Освоить порядок создания дочерних процессов с помощью функции fork(). Описать, что наследуется и что не наследуется при порождении процесса.

1. Освоить функции ожидания и определения статуса завершения дочернего процесса.
2. Определить, какую информацию о процессе можно получить из каталога /proc.
3. Освоить порядок действий для обработки события завершения процесса с помощью функции atexit().

# Порядок работы создания дочерних процессов с помощью функции fork()

Системный вызов fork() позволяет создать новый процесс, который называется дочерним, который выполняется одновременно с процессом, вызвавшим fork() (родительским). После создания дочернего процесса оба процесса будут выполнять следующую инструкцию после системного вызова fork().

Функция демонстрирующая порядок создания дочерних процессов с помощью системного вызова fork() представлена в листинге 1.

Листинг 1. Функция демонстрирующая порядок создания дочернего процесса с помощью fork()

void create\_child() {

atexit(my\_exit);

pid\_t pid = fork();

if(pid < 0) {

fprintf(stderr,"Error on fork: %s\n", strerror(errno));

exit(EXIT\_FAILURE);

}

if (pid == 0) {

printf("That's Child: PID - %d, PPID - %d\n", getpid(), getppid());

exit(EXIT\_SUCCESS);

}

else {

printf("That's Parent: PID - %d, child PID %d\n", getpid(), pid);

exit(EXIT\_SUCCESS);

}

}

При порождении дочернего процесса наследуется:

1. все открытые файлы и открытые каталоги;
2. установки umask;
3. текущий рабочий каталог;
4. корневой каталог;
5. текущее значение относительного приоритета;
6. управляющий терминал;
7. ID группы (pgid) и сеанса (sid) процесса;
8. сигнальная маска;
9. реальные ID пользователя и ID группы, а также дополнительный набор групп;
10. эффективные и сохраненные ID пользователя и групп;

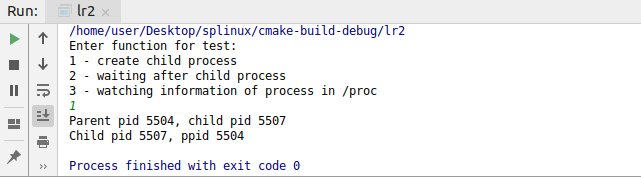


Рисунок 1 – Результат выполнения функции, создающей дочерний процесс с помощью fork()

При порождении дочернего процесса не наследуется:

1. ID процесса (pid);
2. ID родительского процесса (ppid);
3. ждущие сигналы, т.е. набор сигналов, ожидающих доставки в родительский процесс;
4. время посылки предупреждающего сигнала (устанавливается системным вызовом alarm, в порожденном процессе сбрасывается в ноль);
5. блокировки файлов;
6. статистика использования времени процессора;

# Функции ожидания и определения статуса завершенного дочернего процесса

Системный вызов wait(&status) приостанавливает выполнение вызывающего процесса, пока его дочерний процесс не завершится, значение status определяет статус завершенного процесса. Макросы WIFEXITED, WIFSIGNALED, WIFSTOPPED позволяют определить статус завершения дочернего процесса. В случае если WIFEXITED(status) вернуло TRUE значит дочерний процесс завершился сам с кодом возврата WIEXITSTATUS(status).В случае если WIFSIGNALED(status) вернуло TRUE значит дочерний процесс был завершен по сигналу, с номером сигнала, который завершил процесс – WTERMSIG(status). В случае если WIFSTOPPED(status) вернуло TRUE значит дочерний процесс был остановлен по сигналу, с номером сигнала, который остановил процесс WSTOPSIG(status).

Системный вызов waitpid(pid, &status, options) приостанавливает выполнение вызвавшего процесса до тех пор, пока не изменится состояние потомка, заданного аргументом pid.

Функция, демонстрирующая ожидание и определение статуса завершения дочернего процесса представлена на листинге 2.

Листинг 2. Функция, демонстрирующая ожидание и определение статуса завершения дочернего процесса.

void create\_and\_wait() {

pid\_t pid = fork();

if(pid < 0) {

fprintf(stderr,"Error on fork: %s\n",strerror(errno));

exit(EXIT\_FAILURE);

}

if (pid == 0) {

printf("That's Child: PID - %d, PPID - %d\n", getpid(), getppid());

sleep(60);

exit(EXIT\_SUCCESS);

}

else {

printf("That's Parent: PID - %d, child PID %d\n", getpid(), pid );

int status;

if (waitpid(pid,&status,WUNTRACED) < 0) {

perror("error on wait");

exit(EXIT\_FAILURE);

}

if (WIFEXITED(status))

printf("Child was exited with %d\n",WEXITSTATUS(status));

else if (WIFSTOPPED(status))

printf("Child was stopped by %d\n", WSTOPSIG(status));

else if (WIFSIGNALED(status))

printf("Child was killed by %d\n", WTERMSIG(status));

}

}

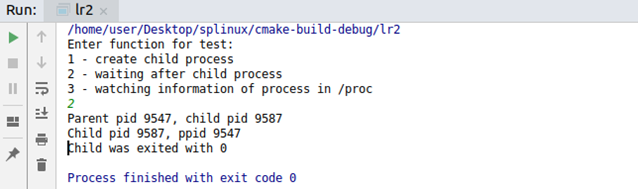


Рисунок 2.1 - Результат выполнения функции, при условии, что дочерний процесс завершился сам.

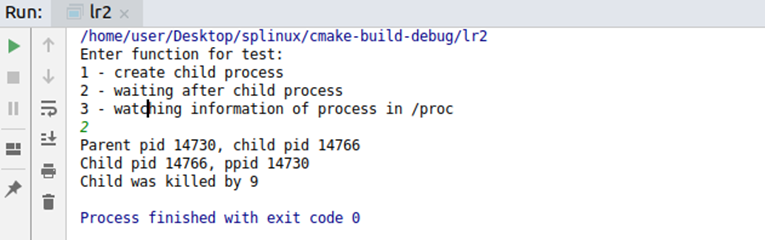


Рисунок 2.2 - Результат выполнения функции, при условии, что дочерний процесс был завершен по сигналу.

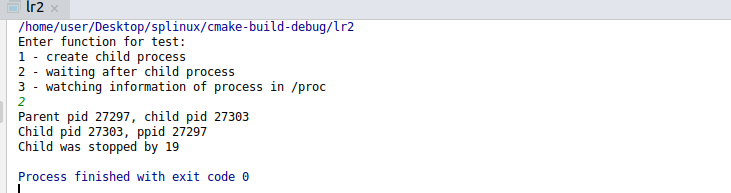


Рисунок 2.3 - Результат выполнения функции, при условии, что дочерний процесс был остановлен по сигналу.

# Информация о процессе из каталога /proc

Файловая система /proc является особой для Linux. Это виртуальная файловая система. Информация о процессах хранится в директориях /proc/N, где N - числовой идентификатор процесса. В этой директории содержатся различные псевдо-файлы, которые содержат информацию о самом процессе и связанном с ним окружении.

Листинг 3. Функция, демонстрирующая информацию о процессе из каталога /proc

void info\_of\_proc() {

pid\_t pid = fork();

int status;

if(pid < 0) {

perror("Error on fork");

exit(EXIT\_FAILURE);

}

if (pid == 0) {

printf("That's Child: PID - %d, PPID - %d\n", getpid(), getppid());

char str\_pid[6]; sprintf(str\_pid, "/%d", getpid());

char path[27]; sprintf(path,"/proc/%s/fd", str\_pid);

char \*args[] = { "ls", "-la", path, nullptr };

execve("/bin/ls", args, environ);

sleep(60);

}

wait(&status);

sleep(60);

if (WIFEXITED(status))

printf("Child was exited with %d\n",WEXITSTATUS(status));

else if (WIFSIGNALED(status))

printf("Child was killed by %d\n", WTERMSIG(status));

else if (WIFSTOPPED(status))

printf("Child was stopped by %d\n", WSTOPSIG(status));

}

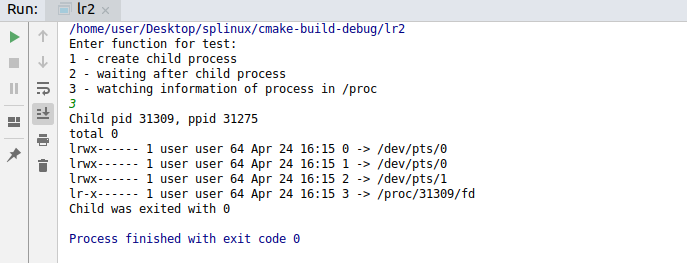


Рисунок 3 - Результат работы функции, в которой определяется информация о процессе из каталога /proc

Функция, демонстрирующая информацию из каталога /proc об используемых процессом файловых дескрипторов представлена на листинге 3.

# Порядок действий для обработки события по завершению процесса с помощью функции atexit()

Функция atexit(void(\*func)) регистрирует функцию, переданную в качестве параметра, которая будет выполняться при успешном завершении процесса.

Функция, в которой порожденный процесс записывает в файл информацию о своем pid и pid его родительского процесса, демонстрирующая вызов функции atexit() представлена в листинге 4.

Листинг 4. Функция, демонстрирующая вызов atexit()

void process\_write() {

atexit(close\_fdesc);

pid\_t pid = fork();

if(pid < 0) {

fprintf(stderr,"error on fork: %s\n",strerror(errno));

exit(EXIT\_FAILURE);

}

if (pid == 0) {

fdesc = open("/home/user/Desktop/test.txt",O\_RDWR | O\_CREAT | O\_TRUNC, 0666);

if(fdesc < 0) {

perror("Error on open\n");

exit(EXIT\_FAILURE);

}

char buff[33];

sprintf(buff, "%s%d%s%d", "child pid: ", getpid(), " parent pid: ", getppid());

write(fdesc, buff, sizeof(buff));

sleep(30);

exit(EXIT\_SUCCESS);

}

else {

printf("That's Parent: PID - %d, child PID %d\n", getpid(), pid );

int status;

if (waitpid(pid,&status,WUNTRACED) < 0) {

perror("Error on wait");

exit(1);

}

if (WIFEXITED(status))

printf("Child was exited with %d\n",WEXITSTATUS(status));

else if (WIFSIGNALED(status))

printf("Child was killed by %d\n", WTERMSIG(status));

else if (WIFSTOPPED(status))

printf("Child was stopped by %d\n", WSTOPSIG(status));

}

}

Функция, выполняемая при успешном завершении процесса и зарегистрированная функцией atexit() представлена в листинге 5.

Листинг 5. Функция, выполняемая при успешном завершении процесса, зарегистрированная функцией atexit()

void close\_fdesc()

{

close(fdesc);

printf("Process %d exit with closing file\n",getpid());

}

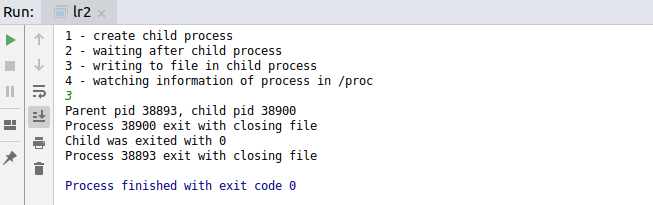


Рисунок 4 - Результат работы функции, которая завершается при успешном завершении процесса с помощью функции atexit().

# Вывод

1. Приобретены навыки по созданию дочерних процессов с помощью системного вызова fork() на языке C.
2. Получены знания о том, что наследуется и не наследуется при создании дочерних процессов.
3. Сформированы навыки по работе с функциями, ожидающими и определяющими статус завершения процесса на языке C.
4. Освоена информация, которую можно получить о процессе из каталога /proc.
5. Освоен порядок действий для обработки события завершения процесса на языке C с помощью системного вызова atexit().
6. Приобретены навыки по работе с системным вызовом exec() на языке C.