Systemy wbudowane - wykład do samodzielnego opracowania
Systemy czasu rzeczywistego

Przemek Błaśkiewicz

29 kwietnia 2020

Systemy czasu rzeczywistego

Popatrzmy na systemy (nie tylko sterowania), gdzie tzw. *timing* wydaje się ważną sprawą:

- sterowanie silnikiem rakietowym;
- system kontroli ABS;
- kontrola rdzenia reaktora;
- robot spawalniczy;
- system podtrzymywania życia pacjenta;
- przenośny odtwarzacz muzyki/telefon/aparat;

Może się zdawać, że w takich systemach ważna jest *szybkość* reakcji (przetwarzania danych). Jednak określenie „jak najszybciej" jest nieprecyzyjne, bo nie ustala żadnych sztywnych granic[1]. Dlatego w takich przypadkach mówi się o *gwarantowanym, ustalonym* czasie odpowiedzi[2].

[2] Podobnie jak w języku prawniczym „bez zbędnej zwłoki" jest przedmiotem oceny subiektywnej.

[2] Czyli urzędowo np. „w nieprzekraczalnym terminie 14 dni roboczych".

Definicja

System czasu rzeczywistego

Real-time system to system, którego poprawność działania zależy nie tylko od poprawności rezultatów (obliczeń, decyzji, akcji), ale również od czasu, kiedy zostaną one wypracowane (*czas reakcji*).

O ile poprawność obliczeń nie podlega dyskusji, to kwestia czasu reakcji implikuje dwa inne pojęcia, tj. gotowość i responsywność.

Gotowość

Program (system) jest zawsze gotowy na napływające dane

- Synchronicznie czy asynchronicznie
- Wiąże się to z np. wbudowaniem dodatkowych buforów danych, czyli np. rejestrów (♠ przypomnij sobie rejestry WE/WY w UART).
- Jeśli dana pojawi się w środowisku systemu, to zostanie przez niego przepracowana (bo to może być b. ważna dana).

Jest to realne, bo projektant systemu ma (powinien mieć...) pełną świadomość zdarzeń, jakie mogą w systemie zajść. Dysponuje szacunkami (doświadczeniem), wyliczeniami dotyczącymi takich zdarzeń oraz tak dobiera sprzęt, żeby nie „zatykał się" przy maksymalnym przewidywalnym obciążeniu.

Responsywność

Proces/system przetwarza dane w sposób przewidywalny, czyli:

- długość czasu przetwarzania jest znana;
- przetwarzanie odbywa się na bieżąco (maksymalne długości czasu oczekiwania na rozpoczęcie przetwarzania są określone);
- spełnione są dodatkowych założenia np. kolejności wykonywania.

Realizacja tych założeń jest możliwa na podstawie wiedzy o środowisku systemu, pracom symulacyjnym określającym efektywność (szybkość) przetwarzania danych przez dany sprzęt (prototypowanie) oraz realizacji odpowiednich algorytmów szeregowania zadań (*scheduling*).

Szybkość RT-OS

Pojęcie „*real-time*” jest często kojarzone z szybkością przetwarzania.

- Pojęcie „real-time” nie oznacza “szybki”. „Normalne” systemy operacyjne mogą być szybsze, bo:
  - korzystają z pamięci cache (♠ *locality of reference*);
  - mają wielopotokowe procesory;
  - mogą korzystać z akceleratorów (np. karty graficzne).

Klasyczne systemy tworzy się w ogólnym celu (np. systemy do pracy biurowej, systemy gamingowe, systemy serwerowe...). Ale pod tymi pojęciami kryje się różnorodność zastosowań (serwer CI z obciążeniem 100% vs. serwer WWW z obciążeniem 15%...), a także na serwerze można pociąć w Prince of Persia ;)

steigerdynamics.com

Szybkosć RT-OS (2)

ALE Windows po szeregu aktualizacji i instalacji sterowników wyraźnie zwalnia... **ALE** sposoby przyśpieszania działają dobrze, ale wprowadzają zależność czasu wykonania działań od okoliczności:

- Działanie cache jest złożone i trudno określić, kiedy przyspieszy, a kiedy opóźni (♠!!) działanie programu przy *dowolnej* sekwencji wejść.
- Wielopotokowość przyspiesza działanie, gdy ścieżka wykonania jest przewidywalna, ale gdy występują zdarzenia nagłe, to opóźnienia mogą być znaczące.
- Różny sprzęt to różne sterowniki, nie zawsze utrzymywana jest kompatybilność nawet w tej samej rodzinie urządzeń, co daje efekt nieprzenaszalności (kodu, sprzętu) między poszczególnymi egzemplarzami systemu.

RT-OS mają gwarantowany *pesymistyczny czas reakcji*.

“PC” vs. SW vs. RTOS

- Systemy PC są oceniane pod względem prędkości działania, przepustowości, wszechstronności *w średnim przypadku*.
- Dlatego są one zmiennicze pod względem sprzętu i oprogramowania, które jest na nim uruchomione co daje różnorodność konfiguracji, kontrolowanych przez nie procesów oraz sposobu, w jaki się pojawiają w systemie.
- System wbudowany jest często zamknięty pod względem sprzętu i oprogramowania, jak i środowiska, w którym pracuje.
- Można więc przedstawić lepsze mechanizmy planowania wykonania zadań, bo:
  - znamy charakter zadań (okresowość, wymagania obliczeniowe);
  - wiemy jak szybko trzeba je wykonać;
  - inaczej SW nie ma sensu (?).

Podział RTOS

Rygorystyczne (hard)

- gwarantują wypełnienie zadań krytycznych na czas;
- ograniczają opóźnienie wszystkich zadań w systemie;
- dane przechowywane w szybkiej pamięci lub pamięci ROM;
- brak wirtualizacji (bo wprowadza niedeterministyczne opóźnienia);
- *NIE* współpracują z systemami z podziałem czasu ♠!

Podział RTOS

Łagodne (soft)

- zadania krytyczne otrzymują i utrzymują pierwszeństwo przed innymi;
- opóźnienia są ograniczone - zadania mają skończony czas oczekiwania na wykonanie;

Mocne (firm)

- pośrednie między rygorystycznymi a łagodnymi
- niewykonanie zadania $\rightarrow$ wyniki nieprzydatne, ale OK

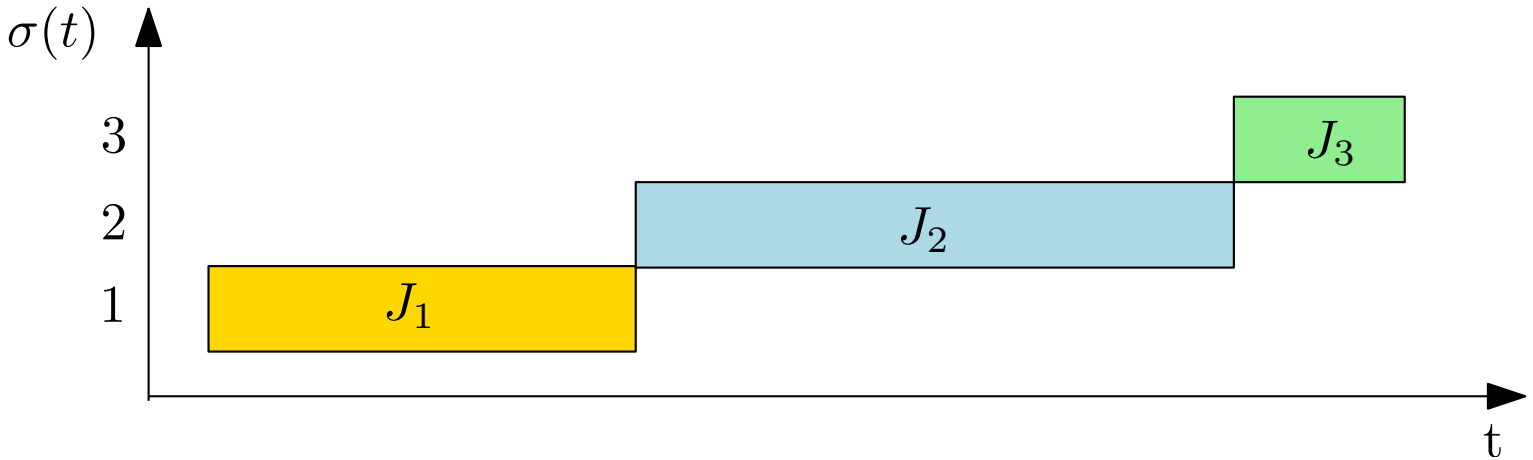
Plan

Plan (*schedule*) dla zbioru zadań $(J_1, J_2, \ldots, J_n)$ to pewna funkcja: $\sigma : R^+ \to \{0, \ldots, n\}$, taka, że:

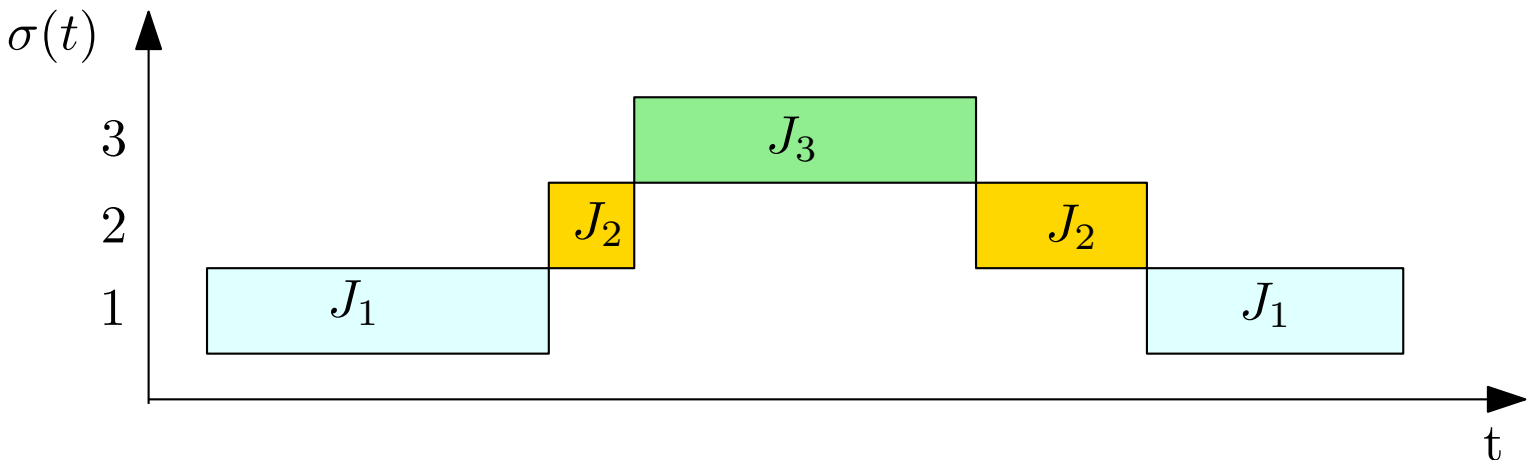
$$\forall t \in R^+, \exists t_1, t_2 \in R^+ : t \in [t_1, t_2), \forall t' \in [t_1, t_2) : \sigma(t) = \sigma(t').$$

Czyli, jeśli $\sigma(t) = j$ to wykonywane jest zadanie J_j, jeśli $\sigma(t) = 0$, to nie jest wykonywane żadne zadanie.

Różne planowanie



System bez wywłaszczeń. Zadania wykonywane bez przerwy do końca.



System wywłaszczeniowy. Zadanie może być przerwane, gdy pojawi się w systemie inne, ważniejsze (o wyższym priorytecie).

Cechy charakterystyczne zadań

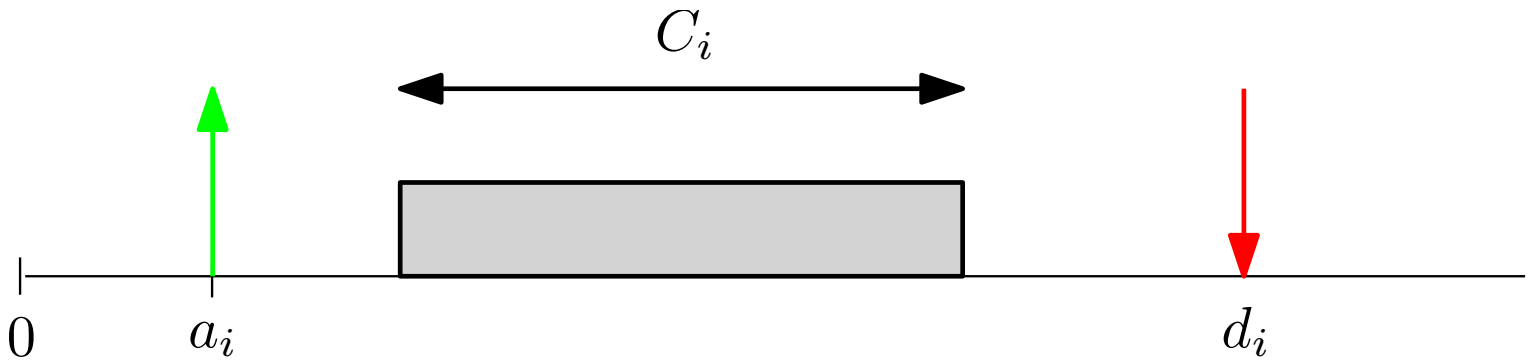
Aby móc skonstruować plan wykonania zadań, potrzebna jest wiedza na ich temat. Każde zadanie ma swoje właściwości a priori (czyli znane z analizy systemu):

pojawienie się w systemie – a_i;

czas obliczenia – C_i - wymagany czas (max) na wykonanie obliczenia;

czas zakończenia (deadline) – d_i - maksymalny czas, przed upływem którego zadanie musi zostać przetworzone;

czas luzu (slack time) – $X_i = d_i - a_i - C_i$, maksymalny czas opóźnienia wykonania zadania po jego nadejściu, nadal umożliwiający jego wykonanie;



Cechy charakterystyczne zadań

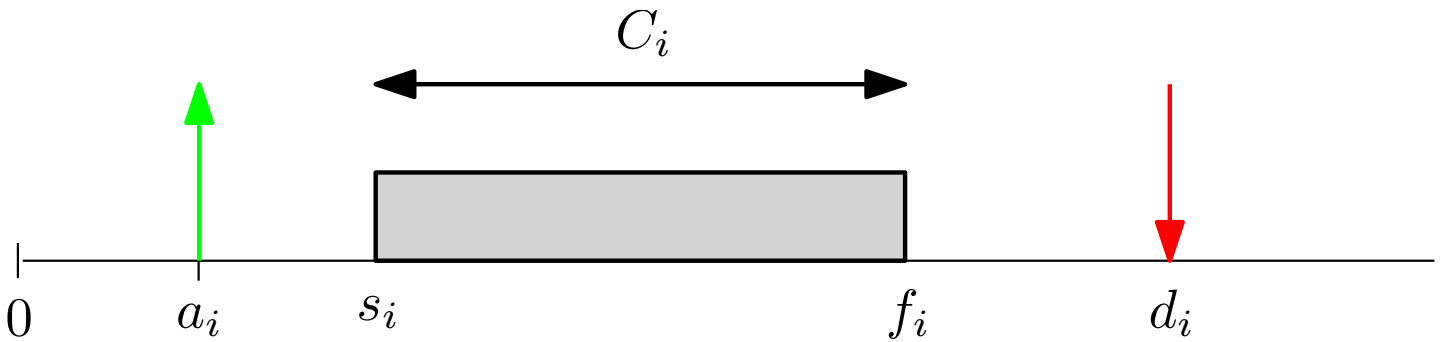
Zaplanowane zadanie otrzymuje dodatkowe właściwości:

start wykonania – s_i;

koniec wykonania – f_i;

opóźnienie - (lateness) – $L_i = f_i - d_i$;

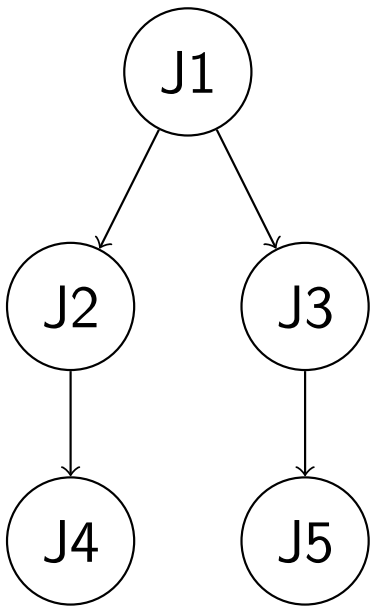
przekroczenie czasu - (exceeding time) – $E_i = \max(0, L_i)$;



Ograniczenia na zadania – ograniczenie kolejności

Pewne zadania muszą być wykonane przed innymi, co może się wiązać np. z potrzebą przygotowania zasobów przed ich wykorzystaniem (np. załadowanie rejestru wiadomości musi nastąpić zanim na tej wiadomości obliczony będzie CRC (albo zostanie uruchomione zadanie szyfrowania na osobnym procesorze).

$$J_1 \prec J_2, J_1 \prec J_3, J_2 \prec J_4, J_3 \prec J_5$$



Zadanie J_1 jest wymagane, by można było uruchomić zadania J_2 i J_3. Zadanie J_4 może wystartować dopiero po zakończeniu J_2 (a więc również po J_1), a J_5 musi czekać na zakończenie J_3. Planowanie takich zadań jest **NP-trudne**, ale istnieją heurystyki (nast. wykład).

Algorytmy planowania (1)

Realizowalność planu

(*feasibility*) – plan jest realizowalny, jeśli jego wykonanie powoduje że wszystkie zadania kończą się nie później niż w swoim terminie wykonania.

Zestaw zadań jest *planowalny* jeśli istnieje co najmniej jeden realizowalny plan ich wykonania.

Algorytmy planowania (2)

Istnieje szereg różnych algorytmów planowania zadań. Omówimy je na następnym wykładzie. Wprowadźmy tymczasem podział, aby zarysować skalę problemu:

- wywłaszczające i niewywłaszczające – zadania mogą być przerwane przez wykonanie innych zadań (lub nie, odpowiednio);
- statyczne i dynamiczne – decyzja planowania podjęta w momencie kompilacji (tworzenia sytemu), lub w trakcie działania, "na bieżąco";
- dla systemów jednoprocesorowych, wieloprocesorowych;
- optymalne (dowodliwe), heurystyczne;
- dla zadań okresowych (periodycznych) i asynchronicznych;

Lemat o wywłaszczaniu

A na koniec wprowadźmy lemat, który uprości znacznie (podzieli na pół) cały świat algorytmów planowania:

Jeśli czasy nadejścia zadań są synchroniczne, to mechanizm wywłaszczenia nie polepsza maksymalnego opóźnienia. Czyli jeśli istnieje algorytm wywłaszczeniowy o opóźnieniu L_{max} to istnieje również algorytm niewywłaszczeniowy o tym samym opóźnieniu dla tego samego zestawu zadań.

Dowód (szkic).

1. Załóżmy że istnieje plan wywłaszczający z opóźnieniem L_{max}.
2. Jeśli istnieje zadanie wywłaszczone (np. J_w) w czasie t i powraca w czasie t', to przesuń całość zadania J_w sprzed t na tuż przed t' (uniknięcie wywłaszczenia).
3. Nie pogorszy to opóźnienia J_w, co najwyżej zmniejszy opóźnienie innych zadań.

♠ Spróbuj przeprowadzić dowód. Narysuj plan wykonania zadań z wywłaszczeniem, następnie wykonaj punkt 2. powyższej procedury. Wykaż, że stwierdzenie z p. 3. jest silne.