

Capturas de Javier Mendez – A01703446

The screenshot shows a dual-pane interface. On the left, a web browser window displays a Google search results page for "hora queretaro". The top result is a link to "www.worldtimeserver.com" showing the local time in Querétaro, Mexico. Below it, there are links to "www.worldtimeserver.com" and "time.is" both listing the current time in Querétaro. On the right, the main Visual Studio Code window shows the file "execute.c" with its code content. The code includes imports for stdio.h, stdlib.h, and string.h, and defines a main() function that executes system commands like ls and date.

```
#include <stdio.h>
#include <stdlib.h> //to use system()
#include <string.h> //to use strcpy()

int main()
{
    char *command = (char *) malloc(100);
    //executing ls command
    strcpy(command, "ls");
    printf("ls command...\n");
    system(command);
    //executing date command
    strcpy(command, "date");
    printf("date command...\n");
    system(command);
    return 0;
}
```

This screenshot is nearly identical to the one above, showing the same search results on Google and the same version of the "execute.c" code in Visual Studio Code. The code remains the same, demonstrating a basic exploit by concatenating user input directly into system calls.

```
#include <stdio.h>
#include <stdlib.h> //to use system()
#include <string.h> //to use strcpy()

int main()
{
    char *command = "ls";
    char *arg = "-l";
    //executing ls command
    printf("ls command...\n");
    execvp(command, command, arg, NULL);
    return 0;
}
```

Google hora queretaro

Cerca de 8,570,000 resultados (0.36 segundos)

9:31
viernes, 17 de abril de 2020 (GMT-5)
Hora en Santiago de Querétaro, Qro.

www.worldtimeserver.com › hora-exacta-MX-QUE › cl... ▾
Hora local actual en Querétaro, Querétaro, México
DST. Comienza el abril 5, 2020 02:00. Establezca su reloj vent...

www.worldtimeserver.com › hora-exacta-MX-QUE ▾
Hora local actual en en Querétaro, México
Querétaro, México - obtener la hora actual y la fecha exactas, telefónica o hacer planes de viaje para un vuelo barato o un ho...

time.is › Querétaro
La hora actual en Querétaro, México - Time.
Hora exacta, zona horaria, diferente zona horaria, salida del so...

www.zeltverschiebung.net › ... › Querétaro › Querétaro ▾
Hora actual en Santiago de Querétaro, México

execute.c - actividades colaborativas - Visual Studio Code

```
int main(int argc, char* argv[]) {  
    if (argc != 4) {  
        fprintf(stderr, "usage: %s file1 file2\n", argv[0]);  
        return -2;  
    }  
    int f1, f2, f3;  
    if ((f1 = open(argv[1], O_RDONLY)) < 0) {  
        fprintf(stderr, "%s: the file %s does not exist\n", argv[0], argv[1]);  
        return -3;  
    }  
    if ((f2 = open(argv[2], O_RDONLY)) < 0) {  
        fprintf(stderr, "%s: the file %s does not exist\n", argv[0], argv[2]);  
        return -4;  
    }  
    if ((f3 = open(argv[3], O_RDONLY)) < 0) {  
        fprintf(stderr, "%s: the file %s does not exist\n", argv[0], argv[3]);  
        return -5;  
    }  
}
```

In 91, Col 1 Spaces 2 UTF-8 CR LF C Win32

Google hora queretaro

Cerca de 8,570,000 resultados (0.38 segundos)

9:44
viernes, 17 de abril de 2020 (GMT-5)
Hora en Santiago de Querétaro, Qro.

www.worldtimeserver.com › hora-exacta-MX-QUE › cl... ▾
Hora local actual en Querétaro, Querétaro, México
DST. Comienza el abril 5, 2020 02:00. Establezca su reloj vent...

www.worldtimeserver.com › hora-exacta-MX-QUE ▾
Hora local actual en en Querétaro, México
Querétaro, México - obtener la hora actual y la fecha exactas, telefónica o hacer planes de viaje para un vuelo barato o un ho...

time.is › Querétaro
La hora actual en Querétaro, México - Time.
Hora exacta, zona horaria, diferente zona horaria, salida del so...

www.zeltverschiebung.net › ... › Querétaro › Querétaro ▾
Hora actual en Santiago de Querétaro, México

execute.c - actividades colaborativas - Visual Studio Code

```
if (argc != 4) {  
    fprintf(stderr, "usage: %s file1 file2\n", argv[0]);  
    return -2;  
}  
int f1, f2, f3;  
if ((f1 = open(argv[1], O_RDONLY)) < 0) {  
    fprintf(stderr, "%s: the file %s does not exist\n", argv[0], argv[1]);  
    return -3;  
}  
if ((f2 = open(argv[2], O_RDONLY)) < 0) {  
    fprintf(stderr, "%s: the file %s does not exist\n", argv[0], argv[2]);  
    return -4;  
}  
if ((f3 = open(argv[3], O_RDONLY)) < 0) {  
    fprintf(stderr, "%s: the file %s does not exist\n", argv[0], argv[3]);  
    return -5;  
}  
ulong f_size;  
f_size = lseek(f1, 0, SEEK_END);  
if (f_size < 1) {  
    fprintf(stderr, "%s: the file %s is empty\n", argv[0], argv[1]);  
    return -6;  
}  
f_size = lseek(f2, 0, SEEK_END);  
if (f_size < 1) {  
    fprintf(stderr, "%s: the file %s is empty\n", argv[0], argv[2]);  
    return -7;  
}  
f_size = lseek(f3, 0, SEEK_END);  
if (f_size < 1) {
```

In 66, Col 18 Spaces 2 UTF-8 CR LF C Win32

Google hora queretaro

Cerca de 8,650,000 resultados (0.46 segundos)

15:00
viernes, 17 de abril de 2020 (GMT-5)
Hora en Santiago de Querétaro, Qro.

www.worldtimeserver.com > hora-exacta-MX-QUE · cl... ▾
Hora local actual en Querétaro, Querétaro, México
DST. Comienza el abril 5, 2020 02:00. Establezca su reloj ventaja de 1 hora.

www.worldtimeserver.com > hora-exacta-MX-QUE ▾
Hora local actual en Querétaro, México
Querétaro, México - obtener la hora actual y la fecha exactos antes de la telefonía o hacer planes de viaje para un vuelo barato o un hotel ...

time.is > Querétaro
La hora actual en Querétaro, México - Time.is
Hora exacta, zona horaria, diferente zona horaria, salida del sol/puesta para Querétaro, México.

execute1.c

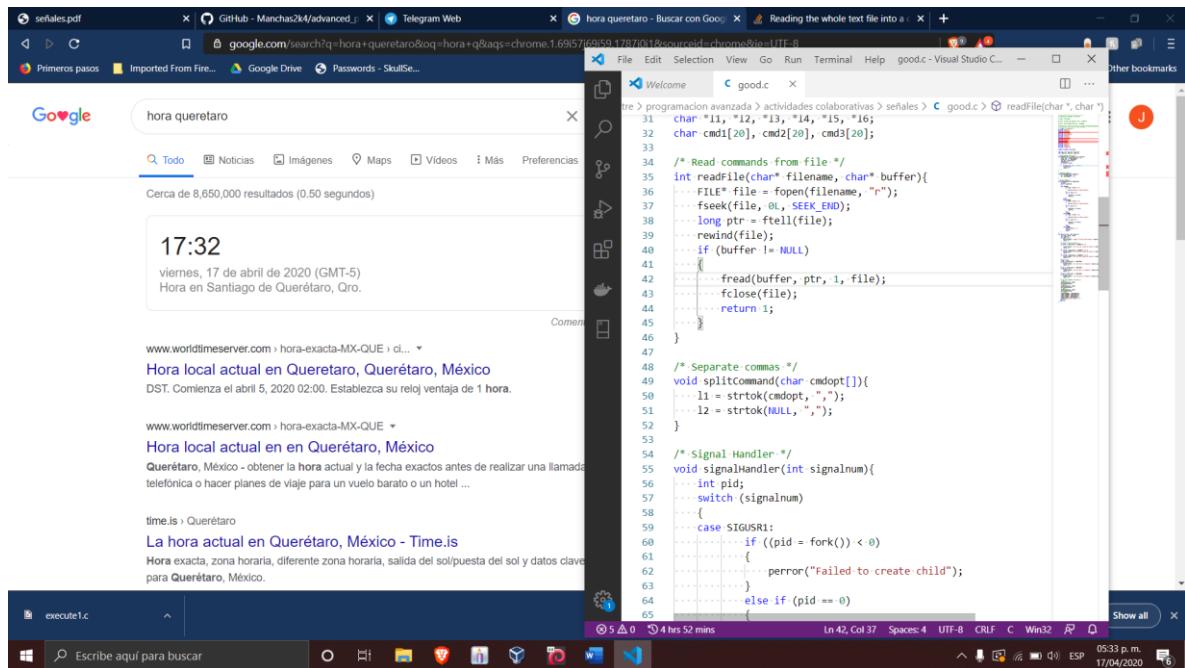
```

File Edit Selection View Go Run Terminal Help
OPEN EDITOR... 1 UNSAVED
C x.c sefales 3
C execute.c seh... 2
C execute1.c se... 2
ACTIVIDADES COLABORAT...
sefales
capturas.docx
e
empty1.txt
empty2.txt
empty3.txt
execute
C execute.c 2
C execute1.c 2
file1.txt
file2.txt
file3.txt
sefales.mp4
test.sh
x
C x.c 3
decripta.c
find.c
main.c

```

```

execute.c > ...
6 #include <setjmp.h>
7 #include <signal.h>
8 #include <fcntl.h>
9 #include <sys/types.h>
10
11 typedef unsigned long ulong;
12
13 void handler_sigusr1(int sig) {
14 }
15
16 void handler_sigusr2(int sig) {
17 }
18
19 void handler_sigpwr(int sig) {
20 }
21
22 void fun1() {
23
24
25
26 void fun2() {
27     setjmp(env2);
28     execvp("ps","ps","-f","./",NULL);
29 }
30
31 void fun3() {
32     setjmp(env3);
33     execvp("uname","uname","-r","./",NULL);
34 }
35
36 int main(int argc, char* argv[]) {
37
38     if (pid < 0) {
39         perror("Error on child");
40     } else {
41         if (pid == 0) {
42             // child
43             execvp("ls","ls -l -a", NULL); // child is replaced
44             _exit(1);
45         }
46         break;
47     }
48
49     case SIGUSR2:
50         pid = vfork();
51         if (pid < 0) {
52             perror("Error on child");
53         } else {
54             if (pid == 0) {
55                 execvp("ps","ps -f", NULL); // child is replaced
56                 _exit(1);
57             }
58             break;
59         }
60     case SIGPWR:
61         pid = vfork();
62         if (pid < 0) {
63             perror("Error on child");
64         } else {
65             if (pid == 0) {
66                 execvp("uname","uname -r", NULL); // child is replaced
67                 _exit(1);
68             }
69             break;
70         }
71     case SIGINT:
72         break;
73
74     default:
75         if (pid < 0) {
76             perror("Error on child");
77         } else {
78             if (pid == 0) {
79                 execvp("ls","ls -l -a", NULL); // child is replaced
80                 _exit(1);
81             }
82         }
83     }
84
85     if (pid < 0) {
86         perror("Error on child");
87     } else {
88         if (pid == 0) {
89             execvp("ps","ps -f", NULL); // child is replaced
90             _exit(1);
91         }
92     }
93
94     if (pid < 0) {
95         perror("Error on child");
96     } else {
97         if (pid == 0) {
98             execvp("uname","uname -r", NULL); // child is replaced
99             _exit(1);
100        }
101    }
102
103    if (pid < 0) {
104        perror("Error on child");
105    } else {
106        if (pid == 0) {
107            execvp("ls","ls -l -a", NULL); // child is replaced
108            _exit(1);
109        }
110    }
111
112    if (pid < 0) {
113        perror("Error on child");
114    } else {
115        if (pid == 0) {
116            execvp("ps","ps -f", NULL); // child is replaced
117            _exit(1);
118        }
119    }
120
121    if (pid < 0) {
122        perror("Error on child");
123    } else {
124        if (pid == 0) {
125            execvp("uname","uname -r", NULL); // child is replaced
126            _exit(1);
127        }
128    }
129
130    if (pid < 0) {
131        perror("Error on child");
132    } else {
133        if (pid == 0) {
134            execvp("ls","ls -l -a", NULL); // child is replaced
135            _exit(1);
136        }
137    }
138
139    if (pid < 0) {
140        perror("Error on child");
141    } else {
142        if (pid == 0) {
143            execvp("ps","ps -f", NULL); // child is replaced
144            _exit(1);
145        }
146    }
147
148    if (pid < 0) {
149        perror("Error on child");
150    } else {
151        if (pid == 0) {
152            execvp("uname","uname -r", NULL); // child is replaced
153            _exit(1);
154        }
155    }
156
157    if (pid < 0) {
158        perror("Error on child");
159    } else {
160        if (pid == 0) {
161            execvp("ls","ls -l -a", NULL); // child is replaced
162            _exit(1);
163        }
164    }
165
166    if (pid < 0) {
167        perror("Error on child");
168    } else {
169        if (pid == 0) {
170            execvp("ps","ps -f", NULL); // child is replaced
171            _exit(1);
172        }
173    }
174
175    if (pid < 0) {
176        perror("Error on child");
177    } else {
178        if (pid == 0) {
179            execvp("uname","uname -r", NULL); // child is replaced
180            _exit(1);
181        }
182    }
183
184    if (pid < 0) {
185        perror("Error on child");
186    } else {
187        if (pid == 0) {
188            execvp("ls","ls -l -a", NULL); // child is replaced
189            _exit(1);
190        }
191    }
192
193    if (pid < 0) {
194        perror("Error on child");
195    } else {
196        if (pid == 0) {
197            execvp("ps","ps -f", NULL); // child is replaced
198            _exit(1);
199        }
200    }
201
202    if (pid < 0) {
203        perror("Error on child");
204    } else {
205        if (pid == 0) {
206            execvp("uname","uname -r", NULL); // child is replaced
207            _exit(1);
208        }
209    }
210
211    if (pid < 0) {
212        perror("Error on child");
213    } else {
214        if (pid == 0) {
215            execvp("ls","ls -l -a", NULL); // child is replaced
216            _exit(1);
217        }
218    }
219
220    if (pid < 0) {
221        perror("Error on child");
222    } else {
223        if (pid == 0) {
224            execvp("ps","ps -f", NULL); // child is replaced
225            _exit(1);
226        }
227    }
228
229    if (pid < 0) {
230        perror("Error on child");
231    } else {
232        if (pid == 0) {
233            execvp("uname","uname -r", NULL); // child is replaced
234            _exit(1);
235        }
236    }
237
238    if (pid < 0) {
239        perror("Error on child");
240    } else {
241        if (pid == 0) {
242            execvp("ls","ls -l -a", NULL); // child is replaced
243            _exit(1);
244        }
245    }
246
247    if (pid < 0) {
248        perror("Error on child");
249    } else {
250        if (pid == 0) {
251            execvp("ps","ps -f", NULL); // child is replaced
252            _exit(1);
253        }
254    }
255
256    if (pid < 0) {
257        perror("Error on child");
258    } else {
259        if (pid == 0) {
260            execvp("uname","uname -r", NULL); // child is replaced
261            _exit(1);
262        }
263    }
264
265    if (pid < 0) {
266        perror("Error on child");
267    } else {
268        if (pid == 0) {
269            execvp("ls","ls -l -a", NULL); // child is replaced
270            _exit(1);
271        }
272    }
273
274    if (pid < 0) {
275        perror("Error on child");
276    } else {
277        if (pid == 0) {
278            execvp("ps","ps -f", NULL); // child is replaced
279            _exit(1);
280        }
281    }
282
283    if (pid < 0) {
284        perror("Error on child");
285    } else {
286        if (pid == 0) {
287            execvp("uname","uname -r", NULL); // child is replaced
288            _exit(1);
289        }
290    }
291
292    if (pid < 0) {
293        perror("Error on child");
294    } else {
295        if (pid == 0) {
296            execvp("ls","ls -l -a", NULL); // child is replaced
297            _exit(1);
298        }
299    }
299
300    if (pid < 0) {
301        perror("Error on child");
302    } else {
303        if (pid == 0) {
304            execvp("ps","ps -f", NULL); // child is replaced
305            _exit(1);
306        }
307    }
307
308    if (pid < 0) {
309        perror("Error on child");
310    } else {
311        if (pid == 0) {
312            execvp("uname","uname -r", NULL); // child is replaced
313            _exit(1);
314        }
315    }
315
316    if (pid < 0) {
317        perror("Error on child");
318    } else {
319        if (pid == 0) {
320            execvp("ls","ls -l -a", NULL); // child is replaced
321            _exit(1);
322        }
323    }
323
324    if (pid < 0) {
325        perror("Error on child");
326    } else {
327        if (pid == 0) {
328            execvp("ps","ps -f", NULL); // child is replaced
329            _exit(1);
330        }
331    }
331
332    if (pid < 0) {
333        perror("Error on child");
334    } else {
335        if (pid == 0) {
336            execvp("uname","uname -r", NULL); // child is replaced
337            _exit(1);
338        }
339    }
339
340    if (pid < 0) {
341        perror("Error on child");
342    } else {
343        if (pid == 0) {
344            execvp("ls","ls -l -a", NULL); // child is replaced
345            _exit(1);
346        }
347    }
347
348    if (pid < 0) {
349        perror("Error on child");
350    } else {
351        if (pid == 0) {
352            execvp("ps","ps -f", NULL); // child is replaced
353            _exit(1);
354        }
355    }
355
356    if (pid < 0) {
357        perror("Error on child");
358    } else {
359        if (pid == 0) {
360            execvp("uname","uname -r", NULL); // child is replaced
361            _exit(1);
362        }
363    }
363
364    if (pid < 0) {
365        perror("Error on child");
366    } else {
367        if (pid == 0) {
368            execvp("ls","ls -l -a", NULL); // child is replaced
369            _exit(1);
370        }
371    }
371
372    if (pid < 0) {
373        perror("Error on child");
374    } else {
375        if (pid == 0) {
376            execvp("ps","ps -f", NULL); // child is replaced
377            _exit(1);
378        }
379    }
379
380    if (pid < 0) {
381        perror("Error on child");
382    } else {
383        if (pid == 0) {
384            execvp("uname","uname -r", NULL); // child is replaced
385            _exit(1);
386        }
387    }
387
388    if (pid < 0) {
389        perror("Error on child");
390    } else {
391        if (pid == 0) {
392            execvp("ls","ls -l -a", NULL); // child is replaced
393            _exit(1);
394        }
395    }
395
396    if (pid < 0) {
397        perror("Error on child");
398    } else {
399        if (pid == 0) {
400            execvp("ps","ps -f", NULL); // child is replaced
401            _exit(1);
402        }
403    }
403
404    if (pid < 0) {
405        perror("Error on child");
406    } else {
407        if (pid == 0) {
408            execvp("uname","uname -r", NULL); // child is replaced
409            _exit(1);
410        }
411    }
411
412    if (pid < 0) {
413        perror("Error on child");
414    } else {
415        if (pid == 0) {
416            execvp("ls","ls -l -a", NULL); // child is replaced
417            _exit(1);
418        }
419    }
419
420    if (pid < 0) {
421        perror("Error on child");
422    } else {
423        if (pid == 0) {
424            execvp("ps","ps -f", NULL); // child is replaced
425            _exit(1);
426        }
427    }
427
428    if (pid < 0) {
429        perror("Error on child");
430    } else {
431        if (pid == 0) {
432            execvp("uname","uname -r", NULL); // child is replaced
433            _exit(1);
434        }
435    }
435
436    if (pid < 0) {
437        perror("Error on child");
438    } else {
439        if (pid == 0) {
440            execvp("ls","ls -l -a", NULL); // child is replaced
441            _exit(1);
442        }
443    }
443
444    if (pid < 0) {
445        perror("Error on child");
446    } else {
447        if (pid == 0) {
448            execvp("ps","ps -f", NULL); // child is replaced
449            _exit(1);
450        }
451    }
451
452    if (pid < 0) {
453        perror("Error on child");
454    } else {
455        if (pid == 0) {
456            execvp("uname","uname -r", NULL); // child is replaced
457            _exit(1);
458        }
459    }
459
460    if (pid < 0) {
461        perror("Error on child");
462    } else {
463        if (pid == 0) {
464            execvp("ls","ls -l -a", NULL); // child is replaced
465            _exit(1);
466        }
467    }
467
468    if (pid < 0) {
469        perror("Error on child");
470    } else {
471        if (pid == 0) {
472            execvp("ps","ps -f", NULL); // child is replaced
473            _exit(1);
474        }
475    }
475
476    if (pid < 0) {
477        perror("Error on child");
478    } else {
479        if (pid == 0) {
480            execvp("uname","uname -r", NULL); // child is replaced
481            _exit(1);
482        }
483    }
483
484    if (pid < 0) {
485        perror("Error on child");
486    } else {
487        if (pid == 0) {
488            execvp("ls","ls -l -a", NULL); // child is replaced
489            _exit(1);
490        }
491    }
491
492    if (pid < 0) {
493        perror("Error on child");
494    } else {
495        if (pid == 0) {
496            execvp("ps","ps -f", NULL); // child is replaced
497            _exit(1);
498        }
499    }
499
500    if (pid < 0) {
501        perror("Error on child");
502    } else {
503        if (pid == 0) {
504            execvp("uname","uname -r", NULL); // child is replaced
505            _exit(1);
506        }
507    }
507
508    if (pid < 0) {
509        perror("Error on child");
510    } else {
511        if (pid == 0) {
512            execvp("ls","ls -l -a", NULL); // child is replaced
513            _exit(1);
514        }
515    }
515
516    if (pid < 0) {
517        perror("Error on child");
518    } else {
519        if (pid == 0) {
520            execvp("ps","ps -f", NULL); // child is replaced
521            _exit(1);
522        }
523    }
523
524    if (pid < 0) {
525        perror("Error on child");
526    } else {
527        if (pid == 0) {
528            execvp("uname","uname -r", NULL); // child is replaced
529            _exit(1);
530        }
531    }
531
532    if (pid < 0) {
533        perror("Error on child");
534    } else {
535        if (pid == 0) {
536            execvp("ls","ls -l -a", NULL); // child is replaced
537            _exit(1);
538        }
539    }
539
540    if (pid < 0) {
541        perror("Error on child");
542    } else {
543        if (pid == 0) {
544            execvp("ps","ps -f", NULL); // child is replaced
545            _exit(1);
546        }
547    }
547
548    if (pid < 0) {
549        perror("Error on child");
550    } else {
551        if (pid == 0) {
552            execvp("uname","uname -r", NULL); // child is replaced
553            _exit(1);
554        }
555    }
555
556    if (pid < 0) {
557        perror("Error on child");
558    } else {
559        if (pid == 0) {
560            execvp("ls","ls -l -a", NULL); // child is replaced
561            _exit(1);
562        }
563    }
563
564    if (pid < 0) {
565        perror("Error on child");
566    } else {
567        if (pid == 0) {
568            execvp("ps","ps -f", NULL); // child is replaced
569            _exit(1);
570        }
571    }
571
572    if (pid < 0) {
573        perror("Error on child");
574    } else {
575        if (pid == 0) {
576            execvp("uname","uname -r", NULL); // child is replaced
577            _exit(1);
578        }
579    }
579
580    if (pid < 0) {
581        perror("Error on child");
582    } else {
583        if (pid == 0) {
584            execvp("ls","ls -l -a", NULL); // child is replaced
585            _exit(1);
586        }
587    }
587
588    if (pid < 0) {
589        perror("Error on child");
590    } else {
591        if (pid == 0) {
592            execvp("ps","ps -f", NULL); // child is replaced
593            _exit(1);
594        }
595    }
595
596    if (pid < 0) {
597        perror("Error on child");
598    } else {
599        if (pid == 0) {
600            execvp("uname","uname -r", NULL); // child is replaced
601            _exit(1);
602        }
603    }
603
604    if (pid < 0) {
605        perror("Error on child");
606    } else {
607        if (pid == 0) {
608            execvp("ls","ls -l -a", NULL); // child is replaced
609            _exit(1);
610        }
611    }
611
612    if (pid < 0) {
613        perror("Error on child");
614    } else {
615        if (pid == 0) {
616            execvp("ps","ps -f", NULL); // child is replaced
617            _exit(1);
618        }
619    }
619
620    if (pid < 0) {
621        perror("Error on child");
622    } else {
623        if (pid == 0) {
624            execvp("uname","uname -r", NULL); // child is replaced
625            _exit(1);
626        }
627    }
627
628    if (pid < 0) {
629        perror("Error on child");
630    } else {
631        if (pid == 0) {
632            execvp("ls","ls -l -a", NULL); // child is replaced
633            _exit(1);
634        }
635    }
635
636    if (pid < 0) {
637        perror("Error on child");
638    } else {
639        if (pid == 0) {
640            execvp("ps","ps -f", NULL); // child is replaced
641            _exit(1);
642        }
643    }
643
644    if (pid < 0) {
645        perror("Error on child");
646    } else {
647        if (pid == 0) {
648            execvp("uname","uname -r", NULL); // child is replaced
649            _exit(1);
650        }
651    }
651
652    if (pid < 0) {
653        perror("Error on child");
654    } else {
655        if (pid == 0) {
656            execvp("ls","ls -l -a", NULL); // child is replaced
657            _exit(1);
658        }
659    }
659
660    if (pid < 0) {
661        perror("Error on child");
662    } else {
663        if (pid == 0) {
664            execvp("ps","ps -f", NULL); // child is replaced
665            _exit(1);
666        }
667    }
667
668    if (pid < 0) {
669        perror("Error on child");
670    } else {
671        if (pid == 0) {
672            execvp("uname","uname -r", NULL); // child is replaced
673            _exit(1);
674        }
675    }
675
676    if (pid < 0) {
677        perror("Error on child");
678    } else {
679        if (pid == 0) {
680            execvp("ls","ls -l -a", NULL); // child is replaced
681            _exit(1);
682        }
683    }
683
684    if (pid < 0) {
685        perror("Error on child");
686    } else {
687        if (pid == 0) {
688            execvp("ps","ps -f", NULL); // child is replaced
689            _exit(1);
690        }
691    }
691
692    if (pid < 0) {
693        perror("Error on child");
694    } else {
695        if (pid == 0) {
696            execvp("uname","uname -r", NULL); // child is replaced
697            _exit(1);
698        }
699    }
699
700    if (pid < 0) {
701        perror("Error on child");
702    } else {
703        if (pid == 0) {
704            execvp("ls","ls -l -a", NULL); // child is replaced
705            _exit(1);
706        }
707    }
707
708    if (pid < 0) {
709        perror("Error on child");
710    } else {
711        if (pid == 0) {
712            execvp("ps","ps -f", NULL); // child is replaced
713            _exit(1);
714        }
715    }
715
716    if (pid < 0) {
717        perror("Error on child");
718    } else {
719        if (pid == 0) {
720            execvp("uname","uname -r", NULL); // child is replaced
721            _exit(1);
722        }
723    }
723
724    if (pid < 0) {
725        perror("Error on child");
726    } else {
727        if (pid == 0) {
728            execvp("ls","ls -l -a", NULL); // child is replaced
729            _exit(1);
730        }
731    }
731
732    if (pid < 0) {
733        perror("Error on child");
734    } else {
735        if (pid == 0) {
736            execvp("ps","ps -f", NULL); // child is replaced
737            _exit(1);
738        }
739    }
739
740    if (pid < 0) {
741        perror("Error on child");
742    } else {
743        if (pid == 0) {
744            execvp("uname","uname -r", NULL); // child is replaced
745            _exit(1);
746        }
747    }
747
748    if (pid < 0) {
749        perror("Error on child");
750    } else {
751        if (pid == 0) {
752            execvp("ls","ls -l -a", NULL); // child is replaced
753            _exit(1);
754        }
755    }
755
756    if (pid < 0) {
757        perror("Error on child");
758    } else {
759        if (pid == 0) {
760            execvp("ps","ps -f", NULL); // child is replaced
761            _exit(1);
762        }
763    }
763
764    if (pid < 0) {
765        perror("Error on child");
766    } else {
767        if (pid == 0) {
768            execvp("uname","uname -r", NULL); // child is replaced
769            _exit(1);
770        }
771    }
771
772    if (pid < 0) {
773        perror("Error on child");
774    } else {
775        if (pid == 0) {
776            execvp("ls","ls -l -a", NULL); // child is replaced
777            _exit(1);
778        }
779    }
779
780    if (pid < 0) {
781        perror("Error on child");
782    } else {
783        if (pid == 0) {
784            execvp("ps","ps -f", NULL); // child is replaced
785            _exit(1);
786        }
787    }
787
788    if (pid < 0) {
789        perror("Error on child");
790    } else {
791        if (pid == 0) {
792            execvp("uname","uname -r", NULL); // child is replaced
793            _exit(1);
794        }
795    }
795
796    if (pid < 0) {
797        perror("Error on child");
798    } else {
799        if (pid == 0) {
800            execvp("ls","ls -l -a", NULL); // child is replaced
801            _exit(1);
802        }
803    }
803
804    if (pid < 0) {
805        perror("Error on child");
806    } else {
807        if (pid == 0) {
808            execvp("ps","ps -f", NULL); // child is replaced
809            _exit(1);
810        }
811    }
811
812    if (pid < 0) {
813        perror("Error on child");
814    } else {
815        if (pid == 0) {
816            execvp("uname","uname -r", NULL); // child is replaced
817            _exit(1);
818        }
819    }
819
820    if (pid < 0) {
821        perror("Error on child");
822    } else {
823        if (pid == 0) {
824            execvp("ls","ls -l -a", NULL); // child is replaced
825            _exit(1);
826        }
827    }
827
828    if (pid < 0) {
829        perror("Error on child");
830    } else {
831        if (pid == 0) {
832            execvp("ps","ps -f", NULL); // child is replaced
833            _exit(1);
834        }
835    }
835
836    if (pid < 0) {
837        perror("Error on child");
838    } else {
839        if (pid == 0) {
840            execvp("uname","uname -r", NULL); // child is replaced
841            _exit(1);
842        }
843    }
843
844    if (pid < 0) {
845        perror("Error on child");
846    } else {
847        if (pid == 0) {
848            execvp("ls","ls -l -a", NULL); // child is replaced
849            _exit(1);
850        }
851    }
851
852    if (pid < 0) {
853        perror("Error on child");
854    } else {
855        if (pid == 0) {
856            execvp("ps","ps -f", NULL); // child is replaced
857            _exit(1);
858        }
859    }
859
860    if (pid < 0) {
861        perror("Error on child");
862    } else {
863        if (pid == 0) {
864            execvp("uname","uname -r", NULL); // child is replaced
865            _exit(1);
866        }
867    }
867
868    if (pid < 0) {
869        perror("Error on child");
870    } else {
871        if (pid == 0) {
872            execvp("ls","ls -l -a", NULL); // child is replaced
873            _exit(1);
874        }
875    }
875
876    if (pid < 0) {
877        perror("Error on child");
878    } else {
879        if (pid == 0) {
880            execvp("ps","ps -f", NULL); // child is replaced
881            _exit(1);
882        }
883    }
883
884    if (pid < 0) {
885        perror("Error on child");
886    } else {
887        if (pid == 0) {
888            execvp("uname","uname -r", NULL); // child is replaced
889            _exit(1);
890        }
891    }
891
892    if (pid < 0) {
893        perror("Error on child");
894    } else {
895        if (pid == 0) {
896            execvp("ls","ls -l -a", NULL); // child is replaced
897            _exit(1);
898        }
899    }
899
900    if (pid < 0) {
901        perror("Error on child");
902    } else {
903        if (pid == 0) {
904            execvp("ps","ps -f", NULL); // child is replaced
905            _exit(1);
906        }
907    }
907
908    if (pid < 0) {
909        perror("Error on child");
910    } else {
911        if (pid == 0) {
912            execvp("uname","uname -r", NULL); // child is replaced
913            _exit(1);
914        }
915    }
915
916    if (pid < 0) {
917        perror("Error on child");
918    } else {
919        if (pid == 0) {
920            execvp("ls","ls -l -a", NULL); // child is replaced
921            _exit(1);
922        }
923    }
923
924    if (pid < 0) {
925        perror("Error on child");
926    } else {
927        if (pid == 0) {
928            execvp("ps","ps -f", NULL); // child is replaced
929            _exit(1);
930        }
931    }
931
932    if (pid < 0) {
933        perror("Error on child");
934    } else {
935        if (pid == 0) {
936            execvp("uname","uname -r", NULL); // child is replaced
937            _exit(1);
938        }
939    }
939
940    if (pid < 0) {
941        perror("Error on child");
942    } else {
943        if (pid == 0) {
944            execvp("ls","ls -l -a", NULL); // child is replaced
945            _exit(1);
946        }
947    }
947
948    if (pid < 0) {
949        perror("Error on child");
950    } else {
951        if (pid == 0) {
952            execvp("ps","ps -f", NULL); // child is replaced
953            _exit(1);
954        }
955    }
955
956    if (pid < 0) {
957        perror("Error on child");
958    } else {
959        if (pid == 0) {
960            execvp("uname","uname -r", NULL); // child is replaced
961            _exit(1);
962        }
963    }
963
964    if (pid < 0) {
965        perror("Error on child");
966    } else {
967        if (pid == 0) {
968            execvp("ls","ls -l -a", NULL); // child is replaced
969            _exit(1);
970        }
971    }
971
972    if (pid < 0) {
973        perror("Error on child");
974    } else {
975        if (pid == 0) {
976            execvp("ps","ps -f", NULL); // child is replaced
977            _exit(1);
978        }
979    }
979
980    if (pid < 0) {
981        perror("Error on child");
982    } else {
983        if (pid == 0) {
984            execvp("uname","uname -r", NULL); // child is replaced
985            _exit(1);
986        }
987    }
987
988    if (pid < 0) {
989        perror("Error on child");
990    } else {
991        if (pid == 0) {
992            execvp("ls","ls -l -a", NULL); // child is replaced
993            _exit(1);
994        }
995    }
995
996    if (pid < 0) {
997        perror("Error on child");
998    } else {
999        if (pid == 0) {
1000            execvp("ps","ps -f", NULL); // child is replaced
1001            _exit(1);
1002        }
1003    }
1003
1004    if (pid < 0) {
1005        perror("Error on child");
1006    } else {
1007        if (pid == 0) {
1008            execvp("uname","uname -r", NULL); // child is replaced
1009            _exit(1);
1010        }
1011    }
1011
1012    if (pid < 0) {
1013        perror("Error on child");
1014    } else {
1015        if (pid == 0) {
1016            execvp("ls","ls -l -a", NULL); // child is replaced
1017            _exit(1);
1018        }
1019    }
1019
1020    if (pid < 0) {
1021        perror("Error on child");
1022    } else {
1023        if (pid == 0) {
1024            execvp("ps","ps -f", NULL); // child is replaced
1025            _exit(1);
1026        }
1027    }
1027
1028    if (pid < 0) {
1029        perror("Error on child");
1030    } else {
1031        if (pid == 0) {
1032            execvp("uname","uname -r", NULL); // child is replaced
1033            _exit(1);
1034        }
1035    }
1035
1036    if (pid < 0) {
1037        perror("Error on child");
1038    } else {
1039        if (pid == 0) {
1040            execvp("ls","ls -l -a", NULL); // child is replaced
1041            _exit(1);
1042        }
1043    }
1043
1044    if (pid < 0) {
1045        perror("Error on child");
1046    } else {
1047        if (pid == 0) {
1048            execvp("ps","ps -f", NULL); // child is replaced
1049            _exit(1);
1050        }
1051    }
1051
1052    if (pid < 0) {
1053        perror("Error on child");
1054    } else {
1055        if
```



execute.c — ad_pr2020

act_colab04 > execute.c > handler(int)

```
21 #include <time.h>
22 #include <pwd.h>
23 #include <grp.h>
24 #include <limits.h>
25 #include <math.h>
26 #include <setjmp.h>
27 #include <signal.h>
28 jmp_buf env1, env2, env3;
29 typedef unsigned long ulong;
30
31 void handler(int sig) {
32     if (sig == SIGUSR1) {
33         longjmp(env1, 1);
34     } else if (sig == SIGUSR2) {
35         longjmp(env2, 1);
36     } else if (sig == SIGHUP) {
37         longjmp(env3, 1);
38     }
39 }
40
41 int main(int argc, char* argv[]) {
42     int file1, file2, file3;
43     signal(SIGUSR1, handler);
44     signal(SIGUSR2, handler);
45 }
```

OUTPUT TERMINAL DEBUG CONSOLE PROBLEMS

1: zsh

m4uz@MacBook-Pro-de-Irving act_colab04 %



execute.c — ad_pr2020

C execute.c act_colab04 ● C execute.c ~/.../Telegram Desktop C main.c C program.c

act_colab04 > C execute.c > main(int, char * [])

```
62     if ((file2 = open(argv[2], O_RDONLY)) < 0) {
63         fprintf(stderr, "%s: the file %s does not exist\n", argv[0], argv[2]);
64         return -4;
65     }
66     if ((file3 = open(argv[3], O_RDONLY)) < 0) {
67         fprintf(stderr, "%s: the file %s does not exist\n", argv[0], argv[3]);
68         return -5;
69     }
70     ulong f_size;
71     f_size = lseek(file1, 0, SEEK_END);
72     if (f_size < 1) {
73         fprintf(stderr, "%s: the file %s is empty\n", argv[0], argv[1]);
74         return -6;
75     }
76     f_size = lseek(file2, 0, SEEK_END);
77     if (f_size < 1) { (char [26])"%s: the file %s is empty\n"
78         fprintf(stderr, "%s: the file %s is empty\n", argv[0], argv[2]);
79         return -7;
80     }
81     f_size = lseek(file3, 0, SEEK_END);
82     if (f_size < 1) {
83         fprintf(stderr, "%s: the file %s is empty\n", argv[0], argv[3]);
84         return -8;
85     }
```

OUTPUT TERMINAL DEBUG CONSOLE PROBLEMS 2

1: zsh

m4uz@MacBook-Pro-de-Irving act_colab04 %



master* 2 ▲ 0 1 hr 2 mins

Ln 78, Col 42 Spaces: 4 UTF-8 LF C Mac



execute.c — ad_pr2020

act_colab04 > C execute.c > `readFile()`

```
30
31 void handler(int sig) {
32     if (sig == SIGUSR1) {
33         longjmp(env1, 1);
34     } else if (sig == SIGUSR2) {
35         longjmp(env2, 1);
36     } else if (sig == SIGHUP) {
37         longjmp(env3, 1);
38     }
39 }
40
41 void readFile(){
42     FILE *file = fopen("file1.txt", "r");
43     if(file){
44         int array[10];
45         char buffer[BUFSIZ], *ptr;
46         for (size_t i = 0; fgets(buffer, sizeof buffer, file); i++)
47         {
48             array[i] = buffer;
49         }
50         for(int k = 0; k < 10; k++){
51             printf("%d ", array[k]);
52         }
53     }
}
```

OUTPUT TERMINAL DEBUG CONSOLE PROBLEMS (2)

```
m4uz@MacBook-Pro-de-Irving act_colab04 % gcc execute.c -o execute
execute.c:48:22: warning: incompatible pointer to integer conversion assigning to 'int' from 'char [1024]' [-Wint-conversion]
    array[i] = buffer;
                  ^~~~~~
```

1 warning generated.

```
m4uz@MacBook-Pro-de-Irving act_colab04 %
```



execute.c — ad_pr2020

C execute.c act_colab04 X C program.c ~/.../archivos_fundamentos C execute.c ~/.../Telegram Desktop C main.c C program.c /Volumes/.../actividad_colaborativa_01 ⓘ ⓘ ...

act_colab04 > C execute.c > readFile()

```
32     longjmp(env1, 1);
33 } else if (sig == SIGUSR2) {
34     longjmp(env2, 1);
35 } else if (sig == SIGHUP) {
36     longjmp(env3, 1);
37 }
38 }
39 }

40 void readFile(){
41     FILE *file = fopen("file1.txt", "r");
42     if(file){
43         char **array = malloc(sizeof(char *) * 10);
44         char buffer[10];
45         for (size_t i = 0; fgets(buffer, 10, file); i++)
46         {
47             array[i] = malloc(10);
48             strcpy(array[i], buffer);
49         }
50         for(int k = 0; k < 10; k++){
51             printf("%s", array[k]);
52         }
53     }
54 }
55
56 }
```

OUTPUT TERMINAL DEBUG CONSOLE PROBLEMS (2)

1: zsh

```
printf("%c ", array[k]);
~~~ ^~~~~~
%s

1 warning generated.
m4uz@MacBook-Pro-de-Irving act_colab04 % gcc execute.c -o execute
m4uz@MacBook-Pro-de-Irving act_colab04 % ./execute
ls,ls -l -a
(null)(null)(null)(null)(null)(null)(null)%
m4uz@MacBook-Pro-de-Irving act_colab04 %
```

master* ⌂ ⌂ 2 ▲ 0 ⌂ 1 hr 33 mins

Ln 50, Col 10 Spaces: 4 UTF-8 LF C Mac ⌂ ⌂



execute.c — ad_pr2020

C execute.c act_colab04 ● C program.c ~/.../archivos_fundamentos C execute.c ~/.../Telegram Desktop C main.c C program.c /Volumes/.../actividad_colaborativa_01 ⓘ ⓘ ...

act_colab04 > C execute.c > readdFile()

```
58
59 }
60
61
62 int readdFile( ){
63     int source;
64     char cmd;
65
66 }
67
68 int main(int argc, char* argv[]) {
69     // int file1, file2, file3;
70     // signal(SIGUSR1, handler);
71     // signal(SIGUSR2, handler);
72     // signal(SIGHUP, handler);
73
74     // if (argc != 4) {
75     //     fprintf(stderr, "usage: %s file1 file2 file3\n", argv[0]);
76     //     return -2;
77     // }
78     // /* Check if the file exists */
79     // if((file1 = open(argv[1], O_RDONLY)) < 0)
80     // {
81         // fprintf(stderr, "%s: the file %s does not exist\n", argv[0], argv[1]);
82     }
```

OUTPUT TERMINAL DEBUG CONSOLE PROBLEMS (2)

1: zsh

```
m4uz@MacBook-Pro-de-Irving act_colab04 % gcc execute.c -o execute
execute.c:54:24: warning: array index 1 is past the end of the array (which contains 1 element) [-Warray-bounds]
    printf("%c\n", filestring[1]);
                           ^ ~~~~~
execute.c:45:9: note: array 'filestring' declared here
    char filestring[1];
                           ^
1 warning generated.
m4uz@MacBook-Pro-de-Irving act_colab04 %
```

master* ⌂ 2 ▲ 0 ⌂ 2 hrs 19 mins

Ln 64, Col 14 Spaces: 4 UTF-8 LF C Mac ⌂ ⌂



execute.c — ad_pr2020

execute.c act_colab04 × execute.c ~/.../señales program.c ~/.../archivos_fundamentos execute.c ~/.../Telegram Desktop main.c program.c /Volumes/.../activida

act_colab04 > execute.c > ...

```
59
60
61 // int readdFile( ){
62 //     int source;
63 //     char *cmd;
64 //     char *argv[100];
65 //     char s[2] = ",";
66 //     FILE *file = fopen("file1.txt", "r");
67 //     if(file){
68 //         char *buffer;
69 //         int size = lseek(source, 0, SEEK_END);
70 //         buffer = (unsigned int*) malloc(sizeof(unsigned int) * size);
71 //         lseek(source, 0, SEEK_SET);
72 //         read(source, buffer, size);
73
74 //         cmd = strtok(buffer, s);
75 //         argv[0] = cmd;
76 //         cmd = strtok(NULL, s);
77 //         int i = 1;
78 //         for (i = 1; cmd != NULL; i++)
79 //         {
80 //             cmd = strtok(NULL, " ");
81 //             argv[i] = cmd;
82 //         }
83 //         argv[i] = 0;
84 //         for (int i = 0; i < sizeof argv; i++)
```

OUTPUT TERMINAL DEBUG CONSOLE PROBLEMS 4

m4uz@MacBook-Pro-de-Irving act_colab04 %

1: zsh + - ×



C execute.c act_colab04 X

C execute.c ~/.../señales

C program.c ~/.../archivos_fundamentos

C execute.c ~/.../Telegram Desktop

C main.c

C program.c /Volumes/.../activida

t1 □ ...

```
act_colab04 > C execute.c > ⚙ main(int argc, char* argv[]) {
140     execlp(argv[1], argv[1], NULL);
141 }
142     sleep(3);
143     break;
144     printf("Ending...");
145     _exit(0);
146 }
147 }

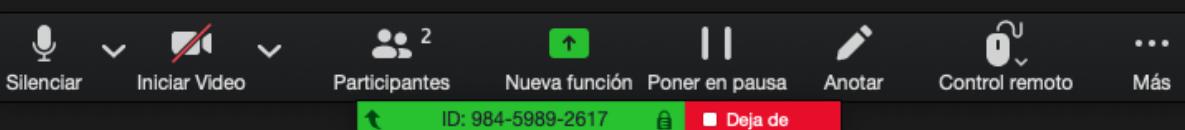
148 int main(int argc, char* argv[]) {
149     // int file1, file2, file3;
150     // signal(SIGUSR1, handler);
151     // signal(SIGUSR2, handler);
152     // signal(SIGHUP, handler);
153
154     if (argc != 4) {
155         fprintf(stderr, "usage: %s file1 file2 file3\n", argv[0]);
156         return -2;
157     }
158     /* Check if the file exists */
159     if ((file1 = open(argv[1], O_RDONLY)) < 0)
160     {
161         fprintf(stderr, "%s: the file %s does not exist\n", argv[0], argv[1]);
162         return -3;
163     }
164     if ((file2 = open(argv[2], O_RDONLY)) < 0)
165         fprintf(stderr, "%s: the file %s does not exist\n", argv[0], argv[2]);
```



OUTPUT TERMINAL DEBUG CONSOLE PROBLEMS 5

1: zsh

m4uz@MacBook-Pro-de-Irving:~/Desktop\$





C execute.c act_colab04 X

C execute.c ~/.../señales

C program.c ~/.../archivos_fundamentos

C execute.c ~/.../Telegram Desktop

C main.c

C program.c /Volumes/.../activida

...

act_colab04 > C execute.c > main(int, char * [])

```
107     l1 = strtok(cmd, ",");
108     l2 = strtok(NULL, ",");
109     switch (signalnum)
110     {
111     case SIGUSR1:
112         if ((pid = fork()) < 0)
113         {
114             perror("No child process");
115         }
116         else if (pid == 0)
117         {
118             execlp(l1, l2, NULL);
119         }
120         sleep(3);
121         break;
122     case SIGUSR2:
123         if ((pid = fork()) < 0)
124         {
125             perror("No child process");
126         }
127         else if (pid == 0)
128         {
129             execlp(l1, l2, NULL);
130         }
131         sleep(3);
132         break;
```



OUTPUT

TERMINAL

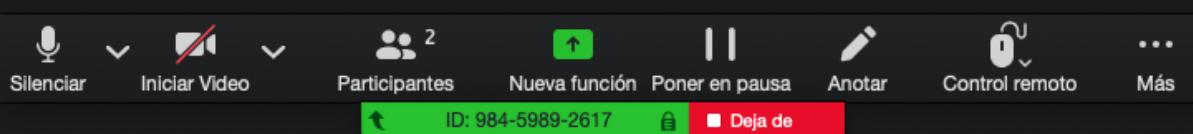
DEBUG CONSOLE

PROBLEMS 5

1: zsh

+

m4uz@MacBook-Pro-de-Irving act_colab04 %



File Edit Selection View Go Run Terminal Help



execute.c

new.c

x

test.c



```
act_colab04 > C new.c > main(int, char *[])
141     f_size = lseek(title3, 0, SEEK_END);
142     if (f_size < 1) {
143         fprintf(stderr, "%s: the file %s is empty\n", argv[0], argv[3]);
144         return -8;
145     }

    // Signals
    signal(SIGUSR1, processSignal);
    signal(SIGUSR2, processSignal);
    signal(SIGPWR, processSignal);
    signal(SIGINT, processSignal);

    puts("Waiting for a signal...");
    while(1);
    return 0;
156 }
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

1: bash

```
Waiting for a signal...
signal 1
Linux
signal 2
signal 3
end
Ending...
proc 3963
UID      PID  PPID  C STIME TTY          TIME CMD
root    2521  2351  0 18:31 pts/3    00:00:00 /bin/bash
root    3399  2521  82 18:53 pts/3    00:18:28 ./execute file1.txt file2.txt file3.txt
root    3418  3399  0 18:54 pts/3    00:00:00 [uname] <defunct>
root    3950  2521  0 19:15 pts/3    00:00:00 /bin/bash ./test.sh execute.c
root    3983  3950  0 19:15 pts/3    00:00:00 ps -f
final grade: 100
root@kali:~/ap2020/act_colab04# ./test.sh execute.c
```

execute.c — ad_pr2020

C execute.c x C execute (2).c

```
act_colab04 > C execute.c > main(int, char * [])
112     }
113     }
114     f_size = lseek(file2, 0, SEEK_END);
115     if (f_size < 1) {
116         fprintf(stderr, "%s: the file %s is empty\n", argv[0], argv[2]);
117         return -7;
118     }
119     f_size = lseek(file3, 0, SEEK_END);
120     if (f_size < 1) {
121         fprintf(stderr, "%s: the file %s is empty\n", argv[0], argv[3]);
122         return -8;
123     }
124     /* Get the commands from files */
125     // FILE 1
126     FILE* f1 = fopen(argv[1], "r");
127     readFromFile(f1, cmd1);
128     l1 = strtok(cmd1, ",");
129     l2 = strtok(NULL, ",");
130     // FILE 2
131     FILE* f2 = fopen(argv[2], "r");
132     readFromFile(f2, cmd2);
133     l3 = strtok(cmd2, ",");
134     l4 = strtok(NULL, ",");
135     // FILE 3
136     FILE* f3 = fopen(argv[3], "r");
137     readFromFile(f3, cmd3);
138     l5 = strtok(cmd3, ",");
139     l6 = strtok(NULL, ",");
140     // Signals
141     signal(SIGUSR1, processSignal);
142     signal(SIGUSR2, processSignal);
143     signal(SIGPWR, processSignal);
144     signal(SIGINT, processSignal);
145     puts("Waiting for a signal...");
```

