LAB4Answers

Implementation:

The implementation is done over the TS solaris scheduler implemented in the last lab.

The implementation of registercb:

For this implementation, I have added 2 entries in the proctab data structure, hascbfunc and regcbfuncptr which stores the flag validating if a callback function has been registered and the pointer to the call back function respectively. When registercb is called this fields are initiailized for the current process. Now, in resched(), when this above process comes into context the registered call back function whose pointer is stored in regcbfuncptr is called. Thus, calling the callback function just after the registering process comes into context and before any part of this process is executed.

The implementation of registercbsig:

Additional fields have to be added in the proctab table for this function. Two more flags are added hassigxcpu and hassigalrm and these are used for MYSIGXPU signaling and MYSIGALRM signaling case respectively. The flag hascbfunc has been extended for the use of MYSIGRECV signaling case. Also, a field called sigarg is added to hold the optarg argument in the case of the MYSIGXCPU signaling and it is used as a flag in the case of MYSIGALRM signaling.

Based on the value of the asig parameter of this function, this implementation can be broken into 3 parts. When asig is equal to:

- MYSIGRECV: This is same as registercb and hence registercb is called and the same function pointer is passed along.
- MYSIGXCPU: For this case, the sigarg stores the number of cpu cycles, the process has to execute before the registered signaling/ callback function is called. In clkhandler.c, this argument is decremented, similar to preempt field. Once this, goes below 0, the registered callback function is fired.
- MYSIGALRM: For this case, an alarm queue was created along with the insertion and deletion process. This alarm queue is a delta queue and works similar to the sleep queue. Additionally alarms() and alrm_wakeup() system calls were implemented on the lines of sleepms() and wakeup(). In the alrm_wakeup(), I have handled sleeping processes by unsleeping (unsleep() call) them and putting them in the ready state. Also the sigarg flag is set to 1 indicating that on the next context switch the callback function should be called.

Test Cases

Uncomment lab4q2 or lab4q3 lines in the main file.

Test case for registercb:

Three case:

1. A callback function is registered with a receiver process and a sender process sends a message which will be printed on both sides of the communication. On the receiver end, the callback function reads and prints the message sent by the sender. An additional receiver process with a valid callback function is registered but no sender.
   Observation:
   The callback function is called on the receiver and the message sent matches the message received. Also, the second receiver with callback function doesn't print anything as it doesn't receive any message (since there is no sender process).
2. A single receiver process receives messages from multiple sender process. One more receiver process is registered with an invalid function pointer.
   Observation:
   The registered callback function reads only one of the message from one of the sending processes. This is expected as once the registered call back function is executed, it is not called again. This clarifies that registerecb is working correctly as the callback was registered only once. For the other receiver, a error message is printed and the registercb fails.
3. 1 receiver process and a sender process is used. But the sender sends multiple messages after a short sleep.
   Observation:
   The callback function is fired only once and hence only one message is read and printed out. This is as required from the registercb function.

Test cases for registercbsig():

- For MYSIGRECV: same as above and same observations
- For MYSIGXCPU:

  One test case in which 2 receiver process and 2 sender process are created making 2 sender-receiver pairs. The first receiver calls registercbsig with 750 cpu cycle wait for signaling to begin and the second receiver calls registercbsig with 0 cpu cycle wait for signaling to begin. Clearly, the second receiver should behave like MYSIGRECV signaling and first should execute 750 cycles before calling the callback signaling function which will read and print the message from the sender.

  Observation:

  Sending message from PID: 4

```
 At time: 128
 And sending msg: 11111

Process
 Sending message from PID: 6
 At time: 135
 And sending msg: 22222

Recieving in Process : 5
At timestamp: 141
Msg received : 22222
with PID: 5 registering receiver to be invoked after 1 cpu cycles

Recieving in Process : 3
At timestamp: 1850
Msg received : 11111
```

This is as expected.

- For MYSIGALRM:

  Two test cases:

  First test case one receiver with registered call back for reading and printing sent messages and one sender process. The alarms time for the receiver is 500ms and it sleeps for 1000ms.

  Observation:

  ```
  Sending message from PID: 4
   At time: 57
   And sending msg: 11111
  ```

Recieving in Process : 3

At timestamp: 557

Msg received : 11111

This is as expected.

Second test case, one receiver with registered call back for reading and printing sent messages and one sender process. The alarms time for the receiver is 500ms but no sleep.

Observation:

Sending message from PID: 4

At time: 217

And sending msg: 11111

Recieving in Process : 3

At timestamp: 557

Msg received : 11111

This is as expected.

BONUS:

registercbsig(2)

NAME
registercbsig - a function to register callback function for signals

SYNOPSIS
registercbsig(asig, *func,optarg)

DESCRIPTION

registercbsig registers callback function for various signals. There is three type of signal. They are receiving signal, usage signal and alarm signal. In case of usage signal, we need to mention amount of cpu utilization after which the callback function had to be called.

OPTIONS

asig

    Three type of signal
    MYSIGRECV- receiving signal
    MYSIGXCPU- usage signal
    MYSIGALRM-alarm signals

func

Points to the callback function

optarg

    used to specify the cpu time utilization after which callback function should be called

EXAMPLE

registercbsig(MYSIGXCPU,*func, 50)

registercbsig is called for registering a callback function pointed by *func which will be called for 50 msec

registercbsig(MYSIGRECV,*func,0)

registercbsig is called for registering a callback function for receiving signal

registercbsig(MYSIGALRM,*func,0)

registercbsig is called for registering a callback function pointed by *func for alarm signal