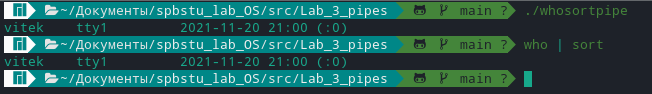
# **Лабораторная работа №3 «ПРОГРАММНЫЕ КАНАЛЫ»**

**Цель работы**

Изучение конвейеров (pipes, программных каналов), как простейшего средства коммуникации запущенных процессов. Исследование различных способов организации каналов и их сопоставление.

**Последовательность выполнения работы:**

1. Войдите в систему и скопируйте с разделяемого ресурса в свой HOMEкаталог набор исходных файлов для третьего занятия.
2. Скомпилируйте и выполните программу whosortpipe.cpp. Сопоставьте результат выполнения программы с выполнением этих же двух команд из shell в конвейерном режиме ( | ). Не забывайте приводить в отчете анализ результатов работы этой программы (как и всех последующих) с соответствующими скриншотами.

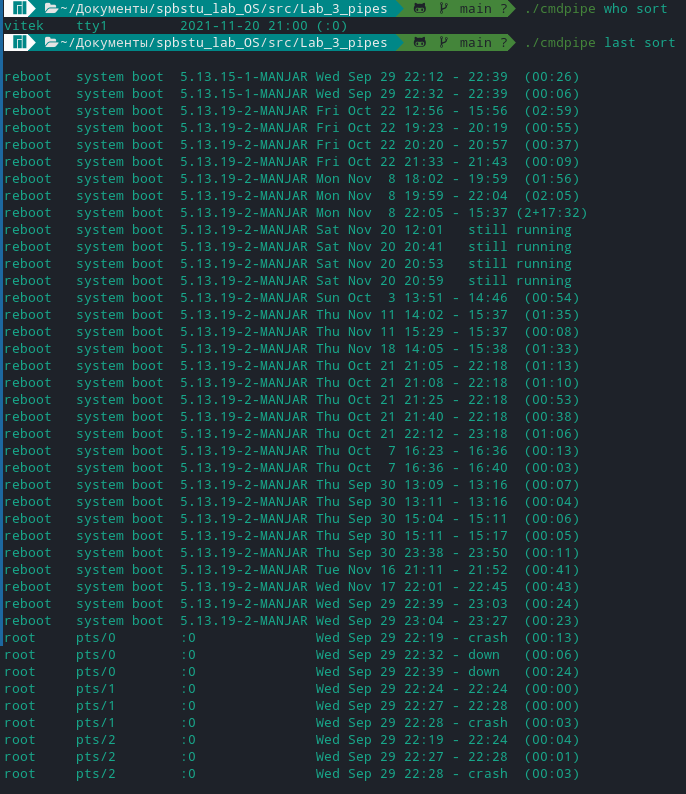


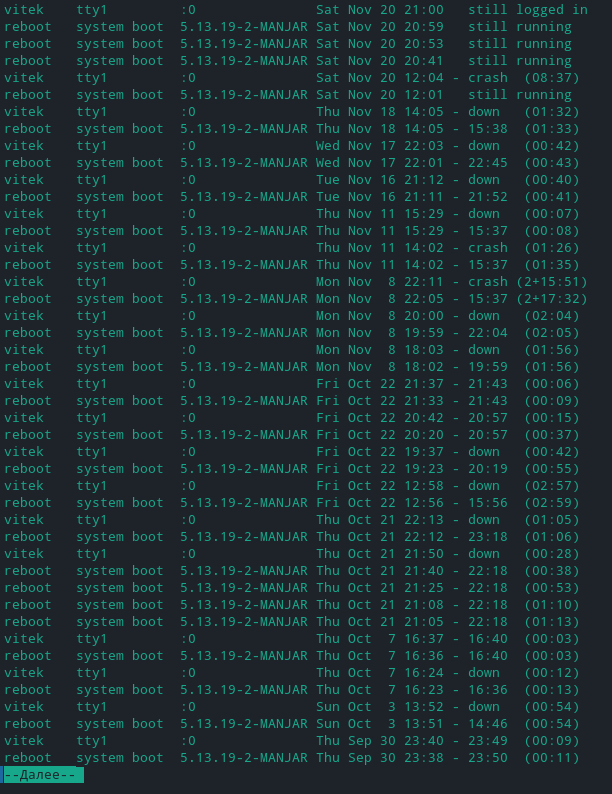
*sort* - простая и очень полезная команда, которая меняет порядок строк в текстовом файле, то есть осуществляет их сортировку по алфавиту или в соответствии с числовыми значениями.

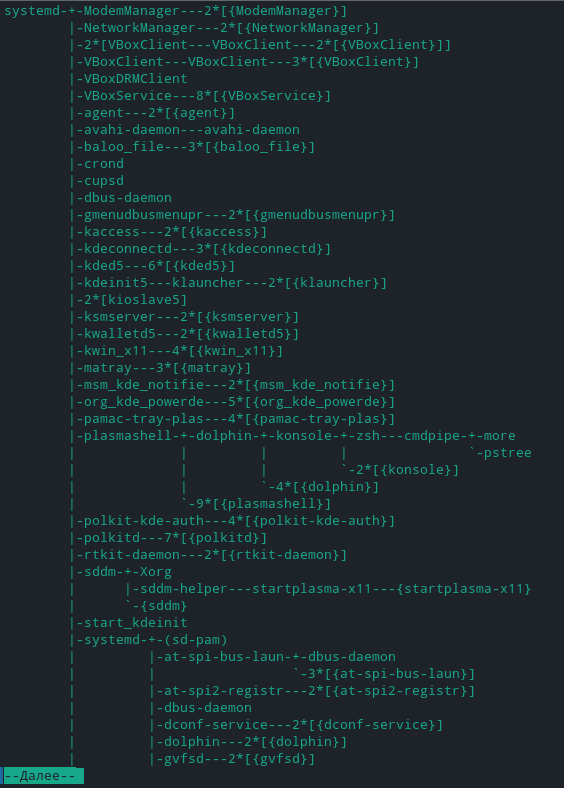
**$ sort опции файл или $ команда | sort опции**

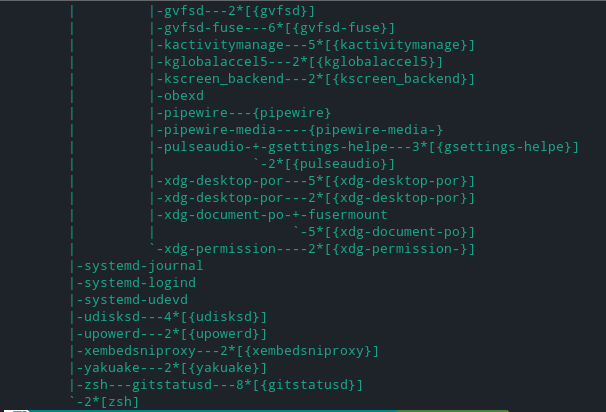
*who -* вывести информацию о пользователях, вошедших в систему.

1. Программу cmdpipe.cpp запускайте после компиляции, задавая ей при стартах в качестве параметров командной строки пары команд shell для конвейеризации (who и sort; last и sort; last и more; pstree и more). Сопоставьте результаты запусков программы с выполнением тех же пар команд из shell в конвейерном режиме. Можно ли с помощью вызова popen() создать программу, организующую конвейер из трех команд shell, передаваемых ей в качестве параметров командной строки при запуске? Если да, то создайте такую программу, если нет, дайте обоснованный ответ, почему нельзя.









*more -* предназначена для постраничного просмотра файлов в терминале Linux. Своим названием она обязана надписи more (в русскоязычном варианте — дальше), появляющейся внизу каждой страницы.

**$ more опции файл**

*last* - утилита командной строки, которая отображает информацию о последних сеансах входа пользователей системы. Это очень полезно, когда вам нужно отслеживать активность пользователя или расследовать возможное нарушение безопасности.

**$ last опции пользователь <TTY>…**

*less -* позволяет перематывать текст не только вперёд, но и назад, осуществлять поиск в обоих направлениях, переходить сразу в конец или в начало файла.

**$ less опции файл**

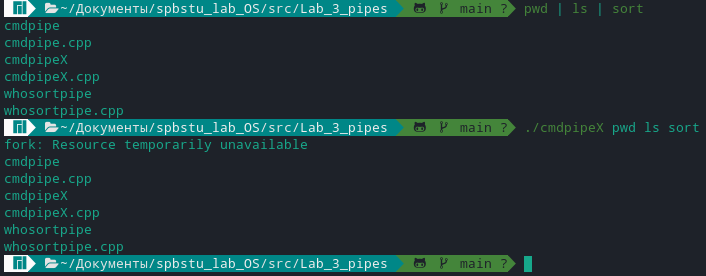
С помощью вызова popen() нельзя создать программу, организующую конвейер из трех команд shell, т.к. метод popen() с аргументом w может только подавать данные на вход указанной команды, но не может считывать ее вывод (результат работы команды автоматически подается на стандартный вывод).

1. Напишите программу (например, на основе вызовов pipe()), воспринимающую варьируемое количество команд, передаваемых ей при запуске в качестве параметров. Каждая последующая команда должна быть соединена с предыдущей с помощью конвейера. Так, при запуске программы

$ ./a.out last sort more

должны выполняться действия, эквивалентные запуску команд из shell:

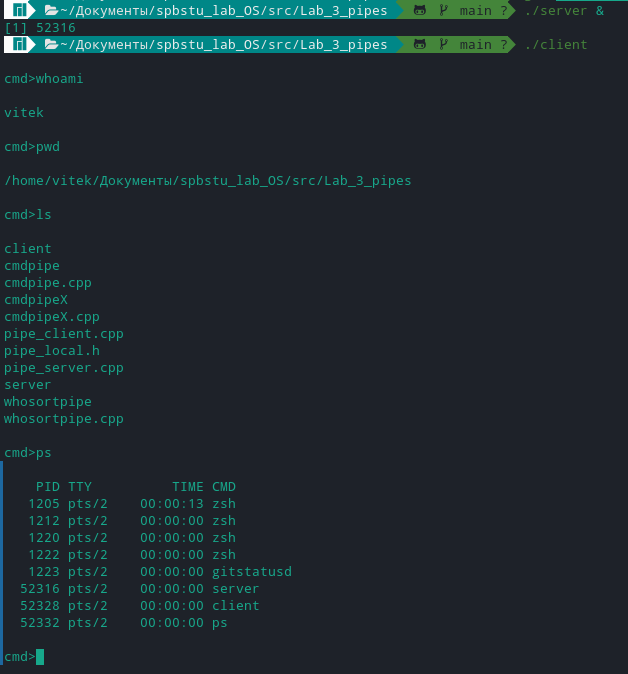
$ last | sort | more.



Программа:

|  |
| --- |
| /\*  \*\* pipex.c - multipipes support  \*/  #include <stdlib.h>  #include <unistd.h>  #include <stdio.h>  #include <sys/wait.h>  /\*  \* loop over commands by sharing  \* pipes.  \*/  static void  pipeline(char \*\*\*cmd)  {  int fd[2];  pid\_t pid;  int fdd = 0; /\* Backup \*/  while (\*cmd != NULL) {  pipe(fd);  if ((pid = fork()) == -1) {  perror("fork");  exit(1);  }  else if (pid == 0) {  dup2(fdd, 0);  if (\*(cmd + 1) != NULL) {  dup2(fd[1], 1);  }  close(fd[0]);  execvp((\*cmd)[0], \*cmd);  exit(1);  }  else {  wait(NULL); /\* Collect childs \*/  close(fd[1]);  fdd = fd[0];  cmd++;  }  }  }  /\*  \* Compute multi-pipeline based  \* on a command list.  \*/  int  main(int argc, char \*argv[])  {  char \*temp[argc][2];  for (int i = 0; i < argc; i++)  {  temp[i][0] = argv[i];  temp[i][1] = NULL;  }  char \*ls[] = {"ls", NULL};  char \*rev[] = {"rev", NULL};  char \*nl[] = {"nl", NULL};  char \*cat[] = {"cat", "-e", NULL};  char \*\*cmd[argc + 1];  for (int i = 0; i < argc; i++)  {  cmd[i] = temp[i];  }  cmd[argc] = NULL;  pipeline(cmd);  return (0);  } |

1. Разберите и выполните пример клиент-серверного взаимодействия, организованного на конвейерах различного типа. Исходный текст примера содержится в файлах pipe\_server.cpp, pipe\_client.cpp и pipe\_local.h и разобран в материалах лекций. Сервер запускается в фоновом режиме. Проанализируйте результаты функционирования данной системы и ее недостатки. Программа-сервер этого примера исполняет каждый командный запрос поочередно. Если какой-либо запрос потребует много времени, все остальные клиентские процессы будут ожидать обслуживания.



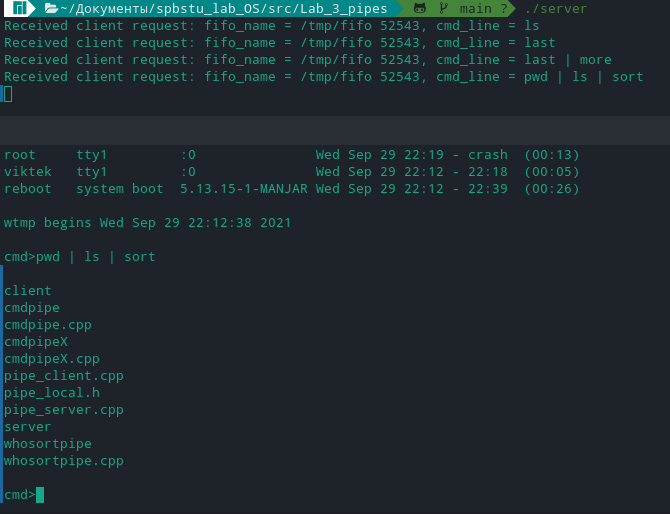
Так как сервер выполняет команды последовательно, то общая скорость работы программы зависит от скорости выполнения каждого из процессов. Такая пропорциональность времени отклика сервера чревата его чрезмерной нагрузкой при трудоемких запросах.

Также, если не запустить сервер, то клиент упадёт.

1. Модифицируйте программу pipe\_server.cpp так, чтобы при получении нового сообщения от очередного клиента сервер порождал очередной дочерний процесс для выполнения задачи обслуживания данного запроса (выполнения переданной от клиента команды и переправки клиенту результата).

Измененная программа pipe\_server , при получении сообщения , операция выполняется и выводится в процессе потомке.

|  |
| --- |
| /\* The server program pipe\_server.cpp \*/  #include"pipe\_local.h"  #include <sys/wait.h>  int publicfifo;  void sigint\_handler(int sig) {  if (sig == SIGINT) {  close(publicfifo);  remove(PUBLIC);  exit(0);  }  }  int main(void)  {  const int max\_forks = 10;  int n, done, dummyfifo, privatefifo;  static char buffer[PIPE\_BUF];  FILE \*fin;  struct message msg;  /\* Generate the public FIFO \*/  mknod(PUBLIC, S\_IFIFO | 0666, 0);  struct sigaction sa;  sa.sa\_handler = sigint\_handler;  sa.sa\_flags = 0;  sigemptyset(&sa.sa\_mask);  if (sigaction(SIGINT, &sa, NULL) == -1) {  perror("sigaction");  exit(1);  }  /\* OPEN the public FIFO for reading and writing \*/  if ((publicfifo=open(PUBLIC, O\_RDONLY))==-1 ||  (dummyfifo=open(PUBLIC, O\_WRONLY | O\_NDELAY))==-1){  perror(PUBLIC);  exit(1);  }  /\* Message can be read from the PUBLIC pipe \*/  int fork\_count = 0;  int pid = 0;  while(read(publicfifo, (char \*) &msg, sizeof(msg))>0){  if (fork\_count < max\_forks) {  pid = fork();  if (pid == 0) {  fork\_count++;  n = done = 0; /\* Clear counters | flags \*/  printf("Received client request: fifo\_name = %s, cmd\_line = %s", msg.fifo\_name, msg.cmd\_line);  do{ /\* Try OPEN of private FIFO \*/  if ((privatefifo=open(msg.fifo\_name, O\_WRONLY | O\_NDELAY))==-1)  sleep(3); /\* Sleep a while \*/  else{ /\* OPEN succesful \*/  fin = popen(msg.cmd\_line, "r"); /\* Execute the cmd \*/  write(privatefifo, "\n", 1); /\* Keep output pretty \*/  while((n=read(fileno(fin), buffer, PIPE\_BUF))>0){  write(privatefifo, buffer, n); /\*to private FIFO \*/  memset(buffer, 0x0, PIPE\_BUF); /\* Clear it out \*/  }  pclose(fin);  close(privatefifo);  done = 1; /\* Record succes \*/  }  }while(++n<5 && !done);  if(!done) /\* Indicate failure \*/  write(fileno(stderr), "\nNOTE: SERVER \*\* NEVER \*\* accessed private FIFO\n", 48);  } else {  wait(0);  fork\_count--;  }  }  }  return 0;  } |



**Вывод**

В ходе лабораторной работы были исследованы способы организации каналов. Изучено понятие конвейер и его свойства. Получены навыки запуска команд по конвейеру и была написана программа для запуска конвейером нескольких команд. Также разобран пример клиент-серверного взаимодействия.