2. Многозадачность в ОС GNU/Linux. Часть 1

Разделы:

- Процессы
  - Создание
  - Уничтожение

Процессы

- У каждого процесса в Linux есть уникальный 16-битный *идентификатор* (PID)
- Он назначается последовательно по мере создания процессов
- У всех процессов, кроме *init*, имеется также родительский процесс
- *init* в иерархии процессов является *корневым* элементом
- У каждого процесса, кроме *init*, есть *идентификатор предка* (PPID)

Процессы

```
/* print-pid.c
 * Print process identifiers
 */

#include <stdio.h>
#include <unistd.h>

int main()
{
  /* Here pid_t isn't used */
  printf("The process ID is %d\n", getpid());
  printf("The parent process ID is %d\n", getppid());
  return 0;
}
```

Сигналы

Имя	Значение	Эффект
SIGHUP	1	Зависание
SIGINT	2	Прерывание от клавиатуры
SIGKILL	9	Удаление процесса
SIGTERM	15	Прерывание процесса
SIGSTOP	19,23	Останов процесса
SIGBUS	7	Ошибка шины
SIGSEGV	11	Ошибка памяти
SIGFPE	8	Ошибка при работе с плавающей точкой
SIGABRT	6	Системный вызов abort()
SIGCHLD	17	Останов дочернего процесса

Создание процессов

```
/* system.c
 * Demonstrate system() function
 */

#include <stdio.h>
#include <unistd.h>

int main()
{
  int returnValue = system("ls -l /");
  return returnValue;
}
```

Создание процессов

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>

int main()
{
  pid_t childPid;

  printf("The main program PID is %d\n", getpid());
  childPid = fork();
  if(0 != childPid)
  {
    printf("This is parent process, its PID is %d\n", getppid());
    printf("The child's PID is %d\n", childPid);
  }
  else
  {
    printf("This is child process, its PID is %d\n", getppid());
  }

  return 0;
}
```

Создание процессов

- Функции *execvp()* и *execlp()* принимают в качестве аргумента имя программы и ищут ее в каталогах, определенных переменной окружения *PATH*
- Функции *execv()*, *execvp()* и *execve()* принимают список аргументов в виде массива указателей на строки, оканчивающегося NULL-указателем
- Функции *execl()*, *execlp()* и *execle()* принимают список аргументов переменного размера
- Функции *execve()* и *execle()* в качестве дополнительного аргумента принимают массив переменных среды

Создание процессов

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
int spawn(char* program, char** argList)
{
  pid_t childPid;
  childPid = fork();
  if (0 != childPid)
  {
    /* This is the parent process.  */
    return childPid;
  }
  else
  {
    execvp(program, argList);
    fprintf(stderr, "an error occurred in execvp\n");
    abort();
  }
}
int main()
{
  char* argList[] =   { "ls", "-l", "/", NULL };
  spawn("ls", argList);
  printf("\ndone with main program.\n");
  return 0;
}
```

Завершение процессов

```
kill(childPid, SIGTERM);
-----------------------------
```

- Функция *wait()* блокирует процесс до тех пор, пока один из его потомков не завершится
- Макрос *WEXITSTATUS* возвращает код завершения
- Макрос *WIFEXITED* позволяет узнать, как именно завершился процесс: обычным образом или аварийно (по сигналу)
- Номер сигнала в последнем случае можно извлечь макросом *WTERMSIG*

Завершение процессов

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <signal.h>
#include <sys/wait.h>
#include <unistd.h>
int spawn(char* program, char** argList)
{
  pid_t childPid;
  childPid = fork();
  if (0 != childPid)
    return childPid;
  else
  {
    execvp(program, argList);
    fprintf(stderr, "an error occurred in execvp\n");
    abort();
  }
}
int main()
{
  int childStatus;
  char* argList[] = { "ls", "-l",  "/", NULL };
  spawn("ls", argList);
  wait(&childStatus);
  if(WIFEXITED(childStatus))
    printf("The child process exited normally with code %d.\n"
           , WEXITSTATUS(childStatus)
          );
  else
    printf("The child process exited abnormally.\n");
  return 0;
}
```

Процессы-зомби

```
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
  pid_t childPid;
  childPid = fork();
  if (childPid > 0)
  {
    sleep(60);
  }
  else
  {
    exit(0);
  }
  return 0;
}
```

Асинхронное удаление дочерних процессов

```
#include <signal.h>
#include <string.h>
#include <sys/types.h>
#include <sys/wait.h>
sig_atomic_t g_ChildExitStatus;
void CleanUpChildProcess(int signalNumber)
{
  int status;
  wait(&status);
  g_ChildExitStatus = status;
}
int main()
{
  struct sigaction sigchldAction;
  memset (&sigchldAction, 0, sizeof (sigchldAction));
  sigchldAction.sa_handler = &CleanUpChildProcess;
  sigaction(SIGCHLD, &sigchldAction, NULL);
/* ...  */
  return 0;
}
```

See also

- Delve into UNIX process creation - http://www.ibm.com/developerworks/aix/lib
- YoLinux Tutorial: Fork, Exec and Process control - http://www.yolinux.com/TUTORIALS/ForkExecProce
- Инструменты Linux для Windows-программистов - http://rus-linux.net/nlib.php?name=/MyLDP/BOOKS/Linux-tools/index.html
- Лав, Р. Linux. Системное программирование/ Р.Лав. – СПб.: Питер, 2008. – 416 с.