OS HW3
Alice Jiang & Sonali Singh

**Part 1 – Nice**

In our implementation, we let the nice value range from **1 to 40**, with 40 being the lowest priority and 1 being the highest priority. We initialize the nice value to 20 as default.

The nice command follows this syntax: **nice pid targetNiceValue**

In the program 'nice_test,' we print out the current process status both before and after setting its nice value to 32, 8, 50, and -5. The output shows the nice value after calling nice: the first two calls are successful, while the last two calls are not because 50 and -5 are out of range. An error message will be generated if the values passed in are out of bounds.

To run the test, please type nice_test

```
init: starting sh
$ nice_test

Running NICE TESTS

PS COMMAND OUTPUT - DEFAULT PROCESSES
Parent PID       PID     NI      Alive   State           Name
80150528         1       10      0       SLEEPING        init
1                2       10      0       SLEEPING        sh
2                3       10      0       RUNNING         nice_test


Updating NICE VALUE to LOWER PRIORITY VALUE LIKE 32
PS COMMAND OUTPUT - AFTER NICE CALL FOR ABOVE OPERATION
Parent PID       PID     NI      Alive   State           Name
80150528         1       10      0       SLEEPING        init
1                2       10      0       SLEEPING        sh
2                3       32      0       RUNNING         nice_test


Updating NICE VALUE to HIGHER PRIORITY VALUE LIKE 8
PS COMMAND OUTPUT - AFTER NICE CALL FOR ABOVE OPERATION
Parent PID       PID     NI      Alive   State           Name
80150528         1       10      0       SLEEPING        init
1                2       10      0       SLEEPING        sh
2                3       8       0       RUNNING         nice_test
```

```
Updating NICE VALUE to LOWER OUT OF BOUND VALUE LIKE 50
PS COMMAND OUTPUT - AFTER NICE CALL FOR ABOVE OPERATION
Invalid nice_value (1-40)!
Parent PID       PID     NI      Alive   State           Name
80150528         1       10      0       SLEEPING        init
1                2       10      0       SLEEPING        sh
2                3       8       0       RUNNING         nice_test


Updating NICE VALUE to HIGHER OUT OF BOUND VALUE LIKE -5
PS COMMAND OUTPUT - AFTER NICE CALL FOR ABOVE OPERATION
Invalid nice_value (1-40)!
Parent PID       PID     NI      Alive   State           Name
80150528         1       10      0       SLEEPING        init
1                2       10      0       SLEEPING        sh
2                3       8       0       RUNNING         nice_test
$
```

**Part 2 - Random Number Generator**

We implemented a random number generator using the xorShift method. The test for the xorShift random number generator can be found in rand_test.c. In rand_test.c, we generate 10000 random numbers in the range [1,10], plot a histogram of the distribution of numbers, and then calculate the number of occurrences of each number.

The result shows that the numbers are uniformly distributed. A randomly generated number has a probability of around 10% falling into an arbitrary bin in the range.

To run rand_test.c, type 'rand_test' in the terminal without any arguments.

```
$ rand_test
TESTING FOR XORSHIFT FOR MAX VALUE OF 10

 -- HISTOGRAM --
1: ********************
2: ********************
3: ********************
4: ********************
5: ********************
6: ********************
7: ********************
8: ********************
9: ********************
10: ********************

Number of occurences of digit 1: 965
Number of occurences of digit 2: 952
Number of occurences of digit 3: 1000
Number of occurences of digit 4: 1002
Number of occurences of digit 5: 1010
Number of occurences of digit 6: 1020
Number of occurences of digit 7: 1008
Number of occurences of digit 8: 1015
Number of occurences of digit 9: 996
Number of occurences of digit 10: 1032
```

**Part 3 Scheduler**

**3.1 Explanation of our approach**

In our implementation, each process is assigned a certain number of tickets based on its nice value. Processes with higher niceness values receive fewer tickets, while processes with lower niceness values receive more tickets. For example, a process with a nice value of 40 will receive ten tickets, while a process whose nice value is 1 will receive 400 tickets.

The function total_tickets_num() is called to calculate the total number of tickets assigned to all runnable processes in the process table. Then, a random number is called to randomly select the winning ticket number in the range [1, total_number_of_tickets].

To maintain the policy of assigning tickets based on nice values, we need to ensure that the total number of tickets is updated to reflect the change whenever a process enters or leaves the system. To achieve this, we recalculate the total number of tickets and generate a new random winning ticket before the scheduler carries out each selection.

The function then loops through the process table, calculating the cumulative number of tickets as it goes, and when it reaches a cumulative number of tickets greater than or equal to the winning ticket number, it selects that process as the winner. We switch to the chosen process, and it then begins executing.

This function also makes use of the acquire() and release() functions to ensure that multiple threads do not simultaneously modify the process table, which could lead to race conditions and other synchronization issues.

**Part 3 Scheduler**

**3.2 Testing**

Our test program ‘lottery_test.c’ contains three test cases, each with different nice value updates. To run the test, type ‘lottery_test test_case_number’ where test_case_number is an integer value from 1 to 3.

1. In the first test case, we create a child process and give higher priority to Parent Process and show that the parent process finishes first.
2. In the second test case, we create a child process and give higher priority to Child Process and show that the child process finishes first.
3. In the third test case, we create a child process and give higher priority to Parent Process. In between the loop, we dynamically update the parent process’ priority to low dynamically using nice() and show that the child process finishes first.

```
$ lottery_test 1
Starting LOTTERY TEST - 1
PID of current main process: 3
PARENT WILL FINISH FIRST: PID: 3
Parent PID       PID     NI      Alive   State           Name
-1996453116      1       20      0       SLEEPING        init
1                2       20      0       SLEEPING        sh
3                4       40      0       RUNNABLE        lottery_test
2                3       1       0       RUNNING         lottery_test
Starting DUMMY LOOP for DUMMY COMPUTATION for process: 3
Parent PID       PID     NI      Alive   State           Name
-1996453116      1       20      0       SLEEPING        init
1                2       20      0       SLEEPING        sh
2                3       1       0       RUNNABLE        lottery_test
3                4       40      0       RUNNING         lottery_test
Starting DUMMY LOOP for DUMMY COMPUTATION for process: 4
DUMMY LOOP done for PROCESS: 3
DUMMY LOOP done for PROCESS: 4
$
```

```
$ lottery_test 2
Starting LOTTERY TEST - 2
PID of current main process: 6
CHILD PROCESS WILL FINISH FIRST: PROCESS ID: 7
Parent PID       PID     NI      Alive   State           Name
-1996453116      1       20      0       SLEEPING        init
1                2       20      0       SLEEPING        sh
6                7       1       0       RUNNABLE        lottery_test
2                6       40      0       RUNNING         lottery_test
Parent PID       PID     NI      Alive   State           Name
-1996453116      1       20      0       SLEEPING        init
1                2       20      0       SLEEPING        sh
2                6       40      0       RUNNABLE        lottery_test
6                7       1       0       RUNNING         lottery_test
Starting DUMMY LOOP for DUMMY COMPUTATION for process: 7
Starting DUMMY LOOP for DUMMY COMPUTATION for process: 6
DUMMY LOOP done for PROCESS: 7
DUMMY LOOP done for PROCESS: 6



$ lottery_test 3
Starting LOTTERY TEST - 3
PID of current main process: 8
CHILD PROCESS WILL FINISH FIRST: PROCESS ID: 9
Parent PID       PID     NI      Alive   State           Name
-1996453116      1       20      0       SLEEPING        init
1                2       20      0       SLEEPING        sh
2                8       20      0       RUNNABLE        lottery_test
8                9       20      0       RUNNING         lottery_test
We are going to give higher priority to PARENT Process: 8 but we will change its
 priority later to lowest.
Parent PID       PID     NI      Alive   State           Name
-1996453116      1       20      0       SLEEPING        init
1                2       20      0       SLEEPING        sh
8                9       40      0       RUNNABLE        lottery_test
2                8       1       0       RUNNING         lottery_test
Starting DUMMY LOOP for DUMMY COMPUTATION for process: 8
PROCESS CAUGHT IN IF: 8
getpid(): 8
c_pid: 9
THIS IS TEST CASE 3. Updating NICE VALUE to give PARENT PROCESS LOWER PRIORITY:
8
PS STATUS After UPStarting DUMMY LOOP for DUMMY COMPUTATION for process: 9
DATING
Parent PID       PID     NI      Alive   State           Name
-1996453116      1       20      0       SLEEPING        init
1                2       20      0       SLEEPING        sh
8                9       1       0       RUNNABLE        lottery_test
2                8       40      0       RUNNING         lottery_test
DUMMY LOOP done for PROCESS: 9
DUMMY LOOP done for PROCESS: 8
$
```

The second test case, ‘lottery_test2,’ tests the behavior of the lottery scheduling algorithm by forking child processes and assigning a different "nice" value to each of them. This program takes two arguments from the command line, which are the "nice" values to assign to each child process. To run the program, you can type:

lottery_test2 nice_num_of_child1 nice_num_of_child2
For example, lottery_test2 40 1

The loop in the main function runs ten times. In each iteration, two children are created, and their nice value is adjusted respectively according to the command arguments. Each one of them does some lengthy computation and prints a message when it finishes.

From the output, we can see that, if the difference between priority or nice value is sufficiently large, the child with a higher priority (a lower nice value) finishes first, and the one with a lower priority will finish later.

```
$ lottery_test2 40 1
Starting LOTTERY TEST - 2
child 2 with nice value 1 has finished | pID is: 5
child 1 with nice value 40 has finished | pID is: 4

child 2 with nice value 1 has finished | pID is: 7
child 1 with nice value 40 has finished | pID is: 6

child 2 with nice value 1 has finished | pID is: 9
child 1 with nice value 40 has finished | pID is: 8

child 2 with nice value 1 has finished | pID is: 11
child 1 with nice value 40 has finished | pID is: 10

child 2 with nice value 1 has finished | pID is: 13
child 1 with nice value 40 has finished | pID is: 12

child 2 with nice value 1 has finished | pID is: 15
child 1 with nice value 40 has finished | pID is: 14

child 2 with nice value 1 has finished | pID is: 17
child 1 with nice value 40 has finished | pID is: 16

child 2 with nice value 1 has finished | pID is: 19
child 1 with nice value 40 has finished | pID is: 18

child 2 with nice value 1 has finished | pID is: 21
child 1 with nice value 40 has finished | pID is: 20

child 2 with nice value 1 has finished | pID is: 23
child 1 with nice value 40 has finished | pID is: 22
```

```
$ lottery_test2 30 10
Starting LOTTERY TEST - 2
child 2 with nice value 10 has finished | pID is: 131
child 1 with nice value 30 has finished | pID is: 130

child 2 with nice value 10 has finished | pID is: 133
child 1 with nice value 30 has finished | pID is: 132

child 2 with nice value 10 has finished | pID is: 135
child 1 with nice value 30 has finished | pID is: 134

child 2 with nice value 10 has finished | pID is: 137
child 1 with nice value 30 has finished | pID is: 136

child 2 with nice value 10 has finished | pID is: 139
child 1 with nice value 30 has finished | pID is: 138

child 2 with nice value 10 has finished | pID is: 141
child 1 with nice value 30 has finished | pID is: 140

child 2 with nice value 10 has finished | pID is: 143
child 1 with nice value 30 has finished | pID is: 142

child 2 with nice value 10 has finished | pID is: 145
child 1 with nice value 30 has finished | pID is: 144

child 2 with nice value 10 has finished | pID is: 147
child 1 with nice value 30 has finished | pID is: 146

child 2 with nice value 10 has finished | pID is: 149
child 1 with nice value 30 has finished | pID is: 148
```

```
$ lottery_test2 2 34
Starting LOTTERY TEST - 2
child 1 with nice value 2 has finished | pID is: 214
child 2 with nice value 34 has finished | pID is: 215

child 1 with nice value 2 has finished | pID is: 216
child 2 with nice value 34 has finished | pID is: 217

child 1 with nice value 2 has finished | pID is: 218
child 2 with nice value 34 has finished | pID is: 219

child 1 with nice value 2 has finished | pID is: 220
child 2 with nice value 34 has finished | pID is: 221

child 1 with nice value 2 has finished | pID is: 222
child 2 with nice value 34 has finished | pID is: 223

child 1 with nice value 2 has finished | pID is: 224
child 2 with nice value 34 has finished | pID is: 225

child 1 with nice value 2 has finished | pID is: 226
child 2 with nice value 34 has finished | pID is: 227

child 1 with nice value 2 has finished | pID is: 228
child 2 with nice value 34 has finished | pID is: 229

child 1 with nice value 2 has finished | pID is: 230
child 2 with nice value 34 has finished | pID is: 231

child 1 with nice value 2 has finished | pID is: 232
child 2 with nice value 34 has finished | pID is: 233
```

‘Lottery_test3’ is a slight variation of the last test. In this test, we fork three children and assign a nice value to each of them.

To run the program, you can type:

> lottery_test3 nice_num_of_child1 nice_num_of_child2 nice_num_of_child3
>
> For example, lottery_test3 5 37 16

Some sample test result is shown below:

```
$ lottery_test3 5 37 16
Starting LOTTERY TEST - 5
child 1 with nice value 5 has finished | pID is: 4
child 3 with nice value 16 has finished | pID is: 6
child 2 with nice value 37 has finished | pID is: 5
$ lottery_test3 12 30 28
Starting LOTTERY TEST - 5
child 1 with nice value 12 has finished | pID is: 8
child 3 with nice value 28 has finished | pID is: 10
child 2 with nice value 30 has finished | pID is: 9
$ lottery_test3 40 11 3
Starting LOTTERY TEST - 5
child 3 with nice value 3 has finished | pID is: 14
child 2 with nice value 11 has finished | pID is: 13
child 1 with nice value 40 has finished | pID is: 12
$
```