wątki

WĄTEK (proces lekki) NT W'95

- pdst. jednostka wykorzystania procesora

  własne: LR, rejestry, stos

  wspólne: sekcja kodu, danych, otwarte pliki, sygnały
- przełączanie procesora między wątkami - tańsze od przełączania kontekstu między procesami ciężkimi
- procesy - reprezentują wykonywanie poszczególnych programów
- wątki - reprezentują oddzielne, współbieżne konteksty w ramach jednego procesu
- procesy - niezależne przestrzenie adresowe
- wątki - ta sama przestrzeń adresowa

Wątki w Linux'ie

- Ta sama wewnętrzna reprezentacja jak dla procesu
- funkcja **clone -** tworzy nowy proces z odrębną tożsamością, dzielący struktury procesu rodzica
- struktura danych procesu - zawiera wskaźniki do:

  kontekstu systemu plików

  tablicy deskryptorów plików

  tablicy obsługi sygnałów

  kontekstu pamięci wirtualnej
- **clone** - nowa tożsamość i kontekst planowania, pozostałe konteksty mogą być wspólne lub skopiowane (**fork -** szczególnym przypadkiem **clone**)

- Wrażenie jednoczesnego wykonywania
- Szeregowanie asynchroniczne
- Istnieją wewnątrz procesów
- Program ->Proces -> wątek wykonujący sekwencyjnie program -> dodatkowe wątki (ten sam program, ten sam proces)
- Proces nie oddziałuje na proces rodzica – kopiowanie pamięci wirtualnej, deskryptorów plików, innych zasobów
- Wątki – współdzielą pamięć, deskryptory plików (np. exec – kończy wszystkie wątki)

GNU/Linux

- Deklaracje funkcji obsługi wątków - <pthread.h>
- Dołączenie biblioteki libpthread -lpthread

- IDwątku - typ pthread_t
- Tworzenie wątku – funkcja pthread_create
  - Parametry:
    - wskaźnik do IDwatku
    - wskaźnik atrybutu (NULL - domyślne atrybuty)
    - wskaźnik funkcji wątku
    - argument – typu void*
  - Zwraca - wartość void*
- Kończenie wątku
  - funkcja wątku kończy działanie
  - funkcja pthread_exit
  - funkcja pthread_cancel – anulowanie
    - Stany anulowalności wątku:
      - asynchronicznie anulowalny
      - synchronicznie anulowalny – można go przerwać w punktach anulowania; domyślnie
      - nie anulowalny

```
#include <pthread.h>
#include <stdio.h>

void* print_1(void* nic)
{
  while (1)
        fputc('1',stderr);
  return NULL;
}

main()
{
  pthread_t idwatek;
  pthread_create(&idwatek, NULL, &print_1, NULL);
  while(1)
        fputc('2', stderr);
  return 0;
}
```

Przekazywanie danych do wątków

- przez argument
- wskaźnik do struktury zawierającej dane
- możliwość wykonywania tego samego kodu dla różnych danych przez wątki

- czekanie na zakończenie wątku (pthread_join)
- ID_watku – funkcja pthread_self
- atrybuty wątku
  - stan odłączenia
    - dołączalny *joinable*

      nie jest czyszczony po zakończeniu działania (pthread_join)
    - odłączony *detached*

      czyszczony po zakończeniu

- Wątki – ta sama przestrzeń adresowa
- Każdy wątek ma własny stos wywołań
  - każdy wywoływany podprogram w każdym wątku ma własne zmienne lokalne przechowywane na stosie wątku
- Obszar danych własnych wątku:
  - powielanie zmiennych dla wszystkich wątków
  - każdy wątek ma swoją kopię
  - tworzenie – funkcja pthread_key_create
  - ustawianie – funkcja pthread_setspecific
  - pobranie – funkcja pthread_getspecific
  - procedury czyszczące

SYNCHRONIZACJA WĄTKÓW

- eliminacja sytuacji wyścigu – niepodzielne wykonywanie działań
  - Muteksy (*MUTual EXclusion locks*)
    - blokada, którą w danej chwili może zamknąć tylko jeden wątek
    - muteks jest odblokowywany przez ten sam wątek
    - próba zablokowania zablokowanego muteksa blokuje wątek
    - po odblokowaniu jeden z czekających - przypadkowy wątek jest wznowiony – może zablokować muteks
  - możliwość zakleszczenia
    - Dwukrotne zablokowanie muteksa przez jeden watek
      - –Szybki muteks – deadlock; opcjonalnie
      - –RECOURCIVE_NP - rekurencyjny muteks – zlicza blokady / nie wystąpi dedlock
      - –ERRORHECK_NP - nie można dwukrotnie zablokować muteksa - błąd
      - NP – GNU/Linuks; nieprzenośne

SYNCHRONIZACJA WĄTKÓW muteksy

- **testowanie muteksów bez blokowania**
  - Aby uniknąć czekania na zablokowanym muteksie
  - funkcja pthread_mutex_trylock
  - jeśli muteks nie jest zablokowany – blokuje go
  - jeśli jest – zwraca kod błędu EBUSY

SYNCHRONIZACJA WĄTKÓW semafory

- zadanie: wątki przetwarzają zadania z kolejki
- rozwiązanie z muteksami: wątki pobierają zadania; jeśli kolejka pusta – kończą się
- problem: kolejka opróżni się chwilowo; po nadejściu nowych zadań – brak wątków
- potrzebny mechanizm blokujący wątki przy pustej kolejce zadań
- SEMAFOR
  - –funkcje:
    - sem_init
    - sem_wait
    - sem_post
    - sem_trywait
    - sem_destroy
    - sem_getvalue

SYNCHRONIZACJA WĄTKÓW
zmienna warunku *condition variable*

- wątek działa w nieskończonej pętli sterowanej za pomocą flagi
- gdy flaga nie ustawiona – wstrzymanie

- bez aktywnego oczekiwania
- zmienna warunku musi być ustawiona przez inny wątek po tym jak dany wątek rozpoczął czekanie na nią
- w przeciwnym razie – sygnał utracony
- aby nie dopuścić do sytuacji wyścigu
  - (wątek wywłaszczony po sprawdzeniu flagi)
    używane wspólnie z muteksami

```
blokuj muteks
testuj flagę
if flaga ustawiona then
      zwolnij muteks; kontynuuj pracę
else
     niepodzielnie zwolnij muteks i czekaj na
zmianę warunku
```

Implementacja wątków w GNU/Linuks

- wątki realizowane jako procesy (getpid)
- nie otrzymują kopii, lecz współdzielą zasoby

Sygnały

- sygnały wysyłane z zewnątrz programu odbierane zwykle przez główny wątek
- wątki mogą wysyłać sygnały do siebie – pthread_kill

PROCESY A WĄTKI

clone fork, pthread_create

- pozwala określać, które zasoby mają być współdzielone

	Procesy	wątki
Program	Różne	Ten sam
Przestrzeń adresowa	Kopia	Wspólna – możliwość uszkodzenia
Koszty	Większe - gdy pamięć zmieniana	
Współdzielenie	IPC	Podobne zadania