Operating Systems Lab 4

Name : Dhruv Subramanian

Part1 : Signal Handling Subsystem

Changes made (following files):

1)registercbsig.c – handled the three signals.Comments for the code are included.
2)resched.c - included all the calculations for clktimefine from lab2 in this file. I also made calls to the callback function here. All comments for my code have been written in resched.c.
3)clkhandler.c - I also wrote the code to make calls to the callback function here, since the granularity is better in clkhandkler. Comments for the code are included.
4)mysignal.h – define the signal flags. All comments have been added in mysignal.h.
5)process.h – I added entries as a field to the process table. The entries were as follows : prcpuused, recvfunc, alrmfunc,sigxcpufunc, mysigalrmtime,optarg. The comments for why these are added have been made in process.h.
6)xinu.h – included mysignal.h.
7)main.c – test cases to test the functionality of my code.

Test cases. For my code, I reran my test cases from lab4, part2 with registercbsig instead of registercb so as to test if it works for MYSIGRECV. The result is the exact same output as that of lab4 part 2. That means that it works properly.

Output of lab4 part2 :

```
 TEST1 started
Time before send is : 70 && the PIMessage received from sender: "A".
D is : 4
Time after receive is : 70 && the PID is : 3
Test1 passed!
TEST1 finished
TEST2 started
Time before send is : 129 && the PMessage received from sender: "A".
ID is : 9
Time after receive is : 131 && the PID is : 5
Time before send is : 195 && the PID is : 10
Message received from sender: "B".
Time after receive is : 199 && the PID is : 6
Time before send is : 269 && the PID is : 11
Message received from sender: "C".
Time after receive is : 275 && the PID is : 7
Time before send is : 351 && the PID is : 12
Message received from sender: "D".
Time after receive is : 355 && the PID is : 8
Test2 passed!
```

TEST2 finished
TEST3 started
Time before send is : 442 && the PID is : 14
Message received from sender: "A".
Time after receive is : 452 && the PID is : 13
Time before send is : 488 && the PID is : 15
Message received from sender: "B".
Time after receive is : 498 && the PID is : 13
Time before send is : 534 && the PID is : 16
Message received from sender: "C".
Time after receive is : 544 && the PID is : 13
Time before send is : 580 && the PID is : 17
Message received from sender: "D".
Time after receive is : 590 && the PID is : 13
Test3 passed!
TEST3 finished.

Output of the test for MYSIGRECV :

----------MYSIGRECV TEST CASES FOR PART1----------
TEST1 started
Time before send is : 232 && the PMessage received from sender: "A".
ID is : 4
Time after receive is : 232 && the PID is : 3
Test1 passed!
TEST1 finished
TEST2 started
Time before send is : 287 && the PMessage received from sender: "A".
ID is : 9
Time after receive is : 289 && the PID is : 5
Time before send is : 345 && the PMessage received from sender: "B".
ID is : 10
Time after receive is : 345 && the PID is : 6
Time before send is : 408 && the PID is : 11
TimMessage received from sender: "C".
e after receive is : 408 && the PID is : 7
Time before send is : 478 && the PID is : 12
Time after receive mMessage received from sender: "D".
s : 478 && the PID is : 8
Test2 passed!
TEST2 finished
TEST3 started
Time before send is : 555 && the PID is : 14
Time after receive mMessage received from sender: "A".
s : 555 && the PID is : 13
Time before send is : 589 && the PID is : 15
Message received from sender: "B".

Time after receive is : 597 && the PID is : 13
Time before send is : 624 && the PID is : 16
Message received from sender: "C".
Time after receive is : 632 && the PID is : 13
Time before send is : 659 && the PID is : 17
Message received from sender: "D".
Time after receive is : 667 && the PID is : 13
Test3 passed!
TEST3 finished.

For MYSIGXCPU I wrote test cases to see when the current process' cpu used is greater than the the optional argument specified. The test case is in main, and it accurately calls the callback function when cpu used becomes atleast equal to the optional argument.I made fileds in my procent to store the cpuused and the optarg of the current process. Thus we can check in this way for every single process that runs at a given time.

Test case output :

----------MYSIGXCPU TEST CASES FOR PART1----------
TEST1 started
mysigxcpu handler test1 passed
Cpu Time is 500
TEST1 finished

In this case, I made a call to the callback when the cputime was supposed to be atleast 500. And it has worked perfectly.

For MYSIGALRM, I wrote two test cases to test its functionality and I completely made use of the implementation of my global clocktime variable without using a deltalist or queue(data structure). I made a procent field mysigalrmtime to store the initial cloctime + optarg. This would then store it for a particular process whenever it is current. Then before calling the callback function, I would check my global clocktime and see if it exceeds the mysigalrmtime stored earlier. This check was implemented in resched after context switching and clkhandler in the current processes reference frame.

Test case output:

----------MYSIGALRM TEST CASES FOR PART1----------
TEST1 started
The inital clocktime is : 2711
The current clocktime is : 3921
myalrmsignal handler test1 passed
TEST1 finished
TEST2 started
The inital clocktime is : 6737
The current clocktime is : 6740

myalrmsignal handler test2 passed
TEST2 finished

The granularity is a bit off, but it is not off by more than time = 10. In the first testcase, optarg was 1200 and the second case it was 0. But it works correctly as per the lab requirements and has a proper functionality.

Part2 : Memory Garbage Collection

Changes made(following file):

1) process.h – I added a linked list structure to implement the garbage collection.
2) initialize.c – initialized the lined list fields.
3)getmem.c - gets additional space for linked list to store allocated memory.
4)freemem.c – frees up space in block, and it may or may not free up all the memory that was gotten through getmem. Depends on how it is called
5)kill.c – frees up whatever freemem hasn't and the linked list becomes empty.
6)main.c -test cases to test functionality of my code

I create multiple test cases in main to test the functionality of my getmem(),freemem() and kill(). For getmem, I call getmem several times and then I print the amount left in my free list once before calling all the getmems and once after calling all the getmems. The difference is equal to the amount of bytes I have requested.

For freemem, I test it in addition with get meme and see after I get memory and then free all of it, the memlist before all operations and the memlist after are of the same size which means it is working correctly.

For kill(), I made print statements within kill function. I allocated a lot of memory and then Ifreed nearly all of it, except for a little bit. I printed out my linked list node size for the particular process and found that it was equal to the the memory that freemem hadn't been called for. This is then cleared and my linked list becomes empty once kill had performed the function I built it for. It also works very well.

Test cases:

Test1 started
the total length is 250028576
the total length is 250022960
TEST1 finished

TEST2 started

the total length is 250022960
the total length is 250021048
the total length is 250022960
TEST2 started

Both the test work accurately and pass the test cases for getmem(),freemem() and kill().

Bonus Problem :

We can implement a form of garbage collection similar to the way we have done it in part 2 of our lab. What I feel is that processes tend to run and after when we get ready to kill them, we should be able to store the list of processes that we should kill, thereby freeing up memory as and when used. Therefore, we dont really need to wait till the end to free up memory, We can do it as processes are no longer used and no longer become necessary to save space for. This is one of the kernel based solutions that I thought of to help prevent unneeded memory and bloated memory usage. This way there will be less memory stress on a system, and thereby making the user experience much better.