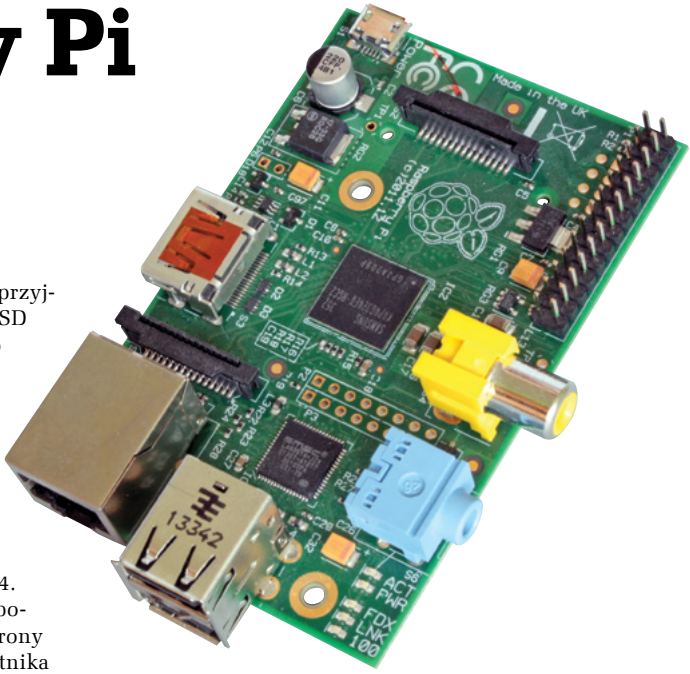
Zastanawialiście się kiedyś, jak właściwie startuje Raspberry? Proces ten nie jest aż tak skomplikowany. Zrozumienie go pomoże Wam diagnozować różne problemy, jakie czasami pojawiają się podczas uruchamiania RPi.

Raspberry Pi
tajemnice startu



Tajemnica 1: uszkodzona karta

Zanim zagłębimy się w szczegóły startu RPi, przyjrzyjmy się sposobowi partycjonowania karty SD z zainstalowanym Raspbianem. Używam jego ostatniej wersji, z 24 grudnia 2014 r. Świeżo wypaloną kartę (stworzoną z użyciem programu *Win32 Disk Imager* i obrazem *2014-12-24-wheezy-raspbian.img*) podzielono na dwie partycje:

- **startową** (ang. *boot*), o rozmiarze ok. 56 MB i systemie plików FAT;
- **główną** (ang. *root*, „/"), o rozmiarze ok. 3,2 GB i linuksowym systemie plików ext4.

Powyższy podział jest ściśle związany ze sposobem, w jaki startuje Raspberry. Z drugiej strony powoduje, że po włożeniu takiej karty do czytnika komputera zarządzanego przez Windows, system zgłosi nam jej uszkodzenie. Co więcej, zaproponuje, że może ją naprawić. Nie róbcie tego. Karta wcale nie jest uszkodzona. Windows potrafi zamontować mniejszą z partycji (w końcu to FAT, do niedawna natywny dla Microsoftu), ale nie rozumie drugiej – dlatego sygnalizuje problemy. Taki podział można zobaczyć w Microsoftowej aplikacji do zarządzania dyskami. Dla Windows 8.1 uruchomicie ją, klikając prawym klawiszem myszy na „Start" i wybierając pozycję „Zarządzanie dyskami" (**1**).

Program „Zarządzanie dyskami" dodatkowo pokaże, że na karcie znajduje się jeszcze 4,2 GB nieprzydzielonego miejsca. Jest to różnica między rozmiarem karty (użyłem 8 GB) a wielkością obrazu Raspbiana (ok. 3,2 GB). Po zalogowaniu do Raspberry możecie odzyskać to miejsce, rozszerzając partycję główną za pomocą programu *raspi-config* (opcja *Expand Filesystem*, **ilustracja 4**). Szczegóły partycjonowania łatwiej zobaczyć, używając aplikacji *GParted* pod Raspbianem (**2**). *GParted* pokazuje nam znacznie więcej szczegółów. Rozumie też system ext4.

Ilustracja 3 prezentuje wygląd karty po rozszerzeniu partycji do rozmiaru karty SD. *GParted* możecie zainstalować poleceniem „sudo apt-get install gparted". Musi być uruchomiony z konta *root*.

Słowniczek

Poniżej znajdziecie wytłumaczenie kilku kluczowych pojęć, które pojawiają się dalej w tekście (w kolejności występowania).

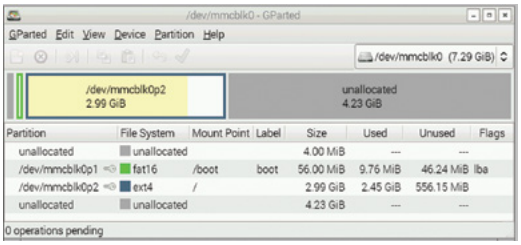
Partycja – wydzielony, logiczny obszar dysku. Jeden dysk może zawierać wiele partycji – podstawowych, rozszerzonych czy logicznych. Każdą z nich można sformatować z innym systemem plików. Windows montuje je zazwyczaj jako kolejne dyski.

FAT (ang. *File Allocation Table*) – rodzaj systemu plików, który powstał już pod koniec lat 70. ub. wieku. Opiera się na tablicy alokacji plików – specjalnej strukturze, która składuje w sobie referencje do plików i zajmowanych przez nie jednostek alokacji (klastrów). Możecie spotkać wersje **FAT12** (adresuje dyski do 16 MB), **FAT16** (maksymalnie do 4 GB), **FAT32** (2 TB) i najnowszy **exFAT/FAT64**. Liczby 12, 16, 32 oznaczają ilość bitów używanych do adresowania klastrów. Im więcej bitów, tym więcej klastrów można zaadresować (czyli większe dyski można obsłużyć). Powstał też VirtualFAT (VFAT), który obsługuje długie nazwy i znaki narodowe. FAT wykorzystywał np. MS-DOS (jeżeli ktoś z Was go jeszcze pamięta) i starsze wersje systemu Microsoft Windows (ostatnio NTFS). Obecnie (głównie FAT32) jest nadal stosowany na kartach flash, pendrivach USB itp.

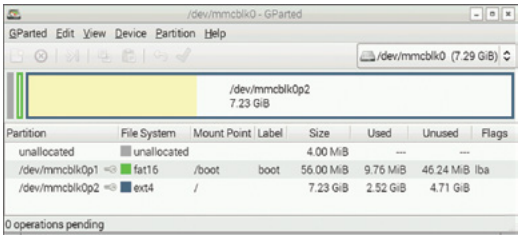
ext4 – najpopularniejszy system plików dla Linuksa. Miał swoją premierę

Skoro już Windowsowi udało się odczytać mniejszą z partycji, warto na nią spojrzeć. Zawiera wiele ciekawych plików. Większością z nich zajmiemy się później, w tym momencie przyjrzyjmy się tylko trzem z nich:

1. Partycje karty SD z Raspbianem, widok pod Windows



2. Partycje karty SD z Raspbianem pod Linuks (GParted)



3. Partycje karty SD z Raspbianem pod Linuks – po rozszerzeniu partycji głównej programem raspi-config (GParted)

- config.txt
- cmdline.txt
- kernel.img

Plik **„config.txt”** to jeden z plików konfiguracyjnych czytany jeszcze przed uruchomieniem CPU (szczegóły poniżej). Zawiera wiele przydatnych ustawień – jak overscan, tryby video itp. Jeżeli użyjecie *raspi-config* do zmiany podziału pamięci RAM między CPU i GPU („Advanced Options>Memory Split”), pojawi się tam wpis, np. „gpu_mem=64” (możliwe wartości to 16/32/64/128/256; zobaczcie „sudo cat /boot/config.txt | grep gpu”). Podział pamięci można sprawdzić za pomocą (nieudokumentowanego) polecenia (parametr „arm” dla CPU i „gpu” dla GPU; [6]):

```
$ sudo vcgencmd get_mem gpu
gpu=64
$ sudo vcgencmd get_mem arm
arm=448
```

Dla konfiguracji wykorzystujących interfejs graficzny, lepiej ustawić „gpu_mem=256”.

Drugi z plików – **„cmdline.txt”** – zawiera opcje dla jądra linuksowego. W jednym z poprzednich tekstów prezentowałem „ip=”. Opcja wymuszała na interfejsie sieciowym przyjęcie podanego adresu. Teraz jednak zwróćcie uwagę na parametry „root=/dev/mmcblk0p2” oraz „rootfstype=ext4”. Nazwa „mmcblk0p2” składa się z: „mmc”, co jest skrótem od „MultiMediaCard”, „blk0”, które pochodzi od pierwszego znalezionego

4. Program raspi-config, wywołanie: „sudo raspi-config”

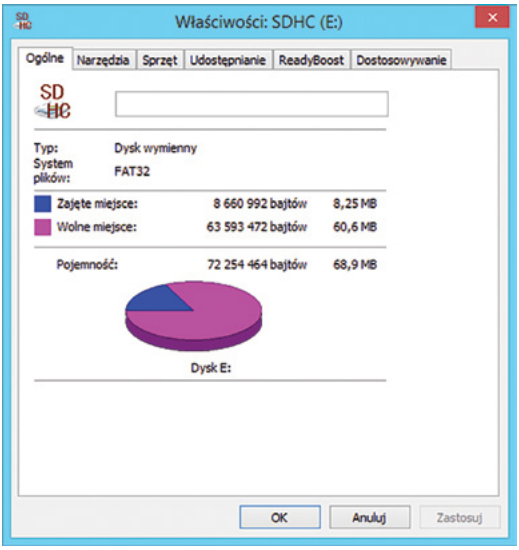
w 2008 r. Obsługuje dyski do 1 eksbibajta (EiB). Zawiera wiele funkcji niedostępnych dla FAT/NTFS, jednak domyślnie nie jest wspierany przez Windows.

root – linuksowe konto uprzywilejowanego użytkownika, który ma pełną kontrolę nad systemem. Zazwyczaj – nawet do administracji systemem – wystarczy korzystać z „sudo”.

Montowanie – proces polegający na przyporządkowaniu dysku lub partycji konkretnego miejsca w systemie plików. Zauważcie, że w przypadku Windows jest to litera dysku („C:”, „E:”). Zasoby w Linuksie montowane są tak, że stają się częścią hierarchii katalogów. Użytkownik często nawet nie wie, że zaczął używać osobnej partycji czy dysku.

Overscan – wobec różnic między możliwościami wyświetlania przez różne odbiorniki (monitory, telewizory), zewnętrzne marginesy obrazu były obcinane, żeby uniknąć zniekształceń. Mogło to dotyczyć nawet 3% powierzchni. To działanie niepożądane w erze telewizji cyfrowej, ale nadal często automatycznie wykonywane przez odbiorniki. Może to pogorszyć jakość obrazu – scena jest rozciągana do wielkości wyświetlacza, żeby przykryć obcięty obszar.

Jądro Linuksa (ang. *kernel*) – centralna część systemu operacyjnego Linuks odpowiedzialna za kontrolowanie sprzętu (za pomocą sterowników).



5. Właściwości FAT-owej partycji pod Windows. Zwróćcie uwagę, że Windows pokazuje FAT32, a Raspbian FAT16

urządzenia blokowego i „p2" – drugiej partycji („p1" to ta mniejsza, którą widzimy pod Windows, zobacz [5]). Porównajcie wartość „root=" z ilustracjami 2 i 3. Od razu zauważycie, że „dev/mmcblk0p2" (system plików ext4) odnosi się do partycji głównej i po starcie Raspbiana zostanie podmontowany jako katalog główny „/". Mniejszą partycję FAT („/dev/mmcblk0p1") znajdziecie za to w katalogu „/boot". Możecie to wszystko wyświetlić na Raspberry poleceniem „lsblk":

```
$ lsblk
NAME        MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
mmcblk0     179:0    0   7.3G  0 disk
├─mmcblk0p1 179:1    0    56M  0 part /boot
└─mmcblk0p2 179:2    0   7.2G  0 part /
```

Nawiasem mówiąc, taka właściwość może się czasami okazać bardzo przydatna. Zdarza mi się używać partycję FAT do przenoszenia plików na Raspberry. Kopiuję je pod Windowsem, a po uruchomieniu Raspberry pliki mam w katalogu „/boot". Nie ma na niej zbyt wiele miejsca – jak pokazują właściwości partycji (**5**), jedynie 60 MB. Ale to zazwyczaj w zupełności wystarcza. Oczywiście zawsze możecie użyć polecenia „wget".

Ostatni ze wspomnianych plików – **„kernel.img"** – to jądro (ang. *kernel*) Linuksa. Zauważcie, jakie jest małe – niewiele ponad 3 MB. Tyle wystarczy, żeby wystartować system i najważniejsze komponenty (dyski, zarządzanie mocą). Reszta modułów znajduje się w katalogu głównym, który kernel montuje zgodnie ze wskazaniami parametru „root=" z pliku „cmdline.txt".

Tajemnica 2: start sprzętu

Jak wspomniałem powyżej, Raspberry Pi napędza SoC Broadcoma, zwany BCM2835. W przypadku Pi składa się on (m.in.) z: procesora głównego ARM1176JZF-S (CPU), jednostki wspomagającej grafikę VideoCore IV (GPU), pamięci nieulotnej ROM oraz przyklejonej nad nimi (PoP) pamięci operacyjnej RAM (konkretnie SDRAM, 512 MB dla wersji B). Start tych układów przebiega w następujących etapach (na podstawie [1][2][3]):

1. Zaraz po zasileniu RPi uruchamiany jest GPU; na ten moment CPU i RAM pozostają jeszcze nieaktywne (w stanie „reset").
2. Wyspecjalizowany, dodatkowy procesor wczytuje z ROM i wykonuje bootloadera pierwszego etapu (ang. *1st stage bootloader*). Kod ten jest wpisywany do ROMu w czasie produkcji układu.
3. Bootloader pierwszego etapu montuje partycję startową FAT karty SD (tę mniejszą), wczytuje zawartość pliku „bootcode.bin" do pamięci podręcznej L2 procesora GPU (ang. *L2 cache*) i wykonuje go (CPU i RAM nadal w stanie resetu).
4. Kod „bootcode.bin" to tzw. bootloader drugiego etapu (ang. *2nd stage bootloader*). Uruchamia pamięć RAM i wczytuje do GPU z karty SD bootloader trzeciego etapu (ang. *3rd stage bootloader*), zapisany w pliku „start.elf". Kod „start.elf" jest jednocześnie systemem operacyjnym GPU (ang. *firmware*), który pozostanie w jego pamięci i pomaga Linuksowi dostawać się do jego zasobów. GPU jest uruchamiane.
5. GPU wczytuje plik konfiguracji RPi „config.txt", zawierający m.in. ustawienia częstotliwości poszczególnych elementów SoC (np. CPU, GPU)

Oryginalnie stworzone przez Linusa Torvaldsa światło dzienne ujrzało w 1991 r. Szybko wsparte przez narzędzia GNU (Richard Stallman), obrosło w liczne aplikacje i powłoki graficzne, powoli wyrosło w dystrybucje takie jak Ubuntu, Fedora czy najpopularniejszy na Raspberry Pi – Raspbian (wywodzący się z Debiana).

SoC (ang. *System On a Chip*) – SoC to dość szeroki termin. Najczęściej odnosi się do kategorii superukładów elektronicznych, które zawierają w sobie dostatecznie dużo komponentów, żeby mogły działać jako autonomiczne „komputery". W przypadku typowej płyty komputerowej, elementy takie jak główny procesor (ang. *Central Processing Unit* – CPU), pamięć operacyjna czy karta graficzna są rozproszone na różne układy (często wymienialne). SoC integruje je w pojedynczy układ (ang. *chip*). SoC charakteryzują się stosunkowo dużą mocą i niewielkim zapotrzebowaniem na prąd. Ich domeną są urządzenia przenośne zasilane bateryjnie. Raspberry Pi napędza SoC Broadcoma BCM2835. Składa się na niego procesor główny typu ARM1176JZF-S (CPU) i procesor graficzny (GPU) Videocore IV. Nad nimi przylutowano pamięć SDRAM – 256 dla wersji A/A+ lub 512 MB dla B/B+ (technologia PoP).

Tabela 1. Pliki na karcie SD Raspberry Pi	
Plik	**Funkcja**
bootcode.bin	Bootloder 2-go etapu
start.elf	Bootloader 3-go etapu; firmware GPU; ładuje plik „config.txt" i jądro linuksowe (z parametrami zapisanymi w „cmdline.txt")
fixup.dat	Dzieli i inicjuje SDRAM między CPU i GPU
kernel.img	Jądro Linuksa; wczytywane do pamięci przez „start.elf"; wykonywane przez CPU
fixup_cd.dat, start_cd.elf	Obcięte wersje plików dla GPUmem=16
fixup_x.dat, start_x.elf	Wersje testowe – np. dodatkowe kodeki; obsługa kamery
cmdline.txt	Parametry startu jądra linuksowego
config.txt	Plik konfiguracyjny czytany przez GPU. Zawiera ustawienia m.in. trybów video, overscanu, taktowania procesorów i podziału pamięci między GPU i CPU

oraz podział pamięci RAM między CPU i GPU (plik „fixup.dat").

6. GPU wczytuje plik konfiguracyjny „cmdline.txt", zawierający parametry do startu jądra Linuksowego.
7. GPU wczytuje do RAM „kernel.img" obraz jądra Linuksa.
8. GPU zmienia stan CPU na aktywny.
9. CPU rozpoczyna wykonywanie kodu jądra linuksowego.

Widzimy więc, że rola GPU nie ogranicza się do wspomagania grafiki. Wykonuje on większość pracy przy starcie systemu. Oprócz wymienionych powyżej zidentyfikujmy inne pliki znajdujące się na karcie SD RPi, a związane ze startem sprzętu:

- „fixup.dat": konfiguruje podział SDRAM między GPU i CPU;
- „fixup_cd.dat" i „start_cd.elf": wersje plików używane dla pamięci GPU zmniejszonej do 16 MB (ustawienie w „config.txt");
- „fixup_x.dat", „start_x.elf": wersje eksperymentalne, często niestabilne, mogą też włączać nowe funkcje;
- „kernel_emergency.img" (w starszych Raspbianach): wersja „awaryjna" jądra, może być używana w przypadku awarii systemu plików.

Tajemnica 3: przebłyski

Raspberry Pi B ma na swojej płytce pięć diód: PWR, ACT i trzy dodatkowe związane z połączeniem sieciowym – LNK, FDX i 100. Wersje A/A+ mają jedynie diody PWR i ACT, a dla B+ sygnalizację sieci przesunięto na wtyk Ethernetowy. W Internecie (np. [4] lub w poprzednich artykułach tej serii) znajdziecie kilka przepisów mówiących o sygnalizacji podstawowych problemów przy starcie. Określona liczba mrugnięć ACT – 3, 4, 7 – sygnalizuje, z jakim problemem mamy do czynienia. Niestety, jest to jedna z tych funkcji, która z wersji na wersję dość dynamicznie się zmienia. W rzeczywistości zaufać można jedynie zachowaniu diody PWR. Jeżeli zacznie mrugać, przygasać i podobne – problem wiąże się z zasilaniem. Zakładając, że Wasza ładowarka działa poprawnie i ma wydajność powyżej 500 mA, z wersją B nie powinno być żadnych problemów. Trochę gorzej to wygląda dla B+. Zauważyłem, że z ładowarką działającą z B (ok. 750 mA), B+ (bez dołączonych żadnych akcesoriów) potrafi „oszaleć". Problemy znikają po zastosowaniu zasilacza 2 A (np. od tabletu).

Wracając do diody ACT. Do doświadczeń użyłem trzech różnych Raspberry: B (rev2), B+ i A+ z ostatnim Raspbianem usuwając lub uszkadzając (wymazane przypadkowe fragmenty plików) kolejne pliki startowe. Okazało się, że każde z urządzeń zachowuje się trochę inaczej (zob. **tabela 2**). Generalnie najbardziej „rozmowna" okazała się najnowsza A+. Sygnalizowała każdą z możliwych sytuacji, tak braku jak i uszkodzenia pliku startowego. Najmniej mrugała najstarsza B – tylko brak „start.elf" lub jądra „kernel.img". B+ zachowuje się pośrednio między B i A+. Niestety, nie mam do dyspozycji wersji A, a rezultaty

Bootloader – program rozruchowy. Zależnie od potrzeb jego zadaniem jest inicjacja układów, ich początkowa konfiguracja, uruchomienie podstawowych usług, załadowanie i przekazanie kontroli do innych programów. Często używa się całego łańcuszka bootloaderów, z których każdy uruchamia następny, bardziej skomplikowany i realizujący coraz więcej funkcji. Taki system nazywamy rozruchem wieloetapowym (ang. *multi-stage bootloader*).

Firmware – specyficzny rodzaj oprogramowania wbudowanego w urządzenie, zapewnia jego inicjację, dostarcza podstawowe procedury obsługi.

PID (ang. *Process Identifier*) – w systemie operacyjnym: unikatowy identyfikator procesu. Na przykład „init" – proces zarządzający uruchamianiem i zatrzymywaniem procesów użytkownika w Linuksie – ma PID=1.

Link symboliczny (ang. *symlink*) – specyficzny rodzaj zasobu, który wskazuje na plik lub katalog. Na poziomie programów odwołanie jest praktycznie przezroczyste i zachowuje się dokładnie tak, jak w przypadku właściwego obiektu. Użycie komendy „ls -l", żeby zobaczyć, na co takie linki wskazują.

Tabela 2. Dioda ACT i popularne problemy dla Raspberry B, B+ i A+

	Usunięty	Uszkodzony
Bootcode.bin	B: ACT nie mruga A+: ACT świeci ciągle B+: ACT świeci ciągle	ACT nie mruga A+: ACT świeci ciągle B+: ACT świeci ciągle
start.elf	B: 4 B+: 4 A+: ACT świeci ciągle	B: ACT nie mruga B+: ACT nie mruga A+: ACT świeci ciągle
fixup.dat	Startuje, ale...	Startuje, ale...
kernel.img	B, B+, A+: 7 błysków ACT	B, B+, A+: ACT mruga podczas ładowania jądra, proces się nie kończy, ze względu na błąd dekompresji (wyświetlany na konsoli)

mogłyby być całkiem interesujące. Najlepiej jednak po prostu pamiętać o robieniu kopii zapasowej.

Pewne wątpliwości mogą dotyczyć plików „fixup.*". Generalnie wszystkie źródła podają, że są one konieczne do uruchomienia urządzenia. Ja usuwałem je z karty... i Raspberry startował. Problemy pojawiały się jednak przy wykrywaniu i dzieleniu pamięci. Bez „fixup.*":

- B+: Raspberry wykrywał jedynie 256 MB całkowitej pamięci (z 512 MB fizycznych);
- B+: podział pamięci nie odzwierciedlał ustawień „/boot/config.txt" (oprócz „gpu_mem=16");
- B: podobnie;
- A+: zachowywał się podobnie (A+ ma 256 MB RAM).

Gdy system startowałem tylko z plikiem „fixup_cd.dat" na karcie – jedynie dla „gpu_mem=16" konfiguracja pamięci wydawała się poprawna. Jego brak (przy jednoczesnej obecności „fixup.dat") sprawiał problemy dla tego ustawienia. Wszystkie te eksperymenty obrazuje **tabela 3**.

Oczywiście powyższe doświadczenia możemy traktować jedynie w kategorii ciekawostek. Musicie wziąć poprawkę na to, że w niestandardowych sytuacjach narzędzia (takie jak „free" czy „vcgencmd") mogą zwracać zafałszowane rezultaty. Dodatkowo przeprowadzałem jedynie testy uruchomieniowe. Głębsza diagnostyka mogłaby wykazać więcej problemów. Do czego więc te informacje mogą się Wam przydać? Jeżeli zauważycie problemy z ilością pamięci dostępnej dla Raspberry lub z jej podziałem – upewnijcie się, że pliki „fixup.*" znajdują się na karcie i są poprawne. Jeżeli to nic nie da – ponownie wypalcie kartę. To zresztą najlepsza rada na większość bolączek.

Tajemnica 4: start Linuksa

Gdy „start.elf" załaduje jądro Linuksa do pamięci, GPU ustawia CPU w stan aktywny. CPU rozpoczyna wykonywanie kodu jądra zgodnie z parametrami z „cmdline.txt". Trzeba by obszernej książki, żeby opisać wszystko, co się wtedy dzieje. Dość wspomnieć, że inicjowane są urządzenia, ładowane sterowniki dla nich, montowane systemy plików (m.in. „/") itd. Przeskoczymy do końca tej procedury, koncentrując się na domyślnym dla Raspbiana sekwencyjnym sposobie inicjacji typu *SystemV*. Wprowadzimy teraz pojęcie poziomów uruchamiania (ang. *runlevel*; [3], [7]):

- S – początkowy, inicjuje sprzęt;
- 0 – zamykanie systemu po komendzie „sudo halt" lub „sudo shutdown now";
- 1 (ang. *single-user*) – tryb pojedynczego użytkownika przeznaczony do administracji systemem; czasami możecie go doświadczyć w przypadku uszkodzenia systemu plików;
- 2 – tryby dla wielu użytkowników (ang. *multiuser*);
- 3, 4, 5 – dla Raspbiana identyczne z 2;
- 6 – restart systemu, po komendzie „sudo reboot" lub „sudo shutdown -r now".

Tabela 3. Doświadczenia z plikami „/boot/fixup*.dat"

gpu_mem w config.txt	Bez fixup.* [gpu MB/cpu MB]	Tylko fixup.dat [gpu MB/cpu MB]	Tylko fixup_cd.dat [gpu MB/cpu MB]
B+: 16 MB	16/240 Tylko 256 MB!	16/240 Tylko 256 MB!	16/496 OK
B+: 32/64/128/256	128/128 Tylko 256 MB!	Zgodnie z ustawieniami, 512 MB – OK	128/128 Tylko 256 MB!
A+: 16 MB	16/240 OK?	16/240 OK?	16/240 OK
A+: 32/64/128	128/128 Niezgodnie z ustawieniami	Zgodnie z ustawieniami, 256 MB – OK	128/128 Niezgodnie z ustawieniami
A+: 256 (eksperyment)	128/128	Ciekawe: 192/64	128/128

Przy **starcie** (podobnie jak w Debianie), system operacyjny Raspbian:

- **krok 1** – wykonuje poziom „S”;
- **krok 2** – jeżeli nie wystąpił błąd przy „S”, uruchamia jeden z poziomów dla wielu użytkowników (2-5).

Poszczególne etapy dostępne są w katalogach „/etc/rcX.d”, gdzie „X” to poziom „S” lub 0-6. Spójrzcie na zawartość katalogu „/etc/rcS.d” (część listingu):

```
$ ls /etc/rcS.d
K05hwclock.sh S01hostname.sh
K12rpcbind
S01mountkernfs.sh
K13nfs-common S02udev
```

System wyświetlił cały zestaw… plików? Nie do końca. Ciekawsze rezultaty przyniesie komenda „ls -l” (część listingu):

```
$ ls -l /etc/rcS.d
lrwxrwxrwx 1 root root  20
Dec 21 12:17 K05hwclock.
sh -> ../init.d/hwclock.sh
lrwxrwxrwx 1 root root  17
Dec 21 12:23 K12rpcbind
-> ../init.d/rpcbind
lrwxrwxrwx 1 root root  20
Dec 21 12:23 K13nfs-common
-> ../init.d/nfs-common
lrwxrwxrwx 1 root root  21
Dec 21 12:08 S01hostname.
sh -> ../init.d/hostname.sh
lrwxrwxrwx 1 root root  24 Dec
21 12:08 S01mountkernfs.sh ->
../init.d/mountkernfs.sh
lrwxrwxrwx 1 root root  14 Dec 21
12:08 S02udev -> ../init.d/udev
```

Wpisy w tym katalogu nie są więc plikami, ale linkami symbolicznymi do skryptów znajdujących się w „/etc/init.d”. Zwróćcie uwagę na to, z czego składają się ich nazwy:

- pierwsza litera – S lub K;
- dwucyfrowy numer;
- nazwa skryptu.

Linki zaczynające się od litery „S” uruchamiane są z parametrem „start” i służą do startowania usług. Linki z literą „K” uruchamiane są przez system z parametrem „stop” – służą do zatrzymywania usług (w konkretnym przypadku rcS nie są brane pod uwagę). Dalej skrypty wykonują się w kolejności dwucyfrowego numeru – np. S01hostname.sh przed S02udev.sh itd.

W ten sposób system inicjuje procesy dla poziomu „S”.

Przejdźmy teraz do **kroku drugiego**. Dla Raspberry Pi domyślny poziom dla wielodostępu to „2”. Można to znaleźć w samym skrypcie „/etc/inititab” (wykorzystywanym przez program „init”):

```
$ cat /etc/inittab |
grep initdefault
id:2:initdefault:
```

ODPOWIEDZI NA PYTANIA

Dostaję od Was listy na temat popularnych problemów dotyczących pracy z Raspberry Pi. Poniżej kilka z pytań:

1. Mój Minecraft na Raspberry Pi B+ bardzo wolno chodzi... Obraz skacze, jest mało płynny... Co mogę zrobić?

Spróbuj zwiększyć ilość pamięci dostępnej dla GPU do 256 MB. Możesz również podnieść taktowanie procesora. Od pewnego czasu jest to opcja konfiguracyjna *raspi-config*, a jej zmiana nie powoduje utraty gwarancji. Może jednak skrócić żywotność Twojego RPi lub negatywnie wpłynąć na jego stabilność.

2. Potrzebuję uruchomić GParted na Raspberry, ale nie pamiętam hasła root. System o nie pyta, ale go nie znam i nie mogę zgadnąć, jak brzmi?

Hasło *root* domyślnie nie jest ustawione (ale nie znaczy to, że jest puste). Wpisz: „sudo passwd root” i ustaw nowe hasło. Dalej powinno już pójść bez problemów (na podstawie: http://goo.gl/5HebiS).

3. Podczas pobytu u znajomego podłączyłem swoje Raspberry do jego sieci domowej. RPi automatycznie dostał adres IP (DHCP), ale przeglądarka Midori nie chciała podłączyć się do żadnego portalu; również „apt-get update” narzekał na brak możliwości rozwiązania nazw serwerów (ang. *can't resolve host name*). W czym problem?

Powodów może być wiele. Zakładając, że RPi faktycznie dostał poprawny adres IP (a nie np. 169.254/16 z autokonfiguracji) – problem może leżeć w DNS. DNS to specjalne serwery, które potrafią tłumaczyć adres IP (np. 188.68.249.117) na nazwę zrozumiałą dla ludzi (np. www.mt.com.pl). Czasami ich ustawienie szwankuje i RPi nie potrafi przetłumaczyć ich nazw – jedynie adresy IP (stosunkowo trudne do zapamiętania).

Wystarczy:
otworzyć plik „/etc/dhcp/dhclient.conf” do edycji:
„sudo nano /etc/dhcp/dhclient.conf”;
odnaleźć linijkę: „prepend domain-name-servers 127.0.0.1;”;
zmienić ją na: „prepend domain-name-servers 8.8.8.8;”
(DNS Google);
zapisać plik: CTRL-X, Y, ENTER;
zrestartować interfejs sieciowy: „sudo ifdown eth0”
i „sudo ifup eth0”.
Powinno pomóc (na podstawie: http://goo.gl/ZY9og0)

Więcej praktycznych porad na temat Raspberry i nie tylko znajdziecie na moim blogu: uczymy.edu.pl

Szanowni Czytelnicy.
Ewentualne pytania do autora można kierować bezpośrednio na adres: arkadiusz.merta@mt.com.pl

Tabela 4. Przydatne polecenia linuksowe użyte w tekście (w kolejności występowania)

Polecenie	Znaczenie i opcje	Przykładowe użycie

vcgencmd get_mem	Pamięć przydzielona dla CPU (parametr „arm") i GPU (parametr „gpu")	Pamięć zarezerwowana dla CPU:
		$ vcgencmd get_mem arm
		Pamięć zarezerwowana dla GPU:
		$ vcgencmd get_mem gpu

lsblk	Lista podmontowanych urządzeń blokowych	Lista urządzeń:
		$ lsblk
		Rozmiary w bajtach:
		$ lsblk -b

ps	Wyświetla informację o aktualnie uruchomionych procesach	Wszystkie uruchomione procesy: $ ps aux
cat	Wypisanie zawartości pliku	Wyświetl zawartość pliku „/boot/config.txt": $ cat /etc/inittab
ls	Wyświetla zawartość katalogu	Dane szczegółowe (rozwinięcie linków): $ ls -l
grep	Filtrowanie strumienia wyjściowego (np. z komendy) zgodnie z podanym kryterium	Wyświetl linie pliku „/boot/config.txt" zawierające łańcuch „gpu": $ cat /boot/config.txt | grep gpu
runlevel	Wyświetla aktualny poziom uruchamiania	Aktualny poziom uruchamiania: $ runlevel
less	Wyświetla strumienie danych (np. pliki) w formie stronicowanej; nawiguj kursorami, PgUp/PgDn; q – wyjście	Zawartość pliku „/var/log/syslog": $ less /var/log/syslog
nano	Prosty, ale użyteczny edytor tekstowy	Edycja pliku „/etc/rc.local": $ sudo nano /etc/rc.local

update-rc.d	Skrypt umożliwiający dodanie (default) lub usunięcie (remove) procesów uruchamianych przy starcie	Instalacja (z: zasób):
		$ sudo update-rc.d z default
		Usuwanie:
		$ sudo update-rc.d z remove
		Włączanie/wyłączanie:
		$ sudo update-rc.d z enable
		$ sudo update-rc.d z disable

man	Informacja o Linuksie i jego komendach	Więcej o starcie systemu: $ man init

Aktualny poziom możecie sprawdzić poleceniem „runlevel":

```
$ runlevel
N 2
```

Podobnie jak w przypadku „S" pliki dla tego poziomu można znaleźć w katalogu „/etc/rc2.d". Skrypty startuje się analogicznie.

Jak możecie tę wiedzę wykorzystać? Przyda się Wam za każdym razem, gdy będziecie chcieli uruchomić jakąś usługę przy starcie systemu. Spójrzmy na komendę „update-rc.d".

Najpierw w katalogu domowym stwórzcie krótki skrypt Pythonowy:

```
$ nano mtcustom.py
print „Witajcie, Mlodzi Technicy!"
```

Następnie w katalogu „/etc/init.d" stwórzcie skrypt startowy:

```
#! /bin/sh
logger „[AM:] Pozdrowienia dla
Mlodych Technikow („$1")!"
case "$1" in
  start)
        /usr/bin/python /
home/pi/mtcustom.py
        ;;
  *)
        logger „[AM:] ten skrypt
reaguje tylko na start"
        exit 1
        ;;
esac
```

Skrypt ten używa systemowego programu „logger", który swoje wpisy wysyła do logu „/var/log/syslog" – razem z wartością parametru, z jakim został wywołany. Gdy parametr równa się „start" – uruchamia się skrypt „~/mtcustom.py". W przypadku innego wywołania – loguje się wiadomość o nieprawidłowym parametrze. Pozostaje dodać go do startu (wpisując fragment „mtcu..." , wciśnijcie klawisz [Tab] – samo się uzupełni):

```
$ sudo update-rc.d mtcustom default
```

Zrestartujcie system („sudo reboot"). Zauważycie, że podczas startu konsola przywita Was wyrażeniem

ze skryptu. Dodatkowo w logu systemowym znajdziecie (niektóre linie usunąłem):

```
$ cat /var/log/syslog | grep [AM:]
Jan  5 21:44:01 raspberry-
pi logger: [AM:] Pozdrowienia
dla Mlodych Technikow (stop)!
Jan  5 21:44:01 raspberry-
pi logger: [AM:] ten skrypt re-
aguje tylko na start
Jan  5 21:44:19 raspberrypi log-
ger: [AM:] Pozdrowienia dla
Mlodych Technikow (start)!
```

Istnieje jeszcze jedna możliwość uruchomienia własnego programu przy starcie. Jest nią skrypt „/etc/rc.local”. Jeżeli wyświetlicie zawartość katalogu „/etc/rc2.d”, zobaczycie tam następujący link do ../init.d/rc.local, który jako „S04” uruchomiony zostanie prawie jako ostatni:

```
$ ls -l /etc/rc2.d
lrwxrwxrwx 1 root root  18
Dec 21 12:36 S04rc.lo-
cal -> ../init.d/rc.local
```

„etc/init.d/rc.local” wskazuje z kolei „/etc/rc.local”. Domyślnie wyświetla on jedynie adres IP komputera, ale możecie go łatwo dostosować do swoich potrzeb. Pamiętajcie jedynie, że wykonuje się z prawami *root*. Musicie również zadbać, żeby nie zamieszczać w nim zbyt długich operacji, gdyż mogą zablokować resztę startu systemu (użyjcie „&” na końcu wywołania, żeby można było oddać kontrolę do systemu).

Podsumowanie

W powyższym artykule prześledziliśmy kilka zagadnień dotyczących startu Raspberry Pi pod kontrolą systemu Raspbian. Zaczęliśmy od sposobu partycjonowania karty (FAT i niewidoczna dla Windows ext4), poprzez wszystkie poziomy ładowania kolejnych bootloaderów, by skończyć na ogólnym opisie systemu SysV. Dużym zaskoczeniem może być rola GPU w procesie startu, które wykonuje praktycznie całą „czarną” robotę. Wiecie już, że jest nie tylko odpowiedzialny za operacje graficzne. Kolejne przydatne informacje wiążą się ze skryptami startowymi jądra linuksowego. Zrozumieliście, jak np. automatycznie uruchomić skrypt Pythonowy podczas inicjacji systemu. Będziemy te umiejętności wykorzystywać w kolejnych tekstach na temat Raspberry Pi. W **tabeli 4** zestawiłem ważniejsze polecenia użyte w tym artykule. ■

Arkadiusz Merta

Źródła:

[1] http://goo.gl/4cZMxB
[2] http://goo.gl/3wh5gD
[3] http://goo.gl/nT5VkJ
[4] http://goo.gl/8MtGDS
[5] http://goo.gl/DqMdIj
[6] http://goo.gl/nPiL0B
[7] http://goo.gl/GnCU1k