

# NOWOŚCI W JĄDRZE

## STEROWNIKI

W ciągu ostatnich dni pojawiło się parę nowych sterowników. Pavel Machek przysłał sterownik obsługujący ekran dotykowy w działającym pod kontrolą systemu Linux palmtopie Sharp Zaurus SL-5500. Sterownik jest zasadniczo modyfikacją kodu Sharpa, przeznaczoną do części jądra odpowiadającej za obsługę architektury ARM. Niestety, wygląda na to, że parę osób, w tym Russel King, autor oryginalnego sterownika, pracowało równolegle nad podobnym zmodyfikowaniem tego kodu tak, aby pasował do infrastruktury sterowników jądra. Większość kwestii dotyczących położenia kodu i tego, jak ogólny powinien być kod, wydaje się już być rozwiązana, a prace nad programem postępują.

Brian Schau wysłał ostatnio swój pierwszy sterownik dla Linuksa, nazwany Wireless Security Lock (dosłownie „bezprowadowa blokada bezpieczeństwa”), czyli w skrócie WSL lub "Weasel" (ang. łasica). Celem tego nowego urządzenia i jego sterownika jest umożliwienie rozpoznawania, czy komputer nie został przeniesiony zbyt daleko od określonego fizycznego miejsca. W takim wypadku podejmowane są środki bezpieczeństwa, takie jak blokowanie ekranu. Wśród przeszkód na drodze do włączenia tej poprawki do jądra znajduje się między innymi fakt, że podobne funkcje oferuje inna poprawka korzystająca z telefonu z technologią Bluetooth. Mimo wszystko reakcje na propozycję Briana były zachęcające.

Stelian Pop wysłał sterownik touchpada USB, który może być wykorzystany w niektórych nowych notebookach Apple Powerbook. Sterownik bazuje na kodzie autorstwa Johanna Berga, który rozgryzł protokół komunikacyjny touchpada. Alex Harper dostarczył informacji o wewnętrznym funkcjonowaniu jego czujników, a Frank Arnold zaproponował różne poprawki kodu. Stelian twierdzi, że sterownik został gruntownie przetestowany i jest gotowy do włączenia do jądra.

## GIT

Linus Torvalds skopiował – za pośrednictwem archiwum CVS – istniejące w archiwum Bit-Keepera drzewo jądra do systemu git. Przepuszczenie całości przez CVS spowodowało, że część historii zmian została usunięta; przetrwał jedynie zbiór pojedynczych poprawek. Ostateczny rezultat zawiera jednak każdą poprawkę w łatwym do zarządzania i użytecznym archiwum, co świadczy o zdolności gita do przechowywania dużych repozytoriów.

Linus przekazał tymczasem opiekę nad gitem Juniowi C. Hamano, który zawsze aktywnie uczestniczył w tym projekcie. Okaże się dopiero, czy będzie to podobnie częściowym działaniem, jak przekazanie gałęzi 2.6 jądra w ręce Andrew Mortona. Junio bardzo poważnie traktuje opiekę nad systemem i zamierza wydać w niedalekiej przyszłości wersję 1.0.

Linus zaczął aktywniej uczestniczyć w rozwoju jądra, ale wydaje się być bardzo zadowolony z obecnego kierunku rozwoju gita. Jak zauważył pod koniec lipca: „Stało się to większe i bardziej profesjonalne niż pierwotnie przewidywałem”. W ciągu paru miesięcy okazało się, że to co rozpoczęło swój żywot jako prowizorka, krótkoterminowe rozwiązanie większego problemu, stało się bardzo znaczącym i posiadającym ogromne możliwości systemem.

Witamy więc bezserwerową kontrolę wersji!

## PLIKI I PARTYCJE WYMIANY

Przez lata sposób obsługi plików i partycji wymiany w Linuksie ulegał poważnym zmianom, a od czasu do czasu ktoś zadawał sakramentalne pytanie o optymalny rozmiar obszaru wymiany w odniesieniu do rozmiaru pamięci RAM albo o to, czy partycja wymiany jest lepszym rozwiązaniem niż plik wymiany. Niedawno postawił to pytanie Mike Richards, a Andrew Morton odpowiedział, że w przypadku Kernela 2.6 niezawodność jest taka sama niezależnie od tego, czy użyto pliku, czy partycji.

Według Andrew Mortona, wydajność powinna być również identyczna, jeśli tylko plik wymiany nie był silnie pofragmentowany w momencie jego zakładania. W tym przypadku, używając partycji wymiany, można osiągnąć wyższą wydajność. Dobrą wieścią jest, że z czasem pliki wymiany nie stają się bardziej pofragmentowane - jeśli początkowo nie były pofragmentowane, to takie zostają.

Andrew stwierdził również, że w przypadku jądra 2.4 pliki wymiany są „gorsze” niż partycje, gdyż wyrzucenie danych do pliku wymiany (swapout) wymaga przydzielenia pamięci przez główny alokator stron. Jest to wydatek, którego jądro 2.6 unika, i który jest najwyraźniej przyczyną tej różnicy.

Zatem werdykt jest następujący: w chwili obecnej w jądrach 2.6 pliki i partycje wymiany są praktycznie równoważne, w jądrach 2.4 partycje są wydajniejsze.

## DEVFS

W niekończącej się sadze o DevFS, pojawił się znowu – po bardzo nagłym zniknięciu w październiku 2002 r. – jego pierwotny autor, Richard Gooch. W swoim krótkim liście na listę dyskusyjną twórców jądra Linuksa Richard bronił jakości kodu DevFS, ale później zniknął znowu, nie odpowiadając na dalsze pytania. Z pewnością kod DevFS musi mieć coś w sobie, jeśli przetrwał bez opiekuna od roku 2002, będąc szczerze zniechęconym przez wielu liczących się programistów jądra. Jak zwrócił uwagę Jan Engelhardt, FreeBSD włączyło nawet DevFS do swojego systemu. Greg Kroah-Hartman natomiast natknął na duży opór, próbując usunąć DevFS z jądra.

## SECURITYFS

Greg Kroah-Hartman stworzył SecurityFS, związany z bezpieczeństwem system plików ogólnego przeznaczenia, który może być używany przez SELinux albo inne planowane linuksowe moduły związane z bezpieczeństwem. Jego zaletą jest to, że każdy podprojekt jądra związany z bezpieczeństwem może używać SecurityFS bez konieczności konstruowania własnego systemu plików. Wadą jest natomiast fakt, że w istniejących projektach, takich jak SELinux, będzie pewnie trzeba wprowadzić wiele poprawek, aby zamienić ich własne rozwiązania na SecurityFS. Podczas gdy programiści SELinux uważają, że koszt migracji jest trochę za duży, inni optymistycznie twierdzą, że końcowy rezultat będzie wielkim uproszczeniem i zaowocuje zmniejszeniem rozmiaru kodu jądra. Obecnie SecurityFS jest jeszcze bardzo świeży i może rozwinąć się w każdym kierunku; może również zniknąć.

## CHECKFS

Firma LinSysSoft Technologies, próbując umożliwić obsługę przyrostowych kopii bezpieczeństwa na poziomie bloków systemu plików, odgałęziła kod systemu plików ext3, tworząc CheckFS. Podczas gdy odgałęzienie to spotkało się z dezaprobatą wielu twórców jądra, podstawowy cel wydaje się wart zachodu. Niektórzy programiści zachęcali LinSysSoft do nadsyłania poprawek, które umożliwią bezpośrednią obsługę tych funkcji w ext3. Pracownicy LinSysSoft, których reprezentują Milind Dumbare i Amit S. Kale, wydają się chętni do przyjęcia takiego rozwiązania.

CheckFS powstał we wczesnej fazie rozwoju jądra 2.6 jako wewnętrzny projekt LinSysSoft, leżał odłogiem przez parę miesięcy i odżył dopiero niedawno. Początkowo zmiana nazwy zapowiadała ułatwienie testowania przez umożliwienie bezpośrednich porównań CheckFS z ext3 w tym samym jądrze. Obecnie, kiedy wykonano już wiele pracy, korzyści z projektu nie przykuwają już tak bardzo uwagi programistów z LinSysSoft, zwłaszcza biorąc pod uwagę podobieństwa ich projektu do ext3.

## USUWANIE PRZESTARZAŁYCH STEROWNIKÓW DŹWIĘKU

Adrian Bunk zaprezentował poprawkę usuwającą z jądra przestarzałe sterowniki dźwięku OSS, mające działające odpowiedniki ALSA. To, które sterowniki są objęte tą poprawką, było przedmiotem poważnej debaty, po części dlatego, że niektóre karty dźwiękowe są trudne do zdobycia, niełatwo więc określić, czy konkretne sterowniki lub ich odpowiedniki działają.

Niezależnie od tego, czy ta poprawka lub jej wariant zostaną zaakceptowane, próby uproszczenia obsługi dźwięku trwają. Od pewnego czasu Adrian postawił sobie za cel wyczyszczenie „brzydkich” części jądra. Bez względu na to, czy chodzi o poprawianie odstępów, wyrzucanie sterowników, czy poprawianie informacji o autorach, Adrian regularnie wysyła poprawki, które wielu twórców jądra uznałoby – niezależnie od ich użyteczności – za zbyt mało istotne.

Nie wszystkie poprawki Adriana są jednak mile widziane. Czasami przesadza i stara się usunąć lub zmienić coś, co jest aktywnie używane. Czasem kod, który planuje zmienić, przechodzi właśnie gruntowne zmiany i nie powinien być jeszcze porządkowany. W innych przypadkach jego zmiany przybierają kierunek, z którym część programistów się nie zgadza. Przez lata nad jego poprawkami odbywały się burzliwe dyskusje. Adrian – aby

uniknąć robienia sobie wrogów – stwierdza od czasu do czasu, że nie chce już dłużej wprowadzać tych mało wyrafinowanych, ale bardzo użytecznych poprawek.

Oto jednak znowu ciężko pracuje, współpracując z opiekunami sterowników OSS i ich poprzednimi opiekunami, obecnymi użytkownikami i ludźmi posiadającymi w domach stare karty dźwiękowe, starając się zbadać, które sterowniki OSS działają, a które są już zastąpione przez sterowniki ALSA.

Jest to przykład jednego z najlepszych aspektów modelu rozwoju Open Source i przełomu, którego Linus Torvalds dokonał w 1991 r., kiedy to zaczął szukać na całym świecie pomocy przy rozwijaniu kodu. Wkład Adriana nie jest kluczowy dla żadnej części jądra i niewiele (jeżeli w ogóle jakiegoś) z jego poprawek jest absolutnie niezbędne. Jeśli wysłałby tego rodzaju poprawki do jakiegokolwiek projektu GNU we wczesnych latach dziewięćdziesiątych, zostałyby one po prostu zignorowane, tak jak ignorowane były praktycznie wszystkie raporty błędów wysyłane do projektów GNU przez ludzi nie związanych z projektem. Doprowadziło to H. J. Lu do stworzenia gałęzi biblioteki libc całkowicie niezależnej od GNU – zgodnie z filozofią „wydawać wcześniej i często, każdy może pomóc, albo zrobimy to za ciebie”.

Obecnie istnieje cała rzesza programistów, których cieszy prokurowanie małych poprawek – albo (tak jak czyni to Adrian) globalnych dla całego jądra, albo w konkretnej dziedzinie. Czasami ludzie przechodzą z jednej dziedziny do innej, jak Rolf Eike Beer, który w ciągu ostatnich lat wniósł do jądra ponad setkę poprawek. Zaczął od kodu SCSI, gdzie zaktualizował dokumentację, poprawił literówki i czasem zaprogramował coś pokazniejszego, później zaczął pracować nad kodem PCI, gdzie wysłał wiele małych poprawek i reorganizacji kodu, stając się w końcu jednym z bardziej płodnych programistów kodu PCI.

Nic z tego nie było możliwe 12 czy 13 lat temu. Nie było to nawet brane pod uwagę. Na tym właśnie polega wielkość wkładu Linusa w rozwój oprogramowania: zasada, że każdy może wnieść coś pożytecznego w dowolnie wybrany sposób. Ideę tę widać także w sposobie prowadzenia przez Linusa projektu git, choć prawdopodobnie zapomniano już o niej w wielu projektach programistycznych na świecie. Może to być jednym z powodów, dla których żadnemu wolnemu rozproszonemu systemowi kontroli wersji nie udało się w ostatecznym rozrachunku zrealizować potrzeb programistów jądra – aż do momentu, kiedy Linus wziął sprawę w swoje ręce.