4. Procesy – pojęcia podstawowe

4.1 Czym jest proces ?

Proces jest czymś innym niż program. Program jest zapisem algorytmu wraz ze strukturami danych na których algorytm ten operuje. Algorytm zapisany bywa zwykle w jednym z wielu języków programowania.

Program = algorytm + struktury danych

Program jest strukturą statyczną zapisaną na jakimś nośniku.
Proces jest wykonującym się programem.

Proces - wykonujący się program

Proces jest aktywną strukturą dynamiczną istniejącą tylko w środowisku działającego komputera.
Aby proces mógł się wykonywać potrzebne są co najmniej następujące zasoby:

- procesor
- pamięć operacyjna
- urządzenia wejścia / wyjścia

Rodzaje procesów:

1. Procesy transformacyjne – Procesy skończone które wykonują obliczenie czyli pobierają dane które mają przekształcić w wyniki. Kryterium poprawności: przekształcenie danych zgodnie ze specyfikacją w skończonym czasie

2. Procesy reaktywne – Wykonują się dowolnie długo (być może w nieskończoność) i ich celem jest interakcja z otoczeniem (wymiana danych). Kryterium poprawności: Prawidłowa interakcja z otoczeniem - czasowa i dotycząca przekształcania danych.

4.2 Podstawowe definicje współbieżności

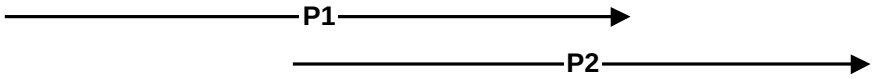
Procesy sekwencyjne (*ang. Sequential processes*)

Procesy są sekwencyjne jeżeli następny proces ze zbioru procesów rozpoczyna się po zakończeniu procesu poprzedniego.

Rys. 1 Procesy P1 i P2 wykonywane są sekwencyjnie

Procesy współbieżne (*ang. Concurrent processes*)

Dwa procesy są współbieżne jeżeli jeden z nich rozpoczyna się przed zakończeniem drugiego.



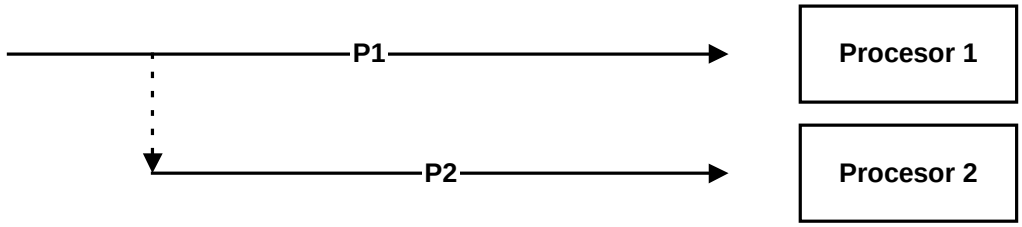
Rys. 2 Procesy P1 i P2 wykonywane są współbieżnie

Procesy równoległe (*ang.Paralell processes*)

Dwa procesy są równoległe jeżeli jeden z nich rozpoczyna się przed zakończeniem drugiego i wykonywane są jednocześnie na oddzielnych procesorach.

Rzeczywista równoległość – procesy wykonywane są na oddzielnych procesorach

Pozorna równoległość – procesy wykonywane są na jednym procesorze pracującym z podziałem czasu.



Rys. 3 Procesy P1 i P2 wykonywane są równolegle.

Rodzaje współbieżności

Współbieżność konkurencyjna – procesy nie współpracują ze sobą. Ich oddziaływanie polega tylko na konkurowaniu o dostęp do zasobów których potrzebują.

Współbieżność kooperacyjna – procesy współpracują ze sobą działając w ramach aplikacji jednej współbieżnej. Komunikują i synchronizują się ze sobą w celu wykonania pewnego zadania.

4.3 Skutki stosowania współbieżności

Korzyści wynikające z zastosowania współbieżności:

1. Polepszenie wykorzystania zasobów. Gdy jakiś proces czeka na niedostępny w danej chwili zasób, procesor może wykonywać inny proces.
2. Umożliwienie przetwarzania równoległego. Podział zadania na procesy umożliwia wykonywanie ich na oddzielnych maszynach. Prowadzi to do zrównoleglenia przetwarzania.
3. Ułatwienia projektowania i implementacji. Podział dużego zadanie na wiele mniejszych komunikujących się procesów prowadzi do dekompozycji problemu. Przez co ułatwia ich implementację, uruchamianie i testowanie przez wielu niezależnych programistów.

Problemy powstające przy implementacji aplikacji współbieżnych:

- problem sekcji krytycznej
- problem synchronizacji procesów
- problem zakleszczenia

Procesy tworzące aplikację nie działają w izolacji. Muszą jakoś ze sobą współpracować co prowadzi do:

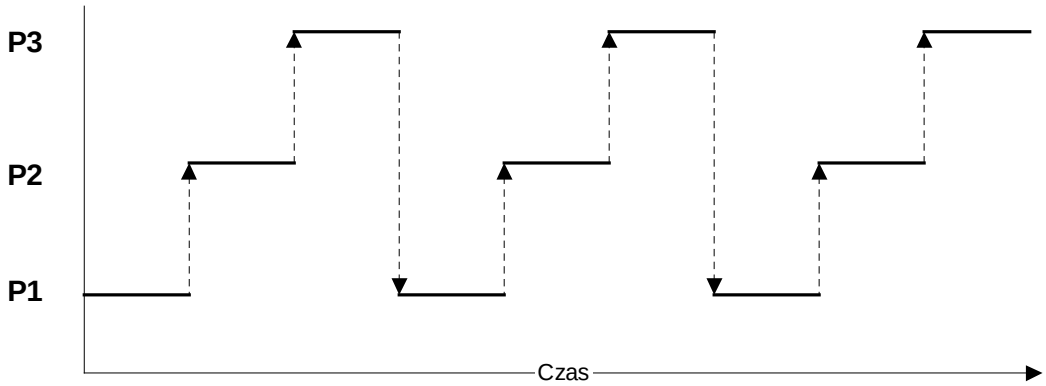
- Konieczności wzajemnej wymiany informacji - komunikacja międzyprocesowa.

- Zapewnienia określonej kolejności wykonania pewnych akcji - problem synchronizacji.

4.4 Przełączanie procesów

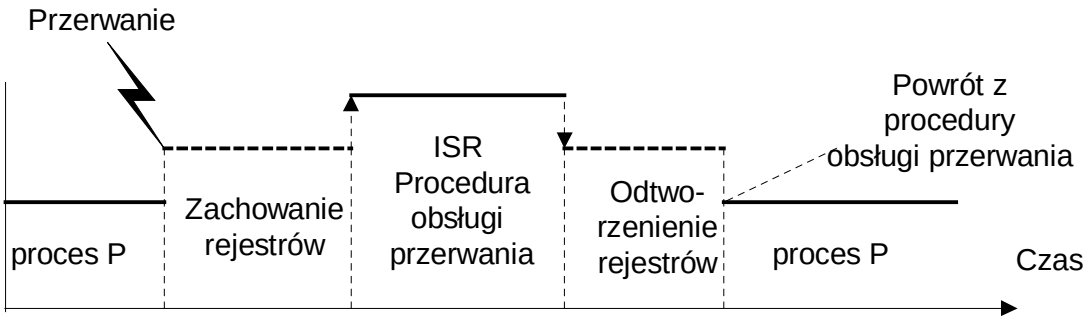
Współczesne komputery są na tyle wydajne że bez trudności mogą wykonywać wiele procesów które współdzielą czas procesora.

Mówimy że procesy wykonywane są w trybie przeplotu.



Procesy P1, P2, P3 wykonujące się w trybie przeplotu

Podstawowym mechanizmem umożliwiającym taki tryb pracy są przerwania.



Obsługa zdarzenia poprzez procedurę obsługi przerwania

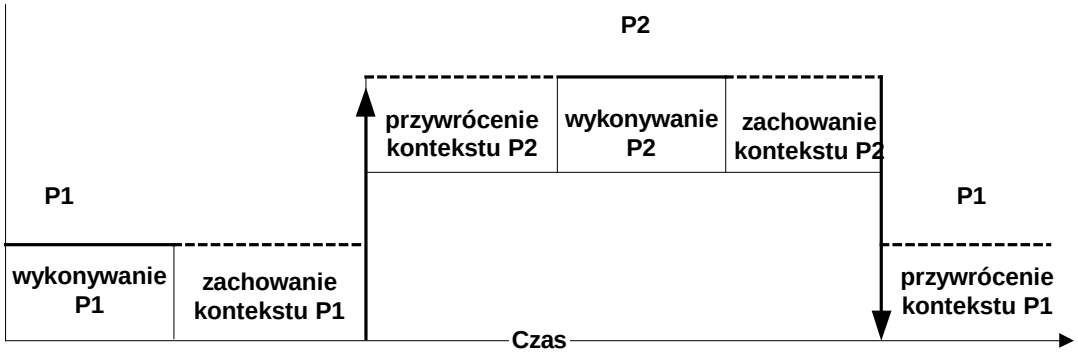
Obsługa przerwania - chwilowe wstrzymanie aktualnie wykonywanego procesu i wykonanie procedury przypisanej zdarzeniu powodującemu przerwanie po zakończeniu której następuje powrót do przerwanego procesu.

Przełączenia procesów wykonywane są przez system operacyjny.

Pojedynczy przełączenie składa się z trzech faz:

1. Zachowania kontekstu procesu dotychczas wykonywanego.
2. Podjęcie decyzji który z procesów wznowić.
3. Przywrócenie kontekstu nowego procesu.

Kontekst procesu – wszystkie informacje potrzebne do wznowienia zawieszonego wcześniej procesu.



Zachowanie kontekstu, wykonywanie i przywrócenie kontekstu procesu

Przełączenia mają miejsce w następujących sytuacjach:

1. Wystąpiło przerwanie zegarowe i system stwierdził że wykonywany proces wyczerpał już swój kwant czasu.
2. Wystąpiło przerwanie zewnętrzne (na przykład od kontrolera wejścia / wyjścia pewnego urządzenia) sygnalizujące zakończenie się zleconej wcześniej operacji.
3. Wystąpiło przerwanie wewnętrzne - proces bieżący wykonał pewną niedozwoloną operację polegającą na naruszeniu systemu ochrony zasobów procesora (na przykład wystąpiła próba odwołania do nieprzydzielonego segmentu pamięci
4. Wykonywany proces wykonał wywołanie systemowe zmieniające status gotowości przynajmniej jednego procesu. Może to być żądanie pewnych niedostępnych w tej chwili zasobów lub też ich zwolnienie.

UWAGA!
Przełączenia procesów występują w nie dających się przewidzieć momentach czasu. Stąd nie można czynić założeń że pewien ciąg instrukcji danego procesu nie zostanie przerwany.

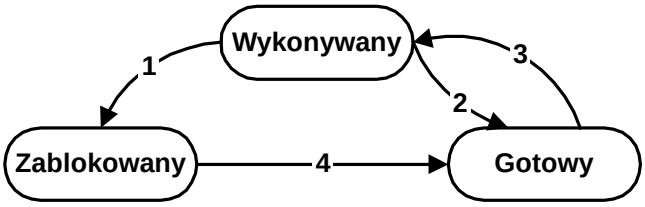
Szeregowanie wywłaszczające – taki sposób organizacji pracy procesów że proces bieżący może być w nie dającym się przewidzieć momencie czasu przełączony na inny proces (wywłaszczony).

Szeregowanie kooperacyjne – taki sposób organizacji pracy procesów że proces bieżący przełączany jest na inny tylko poprzez wykonanie określonych funkcji systemowych.

4.5 Kanoniczne stany procesów

Proces może być w jednym z trzech podstawowych stanów:

- Wykonywany (*ang. Running*),
- Gotowy (*ang. Ready*)
- Zablokowany (*ang. Blocked*).



Rys. 4 Przejścia pomiędzy podstawowymi stanami procesów

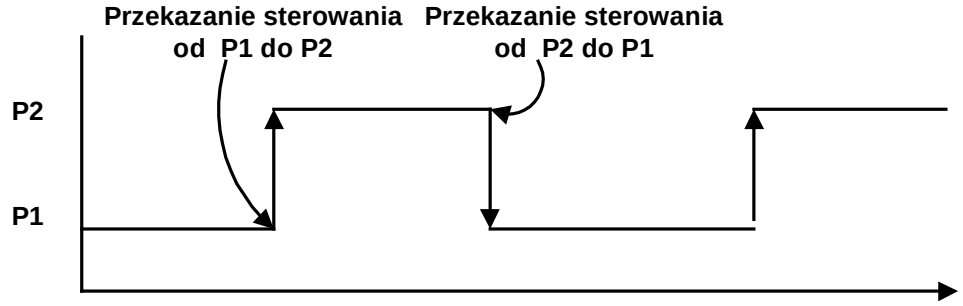
Pokazane na rysunku przejścia mają miejsca w następujących sytuacjach.

1. Proces żąda zasobu który nie jest dostępny.
2. Wystąpiło przerwanie (proces został wywłaszczony) lub też proces dobrowolnie zwolnił procesor.
3. Procedura szeregująca zdecydowała że ten proces ma być wykonywany.
4. Zasób którego brakowało do kontynuacji procesu stał się dostępny. Przejście zostało zainicjowane przez przerwanie od urządzenia wejścia / wyjścia lub też proces aktualnie wykonywany.

4.6 Elementarne operacje na procesach

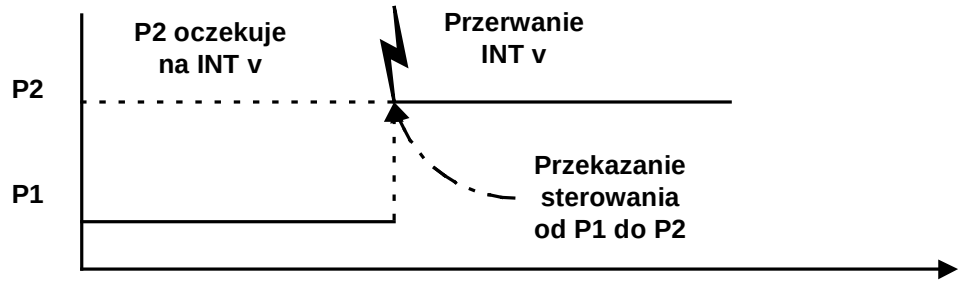
Utworzenie procesu
Operacja powoduje utworzenie deskryptora i alokację pamięci niezbędnej dla procesu.

Synchroniczne przekazanie sterowania od procesu bieżącego do procesu Pi.
Przekazanie sterowania od procesu bieżącego do procesu Pi następuje na żądanie procesu bieżącego. Kontekst procesu bieżącego zostaje zachowany a kontekst procesu Pi odtworzony.



Rys. 5 Synchroniczne przekazanie sterowania pomiędzy procesami

Asynchroniczne przekazanie sterowania od procesu bieżącego do procesu Pi.
Przekazanie sterowania ma miejsce gdy występuje przerwanie. Proces bieżący jest zawieszany a sterowanie przekazywane jest do procesu Pi zainstalowanego wcześniej do obsługi tego przerwania.



Rys. 6 Asynchroniczne przekazanie sterowania pomiędzy procesami

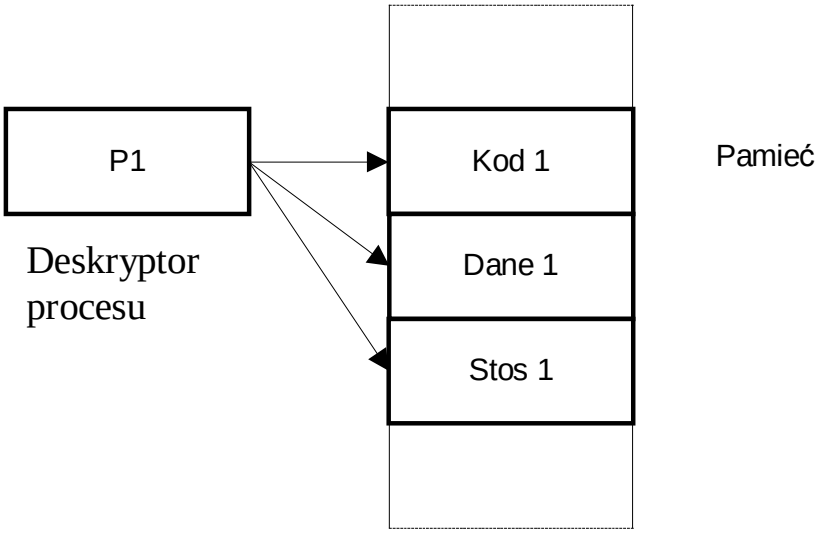
Zakończenie procesu
Przy zakończeniu procesów następuje zwrot zasobów i skasowanie deskryptora procesu.

4.7 Struktury danych używane przez proces

Proces utrzymuje w pamięci następujące struktury danych:

Segment kodu - (*ang. code segment*) segment pamięci w którym przechowywane są instrukcje kodu maszynowego procesu.

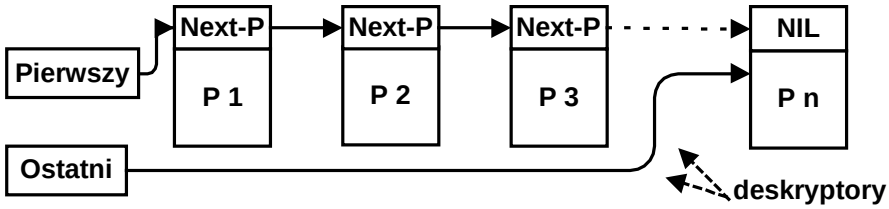
1. Segment danych - (*ang. data segment)* segment pamięci w którym przechowywane są statyczne dane procesu (statyczne znaczy tyle że dane te istnieją poprzez cały czas istnienia procesu)
2. Segment stosu - (*ang. stack segment)* segment pamięci w którym przechowywane są chwilowe dane procesu. Na stosie utrzymywane są zmienne lokalne procedur, parametry procedur i inne chwilowe dane. Przydział i zwalnianie pamięci na stosie odbywa się atomatycznie.
3. Segment sterty - (*ang. heap*) segment pamięci w którym przechowywane są chwilowe dane procesu jawnie przydzielane i zwalniane przez programistę.
4. Deskryptor procesu - (*ang. process descriptor*) rekord w którym system operacyjny utrzymuje wszystkie informacje niezbędne do zarządzania procesem.



Struktury danych procesu

Deskryptor procesu

System operacyjny musi prowadzić administrację procesami.
Procesy są tworzone, wykonywane, wznawiane, zawieszane i kończone.
Muszą być utrzymywane struktury danych zawierające wszystkie informacje ku temu niezbędne.



Rys. 7 Kolejka deskryptorów procesów

Zawartość deskryptora:

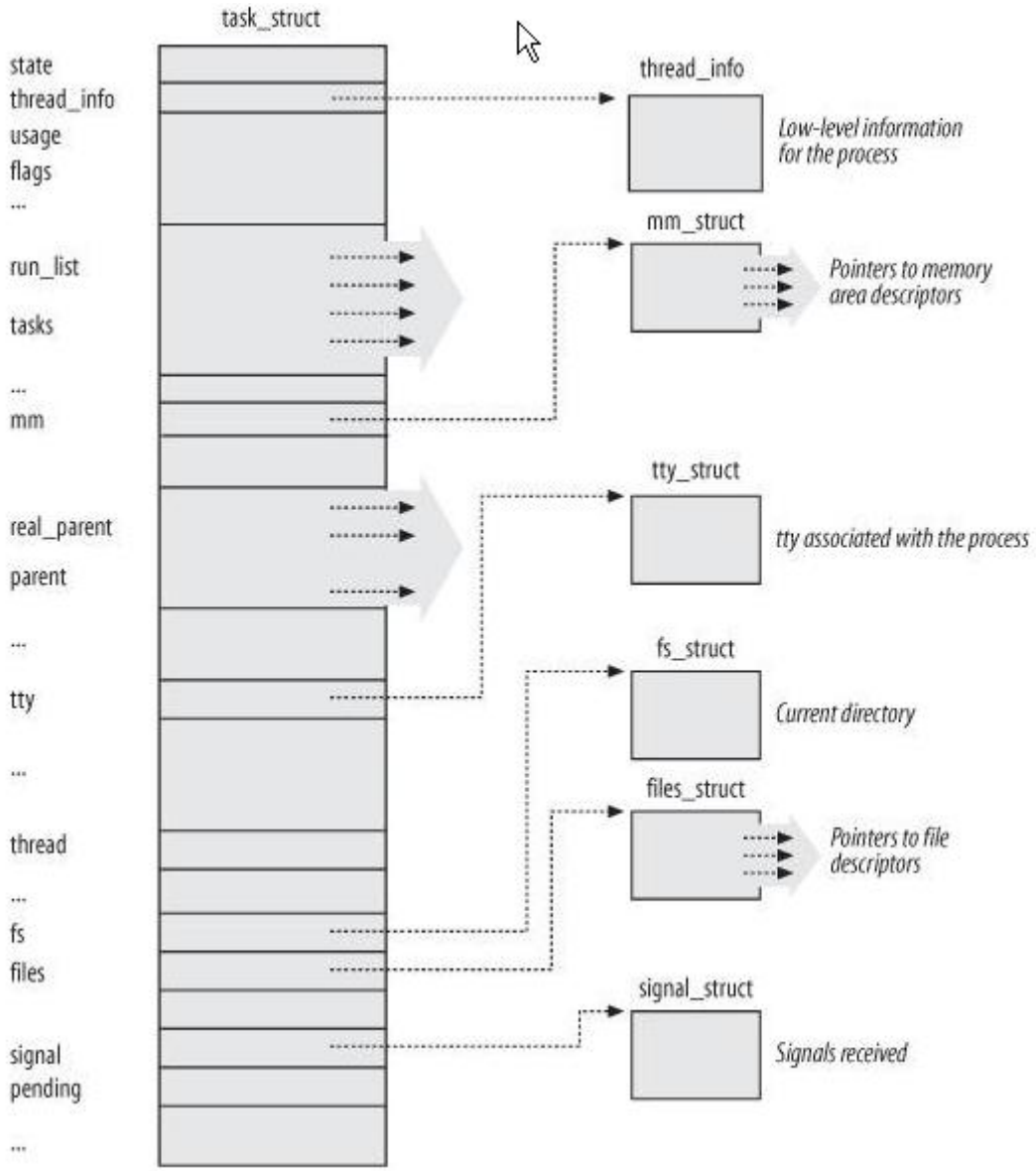
- Identyfikator procesu – PID (*ang. Process Identifier*)
- Bieżący stan procesu (wykonywany, gotowy, zablokowany, itd. …)
- Wskaźniki do poprzedniego i następnego deskryptora w kolejce wszystkich deskryptorów.
- Wskaźniki do poprzedniego i następnego procesu w danej kolejce (procesów gotowych, zablokowanych,itd…).
- Informacje dla procedur szeregowania (priorytet procesu, typ szeregowania).
- Informacje dotyczące obsługi sygnałów (sygnały dostarczone, zablokowane,….).
- Informacje na temat hierarchii procesów (proces macierzysty, potomne, itd…).
- Kontekst sprzętowy procesu (rejestry procesora).
- Informacje rozliczeniowe o czasie procesora zużytym przez proces.
- Nazwa pliku z którego utworzono proces.
- Informacje uwierzytelniające jak rzeczywisty i efektywny identyfikator użytkownika i grupy (UID, GID, EUID, EGID).

Zarządzanie pamięcią:

- Rozmiar segmentu kodu, danych, stosu.
- Położenie segmentu kodu, danych, stosu.
- Informacje o stronach zajmowanych przez proces.

Zarządzanie plikami:

- Katalog bieżący.
- Katalog macierzysty.
- Informacja o terminalu sterującym.
- Wzorzec tworzenia nowych plików (UMASK)
- Wskaźnik na tablicę deskryptorów otwartych plików.



Understanding the Linux Kernel, 3rd Edition

By Daniel P. Bovet, Marco Cesati