

Zaawansowana komunikacja międzyprocesowa - łącza nazwane, kolejki komunikatów		
Dominik Wróbel	09 V 2019	Czw. 17:00

## Spis treści

---

<b>1</b>	<b>Łącza nazwane w powłoce</b>	<b>2</b>
<b>2</b>	<b>Łącza nazwane w API</b>	<b>3</b>
2.1	Zadanie 1 . . . . .	3
2.2	Zadanie 2 . . . . .	4
<b>3</b>	<b>Chatbot Eliza</b>	<b>5</b>

## 1 Łąca nazwane w powłoce

### Zadanie - Łąca nazwane w powłoce

...otwieramy dwie powłoki - tak aby procesy które będą się komunikowały nie miały wspólnego przodka w którym zostało utworzone łącze komunikacyjne i aby wykluczyć dziedziczenie deskryptorów plików.

W dowolnej z nich tworzymy kolejkę FIFO o nazwie test  
mkfifo test

W pierwszej z nich uruchamiamy program ls z parametrami -la i przekierowujemy standardowe wyjście tego polecenia na utworzoną kolejkę:

ls -la > test

W drugiej z nich uruchamiamy program grep z filtrem d i przekierowujemy jego standardowe wejście na utworzoną kolejkę:

grep d < test

The image shows two terminal windows side-by-side. The left window shows the creation of a FIFO named 'test' and the execution of 'ls -la' with its output redirected to 'test'. The right window shows the execution of 'grep "^d" < test', which lists all files and directories in the current directory, including hidden ones like '.cache', '.config', and '.ssh'.

```
dominik@dominik-VirtualBox: ~$ mkfifo test
dominik@dominik-VirtualBox: ~$ ls -la > test
dominik@dominik-VirtualBox: ~$

dominik@dominik-VirtualBox: ~$ grep "^d" < test
drwxr-xr-x 33 dominik dominik 4096 kwi 17 15:56 .
drwxr-xr-x  3 root    root    4096 maj 12 2018 ..
drwxrwxr-x  3 dominik dominik 4096 mar 24 15:34 .cabal
drwx----- 18 dominik dominik 4096 paź 22 16:14 .cache
drwx-----  3 dominik dominik 4096 paź 20 23:45 .compiz
drwx----- 19 dominik dominik 4096 mar 24 15:45 .config
drwxrwxr-x  3 dominik dominik 4096 paź 22 17:27 Databases_PostgreSQL
drwxrwxr-x  3 dominik dominik 4096 lis 24 18:03 DBpostgresRepo
drwxr-xr-x  2 dominik dominik 4096 sty 11 12:20 Desktop
drwxr-xr-x  2 dominik dominik 4096 maj 12 2018 Documents
drwxr-xr-x  2 dominik dominik 4096 mar 24 15:38 Downloads
drwx-----  2 dominik dominik 4096 maj 12 2018 .gconf
drwxrwxr-x  2 dominik dominik 4096 mar 24 15:42 .ghc
drwx-----  3 dominik dominik 4096 kwi 17 15:54 .gnupg
drwxrwxr-x  4 dominik dominik 4096 mar 25 00:20 Haskell
drwxrwxr-x  3 dominik dominik 4096 mar  3 14:54 JęzykC_SO
drwxrwxr-x  2 dominik dominik 4096 sty 16 08:53 Kolokwium
drwx-----  3 dominik dominik 4096 maj 12 2018 .local
drwx-----  5 dominik dominik 4096 paź 17 20:25 .mozilla
drwxr-xr-x  2 dominik dominik 4096 maj 12 2018 Music
drwxrwxr-x 10 dominik dominik 4096 kwi  8 20:03 OperatingSystems
drwxrwxr-x  2 dominik dominik 4096 sty 15 20:38 pcwiczenia
drwxr-xr-x  2 dominik dominik 4096 maj 12 2018 Pictures
drwx-----  3 dominik dominik 4096 mar 24 15:45 .pki
drwxrwxr-x  5 dominik dominik 4096 sty 14 19:39 Prolog
drwxrwxr-x  3 dominik dominik 4096 gru  2 23:33 PrologRepo
drwxr-xr-x  2 dominik dominik 4096 maj 12 2018 Public
drwxr-xr-x  3 dominik dominik 4096 paź 21 18:12 snap
drwx-----  2 dominik dominik 4096 paź 17 19:43 .ssh
drwxrwxr-x  2 dominik dominik 4096 sty 24 00:40 .swipl-dir-history
drwxr-xr-x  2 dominik dominik 4096 maj 12 2018 Templates
drwxr-xr-x  2 dominik dominik 4096 maj 12 2018 Videos
drwxrwxr-x  3 dominik dominik 4096 mar 24 15:45 .vscode
dominik@dominik-VirtualBox: ~$
```

Rysunek 1: FIFO w powłoce

## 2 Łąca nazwane w API

### 2.1. Zadanie 1

#### Zadanie 1

W pierwszym programie proszę wprowadzić modyfikację tak aby:

- Serwer po uruchomieniu nie zatrzymywał się na otwieraniu łącza czytania danych od klienta, tylko przechodził do oczekiwania na dane od klienta.
- Klient po wysłaniu komunikatu nie zatrzymywał się na otwieraniu łącza czytania danych od serwera, tylko przechodził do oczekiwania na dane od serwera.

W kodzie serwera i klienta dodano flagę `O_NONBLOCK` przy otwieraniu pliku do czytania, aby nie zatrzymywać programu na otwieraniu deskryptora. Następnie flaga ta jest ponownie przywracana aby zatrzymać się na czytaniu danych. Otwierany jest również deskryptor pliku do zapisywania aby nie uzyskać EOF. Zmiany zaznaczono kolorem szarym w kodzie.

#### Listing 1: srvfifo.c

```

1  ...
2
3  fdsrv = open(fifosrvname, O_RDONLY | O_NONBLOCK); // setting nonblock flag
4  printf("After read opening\n");
5
6  if(fdsrv == -1)
7  {
8      printf("FAIL!\nError: '%s'\n", strerror(errno));
9      return 0;
10 }
11 printf("OK\n");
12
13 flags = fcntl(fdsrv, F_GETFL); // reset nonblock flag
14 flags &= ~O_NONBLOCK;
15 fcntl(fdsrv, F_SETFL, flags);
16
17 printf("Opening for writing");
18 fdwrite = open(fifosrvname, O_WRONLY); // keep writing descriptor open
19
20 while(1)
21 {
22     ...
23 
```

#### Listing 2: cntfifo.c

```

1  ...
2
3  printf("OK\nOpening client fifo queue '%s' for reading...", fifocntname);
4
5  fdcnt = open(fifocntname, O_RDONLY | O_NONBLOCK);
6
7
8  if(fdcnt == -1)
9  {
10     printf("FAIL!\nError: %s\n", strerror(errno));
11     break;
12 }

```

```

13
14 // unblock
15 flags = fcntl(fdent, F_GETFL);
16 flags &= ~O_NONBLOCK;
17 fcntl(fdent, F_SETFL, flags);
18
19 fdwrite = open(fifocntname, O_WRONLY);
20 printf("OK\n");
21
22
23
24 /* Reading response */
25 printf("Waiting for data ...");
26 bread = read(fdent, &msg, sizeof(msg));
27
28 ...

```

## 2.2. Zadanie 2

### Pytanie - Zadanie 2.1

Program 3. zawiera błąd:  
Na czym on polega?

Błąd znajduje się w dwóch miejscach każdego z programów (klienta i serwera), w obu programach są to dwa te same błędy.

- Zarówno w kliencie jak i serwerze nie uwzględniono przy otwieraniu kolejki do czytania maksymalnego rozmiaru wiadomości, który musi być mniejszy od rozmiaru wysyłanej wiadomości.
- Przy zapisie wiadomości korzysta się z sizeof dla struktury, w tym przypadku zwraca to wartość ponad limit i skutkuje błędem 'message too long'

### Pytanie - Zadanie 2.2

Program 3. zawiera błąd:  
Jak go naprawić?

- Pierwszy z błędów należy naprawić ustawiając parametry struktury mq\_attr na takie aby maksymalny rozmiar wiadomości był mniejszy od rozmiaru bufora.
- Drugi z błędem naprawiamy poprzez ograniczenia rozmiaru wysyłane wiadomości przy wysyłania

### Listing 3: Poprawiony kod POSIX

```

1
2 ...
3 // SERWER
4 struct mq_attr attr;
5 attr.mq_flags = 0;
6 attr.mq_maxmsg = 10;
7 attr.mq_msgsize = 99;
8 attr.mq_curmsgs = 0;
9
10 /* Opening server psx queue */

```

```

11 printf("Opening server queue \'%s\' for reading...", srvpsxqname);
12 mq_unlink(srvpsxqname);
13 qdsrv = mq_open(srvpsxqname, O_RDONLY | O_CREAT, PERM_FILE, &attr);
14 ...
15
16 ...
17 // KLIENT
18 struct mq_attr attr;
19 attr.mq_flags = 0;
20 attr.mq_maxmsg = 10;
21 attr.mq_msgsize = 99;
22 attr.mq_curmsgs = 0;
23
24 /* Opening client queue for reading */
25 printf("Opening client queue \'%s\' for reading...", cntpsxqname);
26 mq_unlink(cntpsxqname);
27 qdcnt = mq_open(cntpsxqname, O_RDONLY | O_CREAT, PERM_FILE, &attr);
28 if(qdcnt == -1)
29 ...
30 // SERWER
31 /* Sending response */
32 printf("OK\nWriting response to client %ld...", (long)msg.pid);
33 bwrite = mq_send(qdcnt, (const char*)&msg, 99, 0);
34
35
36
37 ...
38 // KLIENT
39 printf("Writing message...");
40 bwrite = mq_send(qdsrv, (const char*)&msg, 99, 0);
41 if(bwrite == -1)

```

#### Pytanie - Zadanie 2.3

Gdzie można sprawdzić limity ilości i rozmiaru wiadomości?

Można to zrobić przy użyciu funkcji `mq_getattr`, przekazując do niej strukturę `mq_attr` oraz deskryptor kolejki wiadomości.

```

1 mq_getattr—get message—queue attributes
2 #include <mqueue.h>
3 int mq_getattr(
4     mqd_t mqd, /* message—queue descriptor */
5     struct mq_attr *attr /* attributes */
6 ); /* Returns 0 on success or -1 on error (sets errno) */

```

### 3 Chatbot Eliza

W tym zadaniu zaimplementowano chatbot Eliza przy użyciu komunikacji opartej na kolejkach FIFO. Kod zamieszczono poniżej

Listing 4: `srveliza.c`

```

1
2 #include <stdlib.h>
3 #include <stdio.h>
4 #include <errno.h>
5 #include <fcntl.h>

```

```
6 #include <string.h>
7 #include <ctype.h>
8
9 #include <sys/types.h>
10 #include <sys/stat.h>
11 /* _____ */
12
13 #include "fifo.h"
14 /* _____ */
15
16 #define MAX 100
17 #define SHORT 10
18 #define VERYSHORT 2
19
20 char *response[] = {
21     "How are you this beautiful day ?",
22     "Did you have a happy childhood ?",
23     "Did you hate your father ?",
24     "Did you have a good friend ?",
25     "Did you like your friend ?",
26     "What do you think about your friendship ?",
27     "I'm not sure I understand .",
28     ""
29 };
30
31 char *trans[]={
32     "no","Tell me more about the detail .",
33     "yes","Tell me more about the detail .",
34     "fuck","Don't be so rude again !",
35     "shet","Don't be so rude again !",
36     "you","Let't not talk about me .",
37     "think","Why do you think that ?",
38     "hate","So you hate something — tell me more .",
39     "what","Why do you ask ?",
40     "want","Why do you want that ?",
41     "need","We all need many things — is this special ?",
42     "why","Remember,therapy is good for you .",
43     "know","How do you know that ?",
44     "bye","Your bill will be mailed to you .",
45     "murder","I don't like killing .",
46     "kill","It is wrong to kill .",
47     "jerk","Don't ever call me a jerk !",
48     "can't","Don't be negative — be positive .",
49     "failure","Strive for success .",
50     "never","Don't be negative — be positive .",
51     "unhappy","Why are you unhappy ?",
52     "beautiful","I'm sure you like her,don't you ?",
53     "like","Why do you like that?",
54     "love","Remember,love everthing what you love .",
55     ""
56 };
57
58 char topics[MAX][80];
59 char token[80];
60 char *p_pos;
61 int res=0;
62 int head=0;
63 int tail=0;
64
65 void assert_topic(char *);
66 void get_token(void);
67 int is_in(char, char *);
68 int lookup(char *);
```

```

69 int find_topic(char *);
70 int in_topics(char *);
71
72 /* creat the doctor's response */
73 char* respond (char *s)
74 {
75     char t[80];
76     char *ret;
77     int loc;
78     if (strlen(s)<VERYSHORT && strcmp(s,"bye")) {
79         if (find_topic(t)) {
80             printf ("You just said:");
81             printf ("%s\n",t);
82             printf ("tell me more .\n");
83         }
84         else {
85             if (!*response[res]) res=0; /* start over again */
86             /* printf ("%s\n",response[res++]); */
87             ret = response[res++];
88         }
89         return ret;
90     }
91     if (in_topics(s)) {
92         /* printf ("Stop repeating yourself !\n"); */
93         ret = "Stop repeating yourself !\n";
94         return ret;
95     }
96     if (strlen(s)>SHORT) assert_topic(s);
97     do {
98         get_token();
99         loc=lookup(token);
100         if (loc!=-1) {
101             /* printf ("%s\n",trans[loc+1]); */
102             ret = trans[loc+1];
103             return ret;
104         }
105     } while (*token);
106     /* comment of last resort */
107     if (strlen(s)>SHORT)
108         /* printf ("It's seem intersting , tell me more ...\n"); */
109         ret = "It's seem intersting , tell me more ...\n";
110     /* else printf ("Tell me more ...\n"); */
111     else ret = "Tell me more ...\n";
112
113 return ret;
114 }
115
116
117 /* Lookup a keyword in translation table */
118 int lookup (char *token)
119 {
120     int t;
121     t=0;
122     while (*trans[t]) {
123         if (!strcmp (trans[t],token)) return t ;
124         t++;
125     }
126     return -1;
127 }
128
129 /* place a topic into the topics database */
130 void assert_topic (char *t)
131 {

```

```
132     if (head==MAX) head=0; /* wrap around */
133     strcpy (topics[head], t);
134     head++;
135 }
136
137 /* retrieve a topic */
138 int find_topic (char *t)
139 {
140     if (tail!=head) {
141         strcpy(t, topics[tail]);
142         tail++;
143         /* wrap around if necessary */
144         if (tail==MAX) tail=0;
145         return 1;
146     }
147     return 0;
148 }
149
150 /* see if in topics queue */
151 int in_topics (char *s)
152 {
153     int t;
154     for (t=0; t<MAX; t++)
155         if (!strcmp(s, topics[t])) return 1;
156     return 0;
157 }
158
159 /* return a token from the input stream */
160 void get_token (void)
161 {
162     char *p;
163     p=token;
164     /* skip spaces */
165     while (*p_pos==' ') p_pos++;
166
167     if (*p_pos=='\0') { /*is end of input*/
168         *p++='\0';
169         return;
170     }
171     if (is_in(*p_pos, ".!?")) {
172         *p=*p_pos;
173         p++, p_pos++;
174         *p='\0';
175         return;
176     }
177
178     /*read word until*/
179     while(*p_pos!=' ' && !is_in(*p_pos, ". , ; ? !") && *p_pos) {
180         *p=tolower(*p_pos++);
181         p++;
182     }
183     *p='\0';
184 }
185
186 int is_in(char c, char *s)
187 {
188     while(*s) {
189         if(c==*s) return 1;
190         s++;
191     }
192     return 0;
193 }
194
```



```
195
196
197 int main(void)
198 {
199     struct message msg;
200     char* fifosrvbasename = "srvfifoqueue";
201     char fifosrvname[FIFO_NAME_BUF_SIZE];
202     char fifocntname[FIFO_NAME_BUF_SIZE];
203     int fdsrv,
204         fdcnt,
205         bread,
206         bwrite,
207         fdsrv_w;
208
209     printf("Server started...\n");
210     setbuf(stdout, NULL);
211
212     /* Creating server fifo queue */
213     make_srv_fifo_queue_name(fifosrvname, fifosrvbasename, FIFO_NAME_BUF_SIZE);
214     printf("Creating server fifo queue \'%s\'...", fifosrvname);
215     if((mkfifo(fifosrvname, PERM_FILE) == -1) && (errno != EEXIST))
216     {
217         printf("FAIL!\nError: %s\n", strerror(errno));
218         return 0;
219     }
220
221     /* Opening fifo */
222     printf("OK\nOpening server fifo queue \'%s\' for reading...", fifosrvname);
223     /* O_NONBLOCK - added */
224     fdsrv = open(fifosrvname, O_RDONLY|O_NONBLOCK);
225     if(fdsrv == -1)
226     {
227         printf("FAIL!\nError: \'%s\'", strerror(errno));
228         return 0;
229     }
230
231
232     int flags= fcntl(fdsrv, F_GETFL);
233     flags &= ~O_NONBLOCK;
234     fcntl(fdsrv, F_SETFL, flags);
235     //otworzenie do zapisu
236     fdsrv_w = open(fifosrvname, O_WRONLY);
237     if(fdsrv_w == -1)
238     {
239         printf("FAIL!\nError: \'%s\'", strerror(errno));
240         return 0;
241     }
242
243     printf("OK\n");
244
245     while(1)
246     {
247         /* Reading from queue */
248         printf("Waiting for data...");
249         bread = read(fdsrv, &msg, sizeof(msg));
250         if(bread == -1)
251         {
252             printf("FAIL!\nError: %s\n", strerror(errno));
253             break;
254         }
255         printf("OK\n");
256
257         printf("Message from client [%d]: %s\n", msg.pid, msg.data);
```

```

258
259     /* printf("Your response: ");
260     stdin_readall(msg.data, MESSAGE_BUF_SIZE);
261     */
262
263     /* Creating client fifo name */
264     make_cnt_fifo_queue_name(fifocntname, msg.pid, FIFO_NAME_BUF_SIZE);
265
266     /* Opening client fifo for writing */
267     printf("Opening client fifo '%s' for writing...", fifocntname);
268     fdcnt = open(fifocntname, O_WRONLY);
269     if(fdcnt == -1)
270     {
271         printf("FAIL!\nError: %s\n", strerror(errno));
272         break;
273     }
274
275
276
277
278     /* Sending response */
279     p_pos = msg.data;
280     /* printf("odpowiedz: %s", respond(msg.data)); */
281     strcpy(msg.data, respond(msg.data));
282     printf("OK\nWriting response to client [%ld]...", (long)msg.pid);
283     bwrite = write(fdcnt, &msg, sizeof(msg));
284     if(bwrite == -1)
285     {
286         printf("FAIL!\nError: %s\n", strerror(errno));
287         break;
288     }
289     printf("OK\n");
290     close(fdcnt);
291 }
292
293
294 /* Cleaning up */
295 close(fdsrv);
296 unlink(fifosrvname);
297
298 return 0;
299 }

```

Listing 5: cnteliza.c

```

1
2
3 #include <stdlib.h>
4 #include <stdio.h>
5 #include <errno.h>
6 #include <fcntl.h>
7
8 #include <sys/types.h>
9 #include <sys/stat.h>
10
11 #include "fifo.h"
12 /* ----- */
13
14 int main(void)
15 {
16     struct message msg;
17     char* fifosrvbasename = "srvfifoqueue";
18     char fifosrvname[FIFO_NAME_BUF_SIZE];

```

```
19 char fifocntname[FIFO_NAME_BUF_SIZE];
20 int fdsrv,
21     fdcnt,
22     bread,
23     bwrite,
24     fdcnt_w;
25
26 printf("Client [%ld] started...\n", (long)getpid());
27 setbuf(stdout, NULL);
28 msg.pid = getpid();
29
30 /* Creating client fifo name */
31 make_cnt_fifo_queue_name(fifocntname, msg.pid, FIFO_NAME_BUF_SIZE);
32 printf("Creating client fifo queue \'%s\'...", fifocntname);
33 if((mkfifo(fifocntname, PERM_FILE) == -1) && (errno != EEXIST))
34 {
35     printf("FAIL!\nError: %s\n", strerror(errno));
36     return 0;
37 }
38
39 /* Opening server fifo for writing */
40 make_srv_fifo_queue_name(fifosrvname, fifosrvbasename, FIFO_NAME_BUF_SIZE);
41 printf("OK\nOpening server fifo queue \'%s\' for writing...", fifosrvname);
42 fdsrv = open(fifosrvname, O_WRONLY);
43 if(fdsrv == -1)
44 {
45     printf("FAIL!\nError: %s\n", strerror(errno));
46     return 0;
47 }
48 printf("OK\n");
49
50 while(1)
51 {
52     /* Getting message */
53     printf("Send message: ");
54     if(stdin_readall(msg.data, MESSAGE_BUF_SIZE) == 0)
55         break;
56
57     /* Sending message to server */
58     printf("Writing message to server...");
59     bwrite = write(fdsrv, &msg, sizeof(msg));
60     if(bwrite == -1)
61     {
62         printf("FAIL!\nError: %s\n", strerror(errno));
63         break;
64     }
65
66     /* Opening client fifo for reading */
67     printf("OK\nOpening client fifo queue \'%s\' for reading...", fifocntname);
68     fdcnt = open(fifocntname, O_RDONLY|O_NONBLOCK);
69     if(fdcnt == -1)
70     {
71         printf("FAIL!\nError: %s\n", strerror(errno));
72         break;
73     }
74
75     int flags = fcntl(fdcnt, F_GETFL);
76     flags &= ~O_NONBLOCK;
77     fcntl(fdcnt, F_SETFL, flags);
78     //otworzenie do zapisu
79     fdcnt_w = open(fifocntname, O_WRONLY);
80     if(fdcnt_w == -1)
81     {
```

```
82 printf("FAIL!\nError: \'%s\'\n", strerror(errno));
83 return 0;
84 }
85
86 printf("OK\n");
87
88 /* Reading response */
89 printf("Waiting for data...");
90 bread = read(fdcnt, &msg, sizeof(msg));
91 if(bread == -1)
92 {
93     printf("FAIL!\nError: %s\n", strerror(errno));
94     close(fdcnt);
95     break;
96 }
97     close(fdcnt);
98     printf("OK\nMessage from server: %s\n", msg.data);
99 }
100
101 /* Cleaning up */
102 close(fdsrv);
103 unlink(fifocntname);
104
105 return 0;
106 }
```