Course project: Operating Systems

Topic :

Write a shell script that produces a file of sequential numbers

Desislava Milusheva
471219007

Description of the task

Write a shell script that produces a file of sequential numbers by reading the last number in the file, adding 1 to it, and then appending to the file. Run one instance of the script in the background and one in the foreground, each accessing the same file.

- How long does it take before a race condition manifests itself?
- What is the critical section?
- Modify the script to prevent the race.

Solution of the task

github.com/desi109/race-task

1. Steps to get the project:

```
git clone https://github.com/desi109/race-task
cd race-task
```

2. Create scripts and start the project:

```
sh task_creation.sh
```

Restart: `sh start.sh`
Restart only race task: `sh race_start.sh`
Restart only no race task: `sh no_race_start.sh`

3. Clear folder as it was before executing `` `sh task_creation.sh` `` :

```
sh clear.sh
```

race.sh file

```
#!/bin/bash

echo "--> Start race.sh"
if test ! -f numbers_race
then
    echo "Create the numbers_race file"
    echo 1 > numbers_race
fi

echo "Repeat 100 times - read and increase last number"
for i in `seq 1 100`;
do
        #Read and increase last number
        LASTNUM=`tail -1 numbers_race`
        LASTNUM=$((LASTNUM + 1))

        echo $LASTNUM >> numbers_race
done
echo "--> Finish race.sh"
```

race_start.sh file

```
#!/bin/bash

echo "Start cleaning numbers_race file..."
> numbers_race
echo "File is clean!"

echo "\n...Start the two race programs at same time to see the race"
sh race.sh &
sh race.sh

sleep 3s
echo "...Stop the two race programs at same time to see the race"
exit 0
```

The race condition occurs when two or more threads are able to access shared data and they try to change it at the same time.

The thread scheduling algorithm can swap between threads at any time, because of that we cannot know the order in which the threads will attempt to access the shared data. Therefore, the result of the change in data dependents on the thread scheduling algorithm.

By starting `race_start.sh`, we can see that both threads are '*racing*' to access or change the data. Problem occur when :

- first thread does a '*check-then-act*':
  1. '*check-1*' and get the `LASTNUM` value
  2. then '*act-1*' (increase `LASTNUM` and add it to `numbers_race` file)

- second thread does '*check-2*' and '*act-2*' to the value in `numbers_race` between the '*check-1*' and the '*act-1*'

To summarize

Question: How long does it take before a race condition manifests itself?

Answer: The race condition occurs when two or more threads are able to access shared data and they try to change it at the same time.

Question: What is the critical section?

Answer: A critical section/region is that part of the program where the shared memory is accessed.

no_race.sh file

```
#!/bin/bash
echo "--> Start no_race.sh"
echo "Create the numbers_no_race file"
if test ! -f numbers_no_race; then
    echo 1 > numbers_no_race
fi

echo "Lock numbers_no_race and do not let interruption"
if ln numbers_no_race numbers_no_race.lock; then
       echo "Repeat 100 times - read and increase last number"
       for i in `seq 1 100`;
       do
              #Read and increase last number
              LASTNUM=`tail -1 numbers_no_race`
              LASTNUM=$((LASTNUM + 1))
              echo $LASTNUM >> numbers_no_race
        done

        echo "Unlock numbers_no_race"
        rm numbers_no_race.lock
fi
echo "--> Finish no_race.sh"
```

no_race_start.sh file

```
#!/bin/bash
echo "Start cleaning numbers_no_race file..."
> numbers_no_race
echo "File is clean!"

echo "\n...Start the two no_race programs at same time"
sh no_race.sh &
sh no_race.sh

sleep 3s
echo "...Stop the two no_race programs at same time"

exit 0
```

The solution for the described problem will be to use:

`ln file file.lock`

...to lock the data file and do not let any interruption.

```
$ sh task_creation.sh
        ********************START********************

        Start cleaning numbers_race file...
        File is clean!

        ...Start the two race programs at same time to see the race
        --> Start race.sh
        Repeat 100 times - read and increase last number
        --> Start race.sh
        Repeat 100 times - read and increase last number
        --> Finish race.sh
        --> Finish race.sh
        ...Stop the two race programs at same time to see the race



        Start cleaning numbers_no_race file...
        File is clean!

        ...Start the two no_race programs at same time
        --> Start no_race.sh
        Create the numbers_no_race file
        Lock numbers_no_race and do not let interruption
        --> Start no_race.sh
        Create the numbers_no_race file
        Lock numbers_no_race and do not let interruption
        Repeat 100 times - read and increase last number
        --> Finish no_race.sh
        Unlock numbers_no_race
        --> Finish no_race.sh
        ...Stop the two no_race programs at same time

        ******************FINISH******************


$ sh clear.sh
        Start cleaning folder...
        Folder is clean!
```

It is good to trace the whole process, so in order to do that, there is some simple logging, which will be displayed when the program is started.

Results

NOTE :
The first column of numbers shows the line number of the text editor.

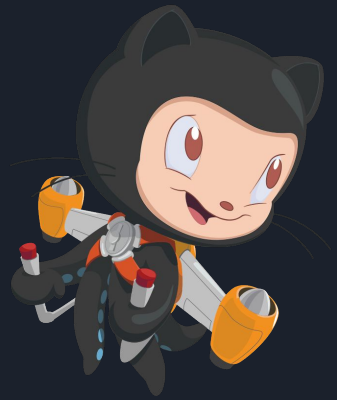
numbers_race file ➟

1 1	49 26	97 70	145 112	192 136
2 1	50 26	98 71	146 112	193 136
3 2	51 27	99 72	147 113	194 137
4 2	52 27	100 73	148 113	195 137
5 3	53 28	101 74	149 114	196 138
6 3	54 29	102 75	150 114	197 138
7 4	55 30	103 76	151 115	198 139
8 4	56 31	104 77	152 115	199 139
9 5	57 32	105 78	153 116	200 140
10 5	58 33	106 79	154 116	201 140
11 6	59 34	107 80	155 117	
12 6	60 35	108 81	156 117	
13 7	61 36	109 82	157 118	
14 7	62 37	110 83	158 118	
15 8	63 38	111 84	159 119	
16 8	64 39	112 85	160 119	
17 9	65 40	113 86	161 120	
18 9	66 41	114 87	162 120	
19 10	67 42	115 88	163 121	
20 10	68 43	116 89	164 121	
21 11	69 44	117 90	165 122	
22 11	70 45	118 91	166 122	
23 12	71 46	119 92	167 123	
24 12	72 47	120 93	168 123	
25 13	73 48	121 94	169 124	
26 13	74 49	122 95	170 124	
27 14	75 50	123 96	171 125	
28 14	76 51	124 96	172 125	
29 15	77 52	125 97	173 126	
30 15	78 53	126 97	174 126	
31 16	79 54	127 98	175 127	
32 16	80 55	128 98	176 127	
33 17	81 56	129 99	177 128	
34 17	82 57	130 100	178 129	
35 18	83 58	131 101	179 130	
36 18	84 59	132 102	180 130	
37 19	85 60	133 103	181 131	
38 19	86 61	134 104	182 131	
39 20	87 61	135 105	183 132	
40 20	88 62	136 106	184 132	
41 21	89 62	137 107	185 133	
42 21	90 63	138 108	186 133	
43 22	91 64	139 109	187 134	
44 22	92 65	140 109	188 134	
45 23	93 66	141 110	189 135	
46 24	94 67	142 110	190 135	
47 25	95 68	143 111	191 136	
48 25	96 69	144 111	192 136	

numbers_no_race file ➟

1 1	49 49	90 90
2 2	50 50	91 91
3 3	51 51	92 92
4 4	52 52	93 93
5 5	53 53	94 94
6 6	54 54	95 95
7 7	55 55	96 96
8 8	56 56	97 97
9 9	57 57	98 98
10 10	58 58	99 99
11 11	59 59	100 100
12 12	60 60	
13 13	61 61	
14 14	62 62	
15 15	63 63	
16 16	64 64	
17 17	65 65	
18 18	66 66	
19 19	67 67	
20 20	68 68	
21 21	69 69	
22 22	70 70	
23 23	71 71	
24 24	72 72	
25 25	73 73	
26 26	74 74	
27 27	75 75	
28 28	76 76	
29 29	77 77	
30 30	78 78	
31 31	79 79	
32 32	80 80	
33 33	81 81	
34 34	82 82	
35 35	83 83	
36 36	84 84	
37 37	85 85	
38 38	86 86	
39 39	87 87	
40 40	88 88	
41 41	89 89	
42 42	90 90	
43 43		
44 44		
45 45		
46 46		
47 47		
48 48		

Thank you for the attention



github.com/desi109/race-task