1. Charakterystyczne cechy systemów wbudowanych - porównanie z innymi systemami komputerowymi

Systemy wbudowane: - **Specjalizowane zastosowanie:** Projektowane do wykonywania jednego lub kilku konkretnych zadań. - **Ograniczone zasoby sprzętowe:** Minimalna ilość pamięci RAM, mocy obliczeniowej, i przestrzeni magazynowej. - **Niska konsumpcja energii:** Efektywność energetyczna jest kluczowa. - **Wysoka niezawodność:** Często muszą działać bez przerwy, w trudnych warunkach. - **Brak lub ograniczony interfejs użytkownika:** Często sterowane automatycznie lub z prostymi interfejsami (np. diody LED, przyciski).

Standardowe systemy komputerowe: - **Wszechstronność:** Mogą wykonywać różnorodne zadania, od przeglądania internetu po edycję wideo. - **Znaczne zasoby sprzętowe:** Duża ilość pamięci RAM, procesory wielordzeniowe, duże dyski twarde. - **Większa konsumpcja energii:** Mniej ważna efektywność energetyczna. - **Mniej niezawodne:** Mogą wymagać częstych restartów i aktualizacji. - **Zaawansowany interfejs użytkownika:** Graficzne interfejsy użytkownika, klawiatury, myszy, ekrany dotykowe.

2. Typowe zastosowania systemów wbudowanych

- **Elektronika użytkowa:** Telewizory, smartfony, odtwarzacze DVD.
- **Przemysł:** Kontrolery PLC, roboty, systemy automatyki.
- **Motoryzacja:** Systemy ABS, kontrola silnika, nawigacja GPS.
- **Telekomunikacja:** Routery, switche, telefony VoIP.
- **Medycyna:** Aparaty EKG, respiratory, pompy infuzyjne.
- **Domowe AGD:** Pralki, zmywarki, lodówki.

3. Porównanie systemów wbudowanych pracujących pod kontrolą standardowego systemu operacyjnego z systemami "bare metal"

Systemy z systemem operacyjnym: - **Zalety:** - **Abstrakcja sprzętu:** Łatwiejsze programowanie dzięki warstwie abstrakcji. - **Multitasking:** Obsługa wielu procesów jednocześnie. - **Standardowe narzędzia i biblioteki:** Lepsze wsparcie programistyczne. - **Wady:** - **Większe wymagania sprzętowe:** Potrzebują więcej pamięci i mocy obliczeniowej. - **Narzut systemowy:** Może obniżać wydajność w porównaniu do rozwiązań bare metal.

Systemy bare metal: - **Zalety:** - **Maksymalna wydajność:** Brak narzutu systemowego. - **Mniejsze wymagania sprzętowe:** Idealne dla systemów z ograniczonymi zasobami. - **Wady:** - **Wyższa złożoność programowania:** Wymaga głębszej znajomości sprzętu. - **Brak abstrakcji sprzętu:** Każdy projekt może wymagać unikalnego podejścia.

4. Jak realia współczesnego rynku podzespołów elektronicznych wpływają na implementację systemów wbudowanych?

- **Niedobory komponentów:** Problemy z dostępnością mogą opóźniać produkcję.
- **Szybki postęp technologiczny:** Wymaga częstych aktualizacji projektów.
- **Globalizacja łańcucha dostaw:** Ryzyko związane z logistyką i cłami.
- **Koszty:** Fluktuacje cen komponentów wpływają na budżety projektowe.

5. Przykłady systemów operacyjnych dla systemów wbudowanych

- FreeRTOS
- Zephyr
- RIOT OS
- VxWorks
- QNX
- uClinux

6. Przykłady platform dla systemów wbudowanych

- **Raspberry Pi**
- **Arduino**
- **BeagleBone**
- **ESP8266/ESP32**
- **STM32 Nucleo**
- **Odroid**

7. Różnice w wykorzystaniu Linuxa w systemie wbudowanym i normalnym systemie komputerowym

Linux w systemie wbudowanym: - **Lekka konfiguracja:** Usunięcie niepotrzebnych komponentów. - **Minimalne zasoby:** Dostosowany do pracy z ograniczonymi zasobami. - **Specyficzne sterowniki:** Skonfigurowany pod konkretne urządzenia wbudowane.

Linux w normalnym systemie komputerowym: - **Pełna funkcjonalność:** Wsparcie dla szerokiego zakresu aplikacji i urządzeń. - **Większe zasoby:** Wymaga więcej pamięci, procesora i przestrzeni dyskowej. - **Standardowe sterowniki:** Ogólnodostępne sterowniki do różnych urządzeń.

8. Zestawy narzędzi do kompilacji systemu Linux dla systemu wbudowanego

- **Buildroot**
- **Yocto Project**
- **OpenEmbedded**
- **LTIB (Linux Target Image Builder)**

9. Trudności przy wykorzystaniu zwykłej dystrybucji Linuxa jako systemu operacyjnego dla systemu wbudowanego

- **Zasoby sprzętowe:** Zwykłe dystrybucje wymagają więcej pamięci i mocy obliczeniowej.
- **Niepotrzebne komponenty:** Wiele usług i programów, które nie są potrzebne w systemach wbudowanych.
- **Optymalizacja:** Typowe dystrybucje nie są zoptymalizowane pod kątem specyficznych zadań.

10. Modyfikacja systemu Linux do wykorzystania go w systemie wbudowanym

- **Usunięcie zbędnych pakietów i usług.**
- **Kompilacja jądra z niezbędnymi modułami.**
- **Optymalizacja systemu plików i bibliotek.**
- **Dostosowanie skryptów startowych do specyficznych wymagań.**

11. Minimalny system Linux dla systemu wbudowanego

- **Jądro Linuxa:** Skonfigurowane i skompilowane do specyficznych potrzeb.
- **Init system:** Busybox jako init system, zapewniający podstawowe narzędzia systemowe.
- **Sterowniki:** Konieczne sterowniki dla specyficznego sprzętu.
- **Biblioteki C:** Używane do podstawowych operacji, np. uClibc lub musl.
- **System plików:** Minimalny system plików zawierający niezbędne narzędzia.

12. Cykl pracy ze środowiskiem Buildroot przy przygotowaniu obrazu systemu Linux dla systemu wbudowanego

1. **Pobranie Buildroot:** Pobierz i rozpakuj Buildroot.
2. **Konfiguracja:** Uruchom `make menuconfig`, aby skonfigurować projekt.
3. **Dostosowanie:** Wybierz docelową platformę, zestaw narzędzi, pakiety i inne opcje.
4. **Kompilacja:** Wykonaj `make`, aby zbudować obraz systemu.
5. **Testowanie:** Przetestuj wynikowy obraz na docelowym sprzęcie lub emulatorze.

13. Środki skracające czas kompilacji środowiska Buildroot

- **Włączenie ccache:** Używanie pamięci podręcznej kompilatora.
- **Wykorzystanie prekompilowanych zestawów narzędzi.**
- **Równoległa kompilacja:** Ustawienie opcji `BR2_JLEVEL` na wartość większą niż 1.

14. Wersje biblioteki libc dla systemu wbudowanego

- **uClibc-ng:** Lekka, szybka, ale mniej funkcjonalna niż glibc.
- **musl:** Nowoczesna, wydajna, lepsza zgodność niż uClibc-ng, ale nie tak kompletna jak glibc.
- **glibc:** Najbardziej kompletna, ale też najcięższa i wymagająca więcej zasobów.

15. Co to jest „busybox" i jaką rolę pełni w systemie Linux dla systemów wbudowanych?

Busybox to zestaw wielu standardowych narzędzi uniksowych skompilowanych w jeden plik wykonywalny. W systemach wbudowanych pełni rolę minimalnego zestawu narzędzi systemowych, zapewniając funkcjonalność przy minimalnym zużyciu zasobów.

16. Narzędzia do konfiguracji środowiska Buildroot

-

make menuconfig - make nconfig - make gconfig - make xconfig

17. Możliwości precyzyjniejszego skonfigurowania środowiska Buildroot

- **Dostosowanie konfiguracji poszczególnych pakietów.**
- **Dodanie własnych skryptów post-build.**
- **Tworzenie i modyfikowanie własnych pakietów.**

18. Modyfikacje do obsługi urządzenia zewnętrznego nieuwzględnionego w konfiguracji domyślnej

- **Dodanie sterowników do jądra.**
- **Modyfikacja plików konfiguracyjnych, np. Device Tree.**

19. Kroki po „make clean" do zapewnienia poprawnej rekompilacji systemu

- **Zapisanie konfiguracji przed `make clean`.**
- **Przywrócenie konfiguracji po `make clean`.**
- **Ponowna konfiguracja i kompilacja systemu.**

20. Dodanie katalogów i zbiorów zawierających uzupełnienia

- **Root filesystem overlay:** Użycie opcji `BR2_ROOTFS_OVERLAY` w Buildroot.

21. Dodanie oprogramowania wymagającego kompilacji ze źródeł

- **Tworzenie własnego pakietu:** Skonfigurowanie odpowiednich plików konfiguracyjnych i Makefile dla nowego pakietu w katalogu `package/`.

22. Dodanie własnych „łat" do standardowych pakietów

- **Utworzenie katalogu z łatami:** Przechowywanie łatek w katalogu z nazwą pakietu.
- **Skonfigurowanie opcji** `BR2_GLOBAL_PATCH_DIR` w Buildroot, aby wskazać ścieżkę do katalogu z łatami.

23. Pełna a częściowa rekompilacja

- **Pełna rekompilacja:** Wymagana przy zmianie konfiguracji toolchaina lub architektury, lub przy usuwaniu pakietów (`make clean all`).
- **Częściowa rekompilacja:** Możliwa przy dodawaniu nowych pakietów lub modyfikacji istniejących (np. `make <pakiet>-rebuild`).

24. Testowanie systemu wbudowanego za pomocą maszyny wirtualnej

- **Narzędzia:** QEMU, VirtualBox, VDE.
- **Możliwości:** Emulacja różnych platform sprzętowych, testowanie sieci, urządzeń USB.
- **Ograniczenia:** Brak pełnej wydajności sprzętowej, niektóre urządzenia mogą nie być w pełni obsługiwane.

25. Proces ładowania systemu operacyjnego w typowym systemie wbudowanym

- **Poziomy programów ładujących:**
  - **Bootloader (np. U-Boot):** Inicjalizacja sprzętu, ładowanie jądra.
  - **Kernel:** Ładowanie systemu operacyjnego.
  - **Init:** Uruchomienie skryptów startowych i aplikacji użytkowych.
- **Powód:** Zapewnienie elastyczności, możliwość aktualizacji poszczególnych komponentów bez wpływu na cały system.

26. Wykorzystanie systemu Linux jako programu ładującego

- **Korzyści:** Możliwość ładowania złożonych środowisk, obsługa różnych systemów plików, możliwość użycia zaawansowanych funkcji.
- **Realizacja:** Konfiguracja bootloadera do uruchamiania jądra Linux, konfiguracja initramfs do ładowania odpowiednich modułów i skryptów startowych.

27. Zalety pracy systemu wbudowanego z głównym systemem plików w „initramfs”

- **Szybki start systemu.**
- **Odporność na uszkodzenia systemu plików.**
- **Łatwość aktualizacji.**
- **Przechowywanie informacji o stanie systemu:** Użycie systemu plików do zapisów tymczasowych lub przeniesienie krytycznych danych na trwałe nośniki pamięci, jak karta SD lub pamięć FLASH.

28. Konfiguracja systemu Linux do pracy z głównym systemem plików w pamięci FLASH

- **System plików:** Użycie systemów plików odpornych na awarie, jak JFFS2 lub UBIFS.
- **Minimalizacja ryzyka uszkodzeń:** Stosowanie technik takich jak transakcyjne zapisy lub log-structured file systems.

29. Zastosowanie debuggera w systemie wbudowanym o ograniczonych zasobach

- **Zdalne debugowanie:** Użycie GDB w trybie zdalnym, debugowanie aplikacji na hosta.
- **Konfiguracja:** Uruchomienie GDB na komputerze hosta, podłączenie do systemu wbudowanego przez sieć lub port szeregowy.

30. Obsługa niestandardowego urządzenia peryferyjnego bez dedykowanego sterownika

- **Użycie standardowych interfejsów:** SPIdev lub I2Cdev do komunikacji z urządzeniem.
- **Programowanie użytkownika:** Pisanie aplikacji użytkownika w C lub Pythonie do bezpośredniej obsługi urządzenia.

31. Automatyczne tworzenie plików specjalnych dla urządzeń USB

- **Konfiguracja udev:** Wkompilowanie narzędzi `udev` i odpowiednich skryptów do systemu.

32. Informowanie jądra systemu o nietypowych urządzeniach

- **Device Tree:** Modyfikacja plików Device Tree.
- **Sterowniki platformowe:** Pisanie i wkompilowanie odpowiednich sterowników do jądra.

33. Obsługa GPIO z poziomu aplikacji użytkownika

- **Biblioteki:** Użycie bibliotek takich jak `libgpiod`.

- **Skrypty powłoki:** Użycie plików specjalnych w `/sys/class/gpio` do sterowania GPIO za pomocą skryptów powłoki.

34. Sterowniki do urządzeń SPI i I2C

- **SPI:** `spidev`.
- **I2C:** `i2c-dev`.
- **Programowanie:** Użycie odpowiednich funkcji systemowych w C, takich jak `ioctl`.

35. System plików dla pamięci FLASH

- **JFFS2:** Journaled Flash File System 2.
- **UBIFS:** Unsorted Block Image File System.

36. System plików dla kart SD i dysków USB

- **ext4:** Wspiera transakcyjne zapisy, lepsza odporność na uszkodzenia.
- **FAT32:** Kompatybilność, ale brak wsparcia dla nowoczesnych funkcji.

37. Modyfikacje w systemie używającym squashfs

- **OverlayFS:** Użycie systemu plików warstwowego do zapisywania zmian.

38. Tryb awaryjny w systemie wbudowanym

- **Dwa systemy plików:** Jeden oryginalny, drugi z modyfikacjami.
- **Skrypty startowe:** Użycie skryptów do wyboru trybu startu w zależności od wciśnięcia przycisku lub innego sygnału.

39. Uruchamianie systemu z głównym systemem plików przez NFS

- **Zastosowanie:** Ułatwienie aktualizacji i debugowania.
- **Realizacja:** Konfiguracja bootloadera i jądra do uruchamiania przez sieć.
- **Zagrożenia:** Ataki na serwer NFS, konieczność zabezpieczeń.

40. Automatyczny restart systemu

- **Watchdog:** Użycie sprzętowego lub programowego watchdog.

41. Zachowanie logów systemowych w initramfs

- **Zewnętrzne logowanie:** Użycie syslog do przesyłania logów na zewnętrzny serwer.
- **Persistent storage:** Zapis logów na trwałych nośnikach pamięci.

42. Podłączenie 16-przyciskowej klawiatury do 8 wyprowadzeń GPIO

- **Matrycowe połączenie:** Połączenie przycisków w układ matrycowy (4x4).

43. Bezpieczny dostęp do systemu przez sieć TCP/IP

- **SSH:** Dla dostępu eksperta.
- **HTTPS:** Dla interfejsu WWW.
- **SSL/TLS:** Zapewnienie szyfrowania transmisji.

44. Różnice między OpenWRT a Buildroot

- **OpenWRT:** Skierowane na routery, dynamiczne zarządzanie pakietami.
- **Buildroot:** Prostsze, bardziej elastyczne, lepsze do jednorazowych wdrożeń.

45. Kompilacja OpenWRT z możliwością dodawania aplikacji

- **SDK OpenW

RT:** Umożliwia kompilowanie aplikacji bez pełnych źródeł systemu.

46. Problemy przy przenoszeniu Buildroot na inny komputer

- **Ścieżki absolutne:** Problemy z zachowaniem ścieżek w konfiguracji.
- **Rozwiązanie:** Poprawa ścieżek i ponowna konfiguracja.

47. Zapis danych użytkownika na karcie SD

- **Atomowe zapisy:** Techniki zapewniające integralność danych.
- **System plików:** Użycie systemu plików wspierającego transakcyjne zapisy, np. ext4.

48. Różnice między Linuxem używającym initramfs a standardowym systemem plików

- **initramfs:** Szybsze uruchamianie, ale ograniczona funkcjonalność.
- **Standardowy system plików:** Większa funkcjonalność, ale wolniejsze uruchamianie.

49. Używanie Linuxa jako bootloadera

- **Kexec:** Uruchamianie nowego jądra bez restartu.

50. Tryb awaryjny w systemie wbudowanym

- **Użycie alternatywnego systemu plików:** Umożliwienie uruchomienia oryginalnej wersji systemu.

51. Zdalny dostęp do systemu o krytycznym znaczeniu

- **VPN:** Zapewnienie bezpiecznego tunelu komunikacyjnego.
- **SSL/TLS:** Szyfrowanie transmisji.

52. Wymagania dla systemu z USB

- **Wkompilowanie wsparcia dla USB i urządzeń masowych.**

53. Podłączenie 32-klawiszowej klawiatury do 13 wyprowadzeń GPIO

- **Matrycowe połączenie:** Połączenie przycisków w układ matrycowy (8x4).

54. Scenariusz użycia unionfs/overlayfs

- **OverlayFS:** Zapewnienie możliwości modyfikacji systemu bez zmiany bazowego obrazu.

55. Dodanie własnej aplikacji do OpenWRT

- **Pakiety:** Tworzenie własnego pakietu i dodanie go do kompilacji OpenWRT.

56. Zalety i wady użycia systemu Linux w systemie wbudowanym

- **Zalety:**
  - **Szeroka gama dostępnych narzędzi i bibliotek.**
  - **Wsparcie dla różnych urządzeń i protokołów.**
- **Wady:**
  - **Większe wymagania sprzętowe.**
  - **Złożoność konfiguracji.**

57. Problem z brakiem biblioteki zlib w Pythonie

- **Nieprawidłowa konfiguracja:** Możliwe pominięcie wymaganych pakietów podczas kompilacji.
- **Rozwiązanie:** Sprawdzenie i ponowna konfiguracja Buildroot.

58. Mniejsza biblioteka libc w Buildroot

- **uClibc-ng lub musl:** Mniejsze, zoptymalizowane pod kątem systemów wbudowanych.

59. Typowy system wbudowany vs. zwykły system komputerowy

- **Wąska specjalizacja:** Systemy wbudowane mają konkretne zadania, podczas gdy zwykłe komputery są uniwersalne.

- **Optymalizacja:** Systemy wbudowane są zoptymalizowane pod kątem zasobów, energii i niezawodności.

60. Komunikacja z niestandardowym urządzeniem I2C

- **Sterownik i2c-dev:** Umożliwia dostęp do urządzeń I2C z poziomu przestrzeni użytkownika.
- **Programowanie:** Użycie standardowych funkcji systemowych w C, takich jak `ioctl`.

61. Kilka poziomów programów ładujących

- **Powód:** Elastyczność, możliwość aktualizacji bez wpływu na cały system.
- **Standardowe bootloadery:** U-Boot, GRUB.

62. Konfiguracja Buildroot dla skryptu Lua

- **Dodanie skryptu:** Użycie `BR2_ROOTFS_OVERLAY` do dodania skryptu i niezbędnych bibliotek Lua.

63. Przywrócenie normalnej pracy po zawieszeniu

- **Watchdog:** Automatyczny restart systemu.
- **Zdalny monitoring:** Narzędzia do zdalnego monitorowania i restartu.

64. Zdalne sterowanie przez przeglądarkę WWW

- **Frameworki:** Flask, Django.
- **Bezpieczeństwo:** Użycie HTTPS i mechanizmów uwierzytelniania.

65. Zmiana konfiguracji jądra w Buildroot

- **Konfiguracja:** Użycie `make linux-menuconfig`.
- **Zachowanie konfiguracji:** Użycie `make linux-update-defconfig`.

66. Automatyczne uruchomienie programu przy starcie

- **Init scripts:** Dodanie skryptu do `/etc/init.d`.
- **Upewnienie się, że nie blokuje konsoli:** Konfiguracja odpowiednich poziomów uruchamiania.

67. Komunikacja z urządzeniem SPI

- **Sterownik spidev:** Umożliwia dostęp do urządzeń SPI z poziomu przestrzeni użytkownika.
- **Programowanie:** Użycie standardowych funkcji systemowych w C, takich jak `ioctl`.

68. Środowisko do autoryzacji w Pythonie

- **Flask-Login:** Umożliwia realizację uwierzytelniania.
- **HTTPS:** Zapewnienie szyfrowania transmisji.

69. Uszkodzone bloki pamięci FLASH

- **Wear leveling:** Techniki zarządzania zużyciem pamięci.
- **ECC:** Kody korekcji błędów.

70. Drzewa urządzeń w systemie Linux

- **Opis sprzętu:** Umożliwia jądru Linuxa rozpoznanie i konfigurację sprzętu.

71. Rola programu busybox

- **Minimalizacja zasobów:** Zawiera wiele narzędzi uniksowych w jednym pliku wykonywalnym.

72. Zachowanie logów w initramfs

- **Zewnętrzne logowanie:** Użycie syslog do przesyłania logów na zewnętrzny serwer.
- **Persistent storage:** Zapis logów na trwałych nośnikach pamięci.

73. Brak partycji wymiany w systemach wbudowanych

- **Ograniczone zasoby:** Zbyt mała ilość pamięci RAM.
- **Żywotność pamięci FLASH:** Unikanie intensywnych operacji zapisu.

74. Różnice między systemami plików

- **initramfs:** Dynamicznie ładowany w pamięci.
- **ext2/3/4:** Tradycyjne systemy plików z dziennikiem.
- **squashfs:** Kompresowany, tylko do odczytu.

75. Dodanie nowego skryptu Python do Buildroot

- **Root filesystem overlay:** Użycie opcji `BR2_ROOTFS_OVERLAY`.

76. Kilka programów ładujących w procesie uruchamiania

- **Powód:** Elastyczność, możliwość aktualizacji bez wpływu na cały system.

77. Konfiguracja urządzenia I2C w Raspberry Pi

- **Device Tree:** Dodanie wpisów do plików Device Tree.

79. Zdalne sterowanie przez sieć

- **VPN:** Bezpieczne połączenie.
- **HTTPS:** Szyfrowanie transmisji.

80. Niezawodne działanie w warunkach zakłóceń

- **Watchdog:** Automatyczny restart systemu.
- **Zasilanie awaryjne:** Użycie UPS.

81. Overlayfs i unionfs

- **OverlayFS:** Umożliwia nakładanie systemów plików.
- **UnionFS:** Łączy wiele systemów plików w jeden widok.

82. Debugowanie aplikacji w systemie wbudowanym

- **Zdalne debugowanie:** Użycie GDB w trybie zdalnym.

83. Automatyczne uruchomienie programu przy starcie

- **Init scripts:** Dodanie skryptu do `/etc/init.d`.

84. Aplikacja opkg

- **Zarządzanie pakietami:** Używana w OpenWRT, ale nie w Buildroot.

85. Modyfikacja źródeł pakietu w Buildroot

- **Patch files:** Dodanie łat do katalogu z pakietem.

86. Różnice między Buildroot a OpenWRT

- **Buildroot:** Prostsze, bardziej elastyczne.
- **OpenWRT:** Skierowane na routery, dynamiczne zarządzanie pakietami.

87. System plików w initramfs a standardowy system plików

- **initramfs:** Szybsze

uruchamianie, ale ograniczona funkcjonalność. - **Standardowy system plików:** Większa funkcjonalność, ale wolniejsze uruchamianie.

88. Używanie Linuxa jako bootloadera

- **Kexec:** Uruchamianie nowego jądra bez restartu.

89. Podłączenie urządzenia przez SPI

- **Device Tree:** Konfiguracja plików Device Tree.

90. Powody, dla których standardowa dystrybucja Linuxa może być nieodpowiednia

- **Zasoby sprzętowe:** Zbyt duże wymagania.
- **Optymalizacja:** Brak dostosowania do specyficznych zadań.
- **Złożoność:** Zbyt wiele niepotrzebnych komponentów.
- **Bezpieczeństwo:** Większa powierzchnia ataku.

91. Dedykowana wersja biblioteki libc w Buildroot

- **uClibc-ng:** Mniejsza, zoptymalizowana wersja libc.

92. Komunikacja z urządzeniem SPI w C

- **Sterownik spidev:** Umożliwia dostęp do urządzeń SPI z poziomu przestrzeni użytkownika.
- **Funkcje systemowe:** Użycie `ioctl` do komunikacji.

93. Kompilacja aplikacji na system wbudowany

- **Cross-compilation:** Użycie narzędzi do kompilacji krzyżowej.

94. Powód stosowania programu busybox

- **Minimalizacja zasobów:** Zawiera wiele narzędzi uniksowych w jednym pliku wykonywalnym.