UML

UML, czyli User Mode Linux, jest koncepcyjnie bardzo różny od większości wymienionych poprzednio narzędzi do wirtualizacji. Podczas gdy praktycznie wszystkie one wirtualizują maszynę wraz z zainstalowanym na niej systemem operacyjnym, UML wcale nie używa maszyny wirtualnej jako takiej. Jest, ściśle rzecz biorąc, portem (http://en.wikipedia.org/Porting) Linuksa na Linuksa. Innymi słowy, UML pozwala na uruchomienie jądra Linuksa jako jednego z programów w trybie użytkownika (stąd nazwa) naszego oryginalnego (zwanego w terminologii programu „host" – „gospodarz") Linuksa.

Uruchomione „gościnne" („guest") jądro jest praktycznie całkowicie wyizolowanym procesem, nieróżniącym się od pozostałych uruchomionych w Linuksie-gospodarzu – w szczególności można wysyłać do niego sygnały (choćby SIGKILL) czy uruchomić za pośrednictwem narzędzia strace, aby poznać jego wywołania przerwań systemowych. Posiada ono jednak przy tym możliwość tworzenia własnych procesów, montowania systemów plików i ogólnie wykonywania większości operacji właściwych Linuksowi. Cały system (lub systemy) plików „gościnnej" dystrybucji zawarty jest w pojedynczym pliku, a jej dostęp do zasobów sprzętowych jest w pełni konfigurowalny. Dodatkową zaletą jest fakt, że jądro w wersji 2.6.0 naniesioną poprawkę UML'ową – czyli możemy samodzielnie skompilować je „pod" używanie w UML'u bez żadnego dodatkowego oprogramowania czy skomplikowanych operacji.

Łatwo zatem zauważyć, że UML posiada niezliczone zastosowania – poza bezpiecznym debuggowaniem jądra (który to temat poruszy następna prezentacja), możemy bez obaw uruchamiać programy, którym nie ufamy, testować nowe dystrybucje Linuksa czy stawiać wirtualne serwery (zob. np. http://en.wikipedia.org/wiki/Honeypot_(computing)).

Stroną, od której najlepiej zacząć naszą przygodę z UML'em jest strona domowa projektu, jego pomysłodawcy i głównego twórcy (developera) – Jeffa Dike'a (http://user-mode-linux.sourceforge.net). Znajdziemy na niej praktycznie wszystko, czego potrzebujemy do stworzenia wirtualnego Linuksa na naszym komputerze – gotowe, skompilowane wersje jąder do ściągnięcia, gotowe systemy plików do zamontowania jako root'y w „gościnnym" systemie, poprawki (patche) pozwalające na „UML'owe" skompilowanie jądra w starszej niż 2.6.0 wersji (nowsze, jak już wspomniano, domyślnie obsługują UML'a) czy wreszcie – choć niestety niezbyt nowy i niekoniecznie do końca przyjazny użytkownikowi – tutorial.

Zademonstrujmy teraz, jak UML wygląda w praktyce.

Kompilacja jądra, które chcemy używać w UML'u przebiega jak klasyczna kompilacja jądra, jedyna różnica to podawanie przy wszystkich etapach parametru *ARCH=um*, a zatem w katalogu ze źródłem jądra wywołujemy kolejno:
make config ARCH=um
[konfigurujemy]
make linux ARCH=um
Wygodnie jest przed rozpoczęciem konfigurowania przekopiować do katalogu ze źródłami plik *config.release* z podkatalogu *arch/um/*, nadając mu nazwę *.config*:
cp arch/um/config.release .config
W ten sposób uzyskujemy rozsądne ustawienia domyślne, chociaż niestety wywołanie *make config* dalej jest konieczne.

Po skompilowaniu jądra potrzebujemy jeszcze systemu plików – wygodnie jest ściągnąć gotowy system plików ze strony domowej projektu – dobrą opcją jest na przykład Fedora (http://uml.nagafix.co.uk/FedoraCore5/FedoraCore5-x86-root_fs.bz2).
Najbardziej podstawowe uruchomienie UML'a to wywołanie w katalogu, w którym kompilowaliśmy źródła polecenia *./linux* – wówczas jako główna partycja zamontowany będzie plik *root_fs* z bieżącego katalogu (jeśli takowy nie istnieje, dostaniemy błąd przy uruchamianiu). Jeśli z jakichś przyczyn chcemy, żeby plik z partycją startową nazywał się inaczej – wywołujemy polecenie:
./linux ubda=nazwa_pliku
Dla UML'a – jak już powiedziano – każdy system plików jest oddzielnym plikiem, kolejne systemy plików (intuicyjnie – „dyski") mają nazwy *ubda*, *ubdb* itd. (zmienia się ostatnia litera) – w szczególności partycja startowa to *ubda*. Ciekawostką jest fakt, że jako „dysk" możemy podpiąć nie system plików, ale np. archiwum *.tar*. Wówczas, będąc zalogowanym w systemie „gościnnym", możemy wywołać przykładowo polecenie:
tar –xf /dev/ubdb .
które spowoduje wypakowanie pliku *.tar* podpiętego jako *ubdb* do bieżącego katalogu.

Po udanym uruchomieniu „gościa" stajemy przed właściwym wybranej dystrybucji (czyli: systemowi plików) ekranem logowania, po zalogowaniu (gotowe systemy plików ze strony mają przede wszystkim konto administratora – root, kolejne możemy łatwo utworzyć) zaś znajdujemy się w mniej lub bardziej funkcjonalnym linuksie, który w szczególności nie ma dostępu do „gospodarza" – czy też w ogóle „świadomości" jego istnienia.

Z poziomu „gospodarza" możemy jednak modyfikować wszelkie ustawienia działającego już „gościa", używając do tego narzędzia *uml_mconsole* (do ściągnięcia również ze strony projektu). I tak na przykład możemy podpiąć nowy system plików poleceniem:
uml_mconsole id_goscia config ubdc=nasz_plik
Parametr *id_goscia* pozwala na zidentyfikowanie konkretnej instancji UML'a (bo możemy uruchomić kilka takowych jednocześnie) – możemy przydzielić go dodając do parametrów uruchomienia parametr *umid=id_goscia*, czyli:
./linux ubda=system_plikow umid=linuksik
i odwoływać się potem do tej instancji przez np.
uml_mconsole linuksik config ubdc=paczka.tar
Jeśli nie nadaliśmy umid'u naszej instancji, UML sam jej takowy nadaje w sposób losowy; jest on wypisywany przez jądro w trakcie startu i możemy go poznać za pomocą polecenia (wywołanego u „gościa"):
gmesg | grep mconsole
wynikiem powinna być linijka podobna do:
mconsole (version 2) initialised on /home/ja/.uml/ID_GOSCIA/mconsole

Kolejną sprawą są terminale lub konsole wirtualne UML'a. Domyślnie otrzymujemy na ekranie (w trybie graficznym) jeden terminal do logowania, więcej możemy stworzyć edytując plik */etc/inittab*. Każdy nowy zadeklarowany wirtualny terminal będzie domyślnie otwierał się w nowym oknie. Terminal możemy łatwo przekierować – czy to za pomocą *uml_mconsole*, czy przez parametry uruchomieniowe (jako zasada – te sposoby się dublują). Możemy choćby napisać (na „gospodarzu"):
uml_mconsole linuksik config con1=port:9042
co, jeśli w */etc/inittab* zadeklarowaliśmy pierwszą konsolę (konsole w UML noszą nazwy *con*x, gdzie x – numer, poczynając od *con0* – konsola na której znajdowały się komunikaty

uruchomieniowe jądra), powinno ją przekierować na port o numerze 9042. Możemy wówczas na linuksie-gospodarzu napisać:
telnet localhost 9042
i w ten sposób „zdalnie” zalogować się do naszej instancji UML’a.

Przy okazji: jeśli w *uml_mconsole* nie wywołujemy przypisania, możemy poznać stan zmiennej, czyli:
uml_mconsole linuksik config con1
odpowiedzią powinno być:
OK port:9000
gdzie OK oznacza fakt otrzymania odpowiedzi od urządzenia, a ciąg dalszy – jego stan.

Oprócz tego, można także przekierować konsolę na jedną z konsoli „gospodarza”. Robi się to bardzo podobnie – usuwamy z */etc/inittab* „gospodarza” deklarację jednej z konsol, zmuszamy proces *init* do zaakceptowania zmian (*kill –HUP 1*), po czym podpinamy (np. w parametrach uruchamiania) ją do jednej z konsol wirtualnych (znów musi ona być zadeklarowana w */etc/inittab* „gościa”:
./linux ubda=system_plikow umid=linuksik con1=tty:/dev/tty1
Uwaga: konsola, którą chcemy oddać UML’owi (tutaj: /dev/tty1) musi mieć ustawione uprawnienia do zapisu dla wszystkich (*chmod 666 /dev/tty1*).
Składnia przypisania jest podobna jak w przypadku portu: najpierw podajemy typ urządzenia, potem jego nazwę (dokładniejszy opis na http://user-mode-linux.sourceforge.net/UserModeLinux-HOWTO-5.html#ss5.1).

Przydatne jest również uzyskanie dostępu do internetu z naszej instancji UML’a. Robimy to dość prosto, mostkując połączenie „gospodarza” – najpierw u tegoż „gospodarza” konfigurujemy kartę sieciową „gościa”, co wygląda praktycznie tak samo jak konfigurowanie konsol:
uml_mconsole linuksik config eth0=tuntap,,,adres_ip
TUN/TAP to sterownik tworzący łącze nienazwane (pipe) między „gospodarzem” a „gościem”, przecinki pomijają parametry którym pozostawiamy ustawienia domyślne, natomiast w miejsce *adres_ip* wpisujemy adres lokalny, pod którym od tej pory nasza instancja UML’a będzie widziała „gospodarza”. W ten sposób „gość” otrzymuje „kartę siecową”, ale nie jest ona jeszcze skonfigurowana. W tym celu u „gościa” wywołujemy polecenie:
ifconfig eth0 inny_adres_ip up
Efekt tego to wywołanie serii poleceń, które powinny spowodować, że „gospodarz” będzie widział „gościa” pod adresem *inny_adres_ip* w sieci lokalnej oraz mostkował połączenia wychodzące z niego lub przychodzące do niego przez własne połączenie internetowe – co możemy sprawdzić choćby za pomocą polecenia *ping*.

UML posiada również wiele innych funkcji (jak np. partycjonowanie dysków (systemów plików), które jednak, z braku czasu, nie zostaną tu omówione. Są one jednak opisane zarówno na stronie projektu (jeszcze raz: http://user-mode-linux.sourceforge.net/), a także w napisanej przez Jeffa Dike’a książce „User Mode Linux”.

Bibliografia:
Jeff Dike, „User Mode Linux”
http://user-mode-linux.sourceforge.net
http://en.wikipedia.org/wiki/User-mode_Linux