# Systemy operacyjne

Laboratorium 8

Mateusz Małek
5 maja 2017

# Laboratorium 8

Wątki - podstawy

# Zanim na poważnie...

*A programmer had a problem. He thought to himself, "I know, I'll solve it with threads!". has Now problems. two he*

# Co wątki współdzielą?

- PID (numer procesu)
- PPID (numer procesu macierzystego)
- PGID (numer grupy procesów)
- SID (identyfikator sesji)
- UID i GID (identyfikatory użytkownika)
- otwarte deskryptory plików
- założone rygle (locki)
- **procedury obsługi sygnałów**
- umask (maksa uprawnień tworzonych plików)
- katalog bieżący (związane z chdir)
- katalog główny (związane z chroot)
- nice (priorytet wykonania - pośrednio!)
- limity zasobów
- zużycie zasobów
- zużycie czasu procesora
- **zmienne globalne**
- **przestrzeń adresowa**

# Co wątki mają własne?

- TID (identyfikator wątku)
- maska sygnałów (ustawiana inną funkcją niż w procesach)
- wartość zmiennej errno
- dane własne (inaczej znane jako zmienne “thread local”)
- zmienne lokalne
- polityki szeregowania
- przypisanie do konkretnych procesorów/rdzeni (tzw. CPU affinity)
- security capabilites
- “alternate signal stack”

# Słowo wstępne

- Żeby używać wątków w programie, musimy go skompilować dołączając do niego bibliotekę pthread:

```
LDLIBS = -lpthread
```

# Tworzenie wątku

- int pthread_create(pthread_t *thread, const pthread_attr_t *attr, void *(*start_routine) (void *), void *arg) - man 3 pthread_create
  - Przez wskaźnik thread zostanie przekazany “uchwyt” do wątku
  - Opcjonalnie możemy w attr podać niestandardowe atrybuty z jakimi zostanie utworzony wątek (zamiast wskaźnika do odpowiedniej struktury można przekazać NULL)
  - start_routine to wskaźnik na funkcję która przyjmuje wskaźnik do *jakiegoś* argumentu i zwraca *jakiś* wskaźnik
  - arg to wskaźnik który ma zostać przekazany jako jedyny argument wywołania funkcji start_routine

# Atrybuty wątku

- Zmienna typu pthread_attr_t
- int pthread_attr_init(pthread_attr_t *attr)
  - Służy do zainicjalizowania zmiennej przed użyciem
- int pthread_attr_destroy(pthread_attr_t *attr)
  - Służy do zwolnienia zasobów kiedy zmienna nie jest już potrzebna
- Obie funkcje opisane w man 3 pthread_attr_init
- int pthread_getattr_np(pthread_t thread, pthread_attr_t *attr) - man 3 pthread_getattr_np
  - Inicjalizuje zmienną attr aktualnymi atrybutami wątku thread (pthread_attr_init zbędne)
  - “np” w nazwie oznacza “non-POSIX” - funkcja ta nie należy do standardu POSIX, ale jest dostarczana w linuksowej implementacji pthreads
- Dostępne są liczne funkcje pthread_attr_set* i pthread_attr_get*; zostanie omówiona jedynie para potrzebna do zadania domowego

# Atrybuty wątku cd. (przykładowe)

- int pthread_attr_setdetachstate(pthread_attr_t *attr, int detachstate)
  - Ustawia nową wartość “detach state” we wskazanej strukturze strukturze opisującej atrybuty wątku
  - Parametr detachstate może przyjąć dwie wartości:
    - PTHREAD_CREATE_JOINABLE - domyślnie stosowane, na zakończenie wątku można (i należy!) zaczekać przy użyciu funkcji pthread_join (która pozwoli też odczytać wartość zwróconą przez wątek)
    - PTHREAD_CREATE_DETACHED - wątek wystartuje jako “wątek odłączony”, po jego zakończeniu nastąpi automatyczne zwolnienie zasobów z nim związanych; odpowiada wywołaniu pthread_detach natychmiast po starcie wątku
- int pthread_attr_getdetachstate(const pthread_attr_t *attr, int *detachstate)
  - Odczytuje do zmiennej wskazanej przez detachstate jaka jest aktualna wartość “detach state” we wskazanej strukturze attr opisującej atrybuty wątku
- Obie funkcje omówione w man 3 pthread_attr_setdetachstate

# “Tożsamość” wątku

- pthread_t pthread_self(void) - man 3 pthread_self
  - Zwraca “uchwyt” wątku który wywołał tą funkcję
- Dwóch zmiennych typu pthread_t nie wolno porównywać przy użyciu ==, mogą to być jakieś złożone struktury (a nie wartości liczbowe)
- int pthread_equal(pthread_t t1, pthread_t t2) - man 3 pthread_equal
  - Jeśli t1 i t2 są uchwytem odnoszącym się do tego samego wątku, zwraca prawdę (niezerową wartość); w przeciwnym razie zwraca 0
  - Tylko w ten sposób wolno porównywać dwie wartości pthread_t

# Dane własne wątku

- int pthread_key_create(pthread_key_t *key, void (*destructor)(void*)) - man 3 pthread_key_create
  - Typowo używane w głównym wątku programu (np. w funkcji main)
  - Inicjalizuje wskazane key jako poprawną wartość klucza (nazwy) dla danych własnych
  - Można opcjonalnie wskazać destruktor który zostanie wywołany przy usuwaniu klucza
- int pthread_key_delete(pthread_key_t key) - man 3 pthread_key_delete
  - Usuwa wskazywany przez key klucz danych własnych i uruchamia destruktory
  - Za “posprzątanie” struktur związanych z danymi własnymi odpowiadają wątki które je ustawiły!
- int pthread_setspecific(pthread_key_t key, const void *value) - man 3 pthread_setspecific
  - Ustawia wartość podanego klucza danych własnych na value w bieżącym wątku
- void *pthread_getspecific(pthread_key_t key) - man 3 pthread_getspecific
  - Zwraca wartość podanego klucza danych własnych w bieżącym wątku

# Wysyłanie sygnałów

- int pthread_kill(pthread_t thread, int sig) - man 3 pthread_kill
  - thread to wątek w tym samym procesie do którego ma zostać dostarczony sygnał
  - sig określa jaki sygnał ma zostać dostarczony
- int pthread_sigqueue(pthread_t thread, int sig, const union sigval value) - man 3 pthread_siqueue
  - thead to wątek w tym samym procesie do którego ma zostać dostarczony sygnał
  - sig i value mają znaczenie dokładnie takie jak w sigqueue
- Zbadanie który wątek/wątki odbiorą sygnał w przypadku wysłania go do całego procesu (czy to funkcjami kill/sigqueue, czy też w przypadku wystąpienia błędu w programie) jest przedmiotem zadania domowego

# Maskowanie sygnałów

- int pthread_sigmask(int how, const sigset_t *set, sigset_t *oldset) - man 3 pthread_sigmask
  - Działanie analogiczne jak sigprocmask (man 3 sigprocmask), ale w odniesieniu do bieżącego wątku (a nie całego procesu)
  - Zbiór sygnałów obsługujemy dobrze znanymi funkcjami: sigemptyset, sigfillset, sigaddset, sigdelset, sigismember (wszystkie opisane w man 3 sigemptyset)
  - Drobne oszustwo: obsługa wątków w NPTL (New POSIX Thread Library) wewnętrznie używa dwóch sygnałów czasu rzeczywistego; próba ich zamaskowania zostanie zignorowana, ale i tak nie korzystamy z nich w programach (SIGRTMIN jest odpowiednio przesunięte, żeby ukryć ich wykorzystanie przez NPTL)

# Kończenie wątku

- void pthread_exit(void *retval) - man 3 pthread_exit
  (równoważnie: *return retval;* w funkcji start_routine wątku)
  - Zwraca wartość (wskaźnik) z bieżącego wątku; wartość odbieramy w funkcji pthread_join
- int pthread_detach(pthread_t thread) - man 3 pthread_detach
  - Oznacza wątek thread jako odłączony; po zakończeniu jego zasoby zostaną automatycznie zwolnione, a my nie mamy możliwości odczytania wartości zwróconej ze start_routine
  - Wywołania pthread_detach nie można w żaden sposób cofnąć
- int pthread_join(pthread_t thread, void **retval) - man 3 pthread_join
  - Oczekuje na zakończenie wątku thread i zapisuje do miejsca wskazanego przez retval wartość (wskaźnik) którą zwrócił wątek

# Przerywanie wątku

- int pthread_cancel(pthread_t thread) - man 3 pthread_cancel
  - thread wskazuje wątek do którego chcemy wysłać (zakolejkować) żądanie przerwania
- int pthread_setcancelstate(int state, int *oldstate) - man 3 pthread_setcancelstate
  - state przyjmuje wartości:
    - **PTHREAD_CANCEL_ENABLE** - wątek będzie reagował na przychodzące żądania przerwania go
    - PTHREAD_CANCEL_DISABLE - żądanie przerwania wątku zostanie zablokowane do momentu przywrócenia wartości PTHREAD_CANCEL_ENABLE
- int pthread_setcanceltype(int type, int *oldtype) - man 3 pthread_setcanceltype
  - type przyjmuje wartości:
    - **PTHREAD_CANCEL_DEFERRED** - żądania muszą poczekać do najbliższego cancellation point
    - PTHREAD_CANCEL_ASYNCHRONOUS - żądanie przerwania wątku zadziała (prawie) natychmiast
- void pthread_testcancel(void) - man 3 pthread_testcancel
  - tworzy “cancellation point” - sprawdza czy w danym punkcie funkcji jest jakieś oczekujące żądanie przerwania wątku

# “Tożsamość” wątku raz jeszcze - pobieranie TID

```
#include <sys/syscall.h>
#include <unistd.h>

long gettid() {
    return syscall(SYS_gettid);
}
```

Dziękuję za uwagę