

Лабораторная работа №14

Дисциплина: Операционные системы

Аксенова Анастасия

Содержание

1	Цель работы	5
2	Задание	6
3	Выполнение лабораторной работы	7
4	Библиография	16
5	Выводы	17

List of Tables

List of Figures

3.1	Создание файлов	7
3.2	Прграмма в файле common.h	8
3.3	Прграмма в файле server.c	8
3.4	Прграмма в файле server.c	9
3.5	Прграмма в файле client.c	9
3.6	Прграмма в файле client.c.....	10
3.7	Прграмма в Mikefile	10
3.8	Команда make all	11
3.9	Команда make all	11
3.10	Проверка длительности работы сервера.....	12

1 Цель работы

Цель данной лабораторной работы — Приобретение практических навыков работы с именованными каналами.

2 Задание

1. Сделать отчёт по лабораторной работе №15 в формате Markdown.
2. Приобрести практические навыки работы с именованными каналами.

3 Выполнение лабораторной работы

1). Для начала я создала необходимые файлы с помощью команды «touch common.h server.c client.c Makefile» и открыла редактор emacs для их редактирования (Рисунок 3.1). Вся необходимая информация про создания каталогов указана в следующем источнике: Программное обеспечение GNU/Linux. Лекция 10. Минимальный набор знаний (Г. Курячий, МГУ)

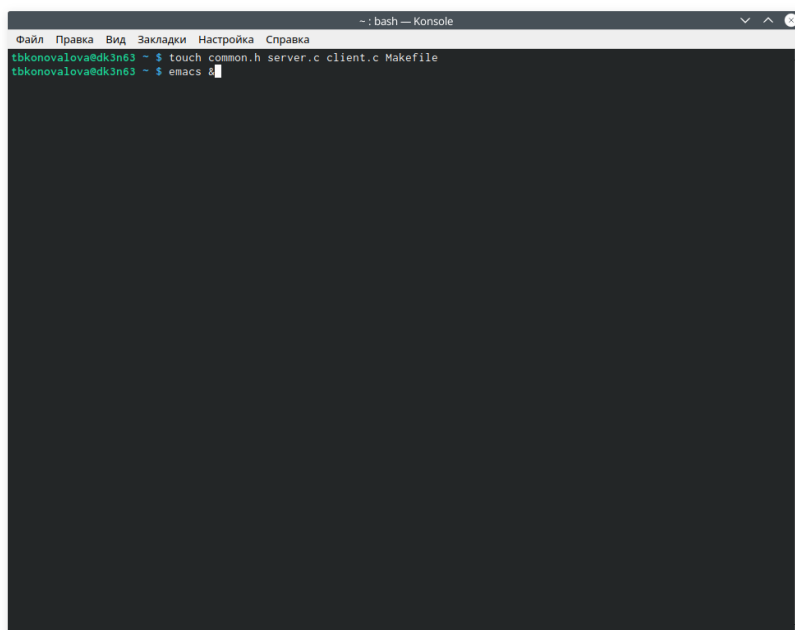


Figure 3.1: Создание файлов

2). Далее я изменила коды программ, представленных в тексте лабораторной работы. В файл common.h добавила стандартные заголовочные файлы unistd.h и time.h, необходимые для работы кодов других файлов. Common.h предназначен для заголовочных файлов, чтобы в остальных программах их не прописывать

каждый раз (Программа представлена на рис. 3.2). Вся необходимая информация про коды программ указана в следующем источнике: Программное обеспечение GNU/Linux. Лекция 12. Выбор дистрибутива (Г. Курячий, МГУ)

Figure 3.2: Прграмма в файле common.h

В файл server.c добавила цикл while для контроля за временем работы сервера. Разница между текущим временем time(NULL) и временем начала работы clock_t start=time(NULL) (инициализация до цикла) не должна превышать 30 секунд (алгоритм действий представлен на рис. 3.3 , 3.4).

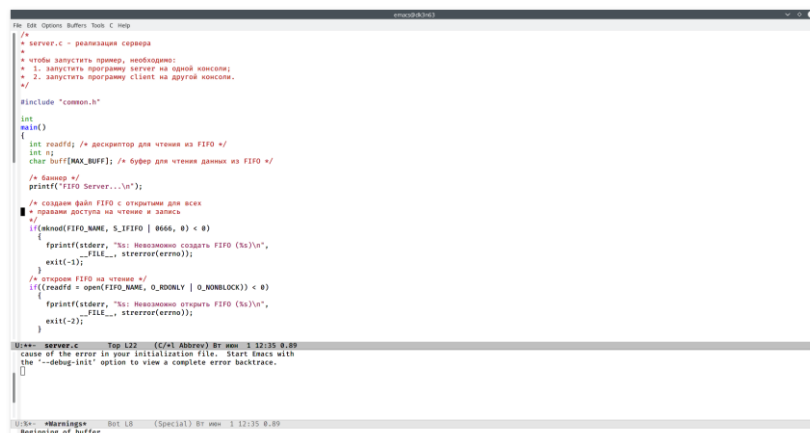


Figure 3.3: Прграмма в файле server.c

8


```

1  exit(-2);
2  }
3  /* начало отсчета времени */
4  clock_t start = time(NULL);
5  /* цикл работает, пока с момента начала отсчета времени прошло меньше 30 секунд */
6  while (time(NULL) - start < 30)
7  {
8      /* чтение данных из FIFO и вывод на экран */
9      while ((n = read(readfd, buff, MAX_BUF)) > 0)
10     {
11         if (write(1, buff, n) != n)
12         {
13             fprintf(stderr, "За: Ошибка вывода (%s)\n",
14                     strerror(errno));
15             exit(-3);
16         }
17     }
18 }
19 close(readfd); /* закроем FIFO */
20 /* удаление FIFO из системы */
21 if (unlink(FIFO_NAME) < 0)
22 {
23     fprintf(stderr, "За: Невозможно удалить FIFO (%s)\n",
24             FIFO_NAME);
25     exit(-4);
26 }
27 exit(0);
28 }
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```

Figure 3.6: Программа в файле client.c

Makefile (файл для сборки) не изменяла (Рисунок 3.7).

```

1  all: server client
2
3  server: server.c common.h
4      gcc server.c -o server
5
6  client: client.c common.h
7      gcc client.c -o client
8
9  clean:
10     rm server client *.o
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```

Figure 3.7: Программа в Makefile

3). После написания кодов, я, используя команду «make all», скомпилировала необходимые файлы (Скриншот 3.8). Подробную информацию по данной теме можно найти в следующих интернет-источниках: <https://habr.com/ru/post/122108/>, https://www.opennet.ru/docs/RUS/linux_parallel/node17.html

```
tbkonovalova@dk3n63 ~$ touch common.h server.c client.c Makefile
tbkonovalova@dk3n63 ~$ emacs &
[1] 15479
tbkonovalova@dk3n63 ~$ emacs &
[2] 16784
[1] Завершён      emacs
tbkonovalova@dk3n63 ~$ make all
gcc server.c -o server
gcc client.c -o client
client.c: В функции «main»:
client.c:34:17: ошибка: expected «=», «.», «;», «asm» or «__attribute__» before «-» token
   34 |         char* text=ctime(&time);
      |                ^
client.c:34:17: ошибка: неверный тип аргумента для унарного минуса
client.c:37:23: ошибка: «text» не описан (первое использование в этой функции)
   37 |         msglen = strlen(text);
      |                       ^
client.c:37:23: замечание: сообщение о каждом неопisanном идентификаторе выдается один раз в каждой функции, где он вс
тречается
make: *** [Makefile:7: client] Ошибка 1
tbkonovalova@dk3n63 ~$ emacs &
[3] 17783
[2] Завершён      emacs
tbkonovalova@dk3n63 ~$ make all
gcc client.c -o client
tbkonovalova@dk3n63 ~$ make all
make: Цель «all» не требует выполнения команд.
tbkonovalova@dk3n63 ~$
```

Figure 3.8: Команда make all

Далее я проверила работу написанного кода. Открыла 3 консоли (терминала) и запустила: в первом терминале – «./server», в остальных двух – «./client». В результате каждый терминал-клиент вывел по 4 сообщения. Спустя 30 секунд работа сервера была прекращена (Рисунок 3.9). Программа работает корректно.

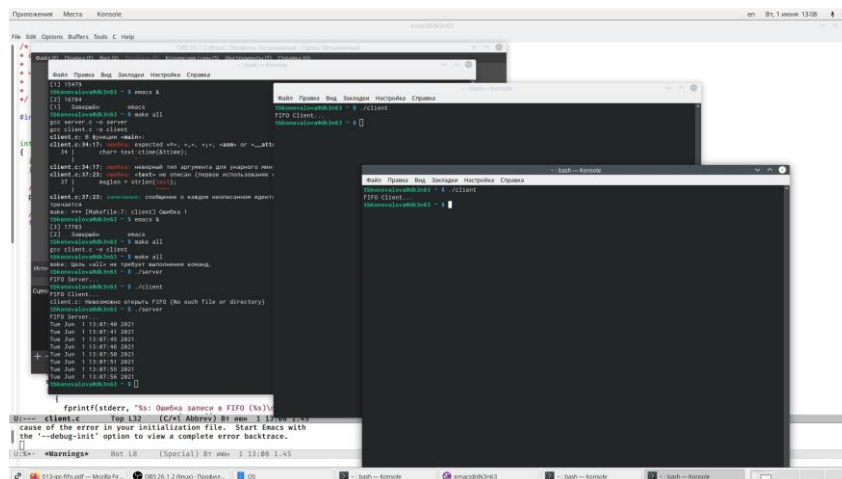
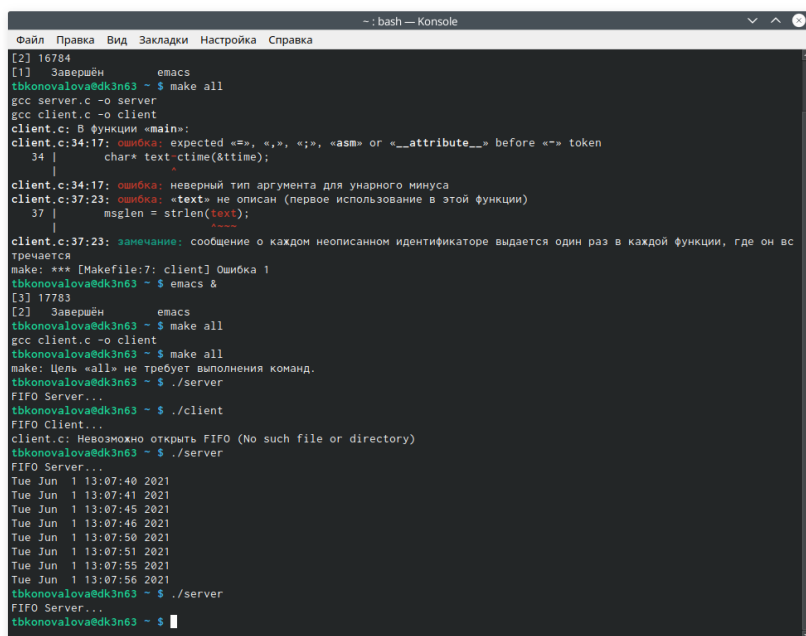


Figure 3.9: Команда make all

Также я отдельно проверила длительность работы сервера, введя команду

«./server» в одном терминале. Он завершил свою работу через 30 секунд (алгоритм действий представлен на рис. 3.10). Если сервер завершит свою работу, не закрыв канал, то, когда мы будем запускать этот сервер снова, появится ошибка «Невозможно создать FIFO», так как у нас уже есть один канал.



```
~: bash — Konsole
Файл  Правка  Вид  Закладки  Настройка  Справка
[2] 16784
[1] Завершён      emacs
tbkonovalova@dk3n63 ~ $ make all
gcc server.c -o server
gcc client.c -o client
client.c: В функции «main»:
client.c:34:17: ошибка: expected «=», «>», «<», «asm» or «__attribute__» before «-» token
   34 |         char* text = ctime(&time);
      |                   ^
client.c:34:17: ошибка: неверный тип аргумента для унарного минуса
client.c:37:23: ошибка: «text» не описан (первое использование в этой функции)
   37 |         msglen = strlen(text);
      |                       ^~~~~
client.c:37:23: замечание: сообщение о каждом неопisanном идентификаторе выдается один раз в каждой функции, где он вс
тречается
make: *** [Makefile:7: client] Ошибка 1
tbkonovalova@dk3n63 ~ $ emacs &
[3] 17783
[2] Завершён      emacs
tbkonovalova@dk3n63 ~ $ make all
gcc client.c -o client
tbkonovalova@dk3n63 ~ $ make all
make: Цель «all» не требует выполнения команд.
tbkonovalova@dk3n63 ~ $ ./server
FIFO Server...
tbkonovalova@dk3n63 ~ $ ./client
FIFO Client...
client.c: Невозможно открыть FIFO (No such file or directory)
tbkonovalova@dk3n63 ~ $ ./server
FIFO Server...
Tue Jun 1 13:07:40 2021
Tue Jun 1 13:07:41 2021
Tue Jun 1 13:07:45 2021
Tue Jun 1 13:07:46 2021
Tue Jun 1 13:07:50 2021
Tue Jun 1 13:07:51 2021
Tue Jun 1 13:07:55 2021
Tue Jun 1 13:07:56 2021
tbkonovalova@dk3n63 ~ $ ./server
FIFO Server...
tbkonovalova@dk3n63 ~ $
```

Figure 3.10: Проверка длительности работы сервера

Контрольные вопросы:

- 1). Именованные каналы отличаются от неименованных наличием идентификатора канала, который представлен как специальный файл (соответственно имя именованного канала — это имя файла). Поскольку файл находится на локальной файловой системе, данное IPC используется внутри одной системы.
- 2). Чтобы создать неименованный канал из командной строки нужно использовать символ `|`, служащий для объединения двух и более процессов: `процесс_1 | процесс_2 | процесс_3...`
- 3). Чтобы создать именованный канал из командной строки нужно использовать либо команду «`mknod`», либо команду «`mkfifo`».
- 4). Неименованный канал является средством взаимодействия между связанными процессами — родительским и дочерним. Родительский процесс создает

канал при помощи системного вызова: `«int pipe(int fd[2]);»`. Массив из двух целых чисел является выходным параметром этого системного вызова. Если вызов выполнен нормально, то этот массив содержит два файловых дескриптора. `fd[0]` является дескриптором для чтения из канала, `fd[1]` – дескриптором для записи в канал. Когда процесс порождает другой процесс, дескрипторы родительского процесса наследуются дочерним процессом, и, таким образом, прокладывается трубопровод между двумя процессами. Естественно, что один из процессов использует канал только для чтения, а другой – только для записи. Поэтому, если, например, через канал должны передаваться данные из родительского процесса в дочерний, родительский процесс сразу после запуска дочернего процесса закрывает дескриптор канала для чтения, а дочерний процесс закрывает дескриптор для записи. Если нужен двунаправленный обмен данными между процессами, то родительский процесс создает два канала, один из которых используется для передачи данных в одну сторону, а другой – в другую.

5). Файлы именованных каналов создаются функцией `mkfifo()` или функцией `mknod`: 1. `«int mkfifo(const char pathname, mode_t mode);»`, где первый параметр – путь, где будет располагаться FIFO (имя файла, идентифицирующего канал), второй параметр определяет режим работы с FIFO (маска прав доступа к файлу), 2. `«mknod (namefile, IFIFO | 0666, 0)»`, где `namefile` – имя канала, `0666` – к каналу разрешен доступ на запись и на чтение любому запросившему процессу), 3. `«int mknod(const char pathname, mode_t mode, dev_t dev);»`. Функция `mkfifo()` создает канал и файл соответствующего типа. Если указанный файл канала уже существует, `mkfifo()` возвращает -1. После создания файла канала процессы, участвующие в обмене данными, должны открыть этот файл либо для записи, либо для чтения.

6). При чтении меньшего числа байтов, чем находится в канале или FIFO, возвращается требуемое число байтов, остаток сохраняется для последующих чтений. При чтении большего числа байтов, чем находится в канале или FIFO, возвращается доступное число байтов. Процесс, читающий из канала, должен соответствующим образом обработать ситуацию, когда прочитано меньше, чем

заказано.

7). Запись числа байтов, меньшего емкости канала или FIFO, гарантированно атомарно. Это означает, что в случае, когда несколько процессов одновременно записывают в канал, порции данных от этих процессов не перемешиваются. При записи большего числа байтов, чем это позволяет канал или FIFO, вызов `write(2)` блокируется до освобождения требуемого места. При этом атомарность операции не гарантируется. Если процесс пытается записать данные в канал, не открытый ни одним процессом на чтение, процессу генерируется сигнал `SIGPIPE`, а вызов `write(2)` возвращает 0 с установкой ошибки (`errno=ERRPIPE`) (если процесс не установил обработки сигнала `SIGPIPE`, производится обработка по умолчанию — процесс завершается).

8). Количество процессов, которые могут параллельно присоединяться к любому концу канала, не ограничено. Однако если два или более процесса записывают в канал данные одновременно, каждый процесс за один раз может записать максимум `PIPE_BUF` байтов данных. Предположим, процесс (назовем его А) пытается записать *X* байтов данных в канал, в котором имеется место для *Y* байтов данных. Если *X* больше, чем *Y*, только первые *Y* байтов данных записываются в канал, и процесс блокируется. Запускается другой процесс (например, В); в это время в канале появляется свободное пространство (благодаря третьему процессу, считывающему данные из канала). Процесс В записывает данные в канал. Затем, когда выполнение процесса А возобновляется, он записывает оставшиеся *X-Y* байтов данных в канал. В результате данные в канал записываются поочередно двумя процессами. Аналогичным образом, если два (или более) процесса одновременно попытаются прочитать данные из канала, может случиться так, что каждый из них прочитает только часть необходимых данных.

9). Функция `write` записывает байты `count` из буфера `buffer` в файл, связанный с `handle`. Операции `write` начинаются с текущей позиции указателя на файл (указатель ассоциирован с заданным файлом). Если файл открыт для добавления, операции выполняются в конец файла. После осуществления операций запи-

си указатель на файл (если он есть) увеличивается на количество действительно записанных байтов. Функция `write` возвращает число действительно записанных байтов. Возвращаемое значение должно быть положительным, но меньше числа `count` (например, когда размер для записи `count` байтов выходит за пределы пространства на диске). Возвращаемое значение `-1` указывает на ошибку; errno устанавливается в одно из следующих значений: `EACCES` – файл открыт для чтения или закрыт для записи, `EBADF` – неверный handle-р файла, `ENOSPC` – на устройстве нет свободного места. Единица в вызове функции `write` в программе `server.c` означает идентификатор (дескриптор потока) стандартного потока вывода.

10). Прототип функции `strerror`: «`char*strerror(int errornum);`». Функция `strerror` интерпретирует номер ошибки, передаваемый в функцию в качестве аргумента `errornum`, в понятное для человека текстовое сообщение (строку). Откуда берутся эти ошибки? Ошибки эти возникают при вызове функций стандартных Си-библиотек. То есть хорошим тоном программирования будет – использование этой функции в паре с другой, и если возникнет ошибка, то пользователь или программист поймет, как исправить ошибку, прочитав сообщение функции `strerror`. Возвращенный указатель ссылается на статическую строку с ошибкой, которая не должна быть изменена программой. Дальнейшие вызовы функции `strerror` перезапишут содержание этой строки. Интерпретированные сообщения об ошибках могут различаться, это зависит от платформы и компилятора.

4 Библиография

1. Программное обеспечение GNU/Linux. Лекция 10. Минимальный набор знаний (Г. Курячий, МГУ)
2. Программное обеспечение GNU/Linux. Лекция 12. Выбор дистрибутива (Г. Курячий, МГУ)
3. Электронный ресурс: <https://habr.com/ru/post/122108/>
4. Электронный ресурс: https://www.opennet.ru/docs/RUS/linux_parallel/node17.html

5 Выводы

В ходе выполнения данной лабораторной работы я приобрела практические навыки работы с именованными каналами.