Приложение, которое позволяет передаваемому процессу работать в фоне

Код программы

#include <stdio.h>

#include <unistd.h>

int main (int argc, char \*argv[])

{

printf("В фоне запущена программа %s\n\n", argv[1]);

if(fork() == 0) /\* дочерний процесс \*/

{

execvp(argv[1], &argv[1]);/\* выполнить программу \*/

fprintf (stderr, "Could not execute %s\n", argv[1]);

}

}

Приложение, которое обеспечивает перенаправление стандартного ввода и вывода

Код программы

#include <stdio.h>

#include <fcntl.h>

#include <unistd.h>

int main (int argc, char \*argv[])

{

int fd;/\* открыть файл для переназначения \*/

fd = open (argv[1], O\_CREAT| O\_TRUNC | O\_WRONLY, 0600);

dup2 (fd, 1); /\* дублировать дескриптор на стандартный вывод\*/

close(fd); /\* закрыть оригинальный дескриптор для сохранения пространства дескрипторов \*/

execvp(argv[2], &argv[2]); /\* вызвать программу; унаследует stdout\*/

perror("main"); /\* не должна выполняться никогда \*/

}

Приложение, которое соединяет процессы через неименованные каналы

Код программы

#include <stdio.h>

#include <unistd.h>

#define READ 0

#define WRITE 1

int main (argc, argv)

int argc;

char\* argv [];

{

int fd [2];

pipe(fd); /\* создатьбезымянныйконвейер\*/

if(fork() != 0) /\* родитель, пишущий \*/

{

close(fd[READ]); /\* закрыть неиспользуемый конец \*/

dup2 (fd[WRITE], 1); /\* дублировать используемый конец stdout\*/

close(fd[WRITE]); /\* закрыть оригинальный используемый конец \*/

execlp(argv[1], argv[1], NULL); /\* выполнить программу-писатель \*/

perror("connect"); /\* не должен никогда выполняться \*/

}

else/\* дочерний, читающий \*/

{

close(fd[WRITE]); /\* закрыть неиспользуемый конец \*/

dup2 (fd[READ], 0); /\* дублировать используемый конец stdin\*/

close(fd[READ]); /\* закрыть оригинальный используемый конец \*/

execlp(argv[2], argv[2], NULL); /\* выполнить программу-читатель \*/

perror("connect"); /\* не должен никогда выполняться \*/

}

}

Два приложения: первое приложение создает именованный канал, второе приложение (писатель) получает дескриптор канала на запись и записывает в него свой идентификатор (pid). Первое приложение (читатель) ожидает поступления данных в канал и выводит их на экран. Для совместной работы приложений, исходя из особенностей алгоритма, необходимо сначала запустить первое приложение.

Код программы

Reader.c

#include <stdio.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <unistd.h>

int readLine(int fd,char\* str);

int main()

{

int fd;

char str[100];

mkfifo ("aPipe", 0660); /\* создать именованный конвейер \*/

fd = open ("aPipe", O\_RDONLY); /\* открыть его для чтения \*/

while (readLine (fd, str)) /\* отобразить полученное сообщение \*/

printf ("%s\n", str);

close (fd); /\* закрыть конвейер \*/

}

int readLine(fd, str)

int fd;

char\* str;

/\* читает отдельную, заканчивающуюся NULL, строку в str из fd \*/

/\* возвращает 0, когда достигнут конец ввода, и 1 - иначе \*/

{

int n;

do /\* читать символы до NULL или конца ввода\*/

{

n = read (fd, str, 1); /\* читать один символ \*/

}

while (n > 0 && \*str++ != 0);

return (n > 0); /\* возвращает ложь, если конец ввода \*/

}

Writer.c

#include <unistd.h>

#include <stdio.h>

#include <fcntl.h>

#include <string.h>

int main ()

{

int fd, messageLen, i;

char message[100];

/\* подготовить сообщение \*/

sprintf(message, "Hello from PID %d", getpid());

messageLen = strlen(message) + 1;

do /\* пытаться открыть файл до успеха \*/

{

fd = open("aPipe", O\_WRONLY); /\* открыть именованный конвейер для

записи\*/

if (fd == -1) sleep(1); /\* пытаться снова через 1 секунду \*/

}

while (fd == -1);

for (i = 1; i <= 3; i++) /\* послать 3 сообщения \*/

{

write(fd, message, messageLen); /\* записать сообщение в конвейер \*/

sleep(3); /\* подождать немного \*/

}

close(fd); /\* закрыть дескриптор конвейера\*/

}