Systemy Operacyjne 2017/18

Wątki - materiały pomocnicze

Informacje wstępne

W obrębie jednego procesu może istnieć wiele wątków, które działają w obrębie jednej przestrzeni adresowej i są wykonywane współbieżnie. Proces kończy swoje działanie, gdy zakończą swoje działanie wszystkie wątki (uwaga - zwrócenie wartości z funkcji `main` kończy działanie wszystkich wątków). Wątki są identyfikowane za pomocą ID typu `pthread_t`. Każdy wątek posiada własny odrębny stos.

Wątki współdzielą:

- przestrzeń adresową (w szczególności zmienne globalne)
- identyfikatory związane z procesem (PID, UID, PPID, ...)
- deskryptory plików
- sposób obsługi sygnałów (signal dispositioin - ignorowanie/obsługa domyślna/handler)
- limity i liczniki zużycia zasobów
- inne: rygle na pliki, umask, katalog główny/bieżący, wartość nice

Każdy wątek posiada własne:

- thread ID (TID)
- **maska** sygnałów
- wartość errno
- dane własne (thread local)
- stos (w szczególności zmienne lokalne)
- inne: polityki szeregowania, CPU affinity, security capabilities, alternate signal stack

Do obsługi wątków w standardzie POSIX służy biblioteka pthreads. Aby jej używać, należy dołączyć plik nagłówkowy pthreads.h oraz dolinkować bibliotekę pthread (`-lpthread`)

Tworzenie wątków

Do tworzenia nowych wątków służy funkcja pthread_create

```
int pthread_create(pthread_t *thread, const pthread_attr_t *attr, void *(*start_routine) (void *), void *arg)
```

gdzie

- `thread` - wskaźnik na miejsce gdzie zapisany zostanie identyfikator utworzonego wątku
- `attr` - dodatkowe ustawienia (opcjonalne, `NULL` = ustawienia domyślne)
- `start_routine` - adres funkcji, która ma zostać wykonana w utworzonym wątku
- `arg` - argument, z którym ma zostać wywołana

Funkcja na którą wskazuje `start_routine` przyjmuje i zwraca wskaźnik na dowolne dane (`void*`). Nie ma gwarancji, że nowo utworzony wątek zacznie swoje działanie natychmiast - nie należy zatem przekazywać jako `arg` adresów zmiennych lokalnych które mogą ulec zniszczeniu zanim nowy wątek zacznie działać.

Typ identyfikatora wątku `pthread_t` jest zależny od implementacji - może nie być to typ całkowitoliczbowy. Do porównywania równości dwóch wartości typu `pthread_t` służy funkcja `pthread_equal`:

```
int pthread_equal(pthread_t tid1, pthread_t tid2);
```

zwracająca 0 jeśli `tid1` nie jest równy `tid2`. Swój własny identyfikator wątek może pobrać przy użyciu `pthread_self`:

```
pthread_t pthread_self(void);
```

Argument attr pozwala kontrolować różne aspekty działania tworzonego wątku. Przed przekazaniem adresu struktury pthread_attr_t należy ją zainicjalizować przy użyciu funkcji `pthread_attr_init`:

```
int pthread_attr_init(pthread_attr_t *attr);
```

Po wywołaniu pthread_create należy zwolnić potencjalnie zaalokowane przez to wywołanie zasoby używając funkcji `pthread_attr_destroy`:

```
int pthread_attr_destroy(pthread_attr_t *attr);
```

Cykl życia wątku

Działanie wątku może zostać zakończone w wyniku kilku zdarzeń:

- jeśli wątek główny zakończy swoje działanie poprzez zwrócenie wartości z funkcji main, cały proces (wszystkie jego wątki) kończy swoje działanie
- jeśli którykolwiek wątek procesu wywoła `exit`, `_Exit` lub `_exit`, cały proces kończy swoje działanie
- jeśli którykolwiek wątek otrzyma sygnał, którego domyślnym sposobem obsługi jest zakończenie procesu, cały proces kończy swoje działanie
- jeśli wątek zwróci wartość ze swojej funkcji, kończy on swoje działanie
- jeśli wątek wywoła `pthread_exit`, kończy on swoje działanie
- jeśli wątek zostanie anulowany przez inny wątek, kończy on swoje działanie

Funkcja

```
void pthread_exit(void *rval_ptr);
```

przyjmuje wartość która zostanie użyta jako wartość zwrócona przez wątek. Zwrócenie wartości `val` z funkcji wątku jest równoważne z wywołaniem `pthread_exit(val)` (poza wątkiem głównym - w nim zwrócenie `val` jest równoważne z `exit(val)`). Wywołanie `pthread_exit` z wątku głównego powoduje zakończenie wątku głównego, ale pozostałe wątki nadal mogą działać - cały proces zakończy się wówczas dopiero gdy zakończą swoje działanie wszystkie utworzone wątki, a kodem wyjścia procesu będzie 0.

Przerwać działanie wątku z innego wątku można przy pomocy funkcji `pthread_cancel`:

```
int pthread_cancel(pthread_t tid);
```

Reakcja wątku na bycie anulowanym zależy od jego ustawień. Wątek może dopuszczać (zachowanie domyślne) lub odrzucać żądania anulowania, sterować tym zachowaniem można przy użyciu funkcji pthread_setcancelstate

```
int pthread_setcancelstate(int state, int *oldstate);
```

podając jako pierwszy argument `PTHREAD_CANCEL_ENABLE` lub `PTHREAD_CANCEL_DISABLE`. Anulowanie wątku z zablokowanym anulowaniem powoduje, że wątek zostanie anulowany w chwili gdy z powrotem odblokuje możliwość anulowania. Jeśli wątek dopuszcza anulowanie, może to robić w dwóch trybach:

- `PTHREAD_CANCEL_DEFERRED` (domyślnie) - wątek kontynuuje swoje działanie do momentu napotkania tzw. *cancellation point*
- `PTHREAD_CANCEL_ASYNCHRONOUS` - wątek kończy swoje działanie natychmiast

Zmienić tryb anulowania można przy pomocy funkcji `pthread_setcanceltype`:

```
int pthread_setcanceltype(int type, int *oldtype);
```

Cancellation point to wywołanie jednej z wymienionej jako takowy przez standard POSIX funkcji (np. `read`, `write`, `pause`). W szczególności cancellation point stanowi wywołanie funkcji `pthread_testcancel`.

Domyślnie po zakończeniu działania wątku wartość którą zwrócił można pobrać przy użyciu funkcji `pthread_join`:

```
int pthread_join(pthread_t thread, void **rval_ptr);
```

Jeśli wątek został anulowany, pod `rval_ptr` zapisana zostanie wartość `PTHREAD_CANCELED`. Jeśli chcemy, by dane zakończonego wątku nie były przechowywane do czasu wywołania pthread_join, a były usuwane natychmiast, możemy wątek uczynić wątkiem odłączonym. Można to uczynić na dwa sposoby: w momencie tworzenia wątku, lub później.

Aby utworzyć wątek od razu jako wątek odłączony, należy do `pthread_create` przekazać adres struktury `pthread_attr_t` po wywołaniu na niej `pthread_attr_setdetachstate`

```
int pthread_attr_setdetachstate(pthread_attr_t *attr, int detachstate);
```

jako drugi argument przekazując `PTHREAD_CREATE_DETACHED`.

Istniejący wątek można po utworzeniu odłączyć przy użyciu funkcji `pthread_detach`:

```
int pthread_detach(pthread_t tid);
```

Wątki i sygnały

Sygnały są dostarczane do jednego wątku w procesie. Sygnały związane ze zdarzeniami sprzętowymi są z reguły dostarczane do wątku, który je spowodował, pozostałe - arbitralnie.

Wątki mają odrębne maski sygnałów, ale wspólne ustawienia ich obsługi (signal disposition). Do ustawienia maski sygnału wątku służy funkcja `pthread_sigmask`:

```
int pthread_sigmask(int how, const sigset_t* set, sigset_t* oset);
```

o sygnaturze identycznej do `sigprocmask`. Działanie `sigprocmask` dla programu wielowątkowego jest niezdefiniowane.

Aby wysłać sygnał do konkretnego wątku, należy użyć funkcji `pthread_kill` lub `pthread_sigqueue`:

```
int pthread_kill(pthread_t thread, int signo);
int pthread_sigqueue(pthread_t thread, int sig, const union sigval value);
```

Ostatnia modyfikacja: wtorek, 15 maja 2018, 00:14

◄ Zadania - Zestaw 8

Przejdź do...

Zadania - Zestaw 9 ►