AGH

AKADEMIA GÓRNICZO-HUTNICZA
IM. STANISŁAWA STASZICA W KRAKOWIE

Systemy operacyjne
Wykład 11

Wersja 2024

dr inż. Marek Wilkus **http://home.agh.edu.pl/~mwilkus**
Wydział Inżynierii Metali i Informatyki Przemysłowej
AGH Kraków

Na podstawie programu opracowanego przez dr inż. Krzysztofa Wilka

Systemy czasu rzeczywistego

- Definicja IEEE:

  System czasu rzeczywistego (real time) to system, którego poprawność działania zależy nie tylko od poprawności logicznych rezultatów, lecz również od czasu, w jakim te rezultaty są osiągane (czasu reakcji).

Rys historyczny

- W pierwszych komputerach oprogramowanie miało pełną kontrolę bezpośrednio nad sprzętem - każdy program odpowiednio napisany był RTOS.
- Późniejsze systemy oferowały możliwości wywłaszczenia systemu przez oprogramowanie. Nie jest to najbezpieczniejsza możliwość, lecz umożliwia np. Zmianę systemu operacyjnego bez przeinicjowania komputera (MonkeyLinux). Programy, które musiały działać jako RT wykorzystywały tą możliwość.
- Wprowadzenie pełnej wielozadaniowości stworzyło potrzebę systemów RT oraz systemów, w których tylko niektóre zadania są RT.

Rys historyczny - wielozadaniowość

- RTOS musi używać wywłaszczania w celu umożliwienia działania procesu RT.
- Proces RT może uruchamiać się z b. wysokim priorytetem (w systemach typu Flex 9) lub informować system o swoim stanie, działając na poziomie jądra (adaptacje Uniksów).
- Systemy „Event-driven” - możliwe do zaimplementowania na mikrokontrolerach. Zdarzenie wyzwala przerwanie, wyzwalające proces RT.
  - Problem: „zapchanie” przerwaniami, patrz zawieszenie komputera podczas lądowania na Księżycu w misji Apollo 11

Rys historyczny

- Systemy „Time Sharing” - z podziałem czasu - umożliwiają płynne mieszanie zadań RT i pozostałych.
  - Problem: Wymagane jest częste zwolnienie mniej istotnych zadań przez zadanie RT.

Systemy czasu rzeczywistego (RTOS)

- Tryb przetwarzania w czasie rzeczywistym jest takim trybem, w którym programy przetwarzające dane napływające z zewnątrz są zawsze gotowe, a wynik ich działania jest dostępny nie później niż po zadanym czasie. Moment nadejścia kolejnych danych może być losowy (asynchroniczny) lub ściśle określony (synchroniczny).
- System czasu rzeczywistego jest systemem interaktywnym, który utrzymuje ciągły związek z asynchronicznym środowiskiem, np. środowiskiem, które zmienia się bez względu na system, w sposób niezależny.
- Oprogramowanie czasu rzeczywistego odnosi się do systemu lub trybu działania, w którym przetwarzanie jest przeprowadzane na bieżąco, w czasie wystąpienia zewnętrznego zdarzenia, w celu użycia rezultatów przetwarzania do kontrolowania lub monitorowania zewnętrznego procesu.

Systemy czasu rzeczywistego

- System czasu rzeczywistego odpowiada w sposób przewidywalny (w określonym czasie) na bodźce zewnętrzne napływające w sposób nieprzewidywalny.
- System komputerowy działa w czasie rzeczywistym, jeżeli wypracowane przez ten system decyzje są realizowane w tempie obsługiwanego procesu. Inaczej mówiąc, system działa w czasie rzeczywistym, jeżeli czas reakcji systemu jest niezauważalny przez proces (decyzja jest wypracowana we właściwym czasie) **

Wg:
Lal K., Rak T., Orkisz K : "RTLinux – system czasu rzeczywistego", HELION, 2003.
** - Plaza R., Wróbel E.: „Systemy czasu rzeczywistego", Wydawnictwo Naukowo – Techniczne, Warszawa 1988.

RTOS - przykłady

- Sekwencja awaryjnego wyłączania silnika rakietowego.
- System zbierania danych (np. pomiarów w procesie produkcyjnym).
- System analizatora danych, oscyloskopu cyfrowego.
- System kontrolny ABS w samochodzie.
- System dostarczania paliwa do silników samolotu.
- Odtwarzanie plików MPEG w stacjonarnych odtwarzaczach.
- Kontroler serwomechanizmu.
- Systemy podtrzymywania życia w urządzeniach medycznych

Czy RTOS muszą być szybkie?

- **Real time** oznacza nie "szybki", lecz "przewidywalny".
- **Gwarantowany pesymistyczny czas reakcji** nie oznacza szybkiego czasu reakcji, a jedynie czas reakcji z góry określony.
- System czasu rzeczywistego może wydawać się wolniejszy od "zwykłego" systemu operacyjnego. Wynika to z faktu, ze techniki stosowane do przyspieszania pracy systemu operacyjnego (pamięć podręczna, wielopotokowe procesory, etc.) wprowadzają element indeterminizmu. Indeterminizm jest niedopuszczalny w przypadku systemu czasu rzeczywistego, gdyż uniemożliwia zapewnienie przewidywalności systemu.

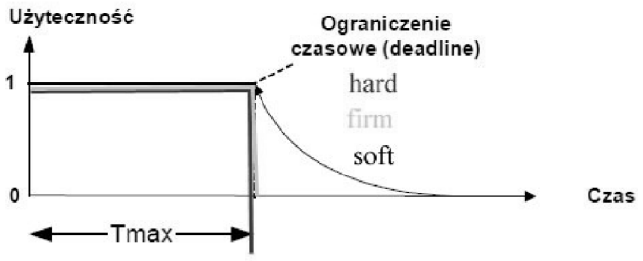
Podział RTOS

- Hard (rygorystyczne)
  - Gwarantują terminowe wypełnianie krytycznych zadań. Wymaga to ograniczenia wszystkich opóźnień w systemie.
  - Pamięć pomocnicza jest na ogół bardzo mała albo nie występuje wcale. Wszystkie dane są przechowywane w pamięci o krótkim czasie dostępu lub w pamięci, z której można je tylko pobierać (ROM).
  - Prawie nie spotyka się w systemach czasu rzeczywistego pamięci wirtualnej.
  - Dlatego rygorystyczne systemy czasu rzeczywistego pozostają w konflikcie z działaniem systemów z podziałem czasu i nie wolno ich ze sobą mieszać.

Podział RTOS

- Soft (łagodne)
  - Krytyczne zadanie do obsługi w czasie rzeczywistym otrzymuje pierwszeństwo przed innymi zadaniami i zachowuje je aż do swojego zakończenia.
  - Opóźnienia muszą być ograniczone - zadanie czasu rzeczywistego nie może w nieskończoność czekać na usługi jądra.
  - Łagodne wymagania dot. czasu rzeczywistego umożliwia godzenie ich z systemami innych rodzajów.
  - Zastosowanie w technikach multimedialnych, kreowaniu sztucznej rzeczywistości itd.
  - Znajdują one swoje miejsce wszędzie tam, gdzie istnieje potrzeba systemów o bardziej rozbudowanych możliwościach.
- Firm (mocne)
  - Wymagania pośrednie pomiędzy hard a soft,
  - Nie wykonanie zadania w terminie skutkuje nieprzydatnością wyników, ale nie zagraża katastrofą.

Użyteczność odpowiedzi



Proces RT w systemie standardowym

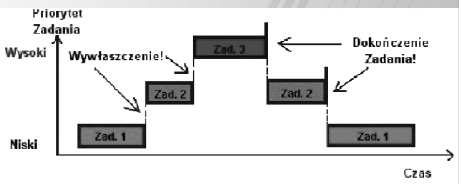
Opóźnienia (w sytuacji, gdy łatwo można było je zmierzyć)

Przykładowe czasy opóźnień dla różnych systemów operacyjnych (Pentium 100MHz)

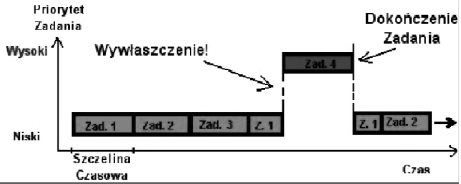
System	Tryb pracy	Opóźnienie
Windows 98/ 2000/ XP	in real time	100µs – 100ms
Linux	soft real-time	1ms
Linux IEEE 1003.1d	hard real-time	10µs – 100µs
Linux RT	hard real-time	1µs – 10µs
Jądro RTOS	hard real-time	1µs – 10µs

Szeregowanie zadań

- Priorytetowe:



- Z podziałem czasu:

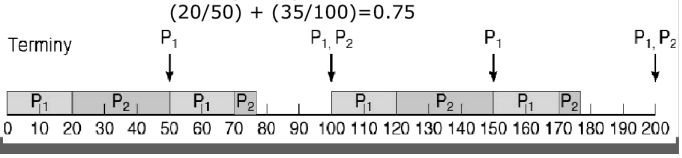


Planowanie procesów

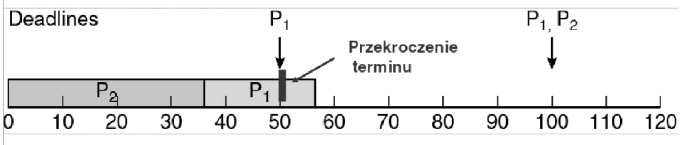
- Przyjmujemy periodyczny (okresowy) model procesów.
- Każdy proces może być opisany przez następujące parametry:
  - Okres **p** (period) tzn. czas pomiędzy kolejnymi zdarzeniami wymagającymi obsługi przez proces.
  - Termin **d** (deadline) w którym zdarzenie musi być obsłużone (od momentu zajścia zdarzenia).
  - Czas **t** (time) potrzebny procesowi na obsługę zdarzenia.
- Zachodzi relacja **0≤t≤d≤p**
- Stopień wykorzystania procesora jest równy **u=t/p**.
- Warunek konieczny wykonywalności szeregowania: suma stopni wykorzystania procesora **Σu≤1**.
- Proces oznajmia swoje parametry t,d,p planiście. Planista albo podejmuje się wykonania procesu gwarantując dotrzymania terminu albo odrzuca proces.

Rate-monotonic scheduling

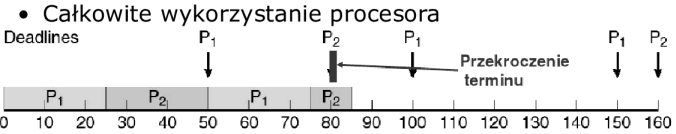
- Procesy są planowane na podstawie statycznego priorytetu równego częstotliwości zdarzeń 1/p.
- Proces o wyższym priorytecie wywłaszcza proces o niższym priorytecie.
- Przykład: dwa procesy:
  - P1: p=50, d=50, t=20 (P1 ma krótszy okres => większą częstotliwość i priorytet).
  - P2: p=100, d=100, t=35
  - Całkowite obciążenie procesora
    (20/50) + (35/100)=0.75



- Spośród klasy algorytmów z priorytetami statycznymi jest to algorytm optymalny, w takim sensie, że jeżeli nie dotrzymuje terminów, to żaden inny algorytm z tej klasy również nie dotrzyma terminów.
- Przykład: zakładamy, że proces P2 ma większy priorytet:



- Przykład, w którym dotrzymanie terminów nie jest możliwe:
  - P1: p=50, d=50, t=25 (wyższy priorytet)
  - P2: p=80, d=80, t=35.
- Całkowite wykorzystanie procesora



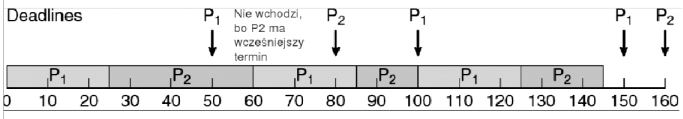
• W pesymistycznym przypadku algorytm nie gwarantuje dotrzymania terminów, gdy całkowite wykorzystanie procesora:

$$u \geq 2(2^{1/N}-1)$$

• Dla liczby procesów N=2 otrzymujemy u≈0.83

Algorytm "Najpierw najwcześniejszy termin" (EDF – Earliest deadline first)

- Przykład:
  - P1: p=50, d=50, t=25 (wyższy priorytet)
  - P2: p=80, d=80, t=35.
- Priorytet przypisywany dynamicznie:



RTOS - WYMAGANIA

- RTOS musi być wielowątkowy i wywłaszczalny.
- W momencie gdy OS nie jest oparty na deadlinach, musi istnieć pojęcie priorytetu wątku.
- OS musi wspierać mechanizm przewidywalnej synchronizacji wątków.
- Musi istnieć dziedziczenie priorytetów.
- Zachowanie OS powinno być znane i przewidywalne.

Dokładne oszacowanie parametrów czasowych

- Opóźnienie przerwań (czyli czas od wygenerowania przerwania do rozpoczęcia wykonywania zadania) - musi być zgodne z wymaganiami aplikacji, i musi być przewidywalne. Wartość ta zależy od liczby jednocześnie oczekujących przerwań.
- Dla każdego przerwania systemowego – maksymalny czas, jaki zajmujemy. Czas powinien być przewidywalny i niezależny od liczby obiektów w systemie.
- Maksymalny czas na jaki OS i sterowniki maskują przerwania.

Tworzenie RTOS

W niektórych przypadkach, aby uzyskać RTOS, podejmuje się próby modyfikacji lub wykorzystania istniejących systemów operacyjnych. Obserwuje się dwa główne podejścia do tej kwestii:

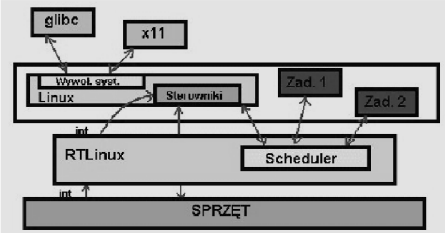
1) Próby balansowania pesymistycznego i średniego czasu reakcji, "robienie ogólno-zadaniowego systemu operacyjnego a`la real-time".
Próby optymalizowania dwóch, zasadniczo przeciwstawnych, parametrów rzadko prowadzą do olśniewających rezultatów, a w przypadku systemów operacyjnych prowadzą raczej do soft RTOSów.

Tworzenie RTOS – drugie podejście

2) Rozbicie systemu na dwa systemy operacyjne - real-time i "zwyczajny" (nie real-time).
Podejście to wydaje się być często skuteczne, czego przykładem jest choćby RTLinux. Niemniej jednak uważane jest za konserwatywne i potencjalnie ograniczające możliwości końcowego użytkownika.

- Windows mogą, przy użyciu specjalnego oprogramowania, być konwertowane do systemów real-time.
Niestety, jedyne co udaje się osiągnąć to soft real-time.

RTLINUX



RTLinux – procesy czasu rzeczywistego

- Procesy cz. rz. to programy zdefiniowane przez użytkownika, które wykonują się zgodnie z podanym harmonogramem (mogą być okresowe, zasypiać się na określony czas) i mają ścisłe wymagania czasowe.
- Procesy czasu rzeczywistego implementuje się jako ładowalne moduły jądra. Z punktu widzenia RTLinuksa procesy RT to wątki jądra RT.
- Scheduler RTLinuksa uważa, że jest tylko jeden prawdziwy proces, który ma wiele wątków. Jeden z tych wątków jest wybierany do wykonania.
- Linux jest tylko jednym z wątków, ma najniższy priorytet.

RTLinux – Procesy czasu rzeczywistego

- Procesy wykonują się w jednej przestrzeni adresowej, zatem przy zmianie kontekstu nie trzeba unieważniać rejestrów asocjacyjnych (TLB – translation look-aside buffers). Gdyby procesy wykonywały się we własnej przestrzeni adresowej, przy każdej zmianie kontekstu trzeba by unieważniać rejestry TLB, co przy częstym przełączaniu kontekstu daje duże obniżenie wydajności.
- Przy wywołaniach systemowych nie trzeba zmieniać poziomu uprzywilejowania.

RTLinux – Procesy czasu rzeczywistego

- Przełączanie kontekstu jest łatwiejsze -- przełącza się tylko kontekst sprzętowy: zachowuje się zawartość rejestrów na stosie i zmienia wskaźnik stosu tak, żeby wskazywał nowy stos nowego procesu. Przełączanie kontekstu jest programowe, a nie sprzętowe, bo sprzętowe przełączenie kontekstu na procesorach x86 jest powolne.
- Program i jego dane nie podlegają stronicowaniu. Nie mogą zatem zostać wysłane na dysk. Nie występują błędy braku strony, więc nie ma opóźnień z tym związanych.

RTLinux – Procesy czasu rzeczywistego

- Błąd w programie użytkownika, jakim jest proces RT, może spowodować załamanie się systemu. Pisanie programów RT wymagana takiego samego natężenia uwagi, co programowanie jądra systemu operacyjnego.
- Dodatkowym ograniczeniem nałożonym na procesy RT jest fakt, że ich zasoby są definiowane statycznie. W szczególności nie ma (w standardowym RTLinuksie) wsparcia dla dynamicznej alokacji pamięci.
- W nowszych wersjach RTLinuksa dostępny jest moduł mbuff, który umożliwia korzystanie z dynamicznie alokowanej pamięci.

QNX
kolejny "uniwersalny" UNIX?

- Został opracowany na początku lat 80 przez założycieli kanadyjskiej firmy Quantum Software System, Limited
- Jego początkowa nazwa QUNIX (Quick UNIX), ze względu na zbyt duże podobieństwo do nazwy UNIX, nie mogła przetrwać zbyt długo i w kilka miesięcy później, za sprawą firmy AT&T, musiała zostać zmieniona i przybrała znane do dzisiaj brzmienie: QNX.
- Pierwsza wersja, była przeznaczona na komputery klasy IBM PC oraz wymagała 64kB pamięci RAM i 180kB napęd dyskietek. Pomimo swojej prostoty, umożliwiała już uruchomienie kilku procesów jednocześnie.
- *"gdyby firma IBM wybrała system QNX dla mikrokomputera IBM PC, wprowadzenie na rynek modelu AT mogłoby zostać opóźnione, gdyż aplikacje uruchamiane pod systemem QNX w komputerach PC zachowują się tak, jakby zostały uruchomione pod systemem DOS w komputerze 386" - PCMag*

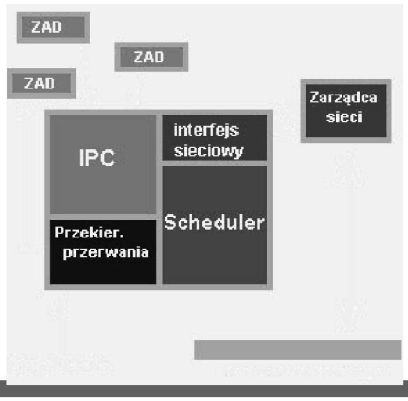
QNX - mikrojądro

- Rozmiar ok. 8kB (jądro UNIX > 700kB), stąd nazwa mikrojądra - Neutrino.
- To, co odróżnia ten system ten od rodziny UNIX, to przede wszystkim struktura modułowa oraz architektura oparta o przesyłanie komunikatów (klient - serwer).
- QNX daje możliwość zdeterminowania czasu reakcji na zdarzenia występujące w systemie.
- Dzięki rozbudowanym możliwościom definiowania priorytetów, QNX jest stosowany jako system służący do sterowania automatyką przemysłową, gdzie pewne zdarzenia są krytyczne (np. otwarcie zaworu bezpieczeństwa w zbiorniku kiedy gwałtownie wzrasta ciśnienie) i muszą być zawsze obsłużone na czas.

Budowa mikrojądra w QNX

- IPC - (ang. Interprocess communication) – obsługuje komunikację między procesami.
- Network Interface - przeźroczysta komunikacja pomiędzy procesami w obrębie sieci lokalnej.
- Hardware Interrupt Redirector – przechwytuje pojawiające się przerwania oraz przekazuje je do odpowiednich procesów obsługujących je. Część ta sama nie obsługuje przerwań!
- Realtime Scheduler - decyduje, który proces ma uzyskać dostęp do procesora w danej chwili (POSIX 1003.4 - dotyczy zagadnień czasu rzeczywistego).

Budowa mikrojądra QNX



Mikrojądro QNX

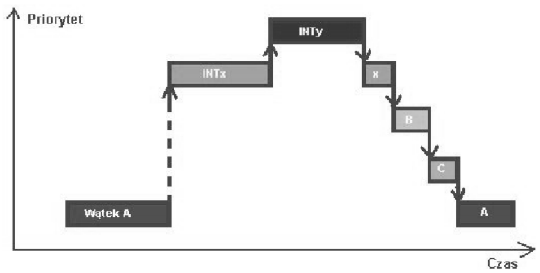
- Procesy zarządzające są traktowane w identyczny sposób jak procesy użytkowe. Można je ładować, uruchamiać, zawieszać oraz usuwać niezależnie od siebie. Poza tym czynności te mogą być wykonywane dynamicznie w czasie normalnej pracy systemu.
- W zależności od wymagań zewnętrznych dany proces zarządzający może zostać zainstalowany bądź usunięty. Wyjątkiem jest tutaj tzw. **Process Manager**, który musi być zawsze obecny w systemie.
- Procesy zarządzające od procesów użytkowych odróżniają priorytety. P.z. mają najwyższe. Dodatkowo zyskują poziomy uprzywilejowania, które zezwalają im na realizację niektórych instrukcji mikroprocesora.
- W systemie QNX zaimplementowano trzy poziomy uprzywilejowania:
  - poziom 0 - mikrojądro
  - poziom 1 - procesy zarządzające (np. Proc, Fsys itp.)
  - poziom 2 - nie jest aktualnie wykorzystywany
  - poziom 3 - procesy użytkowe

Szeregowanie zadań w QNX

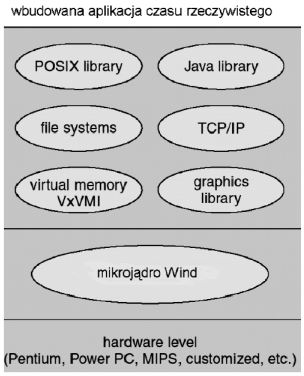
- Algorytmy szeregujące wybierają zadanie gotowe do wykonania i najważniejsze, czyli to o najwyższym priorytecie w danej chwili (priorytet 0÷31).
- Są trzy algorytmy szeregowania zadań (o tym samym priorytecie).

1) Kolejka FIFO. Zadanie wybrane do wykonywania, kontynuuje swoje działania aż samo odda sterowanie (np. wątek zostanie zablokowany lub proces wywoła funkcję systemową) bądź zostanie wywłaszczone przez zadanie o wyższym priorytecie.
2) Przydział czasu procesora (ang. time slice), który wynosi 50ms. Wówczas zadanie zostaje wywłaszczone, poza warunkami opisanymi we wcześniejszym algorytmie, również wtedy, kiedy wykorzysta przydzielony mu czas.
3) Adaptacyjny (ang. adaptive scheduling). Jeśli dane zadanie wykorzysta przydzielony czas procesora i nie zostanie zablokowane, to jego priorytet jest dekrementowany. Wtedy zazwyczaj następuje przełączenie kontekstu do innego zadania. Oryginalna wartość priorytetu jest natychmiast przywracana, kiedy zadanie przechodzi do stanu blokowania. Algorytm ten znajduje zastosowanie w sytuacjach, kiedy zadania wykonujące ogromne ilości obliczeń dzielą czas procesora z zadaniami interaktywnymi.

Obsługa przerwań w QNX



VxWorks 5.x



VxWorks 5.x

- Opcjonalne podsystemy: grafiki, systemy plików, Java, sieci TCP/IP, biblioteka Posix, pamięć wirtualna. Pozwala to na minimalizację zajętości (ang. footprint) pamięci.
- Mikrojądro (ang. microkernel) Wind
  - wywłaszczalne
  - gwarantowany czas reakcji na przerwania
- Zastosowanie: samochody, urządzenia konsumenckie, przełączniki sieciowe oraz Marsjańskie łaziki Spirit i Opportunity.

FreeRTOS

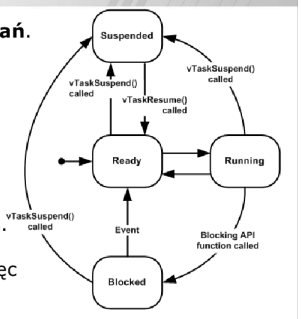
- Wielowątkowy, przeznaczony na systemy wbudowane i mikrokontrolery RTOS.
- Wspiera tryby oszczędzania energii, również działanie „uśpienia” gdy system aktywowany jest przerwaniem.
- Każdy proces czasu rzeczywistego może mieć priorytet.
- Możliwe jest decydowanie o postępowaniu z procesami RT.

Procesy RT w FreeRTOS

- Allocate only - obiekt jest ładowany do pamięci i tam pozostaje. Nie ma strat czasu wynikłych z jego przemieszczania lub przywracania.
- Allocate and free - konwencjonalna, lecz uproszczona możliwość ładowania i zwalniania pamięci np. przez zmienne.
- Blokowe „Allocate and free” - bardziej złożony algorytm likwidujący fragmentację kosztem czasu.
- Jak wyżej, lecz z możliwością dzielenia stosu programów na różne bloki pamięci
- Konwencjonalna alokacja i zwalnianie pamięci.

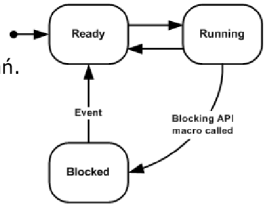
Zadanie w FreeRTOS

- Proces RT składa się z zestawu **zadań**.
- Każde zadanie uruchamia się we własnym kontekście, unikając zależności od stanu innych zadań lub systemu.
  - ...a więc każde zadanie ma własny stos - więcej zajętej pamięci.
- W urządzeniach o mniejszej ilości RAM rolę zadań pełnią **co-routines**. Są one ograniczone pod względem własnego stosu (dzielą wspólny) więc i wywołań innych funkcji, także systemowego API.
- Każde zadanie może znajdować się w konkretnych stanach.



Zadanie (Task) vs co-routine

- Co-routine może być gotowe, działać lub być zablokowana. Nie ma wstrzymywania jak w przypadku zadań.
- Co-routine nie ma własnego stosu. Stąd ograniczone jest wołanie API systemu i innych funkcji.
- Tak jak i zadania, co-routines mogą mieć priorytety, lecz są one zawsze poniżej priorytetów zadań!
  - → Co-routine wykona się gdy nie istnieje żadne zadanie o priorytecie wyższym niż zadanie „bezczynności”.
- Współdzielony stos nie gwarantuje stabilności zmiennych. Najprawdopodobniej zmienne alokowane w jednym uruchomieniu co-routine nie będą zachowane w następnym.



Dziękuję za uwagę