Функції управління життєвим циклом процесів в ОС UNIX

В ОС UNIX (та похідних від неї) всі процеси пов’язані відношенням породивший – породжений (батьківський – синівський). Це дозволяє досить просто організувати колективну роботу кількох процесів (далі П).

Подальший матеріал в основному відповідає стандарту POSIX P1003.1 в редакції 1988 р.

Породження


```
int fork( void );
```

Сворює новий П.

Породжений П є точною копією батьківського П за винятком:
- новий П має унікальний ідентифікатор;
- має ті ж відкриті файли і каталоги що і батьківський;
- всі атрибути часу виконання П встановлені на 0;
- таймер П встановлений на 0;
- блокування ділянок файлів, виконані батьківським П, не успадковуються;
- множина непереданих сигналів нового П порожня.

Породження

`int fork( void );` при успішному завершенні повертає синівському П **0** а батьківському – **ідентифікатор сина**.

Виконання обох П продовжується з однієї й тієї ж точки в функції **fork**.

У випадку помилки новий П не створюється і повертається значення **-1**.

При цьому змінна **`extern int errno;`** приймає одно із двох значень:

[EAGAIN] - в системі недостатньо ресурсів для створення П або досягнуте обмеження на допустиме число П одного користувача.

[ENOMEM] - П необхідно більше пам'яті ніж може надати система.

Мій ідентифікатор


```
#include <sys/types.h>
pid_t getpid( void );
```

Повертає ідентифікатор поточного П.

pid_t це простий тип відповідність якого вбудованому типу мови програмування С вказана в файлі types.h.
Ви можете вважати його "виродженим" абстрактним типом.

Ідентифікатор батька


```
#include <sys/types.h>
pid_t getppid( void );
```

Повертає ідентифікатор **батька** поточного П.

Ніякого іншого способу встановити ідентифікатор батьківського П не передбачено.

Команда


```
#include <stdlib.h>
int system( char* string );
```

Передає системі символьний рядок для виконання **інтерпретатором команд** (командним процесором).

Повертає П значення залежне від реалізації у випадку коли `char* string` не є нульовим покажчиком. Інакше (при `(char*) 0`) значення відмінне від **0** повертається тільки при наявності в системі інтерпретатора команд.

Виклик програми


```
int execve( char* path, char* argv[], char* envp[]
);
```

Заміняє поточний образ процесу новим образом із файла програми, вказаного в першому параметрі, і передає йому управління. Ця програма (на мові С) буде починатись так:

```
extern char** environ;
int main( int argc, char** argv ) {
```

Тут **argc** - кількість числа аргументів, **argv** – масив покажчиків на параметри (аргументи) які передаються програмі. Кожен аргумент є символьним рядком. Зовнішня змінна **environ** в якості початкового значення має покажчик на масив рядків які являються середовищем П. **argv** і **environ** завершаються нульовим покажчиком. Він не враховується в **argc**.

Виклик програми


```
int execve( char* path, char* argv[], char* envp[]
);
```

Параметр **envp** - покажчик на масив рядків які являються середовищем П. З його допомогою ви можете вказати бажаність використання довільного середовища замість того на яке вказує змінна **environ**.

Сімейство функцій виклику програм включає 6 їх варіантів. Відрізняються вони тільки набором параметрів:

```
int execl(char* path, char* arg0, ..., (char*) 0);
int execv(char* path, char* argv[]);
int execle(char* path, char* arg0, ..., (char*) 0,
    char* envp[]);
int execve(char* path, char* argv[], char* envp[]);
int execlp(char* file, char* arg0, ..., (char*) 0);
int execvp(char* file, char* argv[]);
```

Пошук в середовищі


```
#include <stdlib.h>
char* getenv( char* name );
```

Шукає в середовищі П рядок виду “ІМ'Я=ЗНАЧЕННЯ” такий що “ІМ'Я” співпадає з **name**.

Повертає покажчик на символьний рядок що є значенням імені параметра в середовищі П. В разі відсутності такого імені повертає нульовий покажчик.

Завершення процесу


```
#include <stdlib.h>
void exit( status );
```

Нормальне завершення поточного П. При цьому
- викликаються всі ф-ції заявлені раніше в **atexit** ;
- очищуються всі відкриті потоки;
- всі тимчасові файли видаляються;
- всі відкриті П файли та каталоги закриваються;
- аргумент **status** передається батьківському П (або запам'ятовується для подальшої передачі);
- в певних умовах іншим П можуть пересилатись *сигнали* **SIGCHLD, SIGHUP, SIGCONT.**

Завершення П не приводить до завершення його синів. Як правило, вони приєднуються до іншого батьківського П.

Завершення процесу


```
#include <stdlib.h>
int atexit( void (*func)(void) );
```

Об'являє що при завершенні поточного П функцією **`exit`** повинна бути виконана функція **`func`** без параметрів.

При успішному завершенні повертає 0, інакше – відмінне від 0.

Можна *зареєструвати* кілька функцій які будуть виконані в порядку зворотному порядку реєстрації.

Аварійне завершення процесу


```
#include <stdlib.h>
int abort( void );
```

Виконує аварійне завершення поточного П передаючи самому собі сигнал **SIGABRT** (сигнал аварійного завершення).

П завершиться в тому випадку якщо сигнал **SIGABRT** не *ігнорується* і не *перехоплюється*.

Виконує ті ж дії що і функція **exit**.

Очікування завершення процесу


```
#include <sys/types.h>
#include <sys/wait.h>
pid_t wait( int* stat_lock );
```

Поточний П блокується в очікуванні коду завершення одного із синівських П.

```
pid_t waitpid( pid_t pid, int* stat_lock,    int
options );
```