Number of CPU = 1

Default scheduler output:

CPU BOUND PROCESS

Child 0 priority = 10
child 0 start time = 566
Child 1 priority = 30
child 1 start time = 569
Child 2 priority = 50
child 2 start time = 570
Child 3 priority = 70
child 3 start time = 571
Child 4 priority = 90
child 4 start time = 576
child 0 end time = 3707
child 0 total time = 3141
child 2 end time = 3712
child 2 total time = 3142
child 3 end time = 3716
child 3 total time = 3145
child 1 end time = 3720
child 1 total time = 3151
child 4 end time = 3720
child 4 total time = 3144

I/O BOUND PROCESS

child 5 priority = 10
child 5 start time = 6454
child 6 priority = 30
child 6 start time = 6455
child 7 priority = 50
child 7 start time = 6457
child 8 priority = 70
child 8 start time = 6461
child 9 priority = 90
child 9 start time = 6475
child 7 end time = 8285
child 7 total time = 1828
child 8 end time = 8213
child 8 total time = 1752
child 9 end time = 8164
child 9 total time = 1689
child 6 end time = 8305
child 6 total time = 1850
child 5 end time = 8318
child 5 total time = 1864

Modified scheduler output:

CPU BOUND PROCESS

Child 0 priority = 10
child 0 start time = 429
Child 1 priority = 30
child 1 start time = 479
Child 2 priority = 50
child 2 start time = 529
Child 3 priority = 70
child 3 start time = 579
Child 4 priority = 90
child 4 start time = 629
child 4 end time = 2230
child 4 total time = 1601
child 3 end time = 2576
child 3 total time = 1997
child 2 end time = 2882
child 2 total time = 2353
child 1 end time = 3211
child 1 total time = 2732
child 0 end time = 3599
child 0 total time = 3170

I/O BOUND PROCESS

child 5 priority = 10
child 5 start time = 4300
child 6 priority = 30
child 6 start time = 4350
child 7 priority = 50
child 7 start time = 4350
child 9 priority = 90
child 9 start time = 4360
child 9 end time = 4513
child 9 total time = 153
child 8 priority = 70
child 8 start time = 4583
child 8 end time = 4799
child 8 total time = 216
child 7 end time = 5006
child 7 total time = 656
child 6 end time = 5216
child 6 total time = 866
child 5 end time = 5371
child 5 total time = 1071

Difference (all fork are doing same work):

CPU BOUND PROCESS(child0-4)

	priority	old scheduler time	new scheduler time
child0	10	3141	3170
child1	30	3151	2732
child2	50	3142	2353
child3	70	3145	1997
child4	90	3144	1601

I/o BOUND PROCESS(child5-9)

child5	10	1864	1071
child6	30	1850	866
child7	50	1828	656
child8	70	1752	216
child9	90	1689	153

Number of CPU = 2

Old scheduler output:

CPU BOUND PROCESS

Child 0 priority = 10
child 0 start time = 370
Child 1 priority = 30
child 1 start time = 371
Child 2 priority = 50
child 2 start time = 375
Child 3 priority = 70
child 3 start time = 377
Child 4 priority = 90
child 4 start time = 378
child 1 end time = 4274
child 1 total time = 3903
child 3 end time = 4288
child 3 total time = 3911
child 2 end time = 4299
child 2 total time = 3924
child 0 end time = 4301
child 0 total time = 3931
child 4 end time = 4345
child 4 total time = 3967

I/O BOUND PROCESS

child 5 priority = 10
child 5 start time = 4099
child 6 priority = 30

child 6 start time = 4100
child 7 priority = 50
child 7 start time = 4101
child 8 priority = 7child 9 priority = 90
child 9 start time = 4140
0
child 8 start time = 4214
child 8 end time = 7859
child 8 total time = 3645
child 5 end time = 8129
child 5 total time = 4030
child 6 end time = 8664
child 6 total time = 4564
child 7 end time = 8959
child 7 total time = 4858
child 9 end time = 9179
child 9 total time = 5039

Modified Scheduler Output:

CPU BOUND PROCESS

Child 0 priority = 10
child 0 start time = 606
Child 1 priority = 30
child 1 start time = 613
Child 2 priority = 50
child 2 start time = 655
Child 3 priority = 70
child 3 start time = 663
Child 4 priority = 90
child 4 start time = 705
child 4 end time = 3246
child 4 total time = 2541
child 3 end time = 3676
child 3 total time = 3013
child 2 end time = 4807
child 2 total time = 4152
child 1 end time = 5418
child 1 total time = 4805
child 0 end time = 7413
child 0 total time = 6807

I/O BOUND PROCESS

child 5 priority = 10
child 5 start time = 7420
child 6 priority = 30
child 6 start time = 7425
child 7 priority = 50

child 7 start time = 7428
child 9 priority = 90
child 9 start time = 7465
child 8 priority = 70
child 8 start time = 7595
child 9 end time = 10674
child 9 total time = 3209
child 8 end time = 11125
child 8 total time = 3530
child 6 end time = 13655
child 6 total time = 6230
child 7 end time = 12835
child 7 total time = 5407
child 5 end time = 14545
child 5 total time = 7125

Difference (all fork are doing same work):

CPU BOUND PROCESS(child0-4)

	priority	old scheduler time	new scheduler time
child0	10	3931	6807
child1	30	3903	4805
child2	50	3924	4152
child3	70	3911	3676
child4	90	3967	2541

I/o BOUND PROCESS(child5-9)

child5	10	4030	7125
child6	30	4564	6230
child7	50	4858	5407
child8	70	3645	3530
child9	90	5039	3209

Time complexity of new scheduler for finding a process to run = O(n) in the number of processes

Priority can be processed can be set from 1 to 100.

getprio(): return the priority of process

setprio(n): set the priority of the process = n, and return n if the priority is set correctly,
return -1, if n<=0, and set priority to default value
return -2, if n>100, and set priority to default value

default priority = 50, if the priority is not set explicitily in the process

Time taken by same process decreases when priority is increased, using new scheduler.

Scheduling Algorithm:

3 new variables in proc:
int priority
int carry
int has_run

initially, at start all process will have some priority, but carry = 0, and
has_run = 0, for all the process

scheduler will then find RUNNABLE process with maximum value of (priority+carry) and has_run = 0,
and schedule that process
the process is then supposed to run for carry=(priority+carry),
yield function reduces carry by 1 every time, and return to same function is carry>0, else do the normal stuff.
When carry = 0, make has_run of that process = 1, thus process will not picked next time.

Thus carry will take care of priority if a process goes to sleep without completing its run, in the next run.

When no process is found to run by scheduler(might be all process has run once and all RUNNABLE process
will have has_run = 1), so make has_run = 0, for all process, and thus in new loop, again highest (priorty+carry)
process will picked up to run.

When a process get blocked, and has not completed total time it was assigned to, its carry will be > 0(time
remaining to run), and will be taken care in next run by choosing process with highest priority+carry.

highest value of priority+carry (total run time in one loop) can be 2*MAX_PRIORITY(100), so that if process waits
for too long, and multiple rounds have been completed by scheduler, in the mean time, it should not take much of CPU
after waking up.