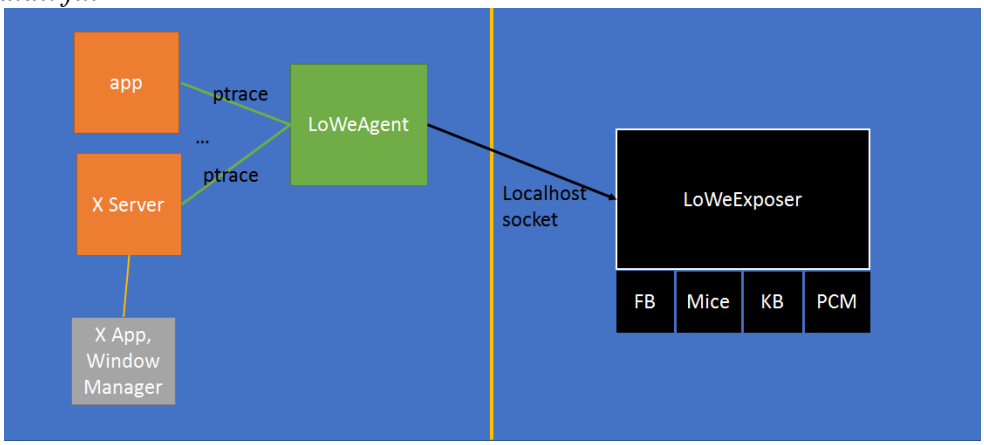
A következő kép *X* serveren futó *Window Maker* ablakkezelőt mutat (23), amely LibreOfficet futtat jelen cikk szerkesztése során:



5.2. Alapvetés és Architektúra

Mint az korábban már megállapításra került, számos alapvető device hiányzik a */dev* alól ahhoz, hogy grafikus és egyéb programok futhassanak.

A LoWe (24) UoW oldali része indulásakor *fizikai fájlként létrehozza* azokat a *hiányzó device állományokat* (/dev/fb0, /dev/snd/controlC0, /dev/snd/pcmC0D0p, /dev/input/mice), amelyekre szükség lehet a felügyelt program futása során. Ezek után *debuggerként csatlakozik* a processhez (X server, mplayer, stb.) a *ptrace* API segítségével, majd rendszerhívásról-rendszerhívásra ugrálva „okos" választ biztosít a program számára. Azaz figyeli az *open hívás*t, és amennyiben olyan /dev kerül megnyitásra (immáron fizikai fájlként), amelyet a LoWe le kell, hogy kezeljen, megjegyzi a fájl handle-jét és a típusát (framebuffer, PCM device, stb.). A *további rendszerhívások* esetében (ioctl, read, write, mmap) pedig *szimulálja* a *megfelelő működést a* device típusától és állapotától függően. Ennek a komponensnek a neve *LoWeAgent* és teljes mértékben *UoW alatt fut*.



A LoWeAgent localhoston nyitott socketen keresztül egy másik, de már *Windows alatt futó LoWeExposer* nevű *WPF* alkalmazás felé továbbítja az kéréseket. *Socket*en tolja át a framebuffer tartalmát, lekérdezi az egér elmozdulását, vagy éppen megszólaltat egy waveform formátumú hangmintát. A LoWeExposer felelős alapvetően a socketen érkezett műveletek elvégzésére és a socketen a válasz visszaadására.

Mindehhez részletes konfigurációra van szükség devicekezelő, alkalmazás valamint X server szinten is.

A *LoWeAgent* egy *C++*-ban készült 64 bites ELF alkalmazás, míg a *LoWeExposer* egy *Windowsos C# program (25)*.

5.3. Működés

Az előzőekben a high-level működés már ismertetésre került, most röviden mélyebb áttekintést adunk a különböző entitások emulációjának részleteibe (24).

A **framebuffer device (/dev/fb0)** segítségével grafikus megjelenítés biztosítható. Az eszköz alapvető *ioctl hívásokat* támogat (grafikus mód lekérdezés, beállítás, stb.) valamint egy bemappelt memória területen keresztüli megjelenítést. A LoWe létrehoz egy akkora *fizikai fájl* a */dev/fb0* útvonalon, amely egy HD-ready, azaz *1280x720x32bpp* kép tárolásához elegendő. Mivel egy fizikai állományról beszélünk ezért az *mmap* a fizikai fájlt fogja bemappelni a grafikus kártya memóriája helyett. Mivel az UoW és a Windows közös fájlrendszeren helyezkedik el, ezért korábban az fb0 állományt olvasta rendszeresen és a tartalmát jelenítette meg a Windows oldal. Teljesítmény okokból a felügyelt process memóriáját a LoWeAgent alkalmazás a *process_vm_readv* másodpercenként 50x olvassa (mmap által visszaadott címről) és localhost socketen keresztül továbbítja a LoWeExposer felé, amely megjeleníti a grafikus képet.

„Hangkeltés” tekintetében az **ALSA (Advanced Linux Sound Architecture)** a PCM (hulllámforma), MIDI, valamint mixer funkciókért felelős. Jelenleg a LoWe néhány alapvető *hangkártya kezelő* funkciót, valamint *PCM funkciókat* (ioctl beállító és lekérdező, lejátszó funkciók) emulál. A LoWeAgent a beállításokat és a hangmintát socketen keresztül továbbítja a LoWeExposer alkalmazásnak, amely elvégzi a szükséges műveleteket.

Az **egeret a /dev/input/mice** eszközön biztosítjuk az X alkalmazások felé. A LoWeExposer grafikus felületén folyamatosan monitorozza az *egér elmozdulás*át valamint a *gombok* állapotát. Az alkalmazás rendszeresen rákérdez az egér állapotára, így a LoWeAgent ezeket a kéréseket socketen a LoWeExposer felé továbbítva szolgálja ki.

A **billentyűzet számára a /dev/input/kb** (nem standard) eszközt hozzuk létre. Az X-et úgy konfiguráljuk, hogy ezt a device-t tekintse a billentyűzetnek, amelyet rendszeresen lekérdez. A lekérdezés a LoWeAgent-en keresztül a

LoWeExposer felé továbbítódik, amely a leütött *billentyűk scan kódját* szolgáltatja válaszként.

6. Irodalomjegyzék

(1) https://arstechnica.com/information-technology/2014/10/microsoft-loves-linux-as-it-makes-azure-bigger-better/, 2014

(2) https://github.com/PowerShell/PowerShell, 2017

(3) https://www.microsoft.com/en-us/sql-server/sql-server-vnext-including-Linux, 2017

(4) https://www.linuxfoundation.org/announcements/microsoft-fortifies-commitment-to-open-source-becomes-linux-foundation-platinum, 2016

(5) https://msdn.microsoft.com/en-us/commandline/wsl/about, 2016

(6) https://www.cygwin.com/, 2017

(7) http://www.mingw.org/, 2017

(8) https://en.wikipedia.org/wiki/Microsoft_POSIX_subsystem, 2017

(9) https://en.wikipedia.org/wiki/Windows_Services_for_UNIX, 2016

(10) https://www.microsoft.com/en-us/research/project/drawbridge/, 2011

(11) https://mspoweruser.com/microsoft-confirms-decision-to-kill-project-astoria/, 2016

(12) https://msdn.microsoft.com/en-us/commandline/wsl/install_guide, 2016

(13) https://blogs.msdn.microsoft.com/wsl/2016/05/23/pico-process-overview/, 2016

(14) https://blogs.msdn.microsoft.com/wsl/2016/04/22/windows-subsystem-for-linux-overview, 2016

(15) https://msdn.microsoft.com/en-us/commandline/wsl/release_notes, 2017

(16) https://blogs.msdn.microsoft.com/wsl/2016/06/08/wsl-system-calls/, 2016

(17) https://blogs.msdn.microsoft.com/wsl/2016/06/15/wsl-file-system-support/, 2016

(18) https://blogs.msdn.microsoft.com/wsl/2016/11/08/225/, 2016

(19) https://blogs.msdn.microsoft.com/wsl/2016/10/19/windows-and-ubuntu-

interoperability/, 2016
(20) http://www.pcworld.com/article/3055403/windows/windows-10s-bash-shell-can-run-graphical-linux-applications-with-this-trick.html, 2016
(21) https://github.com/kpocza/LoWe, 2017
(22) https://github.com/kpocza/LoWe/blob/master/docs/mplayer.md, 2017
(23) https://github.com/kpocza/LoWe/blob/master/docs/x.md, 2017
(24) https://github.com/kpocza/LoWe/blob/master/docs/howitworks.md, 2017
(25) https://github.com/kpocza/LoWe/blob/master/docs/buildrun.md, 2017