1. **Drzewo urządzeń - co to jest, do czego służy, jakie są możliwości jego modyfikacji?**

Drzewo urządzeń – standardowy sposób opisu połączenia między CPU sprzętem, mający hierarchiczną strukturę. Pozwala na łatwe rozbudowywanie i dynamiczną zmianę opisu systemu.

1. **Obsługa magistral zapewniających autodetekcję urządzeń (np. USB, PCI Express) w systemie Linux**

Obsługa magistral zapewniających autodetekcję urządzeń w systemie Linux polega na monitorowaniu magistrali przez jądro systemu i automatycznym wykrywaniu nowych urządzeń poprzez wykorzystanie protokołów komunikacyjnych tej magistrali. Przykładem może być wykorzystanie demona eudev który może automatycznie ładować i usuwać sterowniki oraz tworzyć i usuwać pliki specjalne urządzeń.

1. **Komunikacja z urządzeniami we/wy - rejestry mapowane w pamięć, lub w oddzielną przestrzeń adresową I/O, sposoby ich obsługi**

Mapowanie w pamięć można obsłużyć przez napisanie własnego mikrosterownika korzystając z pliku dev/mem. Zaleta – sterownik pozwala też uruchomić urządzenie zgodnie z drzewem urządzeń.Mapowanie w przestrzeń adresową I/O nie jest dostępne na wszystkich architekturach i obsługuje się przez instrukcje zapisu i odczytu (outb, outw, outl, inb, inw, inl), transferów blokowych (outsb, insb itp.) i przez funkcje dla uprawnienia dostępu do portów I/O (ioperm, iopl).

1. **Szeregowanie procesów - zwykłe tryby szeregowania, tradycyjne tryby szeregowania czasu rzeczywistego, używanie trybu terminowego. Jak ich używać? Jak system radzi sobie z "błędnymi" procesami szeregowanymi w trybach czasu rzeczywistego?**

Zwykłe tryby:

* SCHED\_OTHER – zadania wykonują się po kolei
* SCHED\_BATCH – dla zadań nie interaktywnych
* SCHED\_IDLE – dla zadań realizowanych w tle

Ustawianie priorytetów umożliwiają funkcje: getpriority, setpriority.

Tradycyjne tryby:

* SCHED\_FIFO - Pierwszy zgłoszony proces będzie obsłużony jako pierwszy. Nie ma ograniczeń czasowych. Jeśli proces zostanie wywłaszczony przez proces o wyższym priorytecie, po jego wykonaniu zostanie natychmiast wznowiony. Proces jest wykonywany dopóki nie zakończy się, nie zostanie zawieszony na operacji I/O lub nie wywoła funkcji sched\_yield.
* SCHED\_RR - Jak FIFO, ale po wykorzystaniu kwantu czasu procesora, proces trafia na koniec kolejki procesów o tym samym priorytecie czasu rzeczywistego.
* SCHED\_DEADLINE - dla każdego procesu definiowane runtime, deadline, period. Błędny proces, który się nie skończy przed końcem zadeklarowanego czasu zostaje zawieszony (throttling) do początku następnego okresu i nie przeszkadza wykonaniu następnych procesów.

1. **Jak powinno zachowywać się zadanie czasu rzeczywistego, żeby nie zakłócić działania systemu?**

Zadanie musi unikać blokowania systemu przez długi czas, konfliktów z innymi procesami, błędów i awarii, a także musi być uruchomione tylko wtedy, gdy jest to konieczne, zatrzymując, kiedy nie jest potrzebne.

1. **Emulator QEMU i jego możliwości - emulowane platformy, standardowo dostępne modele urządzeń, możliwości dodawania własnych modeli urządzeń. Sposoby jego wykorzystania do testowania naszego obrazu systemu operacyjnego i aplikacji.**

QEMU pozwala na emulację procesorów ARM, x86, PowerPC, RISC-V itp.. Standardowo dostępne modele urządzeń – dyski twarde, kontrolery sieciowe, USB itd., własne modele można dodawać przy pomocy QEMU Device Tree (QDT). QEMU może być wykorzystany do testowania kompatybilności OS z różnymi platformami lub testowania aplikacji na różnych konfiguracjach sprzętowych

1. **Komunikacja międzyprocesowa - dostępne mechanizmy komunikacji, ich właściwości i sposoby wykorzystania, mechanizmy synchronizacji i ich wykorzystanie. Uruchamianie poszczególnych procesów systemu. Sposoby nawiązywania komunikacji między procesami - np. dostęp przez nazwy do kolejek itp.**

Dostępne mechanizmy komunikacji: sygnały, potoki, kolejki komunikatów, semafory, pamięć dzielona. Właściwości poszczególnych mechanizmów komunikacji:

* Kolejki komunikatów: umożliwiają komunikację jednokierunkową, zorientowana na komunikaty, używamy funkcji poll, epoll, select, identyfikowane przez nazwy, nazwa musi zaczynać się od ukośnika. Przykładowe funkcje: mq\_open, mq\_close
* Pamięć dzielona: pliki tworzone w /dev/shm. Jeśli nie zostaną usunięte, obszary przeżywają zamknięcie używających je procesów. Dostęp wymaga synchronizacji. Przykładowe funkcje: shm\_open, shm\_close

Mechanizmy synchronizacji oraz ich właściwości:

* Semafory: pozwalają pewnej liczbie procesów/wątków na uzyskanie dostępu do zasobu chronionego. Liczbę procesów określamy w funkcji sem\_init. Przykładowe funkcje: sem\_init, sem\_destroy
* Mutexy: pozwalają jednemu procesowi/wątkowi na uzyskanie dostępu do zasobu chronionego. Inne są usypiane przy próbie dostępu. Przykładowe funkcje: pthread\_mutex\_init, pthread\_mutex\_destroy
* Zmienne warunkowe: Pozwalają oczekiwać na spełnienie określonego warunku. Należy ich używać pomiędzy mutexami. Przykładowe funkcje: phtread\_cond\_init, pthread\_cond\_destroy
* Blokady: umożliwiają równoczesny dostęp wielu konsumentów przy równoczesnej synchronizacji producenta. Chodzi o problem zagłodzenia producenta. Przykładowe funckje: phtread\_rwlock\_init, pthread\_rwlock\_destroy
* Bariery: zapewnia synchronizację wykonywania kodu. Przykładowe funckje: phtread\_barrier\_init, pthread\_barrier\_destroy
* Rygle pętlowe: nie usypiają procesów tylko aktywnie oczekują na jej zwolnienie. Dobre gdy systemy wieloprocesorowe i przewidywany czas zwolnienia zasobu jest krótki a czas przełączenia procesu jest długi. Przykładowe funkcje: phtread\_spin\_init, pthread\_spin\_destroy

Uruchamianie procesów: Proces główny uruchamia inne procesy potomne funkcją fork. Proces potomny dziedziczy otwarte pliki i kolejki komunikatów. Lub można użyć innych funkcji np.: posix\_spawn, clone

1. **Możliwości rezerwacji rdzenia CPU - realizacja, do czego się przydaje, jak z tego skorzystać?**

Przydaje się np.: dla zdarzeń wymagających szczególnie szybkiej obsługi, gdy wykorzystujemy aktywne oczekiwanie. Możemy użyć funkcji sched\_setaffinity, sched\_getaffinity. Aby zablokować CPU dla wszystkich innych procesów możemy rezerwować procesor podczas startu systemu.

1. **Oczekiwanie na zdarzenie sprzętowe - oczekiwanie aktywne, wykorzystanie przerwań, porównanie tych rozwiązań. Rola sterownika urządzenia w oczekiwaniu z wykorzystaniem przerwań. Sposoby optymalizacji obsługi przerwań w sterownikach.**

Wykorzystując przerwania nie musimy czekać na “planowe” przydzielenie czasu procesora. Jednak oczekiwanie aktywne w pewnych warunkach daje lepsze wyniki niż przerwania, co może być związane z czasem przełączania kontekstu procesora.

Sterowniki pozwalają przypisywać określone przerwania do określonych CPU. Żeby uniknąć negatywnego wpływania przerwań na siebie, możliwe jest podzielenie ich obsługi na zasadniczą procedurę („górna połówka”) i na odroczoną („dolna połówka”), która obsługuje się w sposoby „zadanka” lub „kolejki prac”.

1. **System przetwarzający dane zbudowany jest z aplikacji dwóch rodzajów - serwera, pobierającego dane z czujników pomiarowych i udostępniającego je przez bufor w pamięci dzielonej, oraz z programów klienckich. Liczba programów klienckich nie jest z góry znana i mogą one być uruchamiane i zamykane niezależnie. W jaki sposób programy klienckie powinny uzyskiwać dostęp do bufora w pamięci dzielonej? W jaki sposób powinniśmy zapewnić zwolnienie tego bufora przy kończeniu działania systemu?**

Dostęp do bufora w pamięci dzielonej może być uzyskany przez funkcję mmap (z warunkiem, że działają odpowiednie metody synchronizacji np. mutexy lub semafory).

Żeby zapewnić zwolnienie tego bufora trzeba zakończyć korzystanie z dostępu przez funkcję munmap, po czym zamknąć ten obszar pamięci dzielonej przez shm\_close.

1. **W jaki sposób standardowe drzewo urządzeń systemu wbudowanego umożliwia opis ''maksymalnej'' konfiguracji tego systemu i ''aktualnie używanej'' konfiguracji? W jaki sposób możemy wybrać jakich podsystemów chcemy używać? W jaki sposób możliwe jest dostosowanie drzewa urządzeń do niestandardowych rozszerzeń, które dodaliśmy do systemu?**

Maksymalna konfiguracja opisuje wszystkie dostępne urządzenia i podsystemy, a używana – te, które są faktycznie używane w danym momencie. Aby wybrać podsystemy, można skorzystać z pliku konfiguracyjnego lub menadżera urządzeń. Dostosowanie drzewa do niestandardowych rozszerzeń jest możliwe przez dodanie odpowiednich sterowników.

1. **Jak możemy zapewnić minimalny czas reakcji naszego oprogramowania (sterownika urządzenia i/lub programu w przestrzeni użytkownika) na oczekiwane zdarzenie sprzętowe. Proszę uwzględnić dwie możliwości:**

a) system z jednym rdzeniem CPU: Możemy nadać naszemu procesowi wyższy priorytet niż inne

b) system z kilkoma rdzeniami CPU: Możemy zarezerwować rdzeń CPU dla naszego oprogramowania

1. **Dlaczego wykorzystanie DMA wymaga sterownika działającego w przestrzeni jądra?**

Dlatego że DMA wymaga dostępu do takich zasobów systemu jak RAM, adresy magistralowe które są dostępne tylko w przestrzeni jądra. Sterownik odpowiada za synchronizacja dostępu DMA do pamięci i przydzielanie jej adresów, a także pomaga uniknąć możliwych błędów i awarii.

1. **Proszę porównać komunikację między procesami za pomocą kolejek komunikatów z komunikacją za pomocą potoków. Proszę wymienić cechy wspólne i różnice.**

Cechą wspólną kolejek i potoków jest ich jednokierunkowość

Kolejki komunikatów: komunikacja jest zorientowana na komunikaty (które mogą być pobierane w dowolnej kolejności), więc procesy nie muszą być bezpośrednio połączone.

Przy korzystaniu z potoków musimy wyodrębniać dane ze strumienia danych (które pobierają się tylko w tej kolejności, w której zostały wysłane), więc jest potrzebne bezpośrednie połączenie procesów

1. **Na czym polega mapowanie rejestrów urządzenia w pamięć? Jak to jest realizowane? Jakie mechanizmy są tu niezbędne? Jakie potencjalne problemy są z tym związane? Jakie funkcje w języku C musimy wykorzystać aby móc odwołać się do tak udostępnionych rejestrów urządzenia?**

Mapowanie rejestrów w pamięć polega na przypisaniu adresów fizycznych rejestrów do adresów wirtualnych procesów, realizowane to jest przez udostępnienie fizycznych adresów przez OS. Mechanizmy niezbędne do tego procesu to mechanizmy synchronizacji, OS oraz specjalne instrukcji procesora. Potencjalne problemy: konflikty dostępów, utrudnienie w debuggowania i testowaniu. Potrzebna funkcja C - mmap(), biblioteka pthread do synchronizacji.

1. **W systemie komputerowym używanym do sterowania działa kilka procesów (nie wątków!). W pewnym momencie konieczne jest wstrzymanie działania pozostałych procesów do chwili, gdy jeden z nich zainicjalizuje pewną strukturę danych. Proszę opisać, jak należy rozwiązać ten problem. Jakiego mechanizmu synchronizującego należy użyć?**

Możemy użyć semafora który umieszczamy dla wszystkich procesów jako zamknięty przed pierwszym dostępem do struktury. Dla procesu tworzącego strukturę, po jej utworzeniu należy wywołać metodę zwalniającą semafor. Ewentualnie możemy również użyć zmiennych warunkowych oraz muteksów.

1. **Co to są bufory rozproszone (SG) z czego wynika potrzeba ich stosowania?**

Proste urządzenia mogą wymagać bufora zajmującego ciągły obszar w przestrzeni adresowej. Rozwiązaniem problemu dużych ciągłych buforów są bufory rozproszone. Symulują długie bufory poprzez listy łączone. Łączymy deskryptory w listę łączoną gdzie mamy wskaźniki na strony pamięci oraz na kolejne deskryptory.

1. **W systemie komputerowym (pracującym pod kontrolą Linuxa) zbierającym dane i sterującym na ich podstawie pewnym urządzeniem działa kilka procesów: Proces "A" obsługuje czujniki pomiarowe, przetwarza ich dane i na ich podstawie generuje sygnały sterujące. Proces "B" obsługuje graficzny interfejs użytkownika, wyświetlając przebiegi mierzonych parametrów dla operatora nadzorującego urządzenie. Proces "C" zajmuje się archiwizacją danych pomiarowych i komend sterujących, w celu ewentualnej późniejszej analizy zaistniałych nieprawidłowości. Proszę zaproponować, uzasadniając, jakie parametry szeregowania powinny zostać nadane poszczególnym procesom.**

Proces A – priorytet wysoki. Proces "A" jest kluczowy dla czasu rzeczywistego operacji systemu, ponieważ bezpośrednio obsługuje czujniki i generuje sygnały sterujące na podstawie danych pomiarowych. Każde opóźnienie w tym procesie może prowadzić do błędów sterowania urządzeniem.

Proces B – priorytet średni. Proces "B" obsługuje graficzny interfejs użytkownika i wyświetla dane pomiarowe dla operatora. Chociaż ważne jest, aby ten proces działał płynnie i reagował na interakcje użytkownika, nie jest tak krytyczny jak proces "A". Dlatego powinien mieć średni priorytet.

Proces C – priorytet niski. Proces "C" zajmuje się archiwizacją danych, co jest ważne, ale nie jest czasowo krytyczne. Można pozwolić, aby ten proces działał w tle i wykonywał swoje zadania, gdy inne, bardziej krytyczne procesy nie potrzebują zasobów CPU.

1. **Czy do synchronizacji dostępu do danych, współużywanych przez dwa procesy, mogę użyć semafora, tworzonego jako globalna zmienna statyczna, przed wywołaniem funkcji "fork"? Jeśli nie, to dlaczego i jak poprawie utworzyć taki semafor?**

Semafor utworzony jako zmienna globalna przed wywołaniem fork nie będzie działał prawidłowo, ponieważ po wywołaniu fork semafor nie będzie współdzielony między procesami (każdy proces będzie miał swoją kopię semafora w swojej przestrzeni adresowej). Aby utworzyć semafor współdzielony między procesami, należy użyć semaforów systemu POSIX przechowywanych w pamięci współdzielonej.

1. **Pewne urządzenie może obsługiwać równocześnie maksymalnie 4 zlecenia. Jaki mechanizm pozwoli najłatwiej zsynchronizować dostęp do niego tak, aby zapewnić spełnienie tego warunku? Czym różni się użycie tego mechanizmu w następujących sytuacjach:**

**a) o dostęp do tego urządzenia konkurują różne wątki tego samego procesu;**

**b) o dostęp do tego urządzenia konkurują różne procesy.**

W obu przypadkach możemy użyć semafora. Prostsze jest to kiedy używamy wątków ponieważ wątki współdzielą przestrzeń adresową. Synchronizacja za pomocą semafora jest bezpośrednia i nie wymaga dodatkowych mechanizmów. Gdy mamy różne procesy musimy wykorzystać dodatkowe mechanizmy np.: wspólną przestrzeń adresową aby procesy mogły dzielić semafor. Jest to bardziej kosztowne, lecz czasem w systemach czasu rzeczywistego lepiej używać wieloprocesowości bo np.: w razie awarii jednego z zadań łatwiej zapewnić poprawną pracę reszty systemu.

1. **Jak działa tryb szeregowania terminowego (SCHED\_DEADLINE)? Jakimi parametrami opisujemy wymagania dotyczące szeregowania procesu? Jakie wymagania są weryfikowane przez system?**

Tryb SHED\_DEADLINE rozwiązuje problem gdzie błędne procesy o wyższych priorytetach mogłyby uniemożliwić wykonanie procesów o niższych priorytetach. Dla każdego procesu definiowane są:

Czas wykonania <= Termin wykonania <= Okres wykonania

Każdy proces wpisuje swoje atrybuty do struktury sched\_attr, gdzie podlegają weryfikacji przez system. Proces, który nie zakończy pracy przed końcem zadeklarowanego czasy wykonania, zostaje zawieszony do początku następnego okresu.

1. **Pewna struktura danych, umieszczona w pamięci współdzielonej (shared memory) jest często odczytywana przez wiele procesów, a dość rzadko modyfikowana przez małą liczbę procesów. Jakiego prymitywu synchronizacyjnego powinniśmy użyć do synchronizacji dostępu wszystkich użytkowników tej struktury? Gdzie powinien on być umieszczony? Jak powinien on być zainicjalizowany?**

Prymityw synchronizacyjny: pthread\_rwlock (czyli blokada czytelnika/pisarza).

Umiejscowienie: Powinien być umieszczony w tej samej pamięci współdzielonej co struktura danych, aby wszystkie procesy miały do niego dostęp i zainicjalizowany za pomocą pthread\_rwlock\_init.

To zapewni, że wiele procesów może odczytywać strukturę równocześnie, a modyfikacja będzie zarezerwowana tylko dla jednego procesu na raz, co minimalizuje czas oczekiwania na dostęp do zasobu.

1. **Strategie szeregowania zadań.**

Szeregowanie na bieżąco:

- brak powtarzalnego schematu wykonania zadań

- czasy wykonania przekraczają okresy powtarzania

- system musi być wielozadaniowy (z wywłaszczeniem)

Szeregowanie stałe:

- znany rozkład zadań

- długie okresy powtarzania

- cykliczny program sekwencyjny.

1. **Systemy związane z bezpieczeństwem, bezpieczeństwo, ryzyko**

System którego prawidłowe działanie jest niezbędne do zapewnienia lub utrzymania bezpieczeństwa ludzi.

Bezpieczeństwo to brak nieakceptowalnego ryzyka śmierci lub uszczerbku na zdrowiu człowieka wynikającego z działania systemu.

Ryzyko = prawdopodobieństwo x stopień szkodliwości

1. **Metoda FTA (Analiza drzewa niezdatności)**

- Analiza systemów i celów jego działania

- Identyfikacja zdarzeń zagrażających

- Budowa drzewa dla każdego zdarzenia

- Badanie drzewa dla określenia tolrencji, p-stwa uszkodzenia i elementów krytycznych systemu

- Podczas projektu wprowadzamy zabezpieczenia, określamy plan spodziewanych napraw

1. **Cykl utrzymania bezpieczeństwa**

- Identyfikacja i ocena zagrożeń

- Analiza bezpieczeństwa

- Określenie alokacja wymagań bezpieczeństwa

- Planowanie i realizacja funkcji bezpieczeństwa

- Ocena i zatwierdzenie

- Eksploatacja, obsługa, naprawy

1. **Cechy zadania**

- czas zgłoszenia

- czas odpowiedzi

- czas wykonania

- dozwolony okres zakończenia

- okres powtarzania

1. **W jakim celu wprowadzono mechanizm podziału procedury obsługi przerwania na dwie części? Proszę opisać przynajmniej dwie możliwe realizacje tego mechanizmu.**

Wprowadzono ten mechanizm aby zmniejszyć niekorzystne wpływanie na siebie różnych przerwań. Dzielimy taką obsługę na zasadniczą procedurę obsługi przerwania oraz odroczoną procedurę obsługi przerwania. Przykładowe realizacje:

- „zadanka”

- „kolejki prac”

1. **Algorytm RMS**

- priorytety zadań są proporcjonalne do częstotliwości

- optymalny algorytm stało-priorytetowy

- gdy obciążenie jest zbyt duże to odrzucenie jest przewidywalne

- warunek dostateczny:

(sumaryczne obciążenie) <= n(2^1/n -1)

1. **Algorytm EDF**

- priorytety zadań związane z terminem zakończenia

- optymalny algorytm zmienno-priorytetowy

- gdy obciążenie jest zbyt duże to odrzuca losowo

- warunek dostateczny: jeśli obciążenie jest mniejsze niż 1 to zawsze uda się uszeregować tak aby wszystkie zadania się wykonały

1. **Zadania sporadyczne**

- nieznany z góry czas zgłoszenia

- możliwe zgłoszenia „stadne”

Warianty obsługi:

- wykonanie w tle: nie narusza terminu zadań cyklicznych, długi czas odpowiedzi na zadania sporadyczne

- wykonania na najwyższym priorytecie: krótki czas odpowiedzi zadań sporadycznych, możliwe opóźnienie zadań cyklicznych

1. **Serwer sporadyczny**

Jest to zadanie obsługujące zgłoszenia sporadyczne. Serwer otrzymuje budżet czasu s oraz okres odnowienia c:

- przez s serwer pracuje na wysokim priorytecie (zadania sporadyczne)

- później priorytet spada do niskiego l (zadania cykliczne)

- odzyskuje wysoki priorytet po okresie odnowienia c

1. **Redukcja ryzyka**

Funkcja bezpieczeństwa to funkcja przeznaczona do utrzymania bezpiecznego stanu instalacji w odniesieniu do konkretnego zdarzenia zagrażającego. Może być integralną częścią sterownika lub odrębnym urządzeniem.

1. **Poziom nienaruszalności bezpieczeństwa**

Prawdopodobieństwo, że system wykona wymagane funkcje bezpieczeństwa w zadanych warunkach i czasie

1. **Tryby pracy bezpieczeństwa**

- na rzadkie przywołanie: prawdopodobieństwo nie zadziałania w trakcie obsługi żądania

- na częste przywołanie: prawdopodobieństwo nie zadziałania na godzinę

1. **Szacowanie poziomu SIL**

Poziom SIL możemy szacować ze względu na: skutki awarii, częstotliwość przebywania ludzi w warunkach zagrożenia, nieuchronność zagrożenia, prawdopodobieństwo zdarzenia

1. **Algorytm dostępu do kabla Ethernet CSMA/CD**

Obserwujemy stan linii przesyłowej i nadajemy gdy wolna. Jednak mogą wystąpić kolizje i wtedy przerywamy nadawanie + losowa zwłoka i ponowna próba. Przez to algorytm nie jest deterministyczny i nie jest używany mimo bardzo dużej szybkości transmisji.

1. **Algorytm dostępu do kabla Ethernet poprzez odpytywanie**

Mamy jeden węzeł Master i kilka węzłów Slave. Master odpytuje a Slave odpowiada na żądanie. Slave nigdy nie nadaje bez zapytania węzła master.

1. **Algorytm dostępu do kabla Ethernet poprzez przekazywanie znacznika**

Podobnie jak w odpytywaniu ale mamy kilka węzłów Master i przechodnie prawo do nadawania „znacznik”.

1. **I2C**

Mamy dwie linie: zegar i linię danych. Może być wiele masterów. Master steruje zegarem, wyznacza adres slave, żądanie zapisu lub odczytu oraz nadaje dane. Gdy dwóch masterów nadaje w tym samym czasie rozwiązujemy arbitraż kolizji w ten sposób że na poziomie wybierania adresu slave tam gdzie pojawi się zero na magistrali to urządzenie przejmuje magistralę.

1. **Używamy systemu wbudowanego z procesorem ARM. Skąd system Linux uzyskuje informacje, jaki sterownik wybrać do obsługi konkretnego urządzenia sprzętowego? W jaki sposób dowiaduje się, jak uzyskać dostęp do jego rejestrów? Co i jak możemy zmienić w konfiguracji systemu, żeby umożliwić obsługę nowo dołączonego urządzenia?**

System Linux może uzyskać te informacje z opisów zawartych w drzewie urządzeń. Każde urządzenie posiada swój wpis w drzewie, tam możemy zawrzeć informacje o sterownikach które mogą takie urządzenie obsłużyć, o adresie i zakresie jego rejestrów. Aby umożliwić obsługę nowego urządzenia możemy dodać nowy wpis do drzewa.

1. **W systemie komputerowym z procesorem ARM, pewne urządzenie peryferyjne jest podłączone do magistrali systemowej, przy czym do pracy wymaga pewnego sygnału zegarowego, który nie jest zawsze aktywny. W jaki sposób system operacyjny powinien zostać poinformowany o potrzebie włączenia tego sygnału, oraz o adresach rejestrów tego urządzenia i wykorzystywanej przez nie linii przerwań?**

Wszystkie powyższe informacje znajdują się w opisie danego urządzenia w drzewie urządzeń np. adresy rejestrów są widoczne w opisie urządzenia w sekcji “reg”. Urządzenie jest podłączone do CPU a sterownik odpowiedzialny za obsługę danego urządzenia powinien powiadamiać OS o potrzebie wygenerownia przerwania.

1. **Jakie jest zastosowanie nakładek na drzewo urządzeń? Kiedy i jak je stosujemy? Odpowiedź może być oparta na konkretnych rozwiązaniach dla Raspberry Pi.**

Nakładki stosujemy aby przygotować drzewo urządzeń do obsługi urządzenia a konfigurowalnym wyposażeniu. Nakładki są dynamicznie ładowane. Aby w bezpieczny sposób je zastosować należy je modyfikować poprzez bootloader drzewa urządzeń dla kernela Linuxa, np. poprzez bootloader płytki RBi.

1. **Co powinniśmy zrobić, żeby w Linuksie zbudowanym za pomocą środowiska OpenWRT (np. w pobranej prekompilowanej wersji) zapewnić automatyczne uruchomienie karty dźwiękowej USB po jej podłączeniu do systemu?**

Możemy upewnić się, że w naszym środowisku działa demon systemowy eudev, który automatycznie ładuje i usuwa sterowniki oraz tworzy i usuwa pliki specjalne urządzeń.

1. **Na czym polega mapowanie rejestrów urządzenia w pamięć aplikacji? W przypadku jak podłączonych urządzeń daje się zastosować takie podejście? Jakie są zalety i wady takiego rozwiązania?**

Takie rozwiązanie możemy zastosować gdy urządzenie jest podłączone do magistrali systemowej. Mapowanie rejestrów urządzenia w pamięć aplikacji polega na przypisaniu adresów fizycznych rejestrów do adresów wirtualnych widzianych z poziomu aplikacji.

Zaleta:

* do takiego urządzenia możemy odwoływać się normalnie jak do pamięci np. za pośrednictwem struktur

Wada:

* Przy mapowaniu musi zostać wyłączone używanie pamięci cache i mechanizmy optymalizacji dostępu do pamięci co pogarsza wydajność

1. **W jaki sposób możemy za pomocą drzewa urządzeń włączać i wyłączać komponenty systemu? W jaki sposób możemy dodawać do drzewa urządzeń informacje o nowo podłączonych urządzeniach?**

Aby włączyć lub wyłączyć urządzenia możemy w DT odwołać się do etykiety danego urządzenia i zmienić jego status na “disabled”. Aby dodać nowo podłączone urządzenia można dodać nowy “node” do DT.

**Urządzenie peryferyjne po odebraniu pakietu danych zgłasza przerwanie i ustawia pewien bit w swoim rejestrze statusu. Jak możemy zapewnić możliwie najszybsze przetwarzanie odebranych danych:**

1. **w systemie z jednym rdzeniem CPU:**
   1. Ustaw najwyższy priorytet dla wątku obsługującego przerwanie (np. SCHED\_FIFO).
   2. Minimalizuj kod w obsłudze przerwania, delegując większe zadania do wątku użytkownika.
   3. Maskuj przerwania tylko w krytycznych momentach.
2. **w systemie z czterema rdzeniami CPU:**
   1. Przypisz przerwanie do jednego rdzenia (np. rdzenia 0).
   2. Ustaw najwyższy priorytet dla wątku obsługującego przerwanie.
   3. Dedykowane rdzenie 1, 2 i 3 do innych zadań.
   4. Użyj wątków na innych rdzeniach do przetwarzania danych.

**System komputerowy jest wyposażony w czterordzeniowy procesor. Dwa urządzenia peryferyjne wymagają bardzo szybkiej obsługi, przy czym potrzebę obsługi zgłaszają przez wygenerowanie przerwania (które można maskować) oraz przez zmianę wartości w swoim rejestrze statusu. Jak skonfigurować system, aby jak najlepiej spełnić wymagania tych dwóch urządzeń, a zarazem umożliwić realizację innych zadań?**Aby zoptymalizować obsługę dwóch urządzeń peryferyjnych na czterordzeniowym procesorze, przypisz przerwania z urządzenia 1 do rdzenia 0, a z urządzenia 2 do rdzenia 1. Użyj trybu SCHED\_FIFO z najwyższym priorytetem dla wątków obsługujących przerwania tych urządzeń. Dedykowane rdzenie 2 i 3 dla zadań ogólnych zapewnią, że te zadania nie będą konkurować z obsługą przerwań. Maskuj przerwania podczas krytycznych operacji, aby uniknąć zakłóceń.

