LAB 4

Anirudh Pal (pal5)

Part 3.1: Testing 1 Sender and 1 Receiver

Output:

```
Testing Single Sender
PID: 4, PName: receiver0, Time: 6ms, Starting Proc
PID: 5, PName: sender0, Msg: 1234, To PID: 4, Time: 56ms, Start Sending
PID: 5, PName: sender0, Msg: 1234, To PID: 4, Time: 57ms, Start Done
PID: 4, PName: receiver0, Msg: 1234, Time: 58ms, From Callback
```

Results: The receiver receives the message and runs the callback immediately when it gets scheduled.

Part 3.2: Testing 1 Sender and 1 Receiver which sleeps for 1 sec.

Output:

```
Testing Single Sender with Sleeping Receiver
PID: 6, PName: receiverslp0, Time: 1086ms, Starting Proc
PID: 7, PName: sender1, Msg: 4321, To PID: 6, Time: 1087ms, Start Sending
PID: 7, PName: sender1, Msg: 4321, To PID: 6, Time: 1088ms, Start Done
PID: 6, PName: receiverslp0, Msg: 4321, Time: 2087ms, From Callback
```

Results: The receiver receives the message and runs the callback immediately when it gets scheduled after sleeping.

Part 3.3: Testing 4 Sender and 1 Receiver.

Output:

```
Testing Multiple Senders
PID: 8, PName: receiver1, Time: 3088ms, Starting Proc
PID: 9, PName: sender2, Msg: 1111, To PID: 8, Time: 3113ms, Start Sending
PID: 9, PName: sender2, Msg: 1111, To PID: 8, Time: 3114ms, Start Done
PID: 8, PName: receiver1, Msg: 1111, Time: 3139ms, From Callback
PID: 10, PName: sender3, Msg: 2222, To PID: 8, Time: 4112ms, Start Sending
PID: 10, PName: sender3, Msg: 2222, To PID: 8, Time: 4113ms, Start Done
```

```
PID: 8, PName: receiver1, Msg: 2222, Time: 4138ms, From Callback
PID: 11, PName: sender4, Msg: 3333, To PID: 8, Time: 5112ms, Start Sending
PID: 11, PName: sender4, Msg: 3333, To PID: 8, Time: 5113ms, Start Done
PID: 8, PName: receiver1, Msg: 3333, Time: 5114ms, From Callback
PID: 12, PName: sender5, Msg: 4444, To PID: 8, Time: 6112ms, Start Sending
PID: 12, PName: sender5, Msg: 4444, To PID: 8, Time: 6113ms, Start Done
PID: 8, PName: receiver1, Msg: 4444, Time: 6138ms, From Callback
```

Results: The receiver receives the message and runs the callback immediately when it gets scheduled and does so successively for each sender.

Part 4.1: Testing SIGRECV using Part 3 tests.

Output:

```
Testing SIGRECV

Testing Single Sender
PID: 19, PName: receiversig0, Time: 13212ms, Starting Proc
PID: 20, PName: sender0, Msg: 1234, To PID: 19, Time: 13215ms, Start Sending
PID: 20, PName: sender0, Msg: 1234, To PID: 19, Time: 13216ms, Start Done
PID: 19, PName: receiversig0, Msg: 1234, Time: 13366ms, From Callback

Testing Single Sender with Sleeping receiversig
PID: 21, PName: receiversigslp0, Time: 14215ms, Starting Proc
PID: 22, PName: sender1, Msg: 4321, To PID: 21, Time: 14216ms, Start Sending
PID: 22, PName: sender1, Msg: 4321, To PID: 21, Time: 14217ms, Start Done
PID: 21, PName: receiversigslp0, Msg: 4321, Time: 15216ms, From Callback

Testing Multiple Senders
PID: 23, PName: receiversig1, Time: 16215ms, Starting Proc
PID: 24, PName: sender2, Msg: 1111, To PID: 23, Time: 16217ms, Start Sending
PID: 24, PName: sender2, Msg: 1111, To PID: 23, Time: 16218ms, Start Done
PID: 23, PName: receiversig1, Msg: 1111, Time: 16392ms, From Callback
PID: 25, PName: sender3, Msg: 2222, To PID: 23, Time: 17217ms, Start Sending
PID: 25, PName: sender3, Msg: 2222, To PID: 23, Time: 17218ms, Start Done
PID: 23, PName: receiversig1, Msg: 2222, Time: 17318ms, From Callback
PID: 26, PName: sender4, Msg: 3333, To PID: 23, Time: 18217ms, Start Sending
PID: 26, PName: sender4, Msg: 3333, To PID: 23, Time: 18218ms, Start Done
PID: 23, PName: receiversig1, Msg: 3333, Time: 18243ms, From Callback
PID: 27, PName: sender5, Msg: 4444, To PID: 23, Time: 19217ms, Start Sending
PID: 27, PName: sender5, Msg: 4444, To PID: 23, Time: 19218ms, Start Done
PID: 23, PName: receiversig1, Msg: 4444, Time: 19368ms, From Callback
```

Results: The receiver receives the message and runs the callback immediately when it gets scheduled and does so successively for each sender.

Part 4.2: Testing SIGXCPU with 2 Procs with Infinite Loops and oargs value of 500.

Output:

```
Testing SIGXCPU with 2 Procs
PID: 13, PName: receivercpu0, Time: 7112ms, CPU: 0ms Starting Proc
PID: 14, PName: receivercpu1, Time: 7137ms, CPU: 0ms Starting Proc
PID: 14, PName: receivercpu1, Time: 9089ms, CPU: 501ms From Callback
PID: 13, PName: receivercpu0, Time: 9163ms, CPU: 501ms From Callback
```

Results: The procs have the callback run immediately after the Gross CPU Usage reaches 500ms.

Part 4.3: Testing SIGTIME with 2 Procs with Infinite Loops and xalarm(clktimemilli + 500).

Output:

```
Testing SIGTIME with 2 Procs without ROP
PID: 15, PName: receiveralm0, Time: 10162ms, CPU: 0ms Starting Proc
PID: 16, PName: receiveralm1, Time: 10187ms, CPU: 0ms Starting Proc
PID: 15, PName: receiveralm0, Time: 10761ms, CPU: 100ms From Callback
PID: 16, PName: receiveralm1, Time: 10786ms, CPU: 100ms From Callback
```

Results: The procs run the callback when clktimemilli increases by 500. This happens in the current process context without ROP as the registering process is current when the timer rings.

Part 4.4: Testing SIGTIME with 2 Procs with sleep(1) and xalarm(clktimemilli + 500).

Output:

```
Testing SIGTIME with 2 Procs with ROP
PID: 17, PName: receiveralmslp0, Time: 11212ms, CPU: 0ms Starting Proc
PID: 18, PName: receiveralmslp1, Time: 11213ms, CPU: 0ms Starting Proc
```

```
PID: 17, PName: receiveralmslp0, Time: 12213ms, CPU: 1ms From Callback
PID: 18, PName: receiveralmslp1, Time: 12214ms, CPU: 1ms From Callback
```

Results: The procs run the callback when clktimemilli increases by 500 and when the process gets context switched in after sleep. This does use the ROP mechanism.

Part 4.5: Testing SIGTIME, SIGCPU, SIGRECV with 1 receiver and 1 sender. The receiver has oargs value of 5 and xalarm(clktimemilli + 500) and sleep(1).

Output:

```
Testing all Signals
PID: 28, PName: receivermix0, Time: 20217ms, Starting Proc
PID: 28, PName: receivermix0, Time: 20223ms, CPU: 6ms From Callback for SIGXCPU
PID: 29, PName: sender6, Msg: 1010, To PID: 28, Time: 20242ms, Start Sending
PID: 29, PName: sender6, Msg: 1010, To PID: 28, Time: 20243ms, Start Done
PID: 28, PName: receivermix0, Msg: 1010, Time: 20441ms, From Callback for SIGRECV
PID: 28, PName: receivermix0, Time: 21442ms, CPU: 26ms From Callback for SIGTIME
```

Results: The procs run the callback for SIGXCPU when it hits 5ms Gross CPU Usage. Then it receives a message and runs callback for SIGRECV. Then is sleeps for 1 sec and while sleeping receives a SIGTIME for which it runs the callback when it wakes up.

NOTE: All of this was run in one go and all receivers were still running. This was done to ensure signals can fly around while the CPU is occupied with other stuff. This is also why there is a small divergence in clktimemilli. The test code has been left uncommented-out in main.c

Part 4.6: Implementation Details

Details:

- The ROP mechanism saves the return address in the process table and modifies it process stack to jump to the handler when the process is context switched in. The handler after completion jumps back to the previously saved return address.
- For SIGXCPU all we do is monitor CPU Usage in clkhandler.c and run the callback if it exceeds the oargs value.

- For SIGTIME I use a delta queue similar to the sleep queue to trigger alarms from the clkhandler.c. Then it uses ROP if the process is not current or uses the current context if the registering process is current.
- For SIGRECV I send a signal when send() executes so ROP can account for the handler. I have also kept legacy support for Part 3 which can be done by seeing what type of callback function was registered in process table entry of the receiver.
- I use sendSignal() as a custom function to handle sending Signals to processes. A process can also send a signal to itself, in which case the callback is run immediately. If not the signal sent goes into a sigs[] array in process table entry so when the process becomes current we can run the handlers in the order they came in.

Part 4.6: BONUS

Issues:

- The same signal is dropped until the callback runs. Instead if the signal is sent by another process we should block and chain further signals, specifically for SIGRECV.
- SIGTIME should be handled immediately even if the process is sleeping as it becomes useless if the process sleeps for a long time. Such a system might use context borrowing or pull the process out of sleep and put it back in after running callback. This could be useful for future signal additions.
- SIGCHLD is a signal that is sent to the parent process when a process dies and this was not handled. It can be used to free memory.
- Additionally I don't think the Kernel should alter the process stack and instead use the process table entry and the scheduler to run the callback(in the right context and user mode) before returning from scheduler. The ROP method is error-prone approach to an error-prone problem.