CS 503 Lab 3 Answers

Prashant Ravi — ravi18@purdue.edu

March 15, 2016

Files modified/added

create.c ; initialize.c ; ts-disptb.h ; multilevelfbq.h ; multilevelfbq.c; sendbt.c ; resched,c ; receivebt.c ; ready.c ; queue.c ; newqueue.c ; main.c ; getitem.c ; xinu.h ;queue.h ; prototypes.h ; process.h ; myqueue.h ; lab3q4.c ; lab3q3.c

TO RUN LAB 3: please make sure to change lab variable in main to equal to 3, lab2flag =0; TO RUN LAB 2: please make sure to change lab variable in main to equal to 2, with lab2flag =4 or 5, based on subpart.

Question 3.2

The priority of the null process is set to 0 in the create.c and for all other processes the priority is set to 10. We don't want to create a bias as to whether it's a cpubound process or an iobound process when the process first gets created so priority 10 is a good place to start. After this if the process behaves more like iobound/cpubound process it will move up or down the dispatch table accordingly. But to set a bias in the create method without knowing anything about the process's real behaviour would be incorrect. The null process would be set with priority 0 and according to the code, in the resched procedure, if the currpid is 0, i.e null process is the current process while resched was called then the priority promotion or demotion(by looking up the tsdtab is not going to be allowed. The null process will always have a priority of 0 and hence, only if no other eligible process is prepared to run will the null process get the CPU.

Question 3.4

1 All cpu bound processes

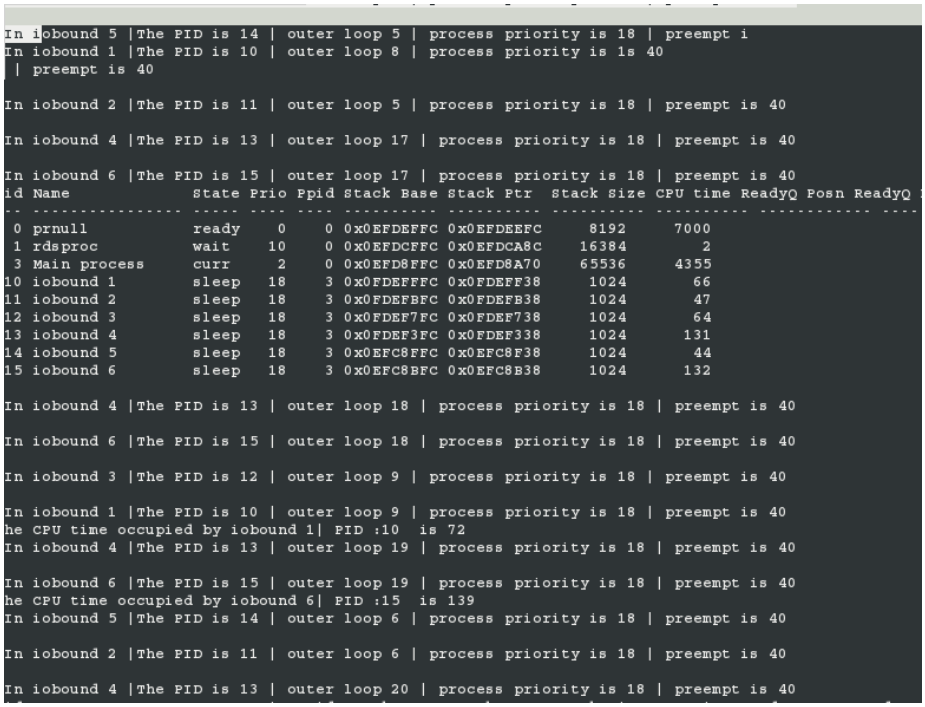
Having all CPU bound processes, their priorities will reduce as they use they consume the quantum on each run. Eventually reaching lowest priority level at which point they will round robin. The total time allocated to all the CPU bound processes is nearly the same, it is more fair than the fair scheduler. This was done by letting main process first resume all the processes and then

scheduling them so they all entered the round robin queue at the same time.

```
------------------------
In lab3q3t1 test | Note: 6 cpubound processes spawned by main process
------------------------
Pid Name              State Prio Ppid Stack Base Stack Ptr  Stack Size CPU time ReadyQ Posn ReadyQ Key
--- ---------------- ----- ---- ---- ---------- ---------- ---------- ---------- ------------ ------------
  0 prnull           ready    0    0 0x0EFDEFFC 0x0EFDEF30      8192          1
  1 rdsproc          wait    10    0 0x0EFDCFFC 0x0EFDCA8C     16384          2
  3 Main process     curr    10    0 0x0EFD8FFC 0x0EFD8ED4     65536         14
  4 cpubound 1       ready    4    3 0x0FDEFFFC 0x0FDEFD64      1024        201
  5 cpubound 2       ready    4    3 0x0FDEFBFC 0x0FDEF964      1024        201
  6 cpubound 3       ready    4    3 0x0FDEF7FC 0x0FDEF564      1024        201
  7 cpubound 4       ready    4    3 0x0FDEF3FC 0x0FDEF164      1024        201
  8 cpubound 5       ready    4    3 0x0EFC8FFC 0x0EFC8D64      1024        201
  9 cpubound 6       ready    4    3 0x0EFC8BFC 0x0EFC8964      1024        201
Pid Name              State Prio Ppid Stack Base Stack Ptr  Stack Size CPU time ReadyQ Posn ReadyQ Key
--- ---------------- ----- ---- ---- ---------- ---------- ---------- ---------- ------------ ------------
  0 prnull           ready    0    0 0x0EFDEFFC 0x0EFDEF30      8192          1
  1 rdsproc          wait    10    0 0x0EFDCFFC 0x0EFDCA8C     16384          2
  3 Main process     curr     4    0 0x0EFD8FFC 0x0EFD8AA0     65536        214
  4 cpubound 1       ready    2    3 0x0FDEFFFC 0x0FDEFD60      1024        361
  5 cpubound 2       ready    2    3 0x0FDEFBFC 0x0FDEF964      1024        361
  6 cpubound 3       ready    2    3 0x0FDEF7FC 0x0FDEF564      1024        361
  7 cpubound 4       ready    2    3 0x0FDEF3FC 0x0FDEF164      1024        361
  8 cpubound 5       ready    2    3 0x0EFC8FFC 0x0EFC8D64      1024        361
  9 cpubound 6       ready    2    3 0x0EFC8BFC 0x0EFC8964      1024        361
Pid Name              State Prio Ppid Stack Base Stack Ptr  Stack Size CPU time ReadyQ Posn ReadyQ Key
--- ---------------- ----- ---- ---- ---------- ---------- ---------- ---------- ------------ ------------
  0 prnull           ready    0    0 0x0EFDEFFC 0x0EFDEF30      8192          1
  1 rdsproc          wait    10    0 0x0EFDCFFC 0x0EFDCA8C     16384          2
  3 Main process     curr     4    0 0x0EFD8FFC 0x0EFD8AA0     65536        214
  4 cpubound 1       ready    2    3 0x0FDEFFFC 0x0FDEFD60      1024        361
  5 cpubound 2       ready    2    3 0x0FDEFBFC 0x0FDEF964      1024        361
  6 cpubound 3       ready    2    3 0x0FDEF7FC 0x0FDEF564      1024        361
  7 cpubound 4       ready    2    3 0x0FDEF3FC 0x0FDEF164      1024        361
  8 cpubound 5       ready    2    3 0x0EFC8FFC 0x0EFC8D64      1024        361
  9 cpubound 6       ready    1    3 0x0EFC8BFC 0x0EFC8960      1024        561
Pid Name              State Prio Ppid Stack Base Stack Ptr  Stack Size CPU time ReadyQ Posn ReadyQ Key
--- ---------------- ----- ---- ---- ---------- ---------- ---------- ---------- ------------ ------------
  0 prnull           ready    0    0 0x0EFDEFFC 0x0EFDEF30      8192          1
  1 rdsproc          wait    10    0 0x0EFDCFFC 0x0EFDCA8C     16384          2
  3 Main process     curr     2    0 0x0EFD8FFC 0x0EFD8AA0     65536        374
  4 cpubound 1       ready    1    3 0x0FDEFFFC 0x0FDEFD60      1024        561
  5 cpubound 2       ready    1    3 0x0FDEFBFC 0x0FDEF964      1024        561
  6 cpubound 3       ready    1    3 0x0FDEF7FC 0x0FDEF564      1024        561
  7 cpubound 4       ready    1    3 0x0FDEF3FC 0x0FDEF164      1024        561
  8 cpubound 5       ready    1    3 0x0EFC8FFC 0x0EFC8D64      1024        561
  9 cpubound 6       ready    1    3 0x0EFC8BFC 0x0EFC8960      1024        561
```

2 All io bound processes

Having all IO bound processes, their priorities will increase each time they don't fully utilize their time quantum and so on relinquishing it's doing these processes good. As can be seen the sharing is not as good as the fair scheduler maybe because the main process isn't getting as much priority as the io processes when resuming the io processes which causes some io processes to start later than the others so their cpu time is lesser. However, the upshot is that once the processes are identified as io bound they all have a preempt of 40ms which means that they are now executing in a round robin fashion.



```
In iobound 5 |The PID is 14 | outer loop 5 | process priority is 18 | preempt i
In iobound 1 |The PID is 10 | outer loop 8 | process priority is 1s 40
| | preempt is 40

In iobound 2 |The PID is 11 | outer loop 5 | process priority is 18 | preempt is 40

In iobound 4 |The PID is 13 | outer loop 17 | process priority is 18 | preempt is 40

In iobound 6 |The PID is 15 | outer loop 17 | process priority is 18 | preempt is 40
id Name             State Prio Ppid Stack Base Stack Ptr  Stack Size CPU time ReadyQ Posn ReadyQ
-- ---------------- ----- ---- ---- ---------- ---------- ---------- ---------- ------------ ----
 0 prnull           ready    0    0 0x0EFDEFFC 0x0EFDEEFC     8192     7000
 1 rdsproc          wait    10    0 0x0EFDCFFC 0x0EFDCA8C    16384        2
 3 Main process     curr     2    0 0x0EFD8FFC 0x0EFD8A70    65536     4355
10 iobound 1        sleep   18    3 0x0FDEFFFC 0x0FDEFF38     1024       66
11 iobound 2        sleep   18    3 0x0FDEFBFC 0x0FDEFB38     1024       47
12 iobound 3        sleep   18    3 0x0FDEF7FC 0x0FDEF738     1024       64
13 iobound 4        sleep   18    3 0x0FDEF3FC 0x0FDEF338     1024      131
14 iobound 5        sleep   18    3 0x0EFC8FFC 0x0EFC8F38     1024       44
15 iobound 6        sleep   18    3 0x0EFC8BFC 0x0EFC8B38     1024      132

In iobound 4 |The PID is 13 | outer loop 18 | process priority is 18 | preempt is 40

In iobound 6 |The PID is 15 | outer loop 18 | process priority is 18 | preempt is 40

In iobound 3 |The PID is 12 | outer loop 9 | process priority is 18 | preempt is 40

In iobound 1 |The PID is 10 | outer loop 9 | process priority is 18 | preempt is 40
he CPU time occupied by iobound 1| PID :10  is 72
In iobound 4 |The PID is 13 | outer loop 19 | process priority is 18 | preempt is 40

In iobound 6 |The PID is 15 | outer loop 19 | process priority is 18 | preempt is 40
he CPU time occupied by iobound 6| PID :15  is 139
In iobound 5 |The PID is 14 | outer loop 6 | process priority is 18 | preempt is 40

In iobound 2 |The PID is 11 | outer loop 6 | process priority is 18 | preempt is 40

In iobound 4 |The PID is 13 | outer loop 20 | process priority is 18 | preempt is 40
```

3 Half-half

With half-half its possible that the cpu bound processes go first, but then their priority drops so once they are out of the way the io bound processes priority rises. So now the io bound processes execute until they are done, which makes sense because these processes have higher priority and we want them to execute, for a short quantum but as soon as they are ready, to improve overall responsiveness for such processes. The multifeed back queue is more fair than the fair scheduler from lab2 and also improves overall responsiveness of the system.

Question 4

Refer to lab3q4.c

1 Overall

I added a new queue called myqueue.c which is the replica of the queue data structure provided. I realized that no process can be in two queues at the same time in the original queue structure so I had to make a new queue to hold the per-receiving process blocked sender sleeperq, which holds the processes that are in PR-SEND state and are currently sleeping waiting for the opportunity by the receiving process to unblock them and execute them within some time

constraints, which if not respected would cause that sending process to drop the message and off the sleeperq aforementioned.

2 Description of the tests

2.1 ASYNCH-TEST1

tests the receive of one word by the sending process. Basic test really.

2.2 ASYNCH-TEST2

tests the receive of multiple senders. Here the receive process is the first to be resumed so there's no chance that the messages can get dropped unless of course the message copy of currently receive execution takes longer than one of the process's timeout. But in our setting one-word message copy cannot take that long.

2.3 ASYNCH-TEST4

tests the dropping of messages. Here testSend 2 and testSend 3 processes have timeouts of 500 and 400 respectively. So by invoking 100 ms sleep and then 400 ms ensures that both processes have timed out and must exit the receiving processes's sleeperq while at the same time also exit with a failure message signifying the dropping of the "are" and "you?" messages.

2.4 ASYNCH-TEST3

tests the FIFO ordering of the senders in the receiver's senderq First resume all the senders and then when they are all ready and have entered the PR-SEND state and have entered the sleeperq then check call start the receiver to check the results are dequeued in a FIFO manner and the message is received correctly. Set large enough timer for the sending processes to ensure the messages don't get dropped.

```
 -- LAB 3.4 ---

-----------ASYNCH SEND TEST 1-----------

RECEIVED: Works!
Send succeeded for PID: 5 with MSG: Works! STARTTIME: 47 ENDTIME: 49 DIFF: 2

-----------ASYNCH SEND TEST 2-----------

RECEIVED: how
Send succeeded for PID: 7 with MSG: how STARTTIME: 3059 ENDTIME: 3061 DIFF: 2

RECEIVED: are
Send succeeded for PID: 8 with MSG: are STARTTIME: 3311 ENDTIME: 3312 DIFF: 1

RECEIVED: you?
Send succeeded for PID: 9 with MSG: you? STARTTIME: 3561 ENDTIME: 3562 DIFF: 1

-----------ASYNCH SEND TEST 3-----------
Send succeeded for PID: 11 with MSG: how STARTTIME: 6814 ENDTIME: 6814 DIFF: 0
 Send request timed out must drop message for 12
Message failed TIMEOUT for PID: 12 with MSG are STARTTIME: 7064 ENDTIME: 7565 DIFF: 501
 Send request timed out must drop message for 13
Message failed TIMEOUT for PID: 13 with MSG you? STARTTIME: 7164 ENDTIME: 7574 DIFF: 410

RECEIVED: how

-----------ASYNCH SEND TEST 4-----------
Send succeeded for PID: 15 with MSG: how STARTTIME: 10567 ENDTIME: 10567 DIFF: 0

RECEIVED: how
 Request buffer cleared up for 16 !

RECEIVED: are
 Request buffer cleared up for 17 !
Send succeeded for PID: 17 with MSG: you? STARTTIME: 10575 ENDTIME: 12078 DIFF: 1503
Send succeeded for PID: 16 with MSG: are STARTTIME: 10575 ENDTIME: 12085 DIFF: 1510

RECEIVED: you?
```

3 Inference from tests

3.1 ASYNCH-TEST1

The message is received in time!

3.2 ASYNCH-TEST2

The messages are received by the receiver process. Signified by the RECEIVED tag in the image above.

3.3 ASYNCH-TEST4

As can be seen the messages "are" and "you?" are dropped. Only the "hello" message goes through.

3.4 ASYNCH-TEST3

The messages are read in the order that they went to the send function. Hence, we are ensured that the messages are read in FIFO order by the receiving process. Signified by the ordering of the correct RECEIVED tags and the ordering of the STARTTIME associated with that process are indeed linearizable.