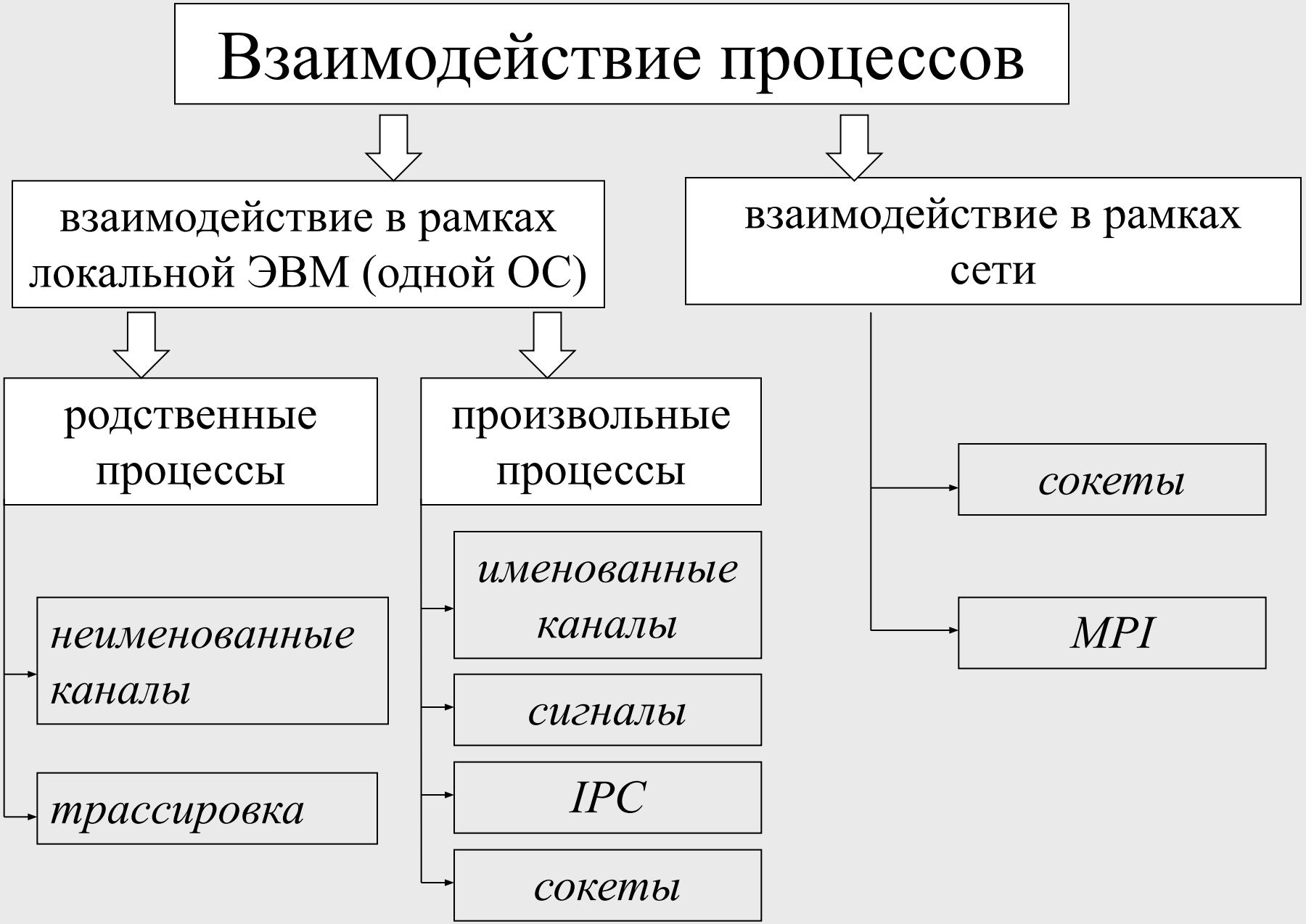
Реализация взаимодействия процессов

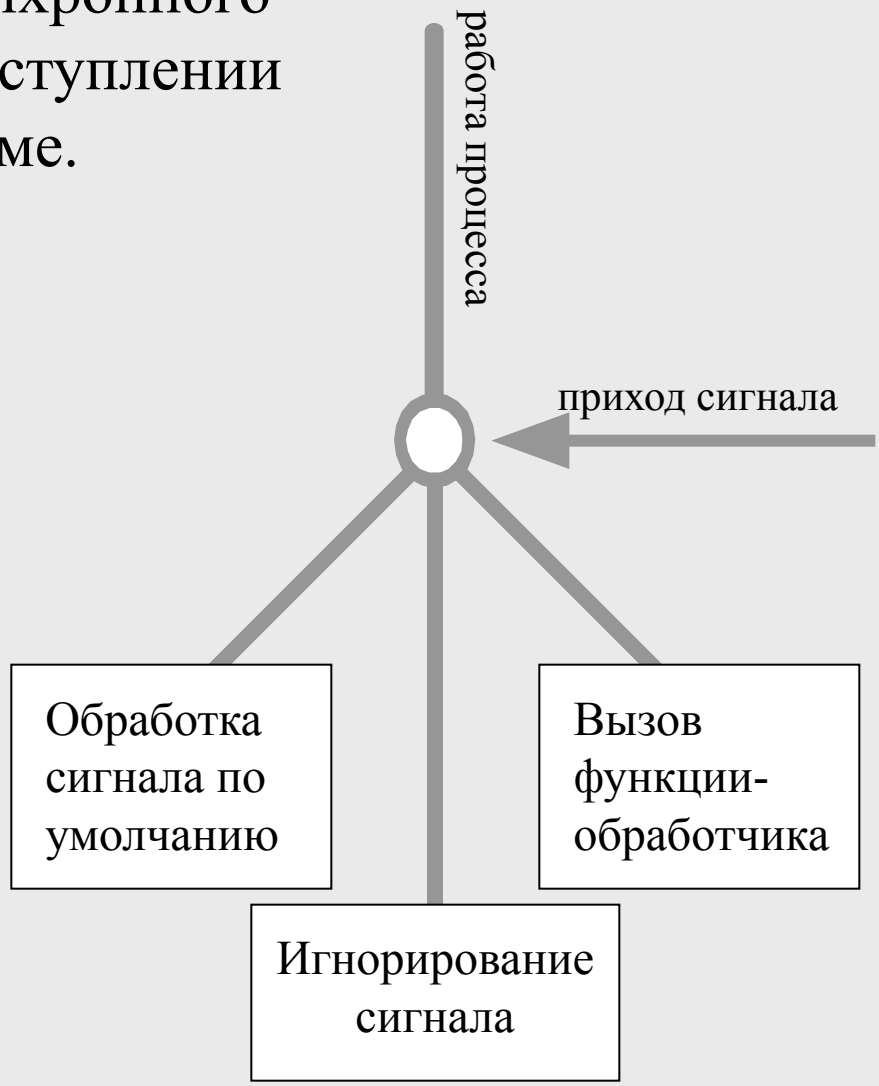

Взаимодействие процессов
взаимодействие в рамках локальной ЭВМ (одной ОС)
взаимодействие в рамках сети
родственные процессы
произвольные процессы
сокеты
именованные каналы
неименованные каналы
MPI
сигналы
трассировка
IPC
сокеты

Сигналы

Сигнал – средство асинхронного уведомления процесса о наступлении некоторого события в системе.

Примеры сигналов <signal.h>

- SIGINT (2)
- SIGQUIT (3)
- SIGKILL (9)
- SIGALRM (14)
- SIGCHLD (18)



Работа с сигналами

```
#include <sys/types.h>
#include <signal.h>
```

int kill (pit_t pid, int sig);

pid – идентификатор процесса, которому посылается сигнал

sig – номер посылаемого сигнала

При удачном выполнении возвращает 0, в противном случае возвращает -1

Работа с сигналами

#include <signal.h>

void (*signal (int sig, void (*disp) (int))) (int)

sig – номер сигнала, для которого устанавливается реакция

disp – либо определенная пользователем функция – обработчик сигнала, либо одна из констант:

SIG_DFL – обработка по умолчанию

SIG_IGN - игнорирование

При успешном завершении функция возвращает указатель на предыдущий обработчик данного сигнала.

Пример. Обработка сигнала.

```
#include <sys/types.h>
#include <signal.h>
#include <stdio.h>
int count=0;
```

```
int main(int argc, char **argv)
{  signal (SIGINT, SigHndlr);
   while (1);/*”тело программы”*/
   return 0;
}
```

```
void SigHndlr (int s)
{printf("\n I got SIGINT %d time(s) \n", count ++);
 if (count==5)
 signal (SIGINT, SIG_DFL); /*   ????    */
}
```

Пример. Программа “будильник”.

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

```
void alrm (int s)
{
  printf(“\n жду имя \n”);
    alarm(5);
}
```

```
int main(int argc, char **argv)
{  char s[80];
    signal(SIGALRM, alrm); alarm(5);
    printf(“Введите имя \n”);
    for (;;) {
      printf(“имя:”);
      if (gets(s) != NULL) break;
     };
    printf(“OK! \n”);
    return 0;
}
```

Пример. Двухпроцессный вариант программы “будильник”.

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

```
void alr(int s)
{
    printf(“\n Быстрее!!! \n”);
}
```

```
int main(int argc, char **argv)
{   char s[80];
    int pid;
    signal(SIGALRM, alr);
    if  (pid=fork()) {/*”отец”*/}
     else {/*”сын”*/}
    return 0;
}
```

Пример. Двухпроцессный вариант программы “будильник”.

```
/*”отец”*/

for (;;) {
    sleep(5);
    kill(pid, SIGALRM);
}
```

```
/*”сын”*/
printf(“Введите имя \n”);
for (;;) {
    printf(“имя:”);
    if (gets(s) != NULL) break;
}
printf(“OK!\n”);
kill(getppid(), SIGKILL);
```

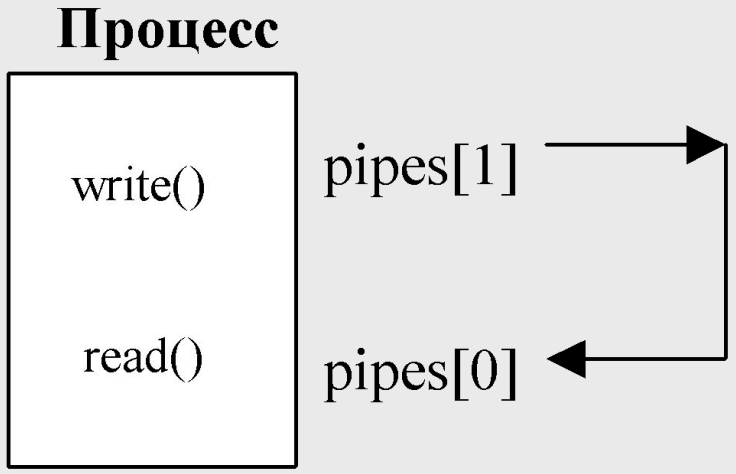
Неименованные каналы.

Неименованные каналы. Системный вызов pipe( )

#include <unistd.h>

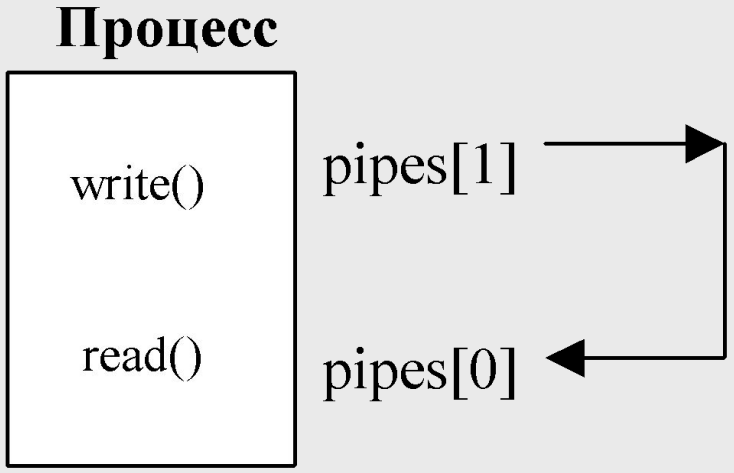
int pipe (int *pipes);

pipes[1] – запись в канал
pipes[0] – чтение из канала



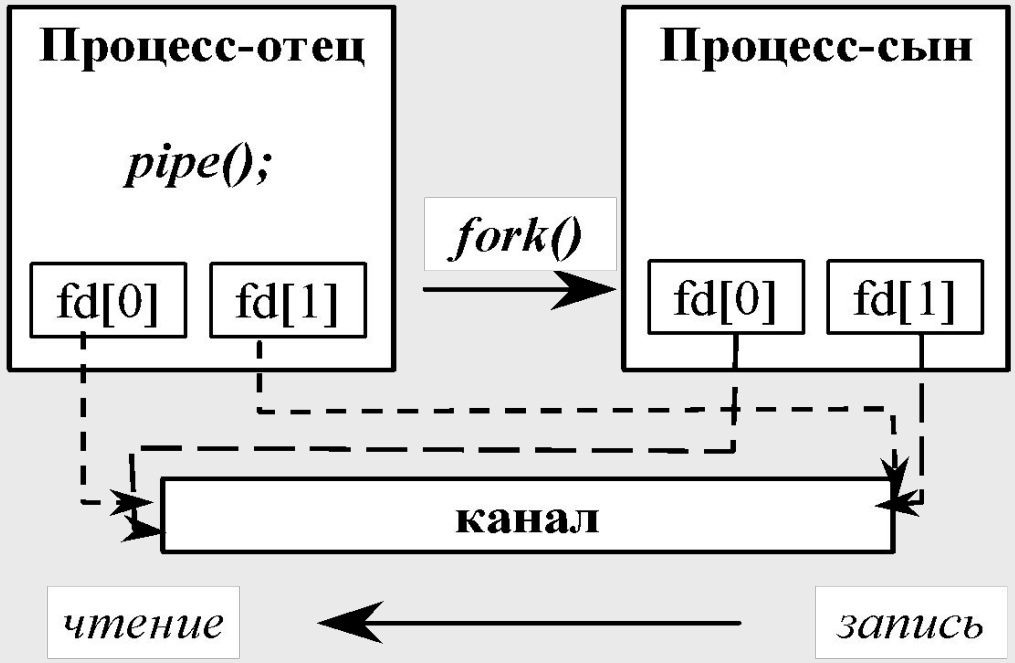
Пример. Использование канала.

```
int main(int argc, char **argv)
{
    char *s=”chanel”;
    char buf[80];
    int pipes[2];
    pipe(pipes);
    write(pipes[1],s,strlen(s)+1);
    read(pipes[0],buf,strlen(s)+1);
    close(pipes[0]);
    close(pipes[1]);
    printf(“%s\n”,buf);
}
```



Пример. Типовая схема взаимодействия процессов с использованием канала.

```
int main(int argc, char **argv)
{
    int fd[2];
    pipe(fd);
    if(fork()) {      close(fd[0]);
            write (fd[1], …);
            …
            close(fd[1]);
            …
             }
    else {close(fd[1]);
      while(read(fd[0],…)) {…}
     …
    }
}
```



Пример. Реализация конвейера.

```
#include <stdio.h>
int main(int argc, char **argv)
{
    int fd[2];

    pipe(fd);
    if(fork() == 0) {  dup2(fd[1],1);
            close(fd[1]);
            close(fd[0]);
            execl(“/usr/bin/print”,”print”,0);
            }
    dup2(fd[0],0);
    close(fd[0]);
    close(fd[1]);
    execl(“/usr/bin/wc”,”wc”,0);
}
```

Пример. Совместное использование сигналов и каналов – «пинг-понг».

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

#define MAX_CNT 100
int target_pid, cnt;
int fd[2];
int status;
```

Пример. Совместное использование сигналов и каналов – «пинг-понг».

```
void SigHndlr (int s)
{
   if (cnt < MAX_CNT)
   {
    read(fd[0], &cnt, sizeof(int));
    printf("%d \n", cnt);
    cnt++;
    write(fd[1], &cnt, sizeof(int));
    kill(target_pid, SIGUSR1);
   } …
```

Пример. Совместное использование сигналов и каналов – «пинг-понг».

```
… else
   if (target_pid == getppid()) /* процесс – сын*/
   {
    printf("Child is going to   be terminated\n");
    close(fd[1]);
    close(fd[0]);
    exit(0);
   } else /* процесс – родитель */
    kill(target_pid, SIGUSR1);
}
```

Пример. Совместное использование сигналов и каналов – «пинг-понг».

```
int main(int argc, char **argv)
{  pipe(fd);
   signal(SIGUSR1, SigHndlr);
   cnt = 0;

   if (target_pid = fork()) { /* процесс – родитель*/
    write(fd[1], &cnt, sizeof(int));
    while(wait(&status)== -1);
    printf("Parent is going to be terminated\n");
    close(fd[1]);
    close(fd[0]);
    return 0; …
```

Пример. Совместное использование сигналов и каналов – «пинг-понг».

```
…
    } else { /* процесс – сын */
     read(fd[0], &cnt, sizeof(int)); /* старт синхр*/
     target_pid = getppid();
     write(fd[1], &cnt, sizeof(int));
     kill(target_pid, SIGUSR1);
     for(;;);
    }
}
```

Именованные каналы.

Именованные каналы. Создание.

int mkfifo (char *pathname, mode_t mode);

pathname – имя создаваемого канала

mode – права доступа + режимы открытия

- блокировка при подключении
- использование флагов:
  - **O_RDONLY** открытие «на чтение»;
  - **O_RDWR** открытие «на чтение+запись»;
  - **O_NONBLOCK** – открытие без блокирования;
  - .....

Пример. «Клиент-сервер». Процесс-сервер:

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/file.h>
```

Пример. «Клиент-сервер». Процесс-сервер:

```
int main(int argc, char **argv)
{  int fd;
   int pid;

   mkfifo("fifo", FILE_MODE | 0666);
   fd = open ("fifo", O_RDONLY | O_NONBLOCK);
   while  ( read (fd, &pid, sizeof(int) ) != -1) {
    printf ("Server %d  got message from %d !\n",
                        getpid(), pid);
   …
   }
   close (fd);
   unlink ("fifo");
}
```

Пример. «Клиент-сервер». Процесс-клиент:

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/file.h>

int main(int argc, char **argv)
{
int fd;

int pid = getpid( );
fd = open ("fifo", O_RDWR);
write (fd, &pid, sizeof(int));
close (fd);
}
```

Взаимодействие «главный-подчинённый».

Главный - Подчиненный

#include <sys/ptrace.h>

int ptrace(int cmd, int pid, int addr, int data);

cmd – код выполняемой команды

pid – идентификатор процесса-потомка

addr – некоторый адрес в адресном пространстве процесса-потомка

data – слово информации.

Главный - Подчиненный

int ptrace(int cmd, int pid, int addr, int data);

cmd – код команды:

- **группа команд чтения** (сегмент кода, сегмент данных, контекст процесса)
- **группа команд записи** (сегмент кода, сегмент данных, контекст процесса)
- **группа команд управления** (продолжить выполнение, продолжить выполнение с заданного адреса, включить «шаговый режим», завершить процесс, разрешить трассировку)

Системный вызов ptrace()

```
#include <sys/ptrace.h>
int ptrace(int cmd, int pid, int addr, int data);
```

cmd=**PTRACE_TRACEME** вызывает сыновний процесс, позволяя трассировать себя

cmd=**PTRACE_PEEKDATA** чтение слова из адресного пространства отлаживаемого процесса

cmd=**PTRACE_PEEKUSER** чтение слова из контекста процесса (из пользовательской составляющей, содержащейся в <sys/user.h>)

cmd=**PTRACE_POKEDATA** запись данных в адресное пространство процесса-потомка

cmd=**PTRACE_POKEUSER** запись данных в контекст трассируемого процесса.

Системный вызов ptrace()

```
#include <sys/ptrace.h>
int ptrace(int cmd, int pid, int addr, int data);
```

cmd=**PTRACE_GETREGS,PTRACE_GETFREGS** чтение регистров общего назначения

cmd=**PTRACE_SETREGS,PTRACE_SETFREGS** запись в регистры общего назначения

cmd=**PTRACE_CONT** возобновление выполнения трассируемого процесса

cmd=**PTRACE_SYSCALL, PTRACE_SINGLESTEP** возобновляется выполнение трассируемой программы, но снова останавливается после выполнения одной инструкции

cmd=**PTRACE_KILL** завершение выполнения трассируемого процесса

Общая схема трассировки процессов

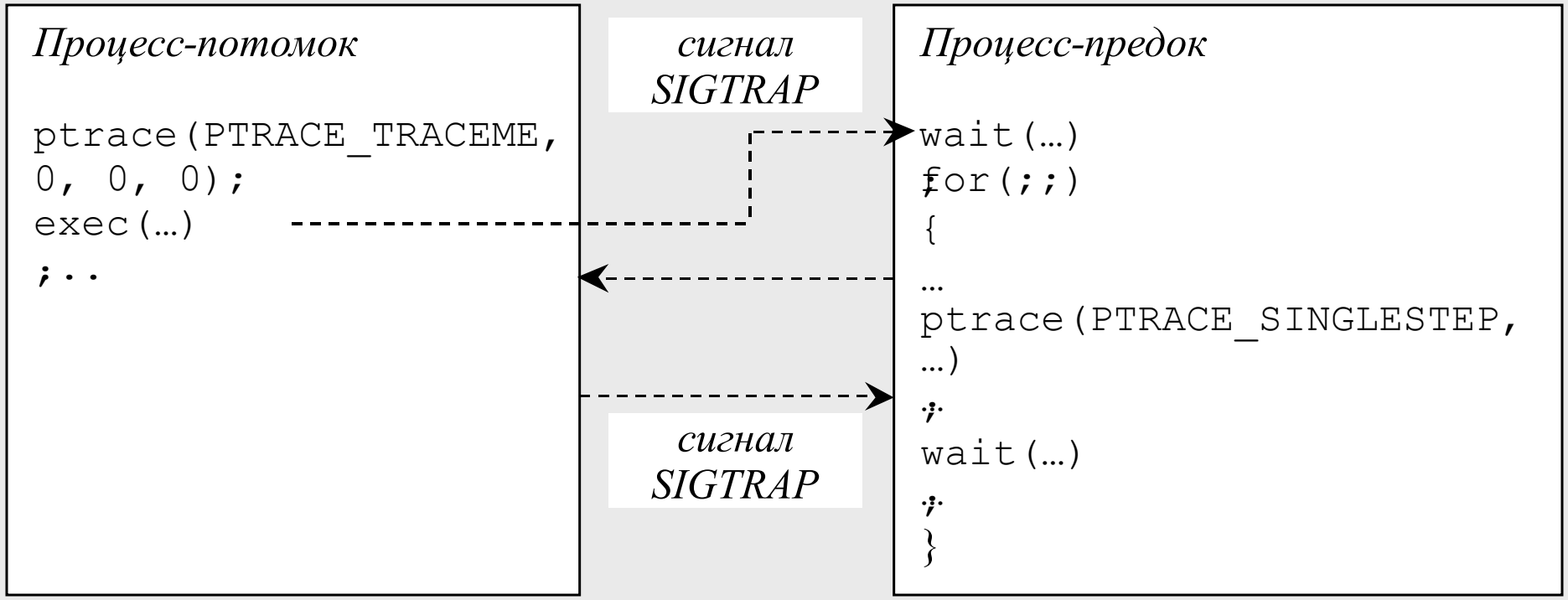


Схема установки контрольной точки по адресу A_{BrPnt}

Установка контрольной точки

Статус отлаживаемого процесса (ОП) **ВЫПОЛНЕНИЕ**

- послать **Sigtrap**
- ждем остановку ОП + анализ точки остановки (статус ОП **ОЖИДАНИЕ**)
- чтение в адресном пространстве ОП, сохранение ($\mathbf{N_{BrPnt}}$, $< A_{BrPnt} >$)
- запись **BrPnt** в A_{BrPnt}
- продолжить с точки останова

Приход в контрольную точку

Статус (ОП)

ВЫПОЛНЕНИЕ

- ждем остановки ОП, остановка (статус ОП **ОЖИДАНИЕ**)
- чтение информации из контекста, анализ точки остановки
- контрольная точка (совпадение адреса остановки + причины остановки)
- действия по отладке ОП в состоянии **ОЖИДАНИЯ**
- …….

Снятие контрольной точки

Статус (ОП) **ОЖИДАНИЕ**

- восстанавливаем содержимое A_{BrPnt} ($\mathbf{N_{BrPnt}}$, $< A_{BrPnt} >$)
- продолжить с адреса A_{BrPnt}

«Движение» через контрольную точку

Статус (ОП) **ОЖИДАНИЕ**

- восстанавливаем содержимое A_{BrPnt} ($\mathbf{N_{BrPnt}}$, $< A_{BrPnt} >$)
- включаем «шаговый» режим
- продолжить с адреса A_{BrPnt}
- ждем остановки ОП (анализ точки остановки)
- запись **BrPnt** в A_{BrPnt}
- продолжаем с точки остановки
-

Пример.

```
int main(int argc, char **argv)
{
   return argc/0;
}
```

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <signal.h>
#include <sys/ptrace.h>
#include <sys/user.h>
#include <sys/wait.h>
```

Пример.

```
int main(int argc, char *argv[])
{
    pid_t pid;
    int status;
    struct user_regs_struct REG;
    if  ((pid = fork()) == 0) {
     ptrace(PTRACE_TRACEME, 0, 0, 0);
     execl(“son”, ”son”, 0);
    }
    …
```

Пример.

```
…
    while (1) {
     wait( &status );
     ptrace(PTRACE_GETREGS, pid, &REG, &REG);
     printf("signal = %d, status = %#x, EIP=%#x,
    ESP=%#x\n“ , WSTOPSIG(status), status,          REG.eip,
    REG.esp);
     if (WSTOPSIG(status) != SIGTRAP) {
          if (!WIFEXITED(status))
               ptrace (PTRACE_KILL, pid, 0, 0);
          break;
     }
     ptrace (PTRACE_CONT, pid, 0, 0);
    }
}
```