# Задание на семинар + домашняя работа

## Теоретическая часть

Для порождения нового процесса (процесс-потомок) используется системный вызов **fork()**. Формат вызова:

**int fork()**

Порожденный таким образом процесс представляет собой точную копию своего процесса-предка. Единственное различие между ними заключается в том, что процесс-потомок в качестве возвращаемого значения системного вызова **fork()** получает 0, а процесс-предок - идентификатор процесса-потомка. Кроме того, процесс-потомок наследует и весь контекст программной среды, включая дескрипторы файлов, каналы и т.д. Наличие у процесса идентификатора дает возможность и ОС UNIX, и любому другому пользовательскому процессу получить информацию о функционирующих в данный момент процессах.

Ожидание завершения процесса-потомка родительским процессом выполняется с помощью системного вызова wait()

**int wait(int \*status)**

В результате осуществления процессом системного вызова **wait()** функционирование процесса приостанавливается до момента завершения порожденного им процесса-потомка. По завершении процесса-потомка процесс-предок пробуждается и в качестве возвращаемого значения системного вызова **wait()** получает идентификатор завершившегося процесса-потомка, что позволяет процессу-предку определить, какой из его процессов-потомков завершился (если он имел более одного процесса-потомка). Аргумент системного вызова **wait()** представляет собой указатель на целочисленную переменную status, которая после завершения выполнения этого системного вызова будет содержать в старшем байте код завершения процесса-потомка, установленный последним в качестве системного вызова exit(), а в младшем - индикатор причины завершения процесса-потомка.

Формат системного вызова **exit()**, предназначенного для завершения функционирования процесса:

**void exit(int status)**

Аргумент status является статусом завершения, который передается родителю процесса, если он выполнял системный вызов **wait()**.

Для получения собственного идентификатора процесса используется системный вызов **getpid()**, а для получения идентификатора процесса-родителя - системный вызов **getppid()**:

**int getpid()**

**int getppid()**

Вместе с идентификатором процесса каждому процессу в ОС UNIX ставится в соответствие также идентификатор группы процессов. В группу процессов объединяются все процессы, являющиеся процессами-потомками одного и того же процесса. Организация новой группы процессов выполняется системным вызовом **setpgrp()**, а получение собственного идентификатора группы процессов - системным вызовом **getpgrp()**. Их формат:

**int setpgrp()**

**int getpgrp()**

С практической точки зрения в большинстве случаев в рамках порожденного процесса загружается для выполнения программа, определенная одним из системных вызовов **execl()**, **execv()**,... Каждый из этих системных вызовов осуществляет смену программы, определяющей функционирование данного процесса:

**execl(char \*name, \*arg0, \*arg1,...,\*argn,0)**

**execv(char \*name, \*argv[])**

**execle(char \*name, \*arg0, \*arg1,...,\*argn,0, \*envp[])**

**execve(char \*name, \*arg[],\*envp[])**

Сигналы - это программное средство, с помощью которого может быть прервано функционирование процесса в ОС UNIX. Механизм сигналов позволяет процессам реагировать на различные события, которые могут произойти в ходе функционирования процесса внутри него самого или во внешнем мире. Каждому сигналу ставятся в соответствие номер сигнала и строковая константа, используемая для осмысленной идентификации сигнала. Эта взаимосвязь отображена в файле описаний <signal.h>. Для посылки сигнала используется системный вызов, имеющий формат

**void kill(int pid, int sig)**

В результате осуществления такого системного вызова сигнал, специфицированный аргументом sig, будет послан процессу, который имеет идентификатор pid. Если pid не превосходит 1, сигнал будет послан целой группе процессов.

Использование системного вызова **signal()** позволяет процессу самостоятельно определить свою реакцию на получение того или иного события (сигнала):

**int \*signal(**int sig, int (\*func)()**) ()**

;

Реакцией процесса, осуществившего системный вызов **signal()** с аргументом func, при получении сигнала sig будет вызов функции func().

Системный вызов **pause()** позволяет приостановить процесс до тех пор, пока не будет получен какой-либо сигнал:

**void pause()**

Системный вызов **alarm(n)** обеспечивает посылку процессу сигнала SIGALARM через n секунд.

В ОС UNIX существует специальный вид взаимодействия между процессами - программный канал. Программный канал создается с помощью системного вызова **pipe()**, формат которого

**int fd[2];**

**pipe(fd)**

# Задание

Разработать программу, вычисляющую плотность нормального распределения в точке x по формуле f(x)=Exp(-x2/2)/Sqrt(2\*Пи).

Для нахождения Пи и Exp(-x2/2) программа должна породить два параллельных процесса, вычисляющих эти величины путем разложения в ряд по формулам вычислительной математики.

<http://ru.wikipedia.org/wiki/Pi>


e^x  = 1 + x + {{x^2 } \over {2!}} + {{x^3 } \over {3!}} + ...
\,\!


e^{ - x}  = 1 - x + {{x^2 } \over {2!}} - {{x^3 } \over {3!}} + ...
\,\!


e^x  = {1 \over {e^{ - x} }} = {1 \over {1 - x + {{x^2 } \over {2!}} - {{x^3 } \over {3!}} + ...}}
\,\!

Разработать программу, с учетом следующих требований:

1. все действия, относящиеся как к родительскому процессу, так и к порожденным процессам, выполняются в рамках одного исполняемого файла;
2. обмен данными между процессом-родителем и процессом-потомком предлагается выполнить посредством временного файла: процесс-родитель после порождения процесса-потомка постоянно опрашивает временный файл, ожидая появления в нем информации от процесса-потомка;
3. если процессов-потомков несколько, и все они подготавливают некоторую информацию для процесса-родителя, каждый из процессов помещает в файл некоторую структурированную запись, при этом в этой структурированной записи содержатся сведения о том, какой процесс посылает запись, и сама подготовленная информация.

# Домашняя работа

После отладки данной программы Модифицировать ее с учетом следующих требований:

1. действия процесса-потомка реализуются отдельной программой, запускаемой по одному из системных вызовов execl(), execv() и т.д. из процесса-потомка;
2. процесс-потомок, после порождения, должен начинать и завершать свое функционирование по сигналу, посылаемому процессом-предком (это же относится и к нескольким процессам-потомкам);
3. обмен данными между процессами необходимо осуществить через программный канал.