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);