Мета роботи: опанувати техніку оброблення сигналів у багатопотоковому середовищі.

Теоретичні відомості

Техніка оброблення сигналів може бути досить складною навіть в однопотоковому застосунку. Наявність декількох потоків ще більш заплутує справу.

Кожен потік має свою власну маску сигналів, але диспозиція сигналів одна для всіх потоків процесу. Це означає, що кожен окремо взятий потік може заблокувати доставку сигналу, але коли потік визначає реакцію на сигнал, вона стає загальною для всіх потоків. Таким чином, якщо один потік встановив диспозицію сигналу так, щоб він ігнорувався, то інший потік може скасувати цю дію, встановивши диспозицію сигналу в значення за замовчуванням або призначивши оброблювач сигналу.

Сигнали доставляються тільки одному потоку процесу. Якщо поява сигналу викликана апаратною помилкою або спрацюванням таймера, то він доставляється тому потоку, який є причиною появи цього сигналу. Однак інші сигнали доставляються будь-якому довільному потоку.

У розділі 9.11 ми розповідали, як можна використовувати функцію *sigprocmask* для блокування сигналів. Поведінка функції *sigprocmask* у багатопотоковому середовищі не визначена. Замість неї потоки повинні використовувати функцію *pthread_sigmask*.

```
#include <pthread.h>
#include <signal.h>
int pthread_sigmask (int how, const sigset_t *set, sigset_t *oset);
/* функція повертає 0 в разі успіху, код помилки – в разі невдачі */
```

Дія функції *pthread_sigmask* ідентична дії функції *sigprocmask*, за винятком того, що вона призначена для роботи в багатопотоковому середовищі й у випадку помилки повертає не -1 із записаним кодом помилки в змінній *errno*, а сам код помилки.

Потік може призупинити своє виконання в очікуванні доставки сигналу, викликавши функцію *sigwait*.

```
#include <pthread.h>
#include <signal.h>
int sigwait (const sigset_t *set, int *signop);
/* функція повертає 0 в разі успіху, код помилки – в разі невдачі */
```

Аргумент *set* визначає набір сигналів, доставки яких очікують. Після повернення з функції за адресою *signop* буде записаний номер доставленого сигналу.

Якщо який-небудь сигнал, що входить до набору *set*, до моменту виклику функції *sigwait* очікує оброблення, то функція поверне керування негайно. Перед поверненням керування функція *sigwait* вилучить сигнал з набору сигналів, що очікують оброблення. Щоб уникнути помилкової реакції на сигнал, потік повинен заблокувати очікувані сигнали перед викликом функції *sigwait*. Ця функція атомарно розблокує сигнали і перейде в режим очікування, поки визначені сигнали не будуть доставлені. Перед поверненням керування функція *sigwait* відновить маску сигналів потоку. Якщо сигнал не буде заблокований до моменту виклику функції, то пройде деякий час, протягом якого сигнал може бути доставлений потоку ще до того, як він викличе функцію *sigwait*.

Перевага використання функції *sigwait* полягає в тому, що вона дозволяє спростити оброблення сигналів і обробляти асинхронні сигнали в синхронному стилі. Щоб не допустити переривання виконання потоку за сигналом, можна додати необхідні сигнали до маски сигналів кожного потоку. Завдяки цьому ми можемо виділити окремі потоки, що будуть займатися тільки обробленням сигналів. Ці спеціально виділені потоки можуть звертатися до будь-яких функцій, які не можна використовувати в оброблювачах сигналів, тому що в цій ситуації функції будуть викликатися в контексті звичайного потоку, а не з традиційного оброблювача сигналу, що переривае роботу потоку.

Якщо відразу декілька потоків виявляться заблокованими функцією *sigwait* в очікуванні того самого сигналу, то функція *sigwait* поверне керування тільки одному з них, коли сигнал буде доставлений процесу. Якщо сигнал перехоплюється процесом (наприклад, коли процес встановив оброблювач сигналу за допомогою функції *sigaction*), і при цьому потік, що звернувся до функції *sigwait*, очікує доставки того ж самого сигналу, то ухвалення рішення про спосіб доставки сигналу визначається реалізацією. Операційна система в цьому випадку може або викликати встановлений оброблювач сигналу, або дозволити функції *sigwait* повернути керування потоку, але ні в якому разі не те й інше разом.

Для надсилання сигналу процесу використовують функцію *kill* (розділ 9.8). Для надсилання сигналу потоку використовують функцію *pthread_kill*.

```
#include <pthread.h>
#include <signal.h>
int pthread_kill (pthread_t thread, int signo);
/* функція повертає 0 в разі успіху, код помилки – в разі невдачі */
```

Можна перевірити існування потоку, передавши в аргументі *signo* значення 0. Якщо дією за замовчуванням для сигналу є завершення процесу, то передача такого сигналу потоку приведе до завершення всього процесу.

Зверніть увагу: таймери є ресурсами процесу, і всі потоки спільно використовують той самий набір таймерів. Отже, у випадку багатопотокового застосунку неможливо використовувати таймери в одному потоці, не впливаючи на інші потоки.

Контрольні запитання

1. Чим є сигнали та з якою метою їх використовують в ОС UNIX?
2. Які види сигналів існують в ОС UNIX та яким чином можна на них реагувати в разі надходження?
3. Окресліть основні принципи розроблення оброблювачів сигналів.
4. Як можна реалізувати синхронне оброблення асинхронних сигналів у багатопотоковій програмі?

Індивідуальні завдання

Пересвідчитись у тому, що потоки в ОС Linux реалізовані як група процесів, яку очолює головний потік. Функція *getpgrp*.

Напишіть багатопотокову програму та з’ясуйте, якому потоку буде доставлений сигнал, якщо декілька потоків одночасно очікують надходження того самого сигналу за допомогою функції *sigwait*.

Напишіть багатопотокову програму та з’ясуйте, в який спосіб буде оброблений доставлений сигнал у випадку одночасного використання деяким потоком функцій *sigwait* та *sigaction*.