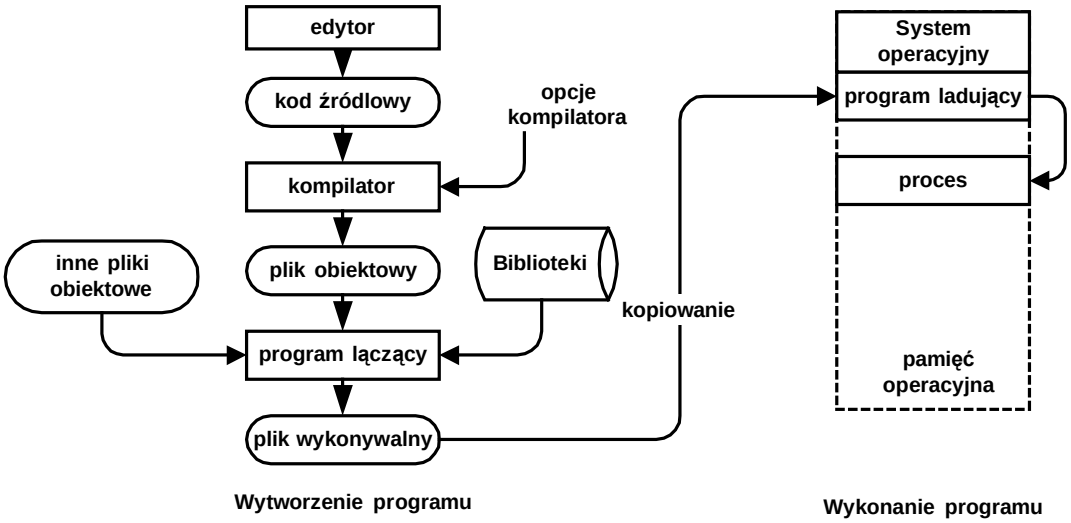
Procesy – pojęcia podstawowe

1.1 Jak kod źródłowy przekształca się w proces

W języku wysokiego poziomu tworzy się tak zwany kod źródłowy który po zapisaniu będzie plikiem z programem źródłowym. Plik źródłowy przetwarzany jest następnie przez kompilator do tak zwanego programu wykonywalnego.



Rys. 0-1 Przebieg procesu wytworzenia i wykonania programu

Przetworzenie kodu źródłowego w wykonywany proces odbywa się kilku etapach. Najważniejsze z nich to:

- kompilacja,
- łączenie
- ładowanie programu.

Celem kompilacji jest transformacja kodu źródłowego będącego zapisem algorytmu w języku wysokiego poziomu (który nie może być wykonany przez procesor) na kod maszynowy danego procesora.

Kompilacja przebiega w kilku etapach i prowadzi ona do wytworzenia tak zwanego pliku obiektowego.

Typowy plik obiektowy składa się z takich części jak:

- nagłówek,
- kod maszynowy,
- dane,
- tablica symboli,
- informacje o relokacji,
- informacje dla programu uruchomieniowego.

Na etapie kompilacji nie sposób określić pod jaki adres w pamięci należy załadować utworzony program, gdyż kompilator nie posiada informacji o stanie pamięci procesora w chwili wykonania programu. Stąd pliki obiektowe i wykonywalne zawierają tak zwaną tablicę relokacji (ang. *relocation table*).

Tablica symboli składa się z pozycji, z których każda zawiera wskaźnik do adresu w kodzie obiektowym, który musi być zmodyfikowany w procesie ładowania programu do pamięci operacyjnej.

W systemie Linux plik obiektowy jak i wykonywalny tworzony jest w tak zwanym formacie ELF (ang. *Executable and Linkable Format*). Informacje o plikach w formacie ELF uzyskać można za pomocą narzędzi Linuksowych takich jak **`readelf`** i **`objdump`** .

```
$objdump –x hello.o
hello.o:     file format elf32-i386
architecture: i386, flags 0x00000011:
HAS_RELOC, HAS_SYMS
start address 0x00000000
Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         0000001c  00000000  00000000  00000034  2**2
  1 .data         00000000  00000000  00000000  00000050  2**2
  2 .bss          00000000  00000000  00000000  00000050  2**2
  3 .rodata       0000000d  00000000  00000000  00000050  2**0
  4 .comment      0000001d  00000000  00000000  0000005d  2**0
  5 .note.GNU-stack 00000000  00000000  00000000  0000007a  2**0
…
SYMBOL TABLE:
…
RELOCATION RECORDS FOR [.text]:
OFFSET   TYPE              VALUE
…
```

Przykład 0-1 Ilustracja działania programu objdump dla pliku obiektowego

Z powyższego przykładu widać jakie informacje zawiera plik obiektowy. Są to nagłówki, segmenty, tablica symboli i dane do relokacji.

Ważniejsze segmenty to:

- text – segment kodu, zawiera instrukcje
- data – segment danych zainicjowanych, dane którym nadano wartości początkowe
- bss – segment danych nie zainicjowanych, dane którym nie nadano wartości początkowych
- rodata – segment danych zawierających stałe (tylko do odczytu)
- comment – komentarze
- note.GNU-stack – informacja że potrzebny będzie stos

Wiele informacji o pliku wykonywalnym można uzyskać za pomocą programów **`file`**, **`size`**, **`readelf`**. Polecenie **`size`** pozwala na uzyskanie informacji o rozmiarach.

```
$size hello
text    data     bss     dec     hex filename
916      264       8    1188     4a4 hello
```

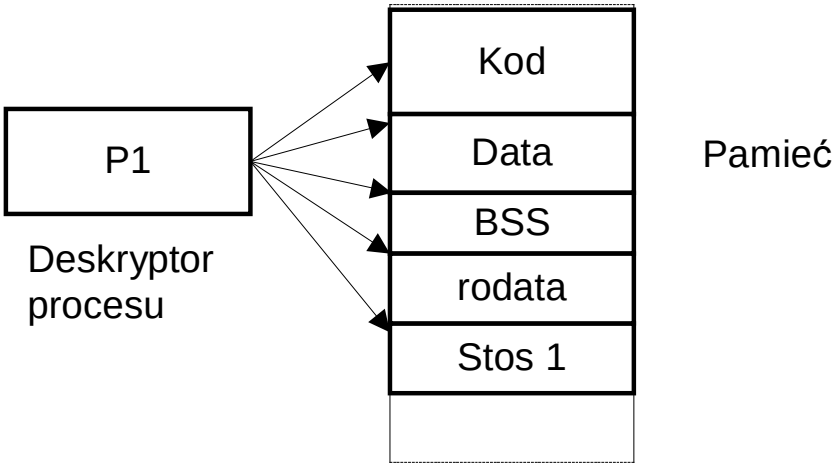
Przykład 0-2 Ilustracja działania programu size dla pliku programu hello

Program	Opis	Przykład
`objdump`	Podaje zawartość plików obiektowych	**`objdump –x hello.o`**
`readelf`	Odczyt pliku w formacie ELF	**`readelf –a hello`**
`size`	Podaje ile pamięci zajmuje proces	**`size hello`**
`file`	Podaje typ pliku	**`file hello`**

Tabela 0-1 Zestawienie narzędzi do analizy plików programowych

1.2 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. 0-2 Struktury danych procesu

Zawartość deskryptora:

Tożsamość procesu:

- Identyfikator procesu – PID procesu, grupa procesów
- Uwierzytelnienia – identyfikator użytkownika i grupy
- Indywidualność (ang. *personality*) – z cechy tej korzystają biblioteki emulacji

Środowisko procesu

Na środowisko procesu składają się dwa wektory:

- Wektor argumentów – wykaz argumentów wywołania programu
- Wektor środowiska – zestaw par NAZWA=WARTOŚĆ

Środowisko procesu dziedziczone jest z procesu macierzystego.

Kontekst procesu

Kontekst procesu zawiera informacje:

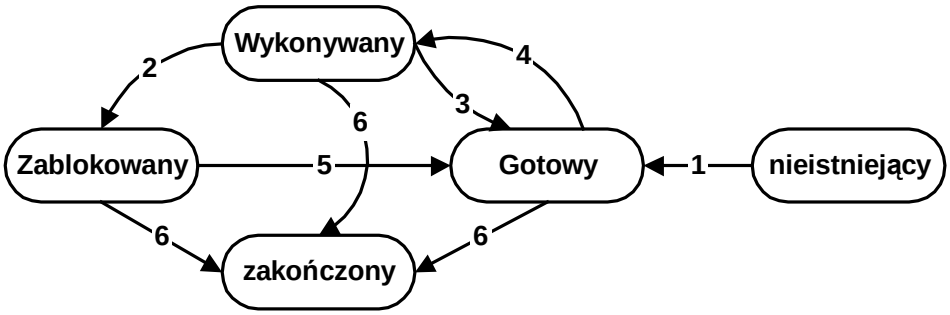
- Kontekst planowania: rejestry, priorytet bieżący, nie obsłużone sygnały

- Informacje rozliczeniowe: skumulowany czas procesora
- Tablica plików: zawiera wskaźniki do otwartych plików utrzymywanej przez jądro.
- Tablica obsługi sygnałów: tablica specyfikująca akcje mające być wykonane przy nadejściu sygnałów
- Kontekst pamięci wirtualnej: opisuje zawartość przestrzeni adresowej procesu

1.1 Kanoniczne stany procesów

Proces może być w jednym z trzech podstawowych stanów:

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



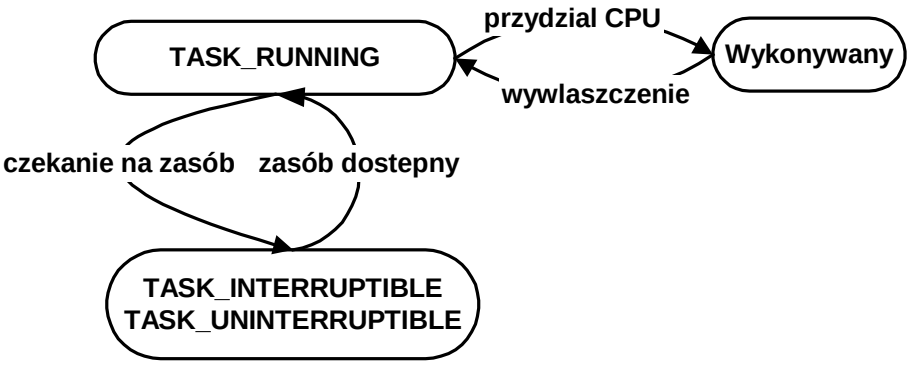
Rys. 1 Przejścia pomiędzy podstawowymi stanami procesów

Pokazane na rysunku przejścia mają miejsca w następujących sytuacjach.

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

1.3 Stany procesów w Linuxie

- TASK_RUNNING, - proces na liście procesów gotowych “Ready list”
- TASK_INTERRUPTIBLE, - proces czeka na sygnał lub zasób – może być wznowiony sygnałem lub przerwaniem
- TASK_UNINTERRUPTIBLE, - proces na zasób – może być wznowiony wprost
- TASK_ZOMBIE – proces bez procesu macierzystego
- TASK_STOPPED – proces debugowany

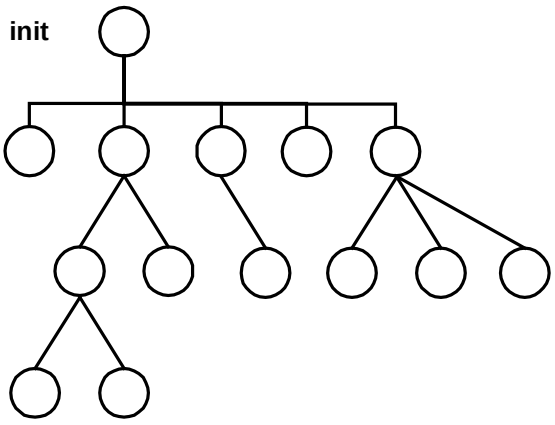


Rys. 2 Przejścia pomiędzy podstawowymi stanami procesów w systemie Linux

1.2 Drzewo procesów

Procesy tworzą drzewo

- Każdy proces ma dokładnie jeden proces macierzysty
- Proces może mieć wiele procesów potomnych



Rysunek 0-1 Drzewo procesów