Zaawansowane programowanie w C++

Michał Tanaś, PhD

Adam Mickiewicz University, Faculty of Physics

http://www.amu.edu.pl/~mtanas

Michal.Tanas@amu.edu.pl

Programy wielowątkowe – pułapki

Najważniejsza cecha programu wielowątkowego

Dla prawidłowego działania programu istotne są zależności czasowe pomiędzy wątkami

a w związku z tym:

- źle napisany program wielowątkowy nie jest deterministyczny
- śledzenie programu wielowątkowego jest bezcelowe
- szukanie błędów w programie wielowątkowym jest bardzo trudne

Programy wielowątkowe – pułapki

Najważniejsza cecha programu wielowątkowego

Dla prawidłowego działania programu istotne są zależności czasowe pomiędzy wątkami

a w związku z tym:

- źle napisany program wielowątkowy nie jest deterministyczny
- śledzenie programu wielowątkowego jest bezcelowe
- szukanie błędów w programie wielowątkowym jest bardzo trudne

Programy wielowątkowe – pułapki

Najważniejsza cecha programu wielowątkowego

Dla prawidłowego działania programu istotne są zależności czasowe pomiędzy wątkami

a w związku z tym:

- źle napisany program wielowątkowy nie jest deterministyczny
- śledzenie programu wielowątkowego jest bezcelowe
- szukanie błędów w programie wielowątkowym jest bardzo trudne

Programy wielowątkowe – pułapki

Najważniejsza cecha programu wielowątkowego

Dla prawidłowego działania programu istotne są zależności czasowe pomiędzy wątkami

a w związku z tym:

- źle napisany program wielowątkowy nie jest deterministyczny
- śledzenie programu wielowątkowego jest bezcelowe
- szukanie błędów w programie wielowątkowym jest bardzo trudne

Programy wielowątkowe – pułapki

Najważniejsza cecha programu wielowątkowego

Dla prawidłowego działania programu istotne są zależności czasowe pomiędzy wątkami

a w związku z tym:

- źle napisany program wielowątkowy nie jest deterministyczny
- śledzenie programu wielowątkowego jest bezcelowe
- szukanie błędów w programie wielowątkowym jest bardzo trudne

Programy wielowątkowe – pułapki

Najważniejsza cecha programu wielowątkowego

Dla prawidłowego działania programu istotne są zależności czasowe pomiędzy wątkami

a w związku z tym:

- źle napisany program wielowątkowy nie jest deterministyczny
- śledzenie programu wielowątkowego jest bezcelowe
- szukanie błędów w programie wielowątkowym jest bardzo trudne

Programy wielowątkowe – pułapki

Pułapka 1 – lokalne kopie danych

Wątek A	Wątek B
mov ax,zmienna	
inc ax	
	mov ax,zmienna
	add ax,5
	mov zmienna,ax
inc ax	
mov zmienna,ax	
zmienna=zmienna+2	zmienna=zmienna+5

a powinno być: zmienna=zmienna+7

Programy wielowątkowe – pułapki

Pułapka 1 – „podręczne” kopie danych

- wątki operują na własnych lokalnych kopiach zmiennej
- zmiany wartości zmiennej dokonane w jednym wątku nie są widoczne w pozostałych
- końcowy wynik jest przypadkowy – wynik tego wątku który ostatni zapisał zmienną
- Wniosek: kompilator MUSI wiedzieć że daną zmienną wykorzystuje więcej niż jeden wątek, w C++ zmienna musi być typu volatile.

Programy wielowątkowe – pułapki

Pułapka 1 – „podręczne” kopie danych

- wątki operują na własnych lokalnych kopiach zmiennej
- zmiany wartości zmiennej dokonane w jednym wątku nie są widoczne w pozostałych
- końcowy wynik jest przypadkowy – wynik tego wątku który ostatni zapisał zmienną
- Wniosek: kompilator MUSI wiedzieć że daną zmienną wykorzystuje więcej niż jeden wątek, w C++ zmienna musi być typu volatile.

Programy wielowątkowe – pułapki

Pułapka 1 – „podręczne” kopie danych

- wątki operują na własnych lokalnych kopiach zmiennej
- zmiany wartości zmiennej dokonane w jednym wątku nie są widoczne w pozostałych
- końcowy wynik jest przypadkowy – wynik tego wątku który ostatni zapisał zmienną
- Wniosek: kompilator MUSI wiedzieć że daną zmienną wykorzystuje więcej niż jeden wątek, w C++ zmienna musi być typu volatile.

Programy wielowątkowe – pułapki

Pułapka 1 – „podręczne” kopie danych

- wątki operują na własnych lokalnych kopiach zmiennej
- zmiany wartości zmiennej dokonane w jednym wątku nie są widoczne w pozostałych
- końcowy wynik jest przypadkowy – wynik tego wątku który ostatni zapisał zmienną
- Wniosek: kompilator MUSI wiedzieć że daną zmienną wykorzystuje więcej niż jeden wątek, w C++ zmienna musi być typu volatile.

Programy wielowątkowe – pułapki

Pułapka 2 – nie atomowy zapis danych

Wątek A					zmienna					Wątek B			
					0x00	0x00	0x00	0x00					
0x01	0x02	0x03	0x04	→	0x01	0x00	0x00	0x00					
0x01	0x02	0x03	0x04	→	0x01	0x02	0x00	0x00					
					0x01	0x02	0x00	0x00	→	0x01	0x00	0x00	0x00
					0x01	0x02	0x00	0x00	→	0x01	0x02	0x00	0x00
					0x01	0x02	0x00	0x00	→	0x00	0x00	0x00	0x00
					0x01	0x02	0x00	0x00	→	0x00	0x00	0x00	0x00
0x01	0x02	0x03	0x04	→	0x01	0x02	0x03	0x00					
0x01	0x02	0x03	0x04	→	0x01	0x02	0x03	0x04					

Programy wielowątkowe – pułapki

Pułapka 2 – nie atomowy zapis danych

- zapis danych do pamięci nie jest operacją atomową
- możliwe jest zatem odczytanie CZĘŚCIOWO zapisanych danych przez inny wątek
- tak odczytane dane mają zazwyczaj przypadkowe wartości
- Wniosek: każda operacja na współdzielonych danych MUSI być zabezpieczona blokadą (mutex albo semafor).

Programy wielowątkowe – pułapki

Pułapka 2 – nie atomowy zapis danych

- zapis danych do pamięci nie jest operacją atomową
- możliwe jest zatem odczytanie CZĘŚCIOWO zapisanych danych przez inny wątek
- tak odczytane dane mają zazwyczaj przypadkowe wartości
- Wniosek: każda operacja na współdzielonych danych MUSI być zabezpieczona blokadą (mutex albo semafor).

Programy wielowątkowe – pułapki

Pułapka 2 – nie atomowy zapis danych

- zapis danych do pamięci nie jest operacją atomową
- możliwe jest zatem odczytanie CZĘŚCIOWO zapisanych danych przez inny wątek
- tak odczytane dane mają zazwyczaj przypadkowe wartości
- Wniosek: każda operacja na współdzielonych danych MUSI być zabezpieczona blokadą (mutex albo semafor).

Programy wielowątkowe – pułapki

Pułapka 2 – nie atomowy zapis danych

- zapis danych do pamięci nie jest operacją atomową
- możliwe jest zatem odczytanie CZĘŚCIOWO zapisanych danych przez inny wątek
- tak odczytane dane mają zazwyczaj przypadkowe wartości
- Wniosek: każda operacja na współdzielonych danych MUSI być zabezpieczona blokadą (mutex albo semafor).

Programy wielowątkowe – pułapki

Pułapka 3 – zakleszczenie (deadlock)

Wątek A	Wątek B
pthread_mutex_lock(&blokada1);	
	pthread_mutex_lock(&blokada2);
pthread_mutex_lock(&blokada2); STOP	
	pthread_mutex_lock(&blokada1); STOP
posiada blokadę 1, czeka na zwolnienie blokady 2	
	posiada blokadę 2, czeka na zwolnienie blokady 1

Zwolnienie żadnej z blokad nigdy nie nastąpi!

Programy wielowątkowe – pułapki

Pułapka 3 – zakleszczenie (deadlock)

- przy źle założonych blokadach wątki mogą wejść w stan w którym kontynuowanie programu nie jest możliwe
- popularna analogia: problem jedzących filozofów (albo bajka o Kozaku i Tatarzynie)
- Wniosek: bardzo uważnie programować zagnieżdżone blokady, w miarę możliwości unikać ich w ogóle

Programy wielowątkowe – pułapki

Pułapka 3 – zakleszczenie (deadlock)

- przy źle założonych blokadach wątki mogą wejść w stan w którym kontynuowanie programu nie jest możliwe
- popularna analogia: problem jedzących filozofów (albo bajka o Kozaku i Tatarzynie)
- Wniosek: bardzo uważnie programować zagnieżdżone blokady, w miarę możliwości unikać ich w ogóle

Programy wielowątkowe – pułapki

Pułapka 3 – zakleszczenie (deadlock)

- przy źle założonych blokadach wątki mogą wejść w stan w którym kontynuowanie programu nie jest możliwe
- popularna analogia: problem jedzących filozofów (albo bajka o Kozaku i Tatarzynie)
- Wniosek: bardzo uważnie programować zagnieżdżone blokady, w miarę możliwości unikać ich w ogóle

Programy wielowątkowe – pułapki

Pułapka 4 – zagłodzenie wątku

Wątek A

```
for (;;) {
if (pthread_mutex_trylock(&blokada1)<0)
continue;
...
pthread_mutex_unlock(&blokada1);
}
```

Wątek B

```
for (;;) {
if (pthread_mutex_trylock(&blokada2)<0)
continue;
...
pthread_mutex_unlock(&blokada2);
}
```

Wątek C

```
for (;;) {
if (pthread_mutex_trylock(&blokada1)<0)
continue;
if (pthread_mutex_trylock(&blokada2)<0) {
pthread_mutex_unlock(&blokada1);
continue;
}
...
}
```

Wątek C zostanie wykonany tylko jeśli instrukcje oznaczone kolorami będą przetwarzane równocześnie!

Programy wielowątkowe – pułapki

Pułapka 4 – zagłodzenie wątku

- zwalnianie blokad w przypadku niemożliwości kontynuowania wątku eliminuje zakleszczenia, ale wprowadza możliwość że jakiś wątek nigdy nie zostanie wykonany
- popularna analogia: problem jedzących filozofów
- Wniosek: w miarę możliwości unikać pollingu blokad

Programy wielowątkowe – pułapki

Pułapka 4 – zagłodzenie wątku

- zwalnianie blokad w przypadku niemożliwości kontynuowania wątku eliminuje zakleszczenia, ale wprowadza możliwość że jakiś wątek nigdy nie zostanie wykonany
- popularna analogia: problem jedzących filozofów
- Wniosek: w miarę możliwości unikać pollingu blokad

Programy wielowątkowe – pułapki

Pułapka 4 – zagłodzenie wątku

- zwalnianie blokad w przypadku niemożliwości kontynuowania wątku eliminuje zakleszczenia, ale wprowadza możliwość że jakiś wątek nigdy nie zostanie wykonany
- popularna analogia: problem jedzących filozofów
- Wniosek: w miarę możliwości unikać pollingu blokad

Programy wielowątkowe

Mechanizmy programowania wielowątkowego w Linuxie

- Standardowe mechanizmy Linuxa (fork, wait i clone)
- Biblioteka PThreads (POSIX Threads)
- System V IPC (Inter Process Communication)

Programy wielowątkowe

Mechanizmy programowania wielowątkowego w Linuxie

- Standardowe mechanizmy Linuxa (fork, wait i clone)
- Biblioteka PThreads (POSIX Threads)
- System V IPC (Inter Process Communication)

Programy wielowątkowe

Mechanizmy programowania wielowątkowego w Linuxie

- Standardowe mechanizmy Linuxa (fork, wait i clone)
- Biblioteka PThreads (POSIX Threads)
- System V IPC (Inter Process Communication)

Programy wielowątkowe – uwaga techniczna

Niektóre funkcje i zmienne systemowe standardowo nie są przystosowane do programów wielowątkowych. Np. errno.

Program wielowątkowy musi być kompilowany z parametrem gcc -D_REENTRANT

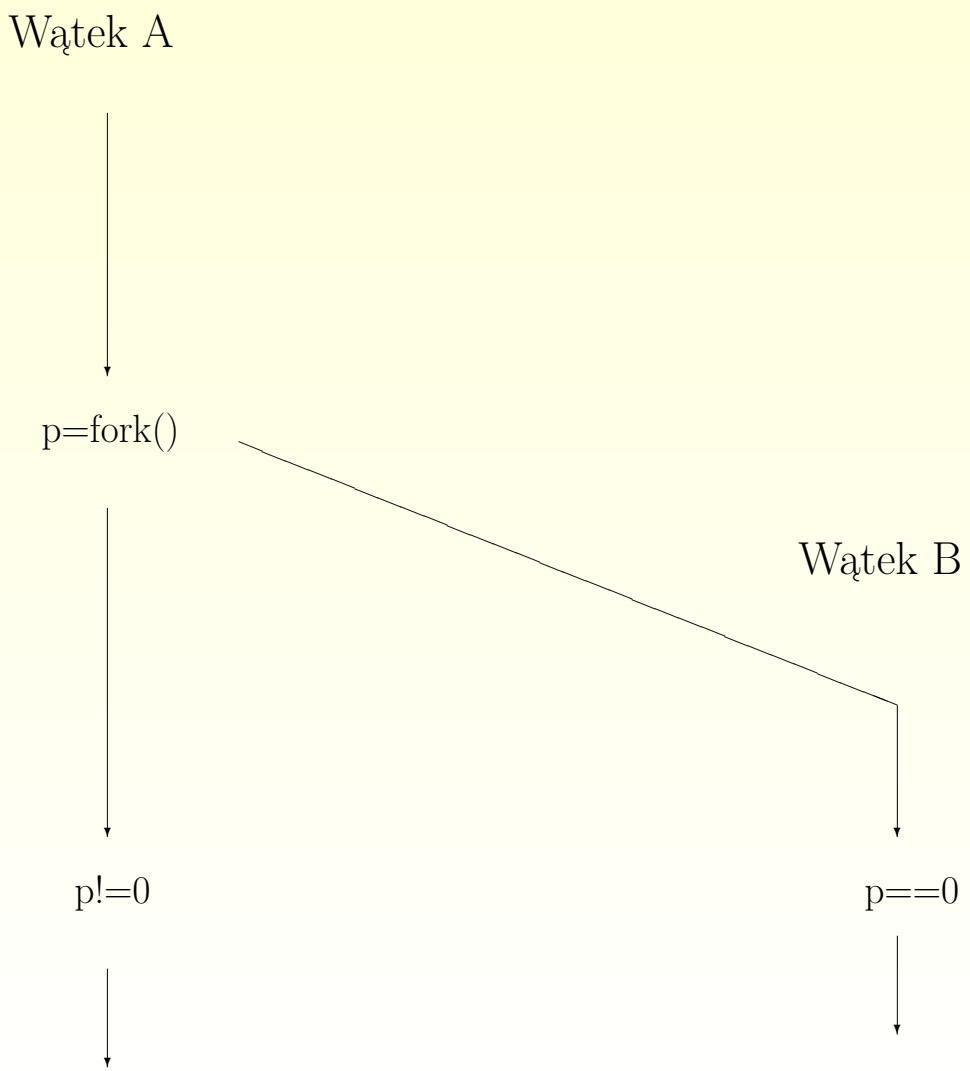
Programy wielowątkowe – uwaga techniczna

Niektóre funkcje i zmienne systemowe standardowo nie są przystosowane do programów wielowątkowych. Np. errno.

Program wielowątkowy musi być kompilowany z parametrem gcc -D_REENTRANT

Programy wielowątkowe – standardowe mechanizmy Unixa

Funkcja fork:



Programy wielowątkowe – standardowe mechanizmy Unixa

int fork()

- tworzy proces potomny identyczny (tzn. z identycznymi stronami kodu) z procesem wywołującym
- oba procesy nie współdzielą żadnych danych (pamięci, otwartych plików, itp.), strony pamięci obu wątków są kopiowane przy pomocy copy-on-write
- do procesu potomnego fork zwraca 0, do procesu macierzystego fork zwraca PID procesu potomnego

Programy wielowątkowe – standardowe mechanizmy Unixa

int fork()

- tworzy proces potomny identyczny (tzn. z identycznymi stronami kodu) z procesem wywołującym
- oba procesy nie współdzielą żadnych danych (pamięci, otwartych plików, itp.), strony pamięci obu wątków są kopiowane przy pomocy copy-on-write
- do procesu potomnego fork zwraca 0, do procesu macierzystego fork zwraca PID procesu potomnego

Programy wielowątkowe – standardowe mechanizmy Unixa

int fork()

- tworzy proces potomny identyczny (tzn. z identycznymi stronami kodu) z procesem wywołującym
- oba procesy nie współdzielą żadnych danych (pamięci, otwartych plików, itp.), strony pamięci obu wątków są kopiowane przy pomocy copy-on-write
- do procesu potomnego fork zwraca 0, do procesu macierzystego fork zwraca PID procesu potomnego

Programy wielowątkowe – standardowe mechanizmy Unixa

Typowe zastosowanie fork

```
pid=fork();
if (pid==0) {
        // proces potomny
} else {
        // proces macierzysty
}
```

Programy wielowątkowe – standardowe mechanizmy Linuxa

```
int clone(int (*fn)(void *), void *child_stack,
          int flags, void *arg);
```

- tworzy proces potomny
- w kontekście procesu macierzystego zwraca PID procesu potomnego
- oba procesy mogą współdzielić dane (pamięć, otwarte pliki, itp.)
- istnieje tylko w Linuxie

Programy wielowątkowe – standardowe mechanizmy Linuxa

```
int clone(int (*fn)(void *), void *child_stack,
          int flags, void *arg);
```

- tworzy proces potomny
- w kontekście procesu macierzystego zwraca PID procesu potomnego
- oba procesy mogą współdzielić dane (pamięć, otwarte pliki, itp.)
- istnieje tylko w Linuxie

Programy wielowątkowe – standardowe mechanizmy Linuxa

```
int clone(int (*fn)(void *), void *child_stack,
          int flags, void *arg);
```

- tworzy proces potomny
- w kontekście procesu macierzystego zwraca PID procesu potomnego
- oba procesy mogą współdzielić dane (pamięć, otwarte pliki, itp.)
- istnieje tylko w Linuxie

Programy wielowątkowe – standardowe mechanizmy Linuxa

```
int clone(int (*fn)(void *), void *child_stack,
          int flags, void *arg);
```

- tworzy proces potomny
- w kontekście procesu macierzystego zwraca PID procesu potomnego
- oba procesy mogą współdzielić dane (pamięć, otwarte pliki, itp.)
- istnieje tylko w Linuxie

Programy wielowątkowe – standardowe mechanizmy Linuxa

```
int clone(int (*fn)(void *), void *child_stack,
          int flags, void *arg);
```

- w kontekście procesu potomnego wywoływana jest funkcja fn z argumentem arg.
- wyjście z funkcji fn w kontekście procesu potomnego jest równoznaczne z zakończeniem tego procesu.
- wartość zwracana przez funkcję fn jest kodem wyjścia procesu potomnego

Programy wielowątkowe – standardowe mechanizmy Linuxa

```
int clone(int (*fn)(void *), void *child_stack,
          int flags, void *arg);
```

- w kontekście procesu potomnego wywoływana jest funkcja fn z argumentem arg.
- wyjście z funkcji fn w kontekście procesu potomnego jest równoznaczne z zakończeniem tego procesu.
- wartość zwracana przez funkcję fn jest kodem wyjścia procesu potomnego

Programy wielowątkowe – standardowe mechanizmy Linuxa

```
int clone(int (*fn)(void *), void *child_stack,
          int flags, void *arg);
```

- w kontekście procesu potomnego wywoływana jest funkcja fn z argumentem arg.
- wyjście z funkcji fn w kontekście procesu potomnego jest równoznaczne z zakończeniem tego procesu.
- wartość zwracana przez funkcję fn jest kodem wyjścia procesu potomnego

Programy wielowątkowe – standardowe mechanizmy Linuxa

```
int clone(int (*fn)(void *), void *child_stack,
          int flags, void *arg);
```

- wskaźnik child_stack określa stos procesu potomnego
- child_stack musi wskazywać na obszar pamięci zaalokowany przez proces macierzysty

Programy wielowątkowe – standardowe mechanizmy Linuxa

```
int clone(int (*fn)(void *), void *child_stack,
          int flags, void *arg);
```

- wskaźnik child_stack określa stos procesu potomnego
- child_stack musi wskazywać na obszar pamięci zaalokowany przez proces macierzysty

Programy wielowątkowe – standardowe mechanizmy Linuxa

```
int clone(int (*fn)(void *), void *child_stack,
          int flags, void *arg);
```

Argument flags określa co będzie współdzielone pomiędzy wątkami (domyślnie: nic). Najważniejsze maski bitowe:

- CLONE_FS – współdzielenie systemu plików. Wywołanie przez dowolny wątek funkcji chdir, chroot lub umask będzie miało efekt również w drugim wątku.
- CLONE_FILES – współdzielenie deskryptorów plików. Wszystkie operacje na deskryptorach plików (również utworzenie nowego deskryptora) wykonane przez dowolny wątek będą miały efekt również w drugim wątku.

Programy wielowątkowe – standardowe mechanizmy Linuxa

```
int clone(int (*fn)(void *), void *child_stack,
          int flags, void *arg);
```

Argument flags określa co będzie współdzielone pomiędzy wątkami (domyślnie: nic). Najważniejsze maski bitowe:

- CLONE_FS – współdzielenie systemu plików. Wywołanie przez dowolny wątek funkcji chdir, chroot lub umask będzie miało efekt również w drugim wątku.
- CLONE_FILES – współdzielenie deskryptorów plików. Wszystkie operacje na deskryptorach plików (również utworzenie nowego deskryptora) wykonane przez dowolny wątek będą miały efekt również w drugim wątku.

Programy wielowątkowe – standardowe mechanizmy Linuxa

```
int clone(int (*fn)(void *), void *child_stack,
          int flags, void *arg);
```

Najważniejsze maski bitowe flags:

- CLONE_SIGHAND – współdzielenie funkcji obsługi sygnałów. Zmiana przez dowolny wątek funkcji obsługi sygnału lub reakcji na sygnał (SIG_IGN, SIG_DFL) będzie miała efekt również w drugim wątku.
- CLONE_VM – współdzielenie pamięci. Zapis do pamięci (oraz wywołanie funkcji mmap i munmap) wykonany przez przez dowolny wątek będzie widoczny również w drugim wątku.

Programy wielowątkowe – standardowe mechanizmy Linuxa

```
int clone(int (*fn)(void *), void *child_stack,
          int flags, void *arg);
```

Najważniejsze maski bitowe flags:

- CLONE_SIGHAND – współdzielenie funkcji obsługi sygnałów. Zmiana przez dowolny wątek funkcji obsługi sygnału lub reakcji na sygnał (SIG_IGN, SIG_DFL) będzie miała efekt również w drugim wątku.
- CLONE_VM – współdzielenie pamięci. Zapis do pamięci (oraz wywołanie funkcji mmap i munmap) wykonany przez przez dowolny wątek będzie widoczny również w drugim wątku.

Programy wielowątkowe – standardowe mechanizmy Linuxa

pid_t wait(int *status);

Funkcja wait zatrzymuje wykonywanie procesu macierzystego do czasu aż nastąpi zakończenie jakiegokolwiek procesu potomnego.

zwraca PID procesu potomnego który się zakończył

Jeżeli argument status jest różny od NULL zostaną w nim zachowane informacje o przyczynach zakończenia procesu potomnego. Wartość zmiennej wskazywanej przez status przed wywołaniem wait jest ignorowana.

Programy wielowątkowe – standardowe mechanizmy Linuxa

pid_t wait(int *status);

Funkcja wait zatrzymuje wykonywanie procesu macierzystego do czasu aż nastąpi zakończenie jakiegokolwiek procesu potomnego.

zwraca PID procesu potomnego który się zakończył

Jeżeli argument status jest różny od NULL zostaną w nim zachowane informacje o przyczynach zakończenia procesu potomnego. Wartość zmiennej wskazywanej przez status przed wywołaniem wait jest ignorowana.

Programy wielowątkowe – standardowe mechanizmy Linuxa

pid_t wait(int *status);

Funkcja wait zatrzymuje wykonywanie procesu macierzystego do czasu aż nastąpi zakończenie jakiegokolwiek procesu potomnego.

zwraca PID procesu potomnego który się zakończył

Jeżeli argument status jest różny od NULL zostaną w nim zachowane informacje o przyczynach zakończenia procesu potomnego. Wartość zmiennej wskazywanej przez status przed wywołaniem wait jest ignorowana.

Programy wielowątkowe – standardowe mechanizmy Linuxa

```
pid_t wait(int *status);
```

Makra testujące status:

- WIFEXITED(status) – zwraca wartość logiczną „prawda” jeżeli proces potomny zakończył się w normalny sposób (tzn. przez wywołanie exit, powrót z funkcji main)
- WIFEXITSTATUS(status) – zwraca kod zakończenia procesu potomnego
  - może być wykonane tylko jeśli WIFEXITED zwróciło warość „prawda”
  - zwraca jedynie 8 najmłodszych bitów kodu zakończenia

Programy wielowątkowe – standardowe mechanizmy Linuxa

```
pid_t wait(int *status);
```

Makra testujące status:

- WIFEXITED(status) – zwraca wartość logiczną „prawda” jeżeli proces potomny zakończył się w normalny sposób (tzn. przez wywołanie exit, powrót z funkcji main)
- WIFEXITSTATUS(status) – zwraca kod zakończenia procesu potomnego
  - może być wykonane tylko jeśli WIFEXITED zwróciło warość „prawda”
  - zwraca jedynie 8 najmłodszych bitów kodu zakończenia

Programy wielowątkowe – standardowe mechanizmy Linuxa

```
pid_t wait(int *status);
```

Makra testujące status:

- WIFEXITED(status) – zwraca wartość logiczną „prawda” jeżeli proces potomny zakończył się w normalny sposób (tzn. przez wywołanie exit, powrót z funkcji main)
- WIFEXITSTATUS(status) – zwraca kod zakończenia procesu potomnego
  - może być wykonane tylko jeśli WIFEXITED zwróciło warość „prawda”
  - zwraca jedynie 8 najmłodszych bitów kodu zakończenia

Programy wielowątkowe – standardowe mechanizmy Linuxa

```
pid_t wait(int *status);
```

Makra testujące status:

- WIFEXITED(status) – zwraca wartość logiczną „prawda” jeżeli proces potomny zakończył się w normalny sposób (tzn. przez wywołanie exit, powrót z funkcji main)
- WIFEXITSTATUS(status) – zwraca kod zakończenia procesu potomnego
  - może być wykonane tylko jeśli WIFEXITED zwróciło warość „prawda”
  - zwraca jedynie 8 najmłodszych bitów kodu zakończenia

Programy wielowątkowe – standardowe mechanizmy Linuxa

pid_t wait(int *status);

Makra testujące status:

- WIFSIGNALED(status) – zwraca wartość logiczną „prawda” jeżeli proces potomny zakończył w skutek otrzymania sygnału (np. SIGTERM, SIGSEGV, itp.)
- WTERMSIG(status) – zwraca numer sygnału który spowodował zakończenia procesu potomnego. Może być wykonane tylko jeśli WIFSIGNALED zwróciło warość „prawda”.

Programy wielowątkowe – standardowe mechanizmy Linuxa

pid_t wait(int *status);

Makra testujące status:

- WIFSIGNALED(status) – zwraca wartość logiczną „prawda” jeżeli proces potomny zakończył w skutek otrzymania sygnału (np. SIGTERM, SIGSEGV, itp.)
- WTERMSIG(status) – zwraca numer sygnału który spowodował zakończenia procesu potomnego. Może być wykonane tylko jeśli WIFSIGNALED zwróciło warość „prawda”.

Programy wielowątkowe – standardowe mechanizmy Linuxa

pid_t waitpid(pid_t pid, int *status, int options);

Funkcja waitpid zatrzymuje wykonywanie procesu macierzystego do czasu aż nastąpi zakończenie procesu potomnego o podanym identyfikatorze.

zwraca PID procesu potomnego który się zakończył

W zależności od wartość parametru pid

- pid>0 – waitpid czeka na zakończenie procesu potomnego i identyfikatorze pid
- pid==-1 – waitpid czeka na zakończenie dowolnego procesu potomnego (tak jak wait).

Programy wielowątkowe – standardowe mechanizmy Linuxa

pid_t waitpid(pid_t pid, int *status, int options);

Funkcja waitpid zatrzymuje wykonywanie procesu macierzystego do czasu aż nastąpi zakończenie procesu potomnego o podanym identyfikatorze.

zwraca PID procesu potomnego który się zakończył

W zależności od wartość parametru pid

- pid>0 – waitpid czeka na zakończenie procesu potomnego i identyfikatorze pid
- pid==-1 – waitpid czeka na zakończenie dowolnego procesu potomnego (tak jak wait).

Programy wielowątkowe – standardowe mechanizmy Linuxa

pid_t waitpid(pid_t pid, int *status, int options);

Funkcja waitpid zatrzymuje wykonywanie procesu macierzystego do czasu aż nastąpi zakończenie procesu potomnego o podanym identyfikatorze.

zwraca PID procesu potomnego który się zakończył

W zależności od wartość parametru pid

- pid>0 – waitpid czeka na zakończenie procesu potomnego i identyfikatorze pid
- pid==-1 – waitpid czeka na zakończenie dowolnego procesu potomnego (tak jak wait).

Programy wielowątkowe – standardowe mechanizmy Linuxa

pid_t waitpid(pid_t pid, int *status, int options);

Funkcja waitpid zatrzymuje wykonywanie procesu macierzystego do czasu aż nastąpi zakończenie procesu potomnego o podanym identyfikatorze.

zwraca PID procesu potomnego który się zakończył

W zależności od wartość parametru pid

- pid>0 – waitpid czeka na zakończenie procesu potomnego i identyfikatorze pid
- pid==-1 – waitpid czeka na zakończenie dowolnego procesu potomnego (tak jak wait).

Programy wielowątkowe – standardowe mechanizmy Linuxa

pid_t waitpid(pid_t pid, int *status, int options);

Funkcja waitpid zatrzymuje wykonywanie procesu macierzystego do czasu aż nastąpi zakończenie procesu potomnego o podanym identyfikatorze.

zwraca PID procesu potomnego który się zakończył

W zależności od wartość parametru pid

- pid>0 – waitpid czeka na zakończenie procesu potomnego i identyfikatorze pid
- pid==-1 – waitpid czeka na zakończenie dowolnego procesu potomnego (tak jak wait).

Programy wielowątkowe – standardowe mechanizmy Linuxa

pid_t waitpid(pid_t pid, int *status, int options);

argument status działa identycznie jak w funkcji wait

najważniejsze maski bitowe parametru options

- WNOHANG – waitpid nie zatrzymuje procesu macierzystego jeżeli żaden z procesów potomnych się nie zakończył. Używane do usuwania zombich.

Programy wielowątkowe – standardowe mechanizmy Linuxa

pid_t waitpid(pid_t pid, int *status, int options);

argument status działa identycznie jak w funkcji wait

najważniejsze maski bitowe parametru options

- WNOHANG – waitpid nie zatrzymuje procesu macierzystego jeżeli żaden z procesów potomnych się nie zakończył. Używane do usuwania zombich.

Programy wielowątkowe – standardowe mechanizmy Linuxa

pid_t waitpid(pid_t pid, int *status, int options);

argument status działa identycznie jak w funkcji wait

najważniejsze maski bitowe parametru options

- WNOHANG – waitpid nie zatrzymuje procesu macierzystego jeżeli żaden z procesów potomnych się nie zakończył. Używane do usuwania zombich.

Programy wielowątkowe – standardowe mechanizmy Linuxa

```
pid_t wait(int *status);
pid_t waitpid(pid_t pid, int *status, int options);
```

Błędy zwracane przez wait i waitpid

- ECHILD dla wait – proces w którym wywołano wait nie ma procesów potomnych
- ECHILD dla waitpid – proces o identyfikatorze pid nie istnieje lub nie jest potomkiem procesu dla którego wywołano wait.

Programy wielowątkowe – standardowe mechanizmy Linuxa

```
pid_t wait(int *status);
pid_t waitpid(pid_t pid, int *status, int options);
```

Błędy zwracane przez wait i waitpid

- ECHILD dla wait – proces w którym wywołano wait nie ma procesów potomnych
- ECHILD dla waitpid – proces o identyfikatorze pid nie istnieje lub nie jest potomkiem procesu dla którego wywołano wait.