CSE 344

SYSTEMS PROGRAMMING

SPRING 2021

HOMEWORK #3

REPORT

TURKER TERCAN

171044032

HOMEWORK'S CHALLANGE:

- A process may start with a hot potato. It needs to be switched between randomly selected processes to be cooled down. Communication between process must be performed via FIFOs and every hot potato's data must be stored Shared Memory Segment.

- If all the hot potatoes have been cooled down, all processes should terminate without any memory leak.

- If a process receives a SIGINT signal with CTRL + C, all processes should terminate immediately.

- Shared Memory should be protected with named semaphore, in order to protect critical region.

DESIGN CHOICES:

- There would be N processes and N will have indicated us with an ASCII file. So, N is equal to how many lines in that file. I used below file during this homework's implementation.

- So N is 20 for my case.

main.c | fifonames.txt

```
/tmp/process_1_fifo
/tmp/process_2_fifo
/tmp/process_3_fifo
/tmp/process_4_fifo
/tmp/process_5_fifo
/tmp/process_6_fifo
/tmp/process_7_fifo
/tmp/process_8_fifo
/tmp/process_9_fifo
/tmp/process_10_fifo
/tmp/process_11_fifo
/tmp/process_12_fifo
/tmp/process_13_fifo
/tmp/process_14_fifo
/tmp/process_15_fifo
/tmp/process_16_fifo
/tmp/process_17_fifo
/tmp/process_18_fifo
/tmp/process_19_fifo
/tmp/process_20_fifo
```

- After that, a process should create shared memory segment and align needed size for this process.

- I implemented a way that which process creates shared memory first, it will truncate that space. It will open named share segment with O_CREAT | O_EXCL with these flags. So other processes won't be able to open memory segment just yet.

- I created a temporary semaphore because while first process creating and arranging space for memory segment other processes should have wait.

- Also, first process creates all fifos with mkfifo function while other processes are waiting.

- To able communicate between multiple processes, every process should have open every other process' fifo as write only and its own fifo as read only. But we have a problem, and it is if a process opens a fifo but fifo's other end doesn't have opened yet, process waits until other end to be opened.

- So instead of randomly selecting a fifo in the read file, I used first index of shared memory segment to which process started when index. According to when the process started, it will pick it up that fifo index.

- After that, if the process started with a potato, it would store it in the shared memory segment. It'd choose a randomly selected process and message to it.

- Process reads its own fifo in endless loop. If a message arrives it will pick it up relevant potato in the shared memory segment and if it has not been cool downed yet, it will send it to random process.

- I protected every shared memory segment store function with named semaphore. It wouldn't be any race condition.

- After every potato is cool downed, last receiver send a message to all other fifos which indicates that they should terminate. And program ends successfully.

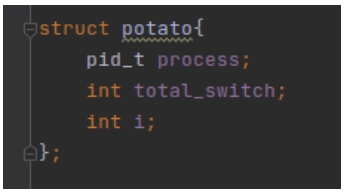
- But if a process ends with a SIGINT signal all the other process also must end. So receviever process must send SIGINT signal for each process ever created since then. But it creates another problem. Where should I know every other process's id? Every process is unrelated.

- So, I used shared memory to store every process' pid and handle it between processes. You can see my memory segment representation in below.

Assume that N = 5 and there are 4 processes created up to now.

INDEX	PROCESS ID	SWITCH	i
0	-1	4	-1
1	4000	20	0
2	4232	15	0
3	0	-1	-1
4	0	-1	-1
5	0	-1	-1
6	4000	null	null
7	4010	null	null
8	4120	null	null
9	4232	null	null
10	null	null	null

- I truncated shared segment as 2N + 1 * sizeof(struct potato).



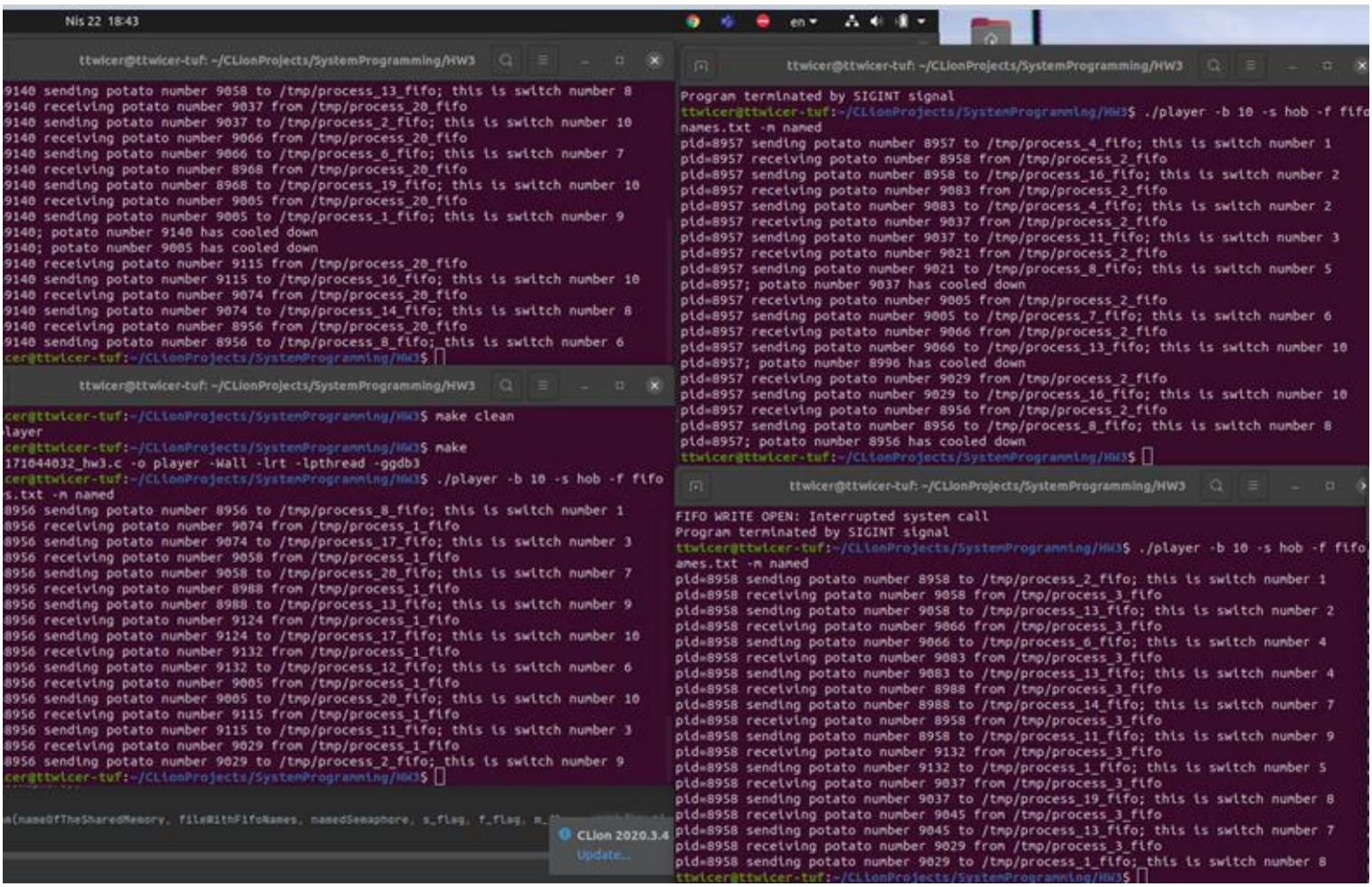
- Index 0 switch indicates how many processes are created up to now.

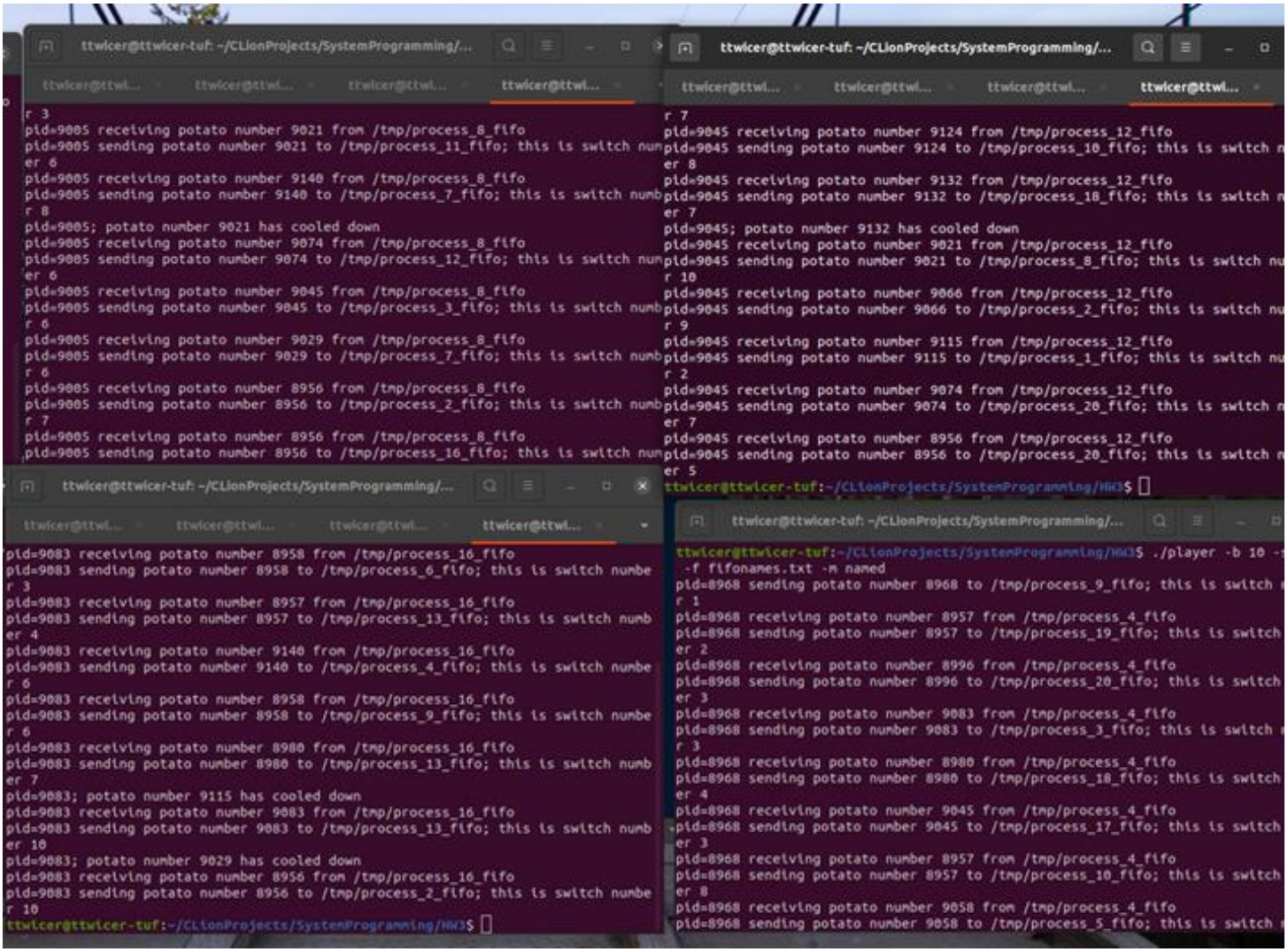
- Index 1 and between N stores hot potatoes it needs to be handled. There are 2 hot potatoes for now as you can see.

- Index N+1 and 2N are used for storing process' pid. So if there is process that receives SIGINT signal, all processes should terminate gracefully.

CONSOLE:

N = 20 PROCESSES





Valgrind Check

```
ttwicer@ttwicer-tuf:~/CLionProjects/SystemProgramming/HW3$ valgrind -s --leak-check=full --track-origins=yes --show-leak-kinds=all ./player -b 10 -s hob -f fifonames.txt -m na
==9115== Memcheck, a memory error detector
==9115== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==9115== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==9115== Command: ./player -b 10 -s hob -f fifonames.txt -m named
==9115==
pid=9115 sending potato number 9115 to /tmp/process_12_fifo; this is switch number 1
pid=9115 receiving potato number 9074 from /tmp/process_17_fifo
pid=9115 sending potato number 9074 to /tmp/process_6_fifo; this is switch number 4
pid=9115 receiving potato number 9045 from /tmp/process_17_fifo
pid=9115 sending potato number 9045 to /tmp/process_4_fifo; this is switch number 4
pid=9115 receiving potato number 9083 from /tmp/process_17_fifo
pid=9115 sending potato number 9083 to /tmp/process_10_fifo; this is switch number 8
pid=9115; potato number 8988 has cooled down
pid=9115 receiving potato number 8996 from /tmp/process_17_fifo
pid=9115 sending potato number 8996 to /tmp/process_2_fifo; this is switch number 10
pid=9115 receiving potato number 9029 from /tmp/process_17_fifo
pid=9115 sending potato number 9029 to /tmp/process_8_fifo; this is switch number 5
pid=9115 receiving potato number 8980 from /tmp/process_17_fifo
pid=9115 sending potato number 8980 to /tmp/process_11_fifo; this is switch number 10
pid=9115; potato number 9124 has cooled down
pid=9115 receiving potato number 8956 from /tmp/process_17_fifo
pid=9115 sending potato number 8956 to /tmp/process_18_fifo; this is switch number 3
==9115==
==9115== HEAP SUMMARY:
==9115==     in use at exit: 0 bytes in 0 blocks
==9115==   total heap usage: 49 allocs, 49 frees, 6,869 bytes allocated
==9115==
==9115== All heap blocks were freed -- no leaks are possible
==9115==
==9115== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
ttwicer@ttwicer-tuf:~/CLionProjects/SystemProgramming/HW3$
```

CTRL + C Signal