Procesy 2

Laboratorium 5 - Procesy i wątki

Maryna Lukachyk (308294)

1 Tworzenie wątku

Proces jest wykonywanym programem.

Wykonanie procesu musi przebiegać w sposób se

Proces służy do organizowania wykonywania programu w ten sposób, że stanowi on powiązanie niezbędnych zasobów systemu komputerowego i umożliwia kontrolę stanu tych zasobów, związaną z wykonywaniem programu.

Koncepcja **wątku** (ang. thread) wiąże się ze współdzieleniem zasobów. Każdy proces (ciężki proces w odróżnieniu od lekkiego, czyli wątku) otrzymuje zasoby od odpowiedniego zarządcy i utrzymuje je do swojej dyspozycji

Wątek korzysta głównie z zasobów przydzielonych procesowi — współdzieli je z innymi wątkami tego procesu. Zasobem, o który wątek rywalizuje z innymi wątkami, jest procesor, co wynika z faktu, że jest on odpowiedzialny za wykonanie fragmentu programu.

pthread_create(3)

Należy podczas kompilacji podlinkować bibliotekę pthreads:

gcc -pthread source.c -o program

Wywolujemy tą funkcję by do procesu dodać nowy wątek.

2 Czekanie na zakończenie wątku

pthread_join(3)

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

- pthread_t thread: wątek na który czekamy (nie wskaźnik)
- void **retval: wartość zwracana przez wątek
- zwraca 0 w wypadku poprawnego wykonania lub kod błędu

3 Synchronizacja wątków (muteksy)

int pthread_mutex_lock(pthread_mutex_t *mutex)

int pthread_mutex_unlock(pthread_mutex_t *mutex)

- pthread_mutex_t *mutex: zamek
- zwraca 0 w wypadku poprawnego wykonania, lub numer błędu w przeciwnym wypadku

pthread_mutex_lock - jeśli obiektu mutex-a jest zablokowany przez inny wątek usypia obecny wątek, aż mutex zostanie odblokowany.

- usypia wywołujący ją wątek (jeśli jest to mutex typu "fast")
- zwraca natychmiast kod błędu EDEADLK (jeśli jest to mutex typu "error checking")
- normalnie kontynuuje prace, zwiększając licznik blokad mutex-a przez dany wątek (jeśli mutex jest typu "recursive"); odpowiednia liczba razy odblokowań musi nastąpić aby mutex powrócił do stanu "unlocked"

4 Zmienne warunkowe

pthread_cond_signal (pthread_cond_t *cond);

- pthread_cond_t *cond: zmienna warunkowa

Sygnalizowanie kończy przywraca do aktywności jeden (dowolny) lub wszystkie (kolejno) czekające wątki.

pthread_cond_wait (pthread_cond_t *cond, pthread_mutex_t *mutex);

- pthread_cond_t *cond: zmienna warunkowa
- pthread_mutex_t *mutex: zamek

Czekanie na zmiennej warunkowej zawsze występuje w sekcji krytycznej zamka (tzn. wątek który rozpoczyna czekanie uprzednio pobrał zamek).

5 Kasowanie wątku

Odwolanie (cancellation) jest mechanizmem, za ktorego pomoca jeden watek może zakonczyc dzialanie innego watku. Dokladniej watek moze wyslac zadanie odwolania do innego watku. Zaleznie od ustawien watek, do ktorego wyslano

takie zadanie moze je zignorowac, uwzglednic (zakonczyc swoje dzialanie) natychmiast lub odwlec zakonczenie az do osiagniecia pierwszego punktu zwanego cancellation point.

pthread_exit (void *retval);

- void *retval: wartość zwracana przez bieżący wątek

Odpowiada za zakończenie aktualnego wątku.

int pthread_cancel(pthread_t **thread**); - zakończenie innego wątku

- pthread_t thread: wątek do zamknięcia

`pthread_testcancel()`
Funkcja `pthread_testcancel` przeprowadza test, czy nie ma zadania odwolania i ewentualnie odpowiednio reaguje w razie jego istnienia. Funkcja ta jest uzywana przy duzych fragmentach kodu, gdzie nie ma wywolan funkcji, ktore sa cancellation points, i nie ma innej mozliwosci sprawdzenia, czy przyszlo zadanie odwolania.

- ✓ The ***pthread_mutex_destroy()*** function shall destroy the mutex object referenced by *mutex*; the mutex object becomes, in effect, uninitialized. An implementation may cause *pthread_mutex_destroy*() to set the object referenced by *mutex* to an invalid value. A destroyed mutex object can be reinitialized using *pthread_mutex_init*(); the results of otherwise referencing the object after it has been destroyed are undefined.