Kyung Jae Cha

Prof. Park

CS354

03/21/18

CS 354 Spring 2018

Lab 5: Enhancing Kernel IPC Services: Blocking and Asynchronous IPC (330 pts)

Due: 04/03/2018 (Tue.), 11:59 PM

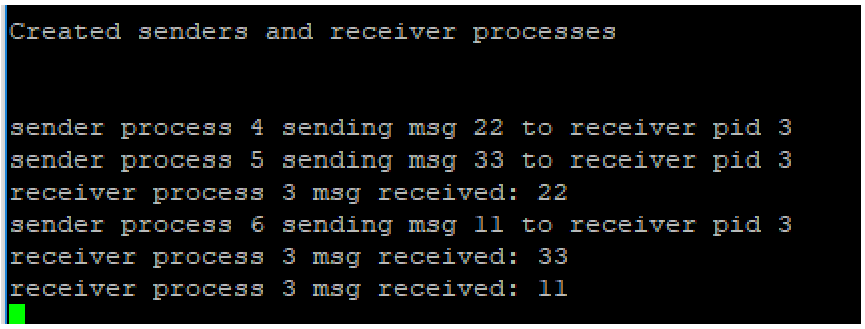
Bonus Problem (man pages for receive() and sendblk() are created in /system)

Problem 3. Blocking message send

Implementation of sendblk()

My implementation toward blocking message send and queueing up to the recipient FIFO waiting queue, started from creating a queue for each process. Then in sendblk(), it checks if the buffer of recipient is full, saves the modified states and flags, queue up the sender process and finally context-switches out sending processes. Because we are queueing the sender processes to the recipient queue, I've created 3 sender processes and one receiver process for my test cases. Each of the sender process sends msg 22, 33 and 11 respectively to receiver process. And the receiver process receives messages every 5 seconds after its woken up from the sleep().

Sendblk() FIFO Queue Results

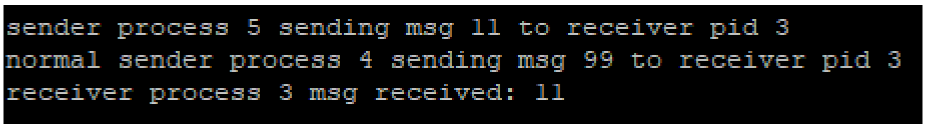


```
Created senders and receiver processes


sender process 4 sending msg 22 to receiver pid 3
sender process 5 sending msg 33 to receiver pid 3
receiver process 3 msg received: 22
sender process 6 sending msg 11 to receiver pid 3
receiver process 3 msg received: 33
receiver process 3 msg received: 11
```

Sendblk() -> Send() Result

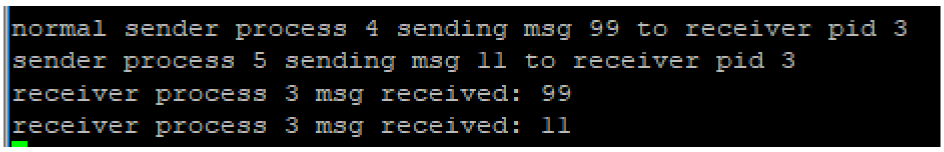
To test normal send() does not get impacted by sendblk(), I've tested two different ways. First, sendblk() followed by send() call was made. And the result came out as sender receiver process grabbing the message sent by sendblk() and ignoring the message sent by send(). The result seems reasonable, because in normal send



```
sender process 5 sending msg 11 to receiver pid 3
normal sender process 4 sending msg 99 to receiver pid 3
receiver process 3 msg received: 11
```

Send() -> Sendblk() Result

Send followed by sendblk() showed a different outcome. This is because normal send finds the buffer empty and sends the message to recipient, and sendblk() queues up the msg because the message buffer is full.



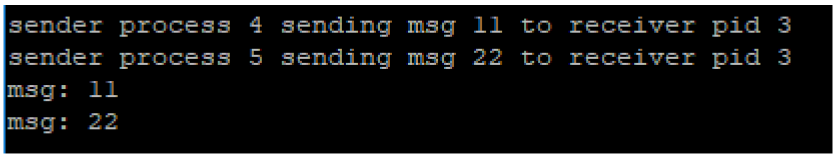
```
normal sender process 4 sending msg 99 to receiver pid 3
sender process 5 sending msg 11 to receiver pid 3
receiver process 3 msg received: 99
receiver process 3 msg received: 11
```

Problem 4. Asynchronous IPC with callback function

Implementation on Asynchronous IPC with CB function

My implementation on designing asynchronous inter-process communication with callbacks is based on the nature of XINU's sleep() function. Because we need to preserve isolation/protection when kernel provides this functionality and while XINU only runs in kernel mode, we need XINU to look at receiver process when it context switches out from sender process. In addition, receiver process gets injected a callback function before resuming the process, so that only after returning from the function, the process can resume. By placing register check at the end of sleep() (that is after the context-switch), I've checked whether receiver process has been registered a callback. If it has, run the function before resuming its process. As mentioned, although XINU runs everything in kernel, isolation and protection are achieved by 1) separating function address pointer and the message buffer that callback function receives away from other memory and 2) controlling interrupts during registration and executing a callback function.

For testing, my receiver registers a simple callback function that receives the message buffer and saves it in the global variable. My receiver that tests asynchronous IPC with callback function first registers a callback function then it goes into endless loop that sleeps every 5 seconds. Because my design checks the registers right before resuming from sleeping (blocking), it will trigger the callback function which receives the message in the global message buffer.



```
sender process 4 sending msg 11 to receiver pid 3
sender process 5 sending msg 22 to receiver pid 3
msg: 11
msg: 22
```