Lab 3

Members: Fangwen Liao (3439869) | Yijin Wang (3476217)

Exercise 1

Inspect the nice command (man nice)

(a)use the nicecommand to create a process that executes ping localhost with a nice value of 19.

According to man page of *nice*, it is used to modify scheduling priority. Run *nice* to assign the value 19 to *ping*.

```
fangwenliao@debian:~$ nice -19 ping localhost
PING localhost(localhost (::1)) 56 data bytes
64 bytes from localhost (::1): icmp_seq=1 ttl=64 time=0.053 ms
64 bytes from localhost (::1): icmp_seq=2 ttl=64 time=0.055 ms
64 bytes from localhost (::1): icmp_seq=3 ttl=64 time=0.065 ms
64 bytes from localhost (::1): icmp_seq=4 ttl=64 time=0.073 ms
64 bytes from localhost (::1): icmp_seq=5 ttl=64 time=0.095 ms
64 bytes from localhost (::1): icmp_seq=6 ttl=64 time=0.073 ms
64 bytes from localhost (::1): icmp_seq=7 ttl=64 time=0.070 ms
64 bytes from localhost (::1): icmp_seq=8 ttl=64 time=0.072 ms
64 bytes from localhost (::1): icmp_seq=9 ttl=64 time=0.074 ms
64 bytes from localhost (::1): icmp_seq=10 ttl=64 time=0.070 ms
64 bytes from localhost (::1): icmp_seq=11 ttl=64 time=0.075 ms
64 bytes from localhost (::1): icmp_seq=12 ttl=64 time=0.074 ms
```

(b)show that the nice value has changed by inspecting it using the ps command.

Use *ps* to find the pid of ping, then in the output format use the *-nice* option to show the niceness.

```
fangwenliao@debian:~$ ps -eo pid,nice,cmd | grep ping
 1007  19 ping localhost
```

Exercise 2

Open the loopdirectory and inspect the files. Compile the source file into a binary using the makecommand. The loopbinary takes one integer (nice value) as command line argument. Run the loop process with nice value 0

(a) Show the output of the lscpu command, and the cat /proc/sys/kernel/sched_autogroup_enabled command.

The autogroup feature according to the man page of sched is to prevent some groups of process from hogging too much CPU-cycles. So we firstly set the *sched_autogroup_enabled* value to 0 to disable this feature.

Firstly we need to access in root to set this value. The *lscpu* command shows that there is only one CPU as required. The autogroup feature is disabled.

```
fangwenliao@debian:~/Downloads/OS_Lab3/loop$ su
Password:
root@debian:/home/fangwenliao/Downloads/OS_Lab3/loop# echo 0 >
/proc/sys/kernel/sched_autogroup_enabled
root@debian:/home/fangwenliao/Downloads/OS_Lab3/loop# exit
exit
fangwenliao@debian:~/Downloads/OS_Lab3/loop$ Process ID: 1039
Requested nice: 0
Original nice: 0
New nice: 0

Use CTRL+C (SIGINT) to exit


fangwenliao@debian:~/Downloads/OS_Lab3/loop$ lscpu
Architecture:          i686
CPU op-mode(s):        32-bit
Byte Order:            Little Endian
CPU(s):                1
On-line CPU(s) list:   0
Thread(s) per core:    1
Core(s) per socket:    1
Socket(s):             1
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 142
Model name:            Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz
Stepping:              10
CPU MHz:               1991.999
BogoMIPS:              3983.99
Hypervisor vendor:     KVM
Virtualization type:   full
L1d cache:             32K
L1i cache:             32K
L2 cache:              256K
L3 cache:              8192K
Flags:                 fpu vme de pse tsc msr mce cx8 apic sep mtrr pge mca cmov
pat pse36 clflush mmx fxsr sse sse2 ht rdtscp constant_tsc xtopology nonstop_tsc
pni pclmulqdq monitor ssse3 cx16 sse4_1 sse4_2 x2apic movbe popcnt aes xsave avx
rdrand hypervisor lahf_lm abm 3dnowprefetch fsgsbase avx2 invpcid rdseed
clflushopt md_clear flush_l1d
fangwenliao@debian:~/Downloads/OS_Lab3/loop$ cat
/proc/sys/kernel/sched_autogroup_enabled
0
```

(b)inspect the current scheduling class of the loopprocess using the chrtcommand. What is the current scheduling class?

According to man page of *chrt*, just use *-p* option to retreive real-time attributes of a task. According to the man page of *sched*, the *SCHED_OTHER* means that this process use the default linux time sharing scheduling, and the priority can only be 0, however a dynamic nice value will be generated every time quantum to decide when to run.

```
fangwenliao@debian:~/Downloads/OS_Lab3/loop$ chrt -p 1039
pid 1039's current scheduling policy: SCHED_OTHER
pid 1039's current scheduling priority: 0
```

(c) Using the same command, change the scheduling class to realtime, what happens to the system's interactiveness(of the GUI for example), and why?

Firstly we check the priority range using *chrt --max*. According to man page of *sched*, there are two real-time scheduling policies in linux: FIFO and RR, both will preempt any other non-real-time tasks, but under RR a task runs in a time slice, which is less aggresive than FIFO. A root access is needed to change the real time scheduling policy. Firstly check the value range of priority. After changing it, the terminal react much slower than usual(GUI is not started on our machine).

```
fangwenliao@debian:~/Downloads/OS_Lab3/loop$ chrt --max
SCHED_OTHER min/max priority    : 0/0
SCHED_FIFO min/max priority : 1/99
SCHED_RR min/max priority   : 1/99
SCHED_BATCH min/max priority    : 0/0
SCHED_IDLE min/max priority : 0/0
SCHED_DEADLINE min/max priority : 0/0
fangwenliao@debian:~/Downloads/OS_Lab3/loop$ su
Password:
root@debian:/home/fangwenliao/Downloads/OS_Lab3/loop# chrt -f -p 99 1039
root@debian:/home/fangwenliao/Downloads/OS_Lab3/loop# chrt -r -p 99 1039
```

In order to look deep into what is happening, a new ssh window is opened and check the interrupts of the system. When doing nothing, the only increasing(also regularly) interrupts are caused by enp0s3 which is ethernet perepherial, Local timer interrupts and ata_piix which is Intel PATA/SATA controllers according to the ata_piix.c, and without running loop, they increase just about the same rate.

The only noticable interrupt that cause lagging is caused by i8042, which according to i8042.c is the keyboard and mouse controller for linux. Each time of a type on our keyboard will cause two(press and release) interrupts, and a obvious lagging in terminal. So it is resonable to guess that the I/O process of keyboard input, which is also a real-time process are preempted by loop.

```
fangwenliao@debian:~/Downloads/OS_Lab3/loop$ watch -n.1 "cat /proc/interrupts"
Every 0.1s: cat /proc/interrupts###################debian: Sun Nov 28 22:59:22
2021

###########CPU0
##0:#########40   IO-APIC   2-edge######timer
##1:########640   IO-APIC   1-edge######i8042
##8:##########0   IO-APIC   8-edge######rtc0
```

```
##9:#########50   IO-APIC   9-fasteoi   acpi
 12:########162   IO-APIC  12-edge######i8042
 14:##########0   IO-APIC  14-edge######ata_piix
 15:#######7627   IO-APIC  15-edge######ata_piix
 18:##########0   IO-APIC  18-fasteoi   vmwgfx
 19:######20212   IO-APIC  19-fasteoi   enp0s3
 20:#######3081   IO-APIC  20-fasteoi   vboxguest
 21:#######9384   IO-APIC  21-fasteoi   ahci[0000:00:0d.0], snd_intel8x0
 22:#########26   IO-APIC  22-fasteoi   ohci_hcd:usb1
NMI:##########0   Non-maskable interrupts
LOC:    2280934   Local timer interrupts
SPU:##########0   Spurious interrupts
PMI:##########0   Performance monitoring interrupts
IWI:##########0   IRQ work interrupts
RTR:##########0   APIC ICR read retries
RES:##########0   Rescheduling interrupts
CAL:##########0   Function call interrupts
TLB:##########0   TLB shootdowns
TRM:##########0   Thermal event interrupts
```

Exercise 3

(a) Start the first loopprocess with nice value 0, use the ampersand (&) to execute it in the background

(b) Now start another loopprocess in the same terminal with nice value 0 and use the timecommand to measure real/user/system time of this new process, after +/-30 seconds terminate the process with CTRL+C. Show and explain the difference between real/user time, explain the ratio between these numbers

According to the *const int sched_prio_to_weight[40]* in kernel/sched/core.c, the ratio can be calculated in a very simple fomular:

> r=weight2/(weight1+weight2).

The theoretical ratio should be *0.5*, and in our test it is *0.4993*.

```
fangwenliao@debian:~/Downloads/OS_Lab3/loop$ Process ID: 1045
Requested nice: 0
Original nice: 0
New nice: 0

Use CTRL+C (SIGINT) to exit


fangwenliao@debian:~/Downloads/OS_Lab3/loop$ time ./loop 0
Process ID: 1047
Requested nice: 0
Original nice: 0
New nice: 0

Use CTRL+C (SIGINT) to exit
```

```
^C

real    0m29.528s
user    0m14.744s
sys 0m0.000s
```

(c) Repeat step (a) four times, using the following nice values for the second process 19, 10, -10, and -20. Show and explain the difference between real/user time, explain also the ratio between these numbers using the CFS Example slide (slide 14)

```
fangwenliao@debian:~/Downloads/OS_Lab3/loop$ ./loop 0 &
[1] 1058
fangwenliao@debian:~/Downloads/OS_Lab3/loop$ Process ID: 1058
Requested nice: 0
Original nice: 0
New nice: 0

Use CTRL+C (SIGINT) to exit


fangwenliao@debian:~/Downloads/OS_Lab3/loop$ time ./loop 19
Process ID: 1059
Requested nice: 19
Original nice: 0
New nice: 19

Use CTRL+C (SIGINT) to exit

^C

real    0m30.858s
user    0m0.444s
sys 0m0.000s
fangwenliao@debian:~/Downloads/OS_Lab3/loop$ time ./loop 10
Process ID: 1062
Requested nice: 10
Original nice: 0
New nice: 10

Use CTRL+C (SIGINT) to exit

^C

real    0m30.169s
user    0m2.920s
sys 0m0.000s
```

```
fangwenliao@debian:~/Downloads/OS_Lab3/loop$ su
Password:
root@debian:/home/fangwenliao/Downloads/OS_Lab3/loop# time ./loop -10
Process ID: 1082
Requested nice: -10
Original nice: 0
New nice: -10

Use CTRL+C (SIGINT) to exit

^C

real    0m30.795s
user    0m27.740s
sys 0m0.028s
root@debian:/home/fangwenliao/Downloads/OS_Lab3/loop# time ./loop -20
Process ID: 1086
Requested nice: -20
Original nice: 0
New nice: -20

Use CTRL+C (SIGINT) to exit

^C

real    0m30.197s
user    0m29.792s
sys 0m0.012s
```

Difference between real/user time:

> Real time is the whole elapsed time from start to the end of a call, including time used by other processes and the time spent on being blocked.
>
> User time is only the CPU time spent in user-mode code (outside the kernel) on executing the process. [1]

According to the table on slide 14, the result has been shown below:

Nice Value	Theoretical Ratio	Test Result
19	15/(15+1024)=1.44%	1.44%
10	110/(110+1024)=10.74%	9.68%
-10	9548/(9548+1024)=90.31%	90.08%
-20	88761/(88761+1024)=98.86%	98.66%

Since negative nice value can only be used by root, we must switch to root in some points. In Gnome terminal, however, the first backgrouond loop process must be executed by root user to get the similar result like below, if switched to root only before the second loop process, the second loop will only get half of real time, like using nice 0. Only when using ssh with GUI off, this problem doesn't exisit.

(d) in loop.c, add a usleep(1)statement in the while loop, build the new binary, and use the timecommand to inspect the real/user/system time when running loopwith nice -20, show and explain the differences with/without usleep

Run the code with usleep(1), this command will suspend the process for 1 microsecond according to the manpage of usleep.

```
fangwenliao@debian:~/Downloads/OS_Lab3/loop$ su
Password:
root@debian:/home/fangwenliao/Downloads/OS_Lab3/loop# time ./loop -20
Process ID: 1146
Requested nice: -20
Original nice: 0
New nice: -20

Use CTRL+C (SIGINT) to exit

^C

real    0m30.373s
user    0m6.492s
sys 0m0.000s
```

Compare to the execution of code without *usleep(1)*:

```
root@debian:/home/fangwenliao/Downloads/OS_Lab3/loop# time ./loop -20
Process ID: 1086
Requested nice: -20
Original nice: 0
New nice: -20

Use CTRL+C (SIGINT) to exit

^C

real    0m30.197s
user    0m29.792s
sys 0m0.012s
```

Difference:

> The execution time ratio of loop.c with *usleep(1)* is much lower than the one without *usleep(1)*.

In order to find what happened, some test are done by changing the code(the modified code for tests is only used in 3(d)). Slightly change the code in the while loop: long i = 1000000000; while(i > 0) { i = i - 1; }

```
root@debian:/home/fangwenliao/Downloads/OS_Lab3/loop# time ./loop -20
Process ID: 1210
Requested nice: -20
Original nice: 0
New nice: -20

Use CTRL+C (SIGINT) to exit


real    0m1.560s
user    0m1.556s
sys 0m0.000s
```

it can be seen that a loop without usleep will take about 1.5e-9 second. Another test is taken with usleep: long i = 400000; while(i > 0) { i = i - 1; usleep(1); }

```
root@debian:/home/fangwenliao/Downloads/OS_Lab3/loop# time ./loop -20
Process ID: 7529
Requested nice: -20
Original nice: 0
New nice: -20

Use CTRL+C (SIGINT) to exit


real    0m30.747s
user    0m5.956s
sys 0m0.000s
```

it runs just about 30 seconds and runs only 400000 loop. So the theoretical running time of 400000 loop should according to our previous test 6e-4 seconds, which is beyond the accuracy of time command according to man page time(7), thus should output 0s in user, however it still has about 6 seconds. It can be refered that there are too many overheads, probably the one caused by context switching.

Differences with/without *usleep*:

> Without *usleep*, all the process switches are involuntary switches. With *usleep*, some process switches become voluntary switches.

Exercise 4

To monitor the context switches of a process, use watch –n.1 grep ctxt/proc/pid/statuswhere pidshould be replaced with an actual process ID. Monitor the context switches (for about 30 seconds) of the original loopprocess without the usleep, and the modified loopprocess with the usleep, both with a nice value of 0. Show and explain the number of voluntary and involuntary context switches in both cases

Differences with/without *usleep*(both in 30-seconds intervals):

> Without *usleep*, nonvoluntary context switch is (4456-2717)=1739 times, voluntary context switch is 0,
>
> With *usleep*, nonvoluntarty context switch is (14-13)=1 time, voluntary context switch is (1664328-1264185)=400143 times, which is roughly the same as the result shown in 3(d).

When *uspleep()* is called, the process is suspended, thus the calling process gives CPU times to other process, thats why it has many voluntary switches and very little nonvoluntary switch.