OS 2020 project

- Project 1 -

B06902019 資工三 洪佳生

Added syscall 334: sys\_my\_printk()

• Use scheduler() funtion to implement sheduling.

• TIME UNIT(): define a basic unit of execution time.

sched\_setscheduler() to raise the priority "pid" of process.

process priority is raised by parent process.

child processes which are are generated by fork().

Prints a string to dmesg

 Kernel Version:

Linux 4.14.25 -> https://cdn.kernel.org/pub/linux/kernel/v4.x/linux-4.14.25.tar.xz Added syscall 333: sys\_my\_clock()

Get the data by getnstimeofday()

• assign\_proc\_core(pid, core): decide which core the process "pid" will run on.

• proc out(pid): use sched setscheduler() to reduce the priority "pid" of process and then

proc\_wakeup(pid): assign the child process to the core where the core 1 and then use

(1) Scheduling process call fork() to simulate the process which is ready and stops child

process by reduccing its priority until parent process wakes it up. Because parent and child

timer, we set a while() loop to avoid the above problem and it will break the loop when child

process are in the same core. So due to the priority, child process won't run if it shouldn't

run. But in case the child process run in the unavailable time and unfortunately start the

(2) When the timer starts, child process will enter a while() loop for execution time of

• First, we will assign a particular core 0 to scheduling process and raise its priority to the

Second, initializes child process by -1 to represent not ready process or already finished

• In while(1) loop, we will kepp doing tje following five steps until all processes are done.

Check whether there are some process are already done in last UNIT\_TIME.If so, label

Use switch to choose the scheduling policy to find the next process to implement it.

When the implementing process i finish, the next implement process will be i+1.

When the implementing process i finish, the next implement process will be the shortest

next implementing process will be select by the index from i+1 (if process (i+1)%n isn't

NO matter the implementing process i is finished, the next implement process will be the

If the next running process isn't the same as now running process, then scheduling process

will reduce the priority of now runnig process and raise the priority of next running process.

run a TIME\_UNIT in parent and now runnig child process simutaneously.

My start time, My end time: the time I get from my program (Initialize by minimun My start

my end

489.92

989.51

1471.77

1944.95

2440.11

my end

81020.72

86600.20

87680.65

88737.23

my end

8272.39

13369.11

16751.26

17781.91

18812.31

19819.16

23677.05

my end

1908.33

2393.82

2580.48

3071.49

my end

7615.50

12521.26

15365.40

16343.53

17308.94

18302.54

22178.37

my end

5772.22

9565.21

15534.46

24435.57

my end

2047.76

4247.36

7204.36

8194.32

11281.43

my end

1058.74

1572.18

2092.58

3587.39

my end

1097.84

3008.82

7102.74

14252.52

my end

time

97.86

288.63

4451.90

8822.97

15905.36

my end

476.97

963.75

1448.80

1931.59

2413.40

my end

7469.94

8740.55

my end

15990.97

18079.26

18412.00

25624.15

27845.31

28304.54

my end

5132.77

5690.23

6146.94

13741.42

17203.33

19318.20

21573.35

my end

5294.98

5743.69

6250.48

13983.14

17334.03

19231.51

22300.76

my end

1924.81

2868.64

6726.39

13442.00

my end

time

93.84

280.43

4076.89

7888.90

14839.23

my end

2922.60

2958.10

2968.18

6903.22

10888.61

15678.89

22392.17

31038.16

my end

2956.21

3949.70

7682.86

8820.35

10685.87

my end

1951.38

2404.55

2924.41

3382.44

my end

time

502.11

1547.33

2584.39

3609.67

4584.37

5581.82

6560.32

7590.78

8607.56

9607.52

time

my exec

time

489.92

487.81

482.11

471.06

484.95

my exec

81020.72

5120.19

1064.84

1050.39

my

exec

time

8272.39

5038.09

3135.01

1002.22

959.52

962.74

3831.10

my exec

1908.33

485.37

186.54

490.91

my

exec

time

7615.50

4836.57

2844.01

969.28

965.29

972.83

3875.72

my exec

2819.26

7580.13

14535.35

24435.57

my exec

1022.42

4247.36

1903.93

948.04

7013.68

my exec

time

531.55

513.25

500.77

3587.39

my exec

time

992.01

3008.82

4093.80

7149.64

my exec

time

97.86

190.51

4163.16

4370.95

7082.27

my exec

time

476.97

486.67

484.87

480.14

479.85

my exec

7469.94

8196.80

my exec

13126.06

18079.26

16989.68

19965.16

22642.87

23560.37

my exec

3718.71

3808.72

3793.92

12804.63

14382.89

18849.67

21573.35

my exec

3842.81

3813.80

3829.52

13016.09

14437.92

18747.42

22300.76

my exec

1924.81

943.72

3844.50

6715.47

my exec

time

93.84

186.37

3794.17

3766.83

6937.66

my

exec

time

9.59

9.58

2922.60

3934.96

3965.34

4790.12

6688.25

8568.68

my exec

2956.21

993.37

3733.03

980.19

1865.41

my exec

1951.38

453.04

489.21

457.93

my exec

time

502.11

520.39

511.50

512.64

478.35

487.52

483.58

517.14

491.50

495.27

time

error

rate

2.02%

2.44%

3.58%

5.79%

3.01%

error

rate

1.28%

2.40%

6.48%

5.04%

error

rate

3.40%

0.76%

4.50%

0.22%

4.05%

3.73%

4.22%

error

rate

4.58%

2.93%

6.73%

1.82%

error

rate

4.81%

3.27%

5.20%

3.07%

3.47%

2.72%

3.11%

error

rate

6.02%

5.25%

3.10%

2.26%

error

rate

2.24%

6.18%

4.80%

5.20%

0.20%

error

rate

6.31%

2.65%

0.15%

2.50%

error

rate

0.80%

0.29%

2.35%

2.14%

error

rate

2.14%

4.74%

4.08%

9.27%

1.18%

error

rate

4.61%

2.67%

3.03%

3.97%

4.03%

error

rate

0.40%

3.57%

error

rate

6.24%

4.85%

5.61%

4.93%

3.65%

5.76%

error

rate

7.03%

4.78%

5.15%

5.15%

4.11%

3.34%

6.20%

error

rate

3.93%

4.66%

4.26%

3.58%

3.75%

3.86%

3.04%

error

rate

3.76%

5.63%

3.89%

4.06%

error

rate

6.16%

6.81%

5.15%

5.83%

0.89%

error

rate

2.58%

4.11%

4.24%

1.63%

0.87%

4.20%

4.45%

4.79%

error

rate

1.46%

0.66%

6.67%

1.98%

6.73%

error

rate

2.43%

9.39%

2.16%

8.41%

error

rate

0.42%

4.08%

2.30%

2.53%

4.33%

2.50%

3.28%

3.43%

1.70%

0.95%

my start

time

0.00

501.70

989.66

1473.90

1955.16

my start

81480.00

86615.81

87686.84

my start

time

0.00

8331.03

13616.25

16779.69

17852.79

18856.43

19845.95

my start

time

0.00

1908.45

2393.94

2580.58

my start

time

0.00

7684.69

12521.39

15374.26

16343.65

17329.71

18302.65

my start

2952.96

1985.08

999.11

0.00

my start

1025.34

5300.43

7246.28

4267.75

my start

time

527.20

1058.93

1591.81

my start

time

0.00

105.83

3008.94

7102.88

my start

time

0.00

98.11

288.74

4452.02

8823.09

my start

time

0.00

477.08

963.93

1451.45

1933.55

my start

time

0.00

543.74

my start

2864.92

1422.31

5658.99

5202.43

4744.18

my start

1414.06

1881.51

2353.02

936.79

2820.43

468.53

my start

1452.16

1929.89

2420.96

967.05

2896.11

484.09

my start

time

0.00

1924.92

2881.88

6726.53

my start

time

0.00

94.06

282.73

4122.07

7901.57

my start

time

0.00

2948.51

2958.60

2968.26

6923.27

10888.77

15703.92

22469.48

my start

2956.33

3949.83

7840.16

8820.46

my start

time

0.00

1951.52

2435.20

2924.52

my start

time

0.00

1026.94

2072.89

3097.03

4106.02

5094.30

6076.74

7073.64

8116.06

9112.25

time

0.00

0.00

time

0.00

time

time

0.00

0.00

time

0.00

time

time

0.00

ready, then check process (i+2)%n) until back to i.If back to i,then choose i as next

RR():In each time slice 500 UNIT\_TIME or the implementing process finish in time slice, the

equal to total process number, break the while (1) loop and finish scheduling.

process's pid into -1 and finished processes number += 1.If finished processes number is

Check whether there are some processes which are ready and if so, implement proc\_exec().

highest level to prevent potential preemptive problem between scheduling process and the

(3) When the timer ends, use system call to output the message into dmesg.

assign the child process back to the core 0 where the parent process is running.

• Read the input (number of process, scheduling policy, process name, ready time, execution

Design

main.c

time)

process.c

Define the followin function:

proc\_exec(Process):

TIME\_UNIT().

scheduler.c

process.

Step1:

Step2:

Step3:

FIFO():

SJF():

PSJF():

Step4:

Step5:

Result

time)

**Unit Time** 

FIFO\_1.txt

process

P1

P2

P3

P4

P5

P1

P2

P3

P4

FIFO\_3.txt

process

P1

P2

P3

P4

**P5** 

P6

P7

P1

P2

P3

P4

FIFO\_5.txt

process

P1

P2

P3

P4

P5

P6

P7

P4

Р3

P2

P1

P2

P1

P4

P5

P3

P2

P3

P4

P1

PSJF\_4.txt

process

P3

P2

P4

P1

P1

P3

P2

P4

P5

RR\_1.txt

P1

P2

Р3

P4

P5

RR\_2.txt

P1

P2

RR\_3.txt

P3

P1

P2

P6

**P5** 

P4

RR\_4.txt

P4

P5

P6

Р3

P7

P2

P1

RR\_5.txt

P4

P6

P3

P7

P2

P1

SJF\_1.txt

P2

P3

P4

P1

P1

P3

P2

P4

P5

SJF\_3.txt

process

P1

P4

P5

P6

P7

P2

P3

P8

SJF\_4.txt

P1

P2

P3

P5

P4

SJF\_5.txt

P1

P2

P3

P4

P0

P1

P2

P3

P4

P5

P6

P7

P8

P9

Conclusion

Correctness

Work Loading

Synchronization

acheivable.

process

process

process

SJF\_2.txt

process

process

process

process

process

process

process

PSJF\_5.txt

process

PSJF\_3.txt

process

PSJF\_2.txt

process

PSJF\_1.txt

process

FIFO\_4.txt

process

FIFO\_2.txt

process

**Unit Time** 

There are four policies:

implementing process.

Assume there are some ready processes.

execution time process in the ready queue.

Error Rate: round off to the 4nd decimal place

Start time, End time: theoretical time (calculate by math)

**Expect exec time**: the time the process should finish

My exec time: the time the process finish in my program

end

time

500

1000

1500

2000

2500

end

time

80000

85000

86000

87000

end

time

8000

13000

16000

17000

18000

19000

23000

end

time

2000

2500

2700

3200

end

time

8000

13000

16000

17000

18000

19000

23000

end

time

6000

10000

16000

25000

end

time

2000

4000

7000

8000

11000

end

time

1000

1500

2000

3500

end

time

1100

3000

7000

14000

end

time

200

400

4400

8400

15400

end

time

500

1000

1500

2000

2500

end

time

8100

9600

end

time

18200

19700

20200

28200

30200

31200

end

time

5500

6000

6500

14500

18500

20000

23000

end

time

5500

6000

7000

14500

18500

20000

23000

end

time

2000

3000

7000

14000

end

time

200

400

4400

8400

15400

end

time

3100

3110

3120

7120

11120

16120

23120

32120

end

time

3000

4000

8000

9000

11000

end

time

2000

2500

3000

3500

end

time

500

1500

2500

3500

4500

5500

6500

7500

8500

9500

process start earlier/later, it will end earlier/later.

would be affected depending on the utilization of CPU.

0.001377

start

time

500

1000

1500

2000

start

time

80000

85000

86000

start

time

8000

13000

16000

17000

18000

19000

start

time

2000

2500

2700

start

time

8000

13000

16000

17000

18000

19000

start

time

3000

2000

1000

start

time

1000

5000

7000

4000

start

time

500

1000

1500

start

time

100

3000

7000

start

time

100

200

400

4400

8400

start

time

500

1000

1500

2000

start

time

600

1100

start

time

4200

1200

2700

8200

6700

6200

start

time

1500

2000

2500

1000

3500

500

start

time

1500

2000

3000

1000

3500

500

start

time

2000

3000

7000

start

time

100

200

400

4400

8400

start

time

100

3100

3110

3120

7120

11120

16120

23120

start

time

3000

4000

8000

9000

start

time

2000

2500

3000

start

time

1000

2000

3000

4000

5000

6000

7000

8000

9000

0

TIME\_MEASUREMENT.txt

0

0

0

0

0

0

0

0

0

0

0

0

0

0

0

Error rate: the error between Expect exec time & My exec time

expect

500

500

500

500

500

expect

80000

5000

1000

1000

expect

8000

5000

3000

1000

1000

1000

4000

expect

2000

500

200

500

expect

8000

5000

3000

1000

1000

1000

4000

expect

3000

8000

15000

25000

expect

1000

4000

2000

1000

7000

expect

500

500

500

3500

expect

1000

3000

4000

7000

expect

100

200

4000

4000

7000

expect

500

500

500

500

500

expect

7500

8500

expect

14000

18500

17500

20000

23500

25000

expect

4000

4000

4000

13500

15000

19500

23000

expect

4000

4000

4000

13500

15000

19500

23000

expect

2000

1000

4000

7000

expect

100

200

4000

4000

7000

expect

3000

10

10

4000

4000

5000

7000

9000

expect

3000

1000

4000

1000

2000

expect

2000

500

500

500

expect

500

500

500

500

500

500

500

500

500

500

In my theoretical time assumption, there is no content switch and some hardware

propagation delay, so each unit time is basicly the same. However, there might be some time

error. But the error rate between theoretical execution time and real execution time aren still

not smaller than 10%. The execution time is smaller than thereotical time because child

further from end time because previous process time error will affect the next one

When CPU was busy or has other jobs, such like I/O input and output, the performance

Because we use two core to simulate parent and child process, maybe child will run on the

unavailable time. So waitpid() is a key funtion in scheduler.c to make synchronization

process, but the my exec time will not be affected by the previous one because if the

process maybe run on the unavailable time before timer start. My end time often be

exec time

shortest execution time process in the ready queue.

<b>⊕</b>