CSCI 4210 — Operating Systems ♢
Spring 2020 Exam 1 (February 24, 2020)

- You have 105 minutes to complete this exam (i.e., 2:00-3:50PM); note that 50% extra time is 160 minutes and 100% extra time is 210 minutes.
- You may use one double-sided or two single-sided 8.5"x11" crib sheets containing anything you would like; crib sheets will not be collected.
- Please silence and put away all laptops, notes, books, phones, electronic devices, etc.
- Questions will not be answered except when there is a glaring mistake or ambiguity in a question. Please do your best to interpret and answer each question.
- This stapled exam packet will not be collected; therefore, all work done here will not be reviewed at any point. Write all answers on the non-stapled answer sheet to be handed in.
- When you hand in your exam, please show the non-stapled answer sheet and this stapled exam packet.
- There are no syntax errors anywhere on this exam. Note that `#include` directives are omitted to save space on the page.
- For all C programs, assume that `fork()`, `wait()`, `waitpid()`, `read()`, `write()`, `dup2()`, `getpid()`, `open()`, `close()`, `printf()`, `fprintf()`, and `pipe()` all return successfully.

1. **(6 POINTS)** In the code below, what is output to `STDOUT`? Be careful. Circle the best answer.

```
int main()
{
char * s = "Operating systems rules";
s[10] = 'S';
s[18] = 'R';
printf("%s\n",s);
}
```

   (a) `Operating systems rules`
   (b) `<No Output to STDOUT>`
   (c) `Operating systems rules\nOperating Systems Rules`
   (d) `Operating Systems Rules`
   (e) `Operating Systems Rules\nOperating systems rules`

2. **(6 POINTS)** For a 64-bit architecture, how many **total** bytes are allocated on the heap and on the stack by the following code. Circle the best answer.

```
int main()
{
  char * c[20];
  char * s = calloc( 10, sizeof( int ) );
  c[15] = s;
  /* ... */
}
```

(a) 80 stack, 80 heap
(b) 160 stack, 40 heap
(c) 0 stack, 120 heap
(d) 200 stack, 0 heap
(e) 80 stack, 160 heap

3. **(6 POINTS)** Given the code below, what is attached to the first five entries in the program file descriptor table? Circle the best answer.

```
int main()
{
  close(1);
  int desc = open("temp.txt", O_CREAT|O_WRONLY|O_TRUNC);
  rc = dup2(desc, 3);
  /* ... */
}
```

(a) 0:STDIN, 1:<empty>, 2:STDERR, 3:"temp.txt", 4:<empty>
(b) 0:STDIN, 1:"temp.txt", 2:<empty>, 3:"temp.txt", 4:<empty>
(c) 0:<empty>, 1:"temp.txt", 2:STDERR, 3:"temp.txt", 4:"temp.txt"
(d) 0:STDIN, 1:"temp.txt", 2:STDERR, 3:"temp.txt", 4:<empty>
(e) 0:"temp.txt", 1:STDOUT, 2:"temp.txt", 3:<empty>, 4:<empty>

4. **(6 POINTS)** Given the code below, how many processes are created? You should include the initial "parent" process in your count. You can assume all process identifiers begin at 100 and count up. Circle the best answer.

```
int main()
{
    int ctr =0;
    while (ctr < 4) ctr += fork() + 1;
    /* ... */
}
```

(a) `2`
(b) `11`
(c) `5`
(d) `14`
(e) `6`

5. **(6 POINTS)** Of the various **non-preemptive** CPU scheduling algorithms we have studied thus far, which scheduling algorithm is guaranteed to have the shortest average wait time? Circle the best answer.

(a) FCFS
(b) SJF
(c) SRT
(d) RR
(e) None of the above

6. **(6 POINTS)** In the RR algorithm, the time slice is typically chosen ... Circle the best answer.

(a) as 20 milliseconds
(b) to allow approximately 80% of CPU bursts to finish in one time slice.
(c) to require approximately 50% of CPU bursts to take 2 or 3 time slices.
(d) to require only the the longest 0.1% of CPU bursts to require more than one time slice.
(e) as the clock speed of the processor times the number of processes in the wait queue.

7. **(6 POINTS)** What is the **exact** terminal output of the code below? Circle the best answer.

```
int main()
{
  char * s = calloc( 100, sizeof( char ) );
  strcpy( s, "RENSSELAER POLYTECHNIC");
  char * r = s + 20;
  strcpy( r+3, "INSTITUTE");
  char * q = r - 5;
  printf( "[%c][%c][%c][%s]\n", *(s+9), *(q-4), *(r+3), s);
  return EXIT_SUCCESS;
}
```

(a) `[R][P][I][RENSSELAER POLYTECHNIC INSTITUTE]`
(b) `[ ][P][N][RENSSELAER POLYTECHNIC INSTITUTE]`
(c) `[R][P][I][RENSSELAER POLYTECHNIC]`
(d) `[E][ ][ ][RENSSELAER POLYTECHNIC]`
(e) None of the above

8. **(6 POINTS)** In SJT and SRT we estimate process running times based on exponential averaging. What is the formula for exponential averaging? Circle the best answer.

   (a) $\tau_n = \tau_{n-1} * n$

   (b) $x = \frac{-log(r)}{\lambda}$

   (c) $F_n = F_{n-2} + F_{n-1}$

   (d) $\tau_{i+1} = \alpha * t_i + (1 - \alpha) * \tau_i$

   (e) None of a-d

For Questions 9 and 10, use the code shown below. As a hint, when `fork()` is called, any buffered output is copied from the parent process to the child process.

```
int main()
{
  setvbuf( stdout, NULL, _IONBF, 0);
  close( 2 );
  close( 1 );
  int fd1 = open( "fd1.txt", O_WRONLY | O_CREAT | O_TRUNC, 0644 );
  int fd2 = open( "fd2.txt", O_WRONLY | O_CREAT | O_TRUNC, 0644 );

  fprintf( stderr, "01234" );

  int rc = fork();

  if ( rc > 0 )
  {
    wait( NULL );
    printf( "56789" );
    dup2(2,1);
  }
  else
  {
    printf( "FGHIJ" );
    fprintf( stderr, "ABCDE" );
  }

  printf( "KLMNO" );

  close(fd1);
  close(fd2);
  return EXIT_SUCCESS;
}
```

9. **(6 POINTS)** What is the **exact** contents of the `fd1.txt` file when this program completes its execution? Circle the best answer.

   (a) `01234ABCDEKLMNO`

   (b) `FGHIJKLMNO56789KLMNO`

   (c) `FGHIJKLMNO56789`

   (d) `FGHIJABCDEKLMNO56789KLMNO`

   (e) `01234ABCDE`

10. **(6 POINTS)** What is the **exact** contents of the `fd2.txt` file when this program completes its execution? Circle the best answer.

    (a) `01234ABCDEKLMNO`

    (b) `FGHIJKLMNO56789KLMNO`

    (c) `FGHIJKLMNO56789`

    (d) `FGHIJABCDEKLMNO56789KLMNO`

    (e) `01234ABCDE`

11. **(6 POINTS)** Given the code below, what is the contents of the `q11.txt` file when the program terminates? Circle the best answer.

```
int main()
{
  int fd = open( "q11.txt", O_WRONLY | O_CREAT | O_TRUNC, 0766 );
  write(fd, "First", 5);
  close(fd);
  fd = open( "q11.txt", O_WRONLY | O_CREAT , 0766 );
  write(fd, "Next", 3);
  close(fd);
  /* ... */
}
```

    (a) `FirstNext`

    (b) `FirstNex`

    (c) `Nex`

    (d) `Nexst`

    (e) None of the above

12. **(6 POINTS)** The result of running this code would be? Circle the best answer.

```
int main()
{
  int t = 19, u = 70;
  int * v = malloc( sizeof( int ) );
  int * w = calloc( t, sizeof( int ) );
  int * x = NULL;

  *v = t;
  free(w);
  w = x;
  x = v;
  printf( stderr, "%d %d\n", *v, *w );

  *x = 73;
  w = &u;
  fprintf( stderr, "%d %d\n", *x, *w );

  free( v );
  return EXIT_SUCCESS;
}
```

(a) A buffer overflow

(b) A fork-bomb

(c) A segmentation fault

(d) A memory leak

(e) Buffered output

13. **(16 POINTS)** What is the **exact** terminal output of the code below? If multiple outputs are possible, succinctly describe all possibilities. Assume that the initial (parent) process ID is 200, with child processes, if any, numbered sequentially 300, 301, 302, etc.

```
int main()
{
  int p[2];
  int rc = pipe( p );

  fprintf( stderr, "%d: %d %d\n", getpid(), p[0], p[1] );

  rc = fork();

  if ( rc == 0 )
  {
    rc = write( p[1], "RPIirlRPIirlRPIirlRPI", 12 );
  }
  else
  {
    rc = fork();
    if (rc == 0)
    {
      rc = write( p[1], "OS-4210:OS-4210:OS-4210", 12 );
    }
    char buffer[13];
    rc = read( p[0], buffer, 12);
    buffer[rc] = '\0';
    fprintf(stderr, "%d: [%s]\n", getpid(), buffer);
  }
  return EXIT_SUCCESS;
}
```

14. **(12 POINTS)** Apply the SJF scheduling algorithm with the addition of aging to the process CPU bursts described in the table below. For each process, calculate the individual turnaround time, wait time, and number of preemptions experienced by each given process. Aging reduces the estimated running time of a process by 1ms for every 1ms spent in the wait queue. Aging is reset to 0 when a process returns to the wait queue and does not affect the actual burst time. Ignore context switch times and other such overhead.

    Note that all "ties" are to be broken based on process ID order, lowest to highest.

♢	**CPU Burst Time**	**Arrival Time**
P1	12 ms	0 ms
P2	14 ms	2 ms
P3	4 ms	6 ms
P4	4 ms	13 ms

This page intentionally left blank

This page intentionally left blank

This page intentionally left blank